Master's Degree Thesis

Self-supervised contrastive learning with hard positive mining for online action detection

Supervisors

Prof. Francesco VACCARINO

(Ext) Andrea LONZA

Candidate

Matteo LATINO

Master's Degree in Data science and engineering

December 2021

Abstract

Nowadays, self-supervised methods, in particular the contrastive learning methods, have largely closed the performance gap with the fully supervised methods, showing their ability to produce good representations without the requirement of a label, especially on the computer vision where the labelling process is quite expensive. That opens up to the possibility of pre-train a model by using a large amount of unlabelled data being generated all the time and further improve its performance on a wide range of downstream tasks.

In this work, are analyzed and tested some of the most promising self-supervised contrastive methods for computer vision, like *MoCo*, *SimCLR*, *SimSiam* and *BYOL*, to perform the downstream task of online action detection, on a dataset of judo moves. Among them, MoCo had shown the larger improvement, with respect to the baseline obtained from ImageNet's pre-trained representation. Then, to further improve the performance on the downstream task, is proposed a variation that exploits the multi-view property of the dataset. It consists of mining the positive from a different camera's angle, this makes the contrastive task harder, since the positive pair are now more different than before, and the model is forced to produce views invariant representation.

The experiments' results had shown a clear improvement on the downstream task, especially on the discrimination between action and non-action moments inside the clips. This suggests that the view invariant's property is beneficial for the online detection task, it forces the model to concentrate more on the position of arms and legs, which are the elements that usually define the start and the end of the judo moves. Moreover, this shows that different views of the same action can effectively have a similar representation.

Acknowledgements

Before starting, I would like to show my gratitude to all the people with whom I shared these years, here at the Polytechnic of Turin and not only.
First of all, to my parents who always supported and encouraged me, to continue on the path to reach this goal. Then to my friends and university classmate with which I shared my free time, and that helps me to both fight the stress and enjoy these years of study.
A special thanks must be given to the Addfor S.p.A. for the opportunity that he gave me to start this thesis project, and to Andrea Lonza, who guide me during the whole path of research and development, by promptly replying to my doubts and questions, helping and teaching me new skills, that have proved to be very helpful to proceed without difficulty on this work.
Thanks also go to professor Francesco Vaccarino, who accepted to follow me in this thesis path and also supervised the thesis writing process.
Last but not least, to my friends of the Mu Nu chapter of the HKN association, with which I shared many joys and effort, and that helps me to feel less the spacing caused by the Covid19 pandemic.
Thanks to my family and my dearest friends, from whom I learned the importance of improving myself, and following my dreams.

"If at first you don't succeed, try, try again. Don't give up too easily; persistence pays off in the end."

Thomas H. Palmer

Table of Contents

List of Tables

List of Figures

Acronyms

AI
Artificial Intelligence

ML
Machine Learning

DL
Deep Learning

MLP
Multilayer Perceptron

CNN
Convolutional Neural Network

RNN
Recurrent Neural Network

GRU
Gated Recurrent Unit

Chapter 1

Introduction

1.1 Artificial Intelligence

Artificial Intelligence (AI) is a branch of computer science concerned with the production of a machine system able to simulate human thought processes such as learning, reasoning, and self-correction. Its goal is the creation of rational agents that can relate to the real world by sensing the environment, planning their goals, and acting to optimally achieve the best expected outcome. To achieve these goals, these agents must be able to think humanely, act humanely, think rationally and act rationally, in a way that becomes impossible to distinguish their behaviour from the human one.

To check if this system is smart enough to match with human intelligence, *Alan Turing*, considered one of the fathers of computer science and artificial intelligence, proposed to play a test game with a real person versus an AI. Then the corresponding scores, obtained on that game, will be used to compare them.
This test is a two side game played by three entities, on one side there is a human interrogator, on the other side there are: a human and an IA system. The interrogator will pose some questions and from the answers received will try to recognize who is the human and who is the IA. A truly intelligent system should be able to fool the interrogator and pass the test, this imitation game is now known also as the *Turing test*. In order to produce AI smart enough to fool the interrogator and pass the test, have been identified a set of skills that, can be considered necessary for an AI to pass it. These skills are:

- *Natural language process*: the ability to generate complex and meaningful sentences to interact in a heterogeneous society.
- *Knowledge representation*: the ability to have and store the knowledge somewhere, in a way that is possible to access it.

- *Automated reasoning*: the ability to takes decision, based on the stored knowledge.
- *Machine learning*: the ability to learn from the environment, adapt to new circumstances and improve the own capability of generating effective behaviour. Following a trial and error process, an IA can learn how to improve herself.
- *Computer vision*: the ability to distinguish and recognize objects in the surrounding area, including also the ability to follow and trace the movement of that objects in the environment.
- *Robotics*: the ability to handle and manipulate objects to enable interaction with the surrounding environment beyond just language communication.

To acquire those skills many subfields of artificial intelligence have been developed. Unfortunately, this test is quite controversial due to some issues regarding the variability in the interrogator's protocols, like the nature of the questions or the metric to evaluate the answers. For example, if the interrogator could access the responding time of the two entities, he can easily recognize the machine's answer, which usually is many orders of magnitude faster than a person. Moreover, depending on the type of question, the AI could find some *"cheap tricks"* to quickly reach the correct answer, and in these cases, it can't be considered really intelligent.
To partially solve this problem, *Hector Levesque*, proposed a variation of the Turing test, called the *Winograd schema challenge* on which the questions are multiple-choice and have a special composition, they are structured in a way that can be easily answered by a person but, for an AI, find some cheap tricks to easily reach the correct answer, result to be practically impossible.

More in general, all Artificial Intelligence can fall under two main categories:

- *Narrow AI*: Sometimes also called *"Weak AI"*, is a type of artificial intelligence that can operate only in a limited set of contexts and can perform solely a small number of tasks, but extremely well. Usually, these AI focus on just a single skill but take it to the highest level. The majority of them are powered by innovation in the fields of study of machine learning and deep learning.
- *Artificial General Intelligence* (AGI): also called *"Strong AI"*, is an artificial intelligence much more similar to human intelligence, it has a self-aware consciousness, is able to think spontaneously and solve almost any type of problem that it might face. Actually, it is more a theoretical concept, than something tangible. There are several conflicting opinions regarding future developments of this type of AI. Some people think that over the next 50-100 years will be possible to achieve it, while others doubt about the real possibility of creating something like that.

1.2 Machine learning

The word *"learning"* describes the process of getting knowledge, a new skill or a new technique. A rational agent can learn something when its performance in a task, measured by a given performance metric, improves thanks to experiences, both positive and negative, that derive from the previous execution of that task.
Machine learning (ML) is a branch of the artificial intelligence, which focuses on the design of computer algorithms that can learn how to be progressively better on a task, without having been explicitly programmed to perform that specific task.
Following Wikipedia's definition [1], machine learning is the study of computer algorithms that automatically improve themselves, through experience and the use of big amounts of data. Machine learning algorithms build a model based on a set of data, called *"training data"*, in order to perform a task without being explicitly programmed to do so. Those training data come from different sources, like IoT devices, social networks, shopping records or scientific computing, and are usually collected on big datasets.
Machine learning algorithms are used in a wide variety of applications, such as e-commerce, email filtering, browser research and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

1.2.1 Data preparation

As can be noted, the majority of the data collected on the datasets are *raw data* obtained from the real world, usually by following some sensing mechanism. Unfortunately, during this process could happen that some pieces of information are lost, wrong or have different data formats for the same type of data. The model will use those data to learn some patterns or rules, so if they contain some misleading information, using them to train a model could result harmful for its performance.
Thus, before using them to train the model, a pre-processing step is necessary to make those data more reliable and readable by the machine learning models. Thus, on these raw data are performed a *data cleaning* and a *data integration* process to:

- Merge data generated from a different source.
- Manages redundancy that could arise from merging different datasets.
- Reduce the effect of noise information.
- Identify, and if necessary, remove outlines.
- Solves data value conflict.

- Solves inconsistencies.
- Integration of metadata.

This pre-process makes it easier to extract correlations or patterns from the data, which after these steps, comes out organized on a set of pairs composed of:

- *A label*: it Is a tag that identifies some specific entry of the dataset, which can be objects, files or events. Usually, this label is also the target of the ML model, which must be able to recognize or predict it. For example, if the entry is a file audio, then the label can be the words registered on it, or, if the entry is an image, the label can be the name of the portrait object. The label may not always be available.
- *A set of features*: are all the attributes of the entry that are useful to recognize and represent it. Following the previous example, for the file audio, a possible set of features can contain the frequencies of the audio, the amplitude of the wave (min and max values), the duration of the registration etc... . In principle, those features could have any type of data format, however, to be processed by the machine learning model it must be numerical. In those cases, a mapping method is used to convert them to the needed format.

The model is fitted on the training data through a training process where it learns the relation between the features and the label, to predict the correct label given a set of features. As already said, the main consequence of this approach is that the performance of the trained model becomes strongly related to the quantity and the quality reached by the training dataset.

1.3 Deep Learning

Deep learning is a subcategory of machine learning based on the use of new types of models called *Artificial Neural Networks* (ANN) together with a *Representation Learning* mechanism.

1.3.1 Representation Learning

Representation learning refers to a set of techniques with which the model itself will automatically discover the optimal representations, for the input data, to perform a specific task. This replaces the manual features engineering, where an ad-hoc program takes care of extract the features, and pass this work directly to the ML model. A representation consists of all the features that the model consider useful or necessary to perform on the given task in an optimal way.

For example, given an image, the model will automatically recognize the salient pattern like the shape or the relative position of the objects in the image and will extract the features from it. Finally, by using the just obtained set of features, the model will directly perform the given task for which it was trained.

1.3.2 Artificial Neural Network

With Artificial Neural Networks we refer to a specific category of models, used in ML, that is inspired by the biological neural network of humans and animals brains. These models are composed of neurons that receive an input signal, and edges that connect them. Typically, both edges and neurons have a weight that defines the actual state of the network and the intensity of a signal in a given connection.

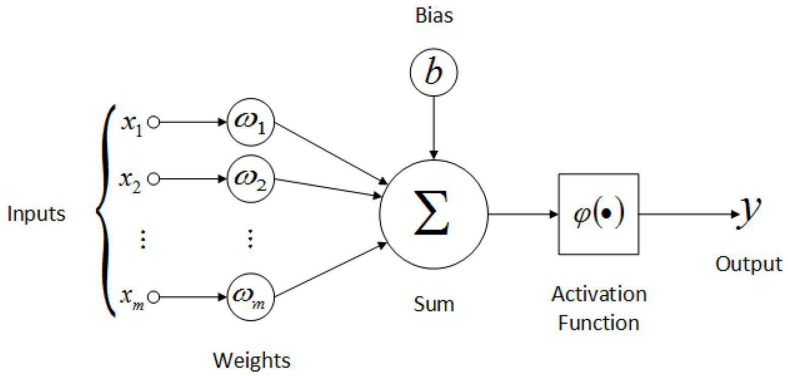

Figure 1.1: Representation of a neuron

The figure 1.1 shows the structure of a single neuron, it will receive as an input signal the vector $x = x_1, x_2, \ldots, x_m$ whose values are multiplied by their corresponding weights $w_1, w_2, \ldots, w_m$. Then, all the elements are summed together with simple arithmetic addition and in the sum is also added a bias value b, which model the sensitivity of the neuron. Finally, the result is fed to an activation function $\varphi(\cdot)$ which will determine the output signal of the neuron. Thus, a single neuron output can be denoted by the following formula:

$$y = \varphi\left(\sum_{i=1}^{m} x_i w_i + b\right) \tag{1.1}$$

Typically, neurons are collected into layers and each layer may perform a different type of operation. Some categories of layers are convolutional layers, fully connected layers, pooling layers and normalization layers.
The values of the weights are adjusted during the training process to minimize an

objective function, also called *loss function*, over all the network weights. Some common loss functions are the mean squared error (MSE), mean absolute error (MAE), adversarial, cross-entropy and quantile loss. To update these weights is used a gradient method with a back-propagation algorithm. With the back-propagation is computed, by a chain rule of the derivative and for a single input-output example, the gradient of the loss function with respect to each weight of the network. This is done so efficiently that becomes feasible to use a gradient method. So, the weights will be updated in relation to the magnitude of their corresponding gradient.
Through the concatenation of different layers, the model becomes able to learn important information about input data, and compress them on a multi-dimensional space. This space is called *latent space* and contains features values that cannot be interpreted directly, but which encodes a meaningful internal representation of examined data.

Three important types of Neural Networks, which form the basic architecture for most of the models in deep learning are the multilayer perceptron, the Convolutional neural network and the recurrent neural network. These models will now briefly described.

Multilayer Perceptron (MLP)

It is one of the most basic feed-forward ANN, and consists of a series of at least three layers, an input layer, one or more hidden layers and an output layer, stacked on top of each other, as shown in figure 1.2. Each one of these layers is a fully connected layer.

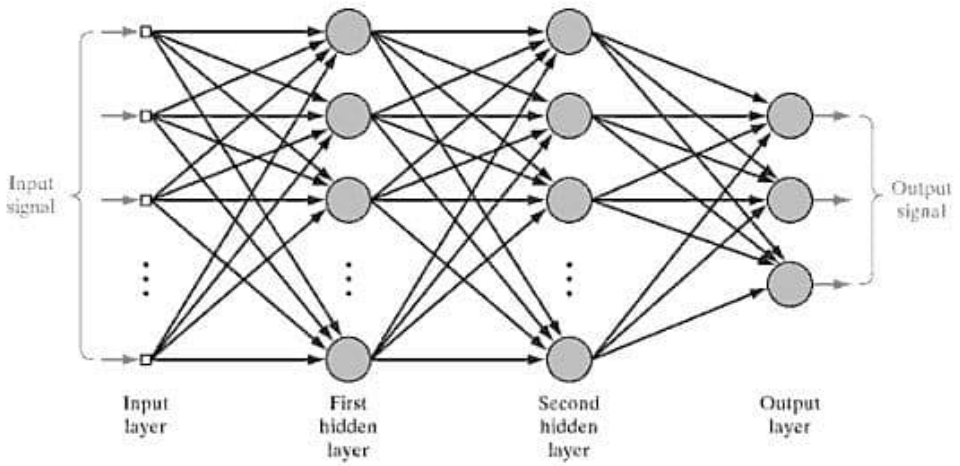

Figure 1.2: Representation of a Multilayer Perceptron Network

On this type of layer, each neuron is connected to all the outputs from the previous layer to processing them through a non-linear activation function. On deep learning, the most used activation function is the *Rectified Linear Unit* ($ReLU$):

$$f(x) = max(0; x) \tag{1.2}$$

Where x is the result of the arithmetic sum of the inputs times the weights. By stacking multiple hidden layers, the model becomes able to produce more complex features, which makes it possible to separate also non-linearly separable data.
When used for the classification task, the output layer must have n outputs neurons, with n equal to the number of classes to be classified. The i^{th} neuron will output the probability percentage that the input belongs to the i^{th} class, so the final predicted class will correspond to the index of the neuron with the highest value.

Convolutional Neural Network (CNN)

Sometimes also called *ConvNet*, is a class of ANN principally designed for working with matrix data, like images or videos. Given an image, it can be represented as a *3-dimensional* matrix where each entry represents the intensity of a colour, between red, green and blue, for each pixel. The architecture's core of CNN is the *convolutional layer*.

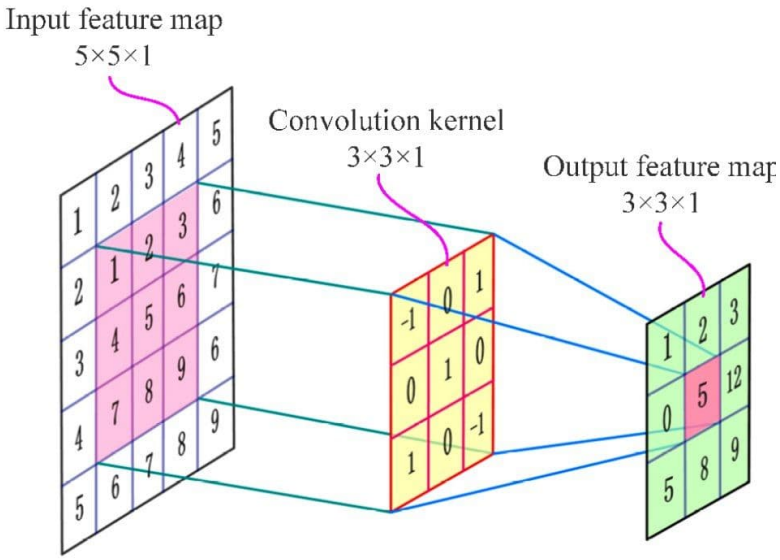

Figure 1.3: Representation of a Convolutional layer

These layers consist of a set of learnable filters, also called *kernels*, which are matrices of weights of dimension equal to the area, of the input matrix, with which these filters will convolve during each step. This convolutional operation consists

of taking a filter, slide it along the input matrix and at each step execute the dot product between the filter and the actually covered region, also called *"receptive field"*. Depending on the number of filters used by the layer, this operation will return one or more features matrices that will contribute to the input of the next layer. The size of the slide is called *stride* and define how many data point the filter must skip after each step.
Given an input matrix of dimension $N \times N$, by applying a filter of size $F \times F$ with a stride S, the size of the output features matrix will be:

$$Size_{output} = \frac{N - F}{S} + 1 \tag{1.3}$$

If the stride is greater than one, sometimes, could be necessary to add a *0-padding* along the margin of the input matrix to ensure that during the sliding of the kernel, it will always perfectly match with the input.

Differently from the fully connected layer, the neurons of the convolutional layer are limited to receive only the input from the receptive field of their filter. Behind these architectural choices, there are two main reasons: the first is that in this way is possible to reduce the number of trainable parameters and reduce the memory size of the neural network, the second reason is to exploit the *local connectivity property* and the presence of *hierarchical patterns* in data. Indeed, each neuron in the first convolutional layer will receive as input, the values of pixels that are near each other and from them will extract features by recognition of simple patterns like points or lines. The following layers will use these initial features to produce more complex ones, that are based on the composition of the pattern recognized on the previous layer. By combining multiple consecutive convolutional layers is possible to assemble complex patterns that are able to produce a high-level representation for the input data.

During the last years have been developed many different models architectures based on the use of the convolutional layers. This thesis will mainly focus on the use of the *ResNet-X* [2], which is an abbreviation of *Residual Network-X*, where the X is a number that indicates the depth of the network. The main component of these models is the *residual block* with the skip connection, which was introduced to solve the vanishing gradient problem.

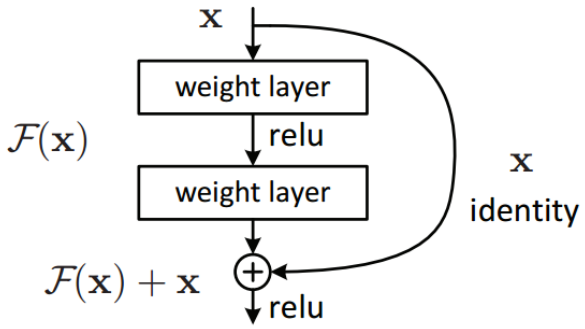

Figure 1.4: Representation of a Residual block

When the gradient is back-propagated from the output to the input layer, it becomes progressively smaller, and with deeper CNN could be reached a point where it is too small to make the update of the weights effective. With the residual block, the signal from the previous layer is added to the actual output, to keep the gradient more stable.

Recurrent Neural Network (RNN)

This type of network was mainly thought to operate on sequential data like sentences, financial data, weather data, video or audio, to solve tasks like machine translation, weather forecasting, speech recognition or handwriting recognition where the temporal position of each input element is meaningful. RNN models are able to keep the memory of past information and use them to infer the final prediction. This memory is maintained thanks to a loopback connection on the recurrent cell and an internal state h which is continuously updated while successive inputs come in.

On figure 1.5 is shown the base structure of a recurrent network. At the time t, the network is on the state h_t, it receives the input vector x_t and will produce the output y_t. If the parameters of the model are defined by W, the state h at the time t can be computed using the following formula:

$$h_t = f_W(h_{t-1}, x_t) \tag{1.4}$$

Where f is a non-linear function with parameters W and depends on both the previous state and the actual input. To notice that, the same function and the same set of parameters will be used for all the time steps, only the internal state and input could change during the process.
While the output y at the time t can be obtained as:

$$y_t = W h_t \tag{1.5}$$

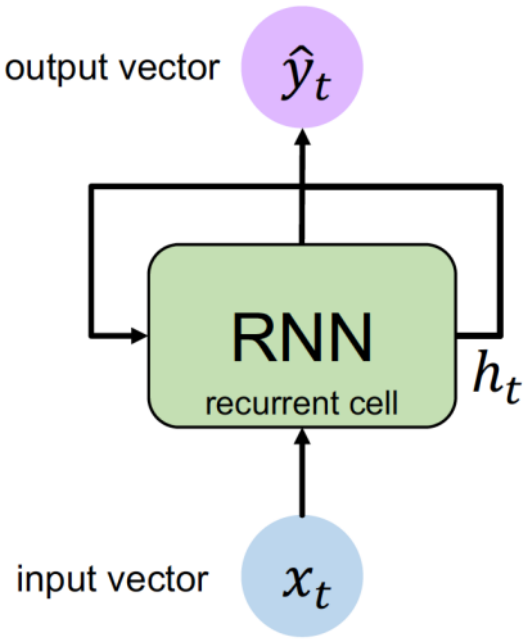

Figure 1.5: Representation of a RNN

It will depend on the parameters of the network and its actual state.
This work will use a *GRU* net (*Gated Recurrent Unit* network), which can be considered as a variation of the *LSTM* net (*Long Short-Term Memory*), where thanks to two gates called update gate and reset gate, is possible to keep memory of long ago information and removes the ones that are useless for the task.

1.3.3 Learning methods

Modern network architectures are able to solve a large number of tasks, from the easiest to the more complex. To train such a neural network there are different algorithms, usually called learning methods. Each one focuses to solve a specific category of tasks, however, the choice of the most adequate method depends not only on the task to perform but also on the availability of labelled data and on the dimension of the training set. These learning methods can be divided into:

1. **Supervised learning:** This method can be used to train models for classification, regression and many other problems, the labels' information is used as supervisory signals to recognize underlying patterns and relationships between

the input data and the output labels.

2. **Unsupervised learning:** Unlike the supervised method, this one is used when the training data comes without the labels, so this approach is mainly used to discover useful patterns into the input data that could help to solve clustering or association problems. This is also particularly useful when subject matter experts are uncertain of common properties within the data.

3. **Semi-supervised learning:** This method is applied when only a small part of the given input data has been labelled, it is a special instance of weak supervision. The two main problems that can be solved using semi-supervised learning are transductive learning and inductive learning.

4. **Self-supervised learning:** Like the unsupervised method, also this method, is used when the training data have no labels. It consists of training a model on a task, usually called *pretext task*, for which the labels associated with the input are automatically generated with a predefined program. However, the objective of this process isn't to obtain a model able to solve that task, which usually is a dummy task, but is to obtain a model able to produce a good representation on a latent space for the samples contained in that dataset.

5. **Reinforcement learning:** It concerns the modelling of an agent that interacts with an environment. This agent takes action that rewards him and modifies the state of the environment on which the agent moves. The objective of the reinforcement learning is to produce an optimal policy for the agent in order to maximize the expected cumulative reward. Differently from the other training methods, this one does not use collected data but focuses on the effect of the action taken by the agent to find a balance between exploration and exploitation of the environment's states.

Among these learning methods, the most common is the supervised learning method. The knowledge of the labels is a very strong supervisory signal, that helps to find the optimal set of parameters for the neural network. When it is combined with a big labelled dataset, the obtained models can easily reach high-level performance in the majority of the tasks. However, obtaining an amount of labelled data, large enough to ensure good performance with the supervised training, is difficult and time-consuming, especially for some particular tasks like autonomous driving, or in particular areas, like the economic and medical one, on which the labelling process is complex, requires some specific domain knowledge, and sometimes even the domain experts can be undecided about the correct labelling process. Moreover, in the last few years, become also relevant the inability of the models, trained in a supervised way, to produce good representation for samples outside or at the edge

of the training data distributions.

This problem is also known as the *overfitting problem* and occurs when, with the aim to perform optimally on the training dataset, the model learns also small details or noise that result useful to operate on those data but are useless or even harmful, when it comes to operating on a little different data distribution. It becomes also more relevant when the training set is small and does not contain enough variance to easily cover all the possible input scenarios. In this case, the deep learning models tend to easily overfit on that little number of samples, by producing features representations that are not generalizable to other similar samples that were not seen during the training phase.

Nowadays, to overcome the overfitting problem, alongside the different regularization techniques like the use of data augmentation function or dropout layer, is progressively increased the use of self-supervised learning methods to learn useful features, transferable on a set of different supervised tasks.
The next chapter will analyze the modern self-supervised learning methods for computer vision when it comes to performing tasks like image classification, object detection or instance segmentation.

Chapter 2

Self-supervised learning methods

As already said, the goal of the self-supervised learning methods is to obtain a model able to produce a good and transferable representation for the data contained on a dataset. This is usually done without the use of any manual obtained label, but through the training of the model to solve a task, called *pretext task*, for which labels are produced thanks to a predefined fixed program and not by manual labelling done by some domain expert. Once the self-supervised training ends, the model can be used to solve a second supervised task, called *downstream task*, like classification, object detection, instance segmentation, etc... .
Thus, used as an initialization point, it will be fully retrained or fine-tuned, in a supervised way, to solve the new task.
Therefore, at this point, some questions could arise:

1. **What is a good representation? When can a representation be considered a good one?**
   As indicated on [22], it can be defined as a representation that is useful for downstream tasks like object detection, image classification, etc. The quality of a representation can be estimated by directly measuring its performance on these downstream tasks, and usually, this evaluation is carried out by following the *linear evaluation protocol*.
   It consists of training, on top of the neural network, previously trained with the self-supervised methods, a new model to perform the new downstream task. In the case of a classification task, will be trained a linear classifier. During this second training, all the weights of the pre-trained neural network are frozen to remove any possible variation of their values, due to the back-propagation of the gradient.
   In this way, will be trained, only the new model, while the pre-train network,

sometimes indicated as *"backbone"*, will work as a fixed features encoder.
This method makes it possible to understand how good the learned representations are to perform the new task, and actually, it is considered the only effective way to directly evaluate the quality of the representation produced by a self-supervised method. The main drawback is that this process is heavy and time-consuming.
Each time that, must be tested a variation on the training method or is done a tuning of a parameter, is needed a full training of two models, first the backbone and then, the model for the downstream task. Moreover, this process does not ensure that the obtained features are truly generalizable to different downstream tasks and data distribution. It is thought to operate on a single downstream task and for a given dataset at a time, use it to check the performance of the same method, on multiple tasks and datasets could result in a very long and unsustainable process.
Sometimes, instead of test directly the learned representation, could be faster and convenient, analyze its invariance property, which is crucial to obtain truly generalizable features. With an invariant representation, a model is able to maps the input x into the output y, even if x is modified by some function, that however does not change the ground-true label for the task. So, even if the model receives as input a variation of x, defined by x^*, he will still return the output y.
Many modern self-supervised methods focus on producing transformation invariant representation, by enforcing some constrain during the training.

2. **Why use self-supervised learning, if in the end is also necessary a supervised training? Why not train directly in a supervised way?**
There are different reasons for which pre-train the model in a self-supervised way can be prefered to the standard supervised training only, the mains reasons are:

   - Self-supervised methods produce more general representations. The supervised method tends to easily overfit the training dataset and as a consequence, the models have difficulty in operating on samples outside the training data distributions. The cause of the overfitting can be related to the supervised signal that comes from the class label, which can give a strong bias to the network's weights. With the self-supervised methods, no class label is used and as consequence also that bias is removed from the representation learned by the network.
   - The same backbone can be reused for different downstream tasks without the necessity of retraining the full model from zero, but with just a small fine-tuning of the weights. This help to reduce the training time, but can

be done only if the distribution of the data used for the pre-training is similar to the distribution of the data used during the fine-tuning.

- Big unlabelled datasets can be easily obtained for free, and by pre-training on these datasets, the model has access to a huge variety of data samples, which otherwise couldn't be accessible because not labelled. Those data could help him to produce very general representations useful on a wide range of tasks.
- A big labelled dataset is not always available and supervised training on a small dataset, usually, have poor performances. However, by using two different learning approaches, two different signals are extracted, thus leverage better the whole dataset, and helps on improving the performance when only the small labelled dataset is available.

All the self-supervised training methods can be divided into three main categories:

- Pretext-task based methods
- Generative methods
- Contrastive learning methods

2.1 Pretext-task based methods

The pretext task-based methods are strongly focused on the assumption that the representation learned by a model to solve a first task, called *"pretext task"*, can be also useful to solve other different *"downstream task"*. They propose to design a well-defined pretext task, which can be solved by the model, only if it becomes able to produces representations with some desired characteristics, like the invariance to a given transformation, the focus on the orientation of the objects in the image or the ability to recognize the relative position between different objects in the same image.
Some of those methods use a given data augmentation transformation to modify the input image, and then, from the knowledge of the variation applied, create the artificial label that will be the supervisory signal for the training process. Others instead, propose to divide the initial data into smaller sets and use one of them to infer some prediction on the attributes of the remaining parts. In this way is possible to obtain the supervisory signals from the data itself, without the use of any label.
Following are described some of the most popular pretext-tasks based methods, present in the literature.

2.1.1 Predicting image rotations

Firstly proposed by Gidaris et al., the input image is randomly rotated by a multiple of 90°, then by having access to both the initial and the rotated version of the same image, the model is trained to predict which degree has been applied. It can be considered as a classification task with 4 classes, one for each possible rotation angle: 0°, 90°, 180° and 270°.

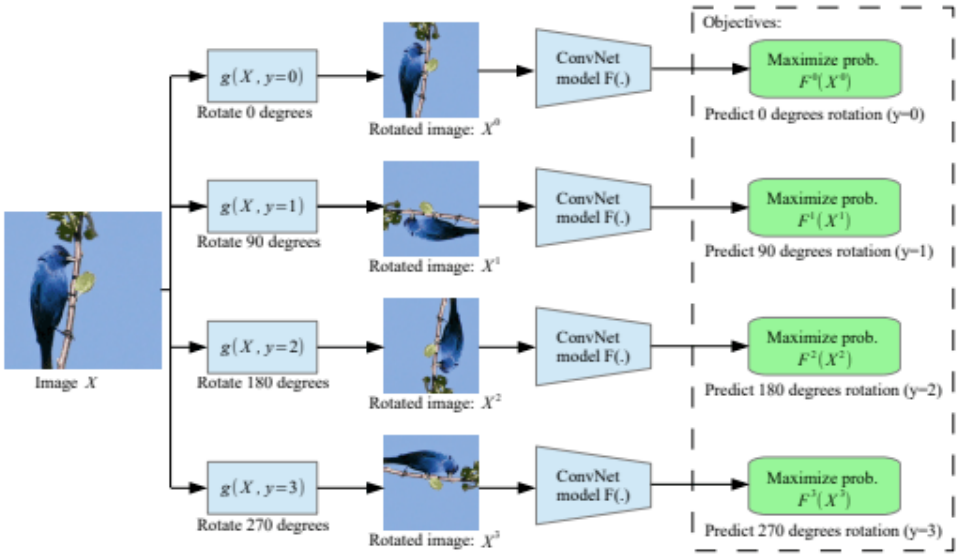

Figure 2.1: Illustration of the self-supervised rotation task

To solve this task the model must learn to recognize high-level details of the image regardless of their absolute position. Moreover, this method enforces the model to produce a representation invariant to the relative rotation of the input image.

2.1.2 Patches relative positioning

Instead of working on the complete image, like on the previous method, where the full image is rotated and analyzed, this one proposes to focus only on a piece of each image at a time, to maximize the spatial information that can be extracted from that portion of data.
Doersch et al. propose to extract multiple patches from the same image, and then ask the model to recognize some relationship between these patches.

As shown in figure 2.2, first is extracted a random patch from the full image, then, by considering it as the centre for a 3×3 grid, is extracted another patch from one of its eight possible neighbours locations. The model will be trained to

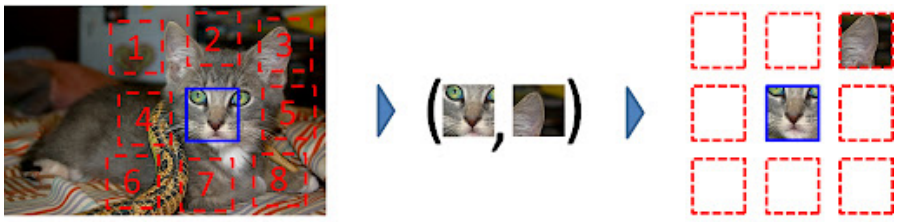

Figure 2.2: Illustration of the self-supervised relative positioning task

recognize the relative position of these two patches, by predicting among which of the 8 possible locations, have been sample the second patch. It looks like a classification task with 8 classes.
To avoid trivial, solutions based on the use of lines across boundaries or matching of local patterns, the sampling introduces a gap between the patches. Moreover, on each patch are added some noises, small jitters and a little shift toward the grey for the green and magenta channels. This is done to avoid trivial solutions caused by the *chromatic aberration* [16], that allow the model to solve the task by analyzing the relative offset between the two colour channels and ignoring the actual patches content.
To perform well on this task, the model must be able to recognize the different part of the same object and their relative position.

2.1.3 Solving Jigsaw puzzle

The jigsaw puzzle is a type of puzzle where different pieces, each one with a particular irregular shape, must be assembled together to form a single block that usually contains an image. Each piece will contain a small portion of the final image, as consequence, its position in the puzzle is unique and not interchangeable with any other pieces.

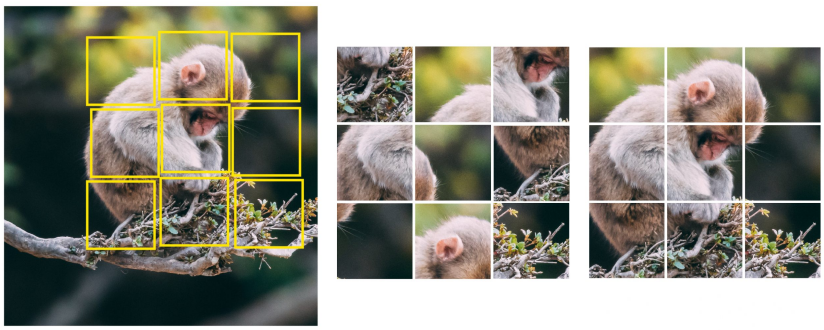

Figure 2.3: Illustration of the Jigsaw puzzle task

Inspired by this puzzle game and following the patches method, Noroozi and Favaro propose a self-supervised method that uses the jigsaw puzzle game as a pretext task.
Given an image and an initial position, is set a 3×3 matrix with that position as the centre, then 9 random patches, one for each location, are sampled and shuffled. Obviously, the task will consist of repositioning all the 9 pieces in their correct position. However, given the high number of possible permutations, Noroozi and Favaro proposed to regulate the difficulty of the task by fixing the possible set of permutations. Moreover, also in this case to avoid trivial solutions thanks to the lines across boundaries, the sampling process includes the addition of some noise, small jitters and small gaps between the patches, as shown in the figure 2.3.

2.2 Generative methods

These methods are based on the use of *Autoencoders* (AE), *Variational Autoencoders* (VAE) and *Generative Adversarial Network* (GAN), all these models are mainly used on the computer vision to create or reconstruct an image from its encoded features.
For example, if it's considered a basic autoencoder like in figure 2.4, it is composed of two main components:

- An encoder G parametrized by θ.
- A decoder F parametrized by ϕ.

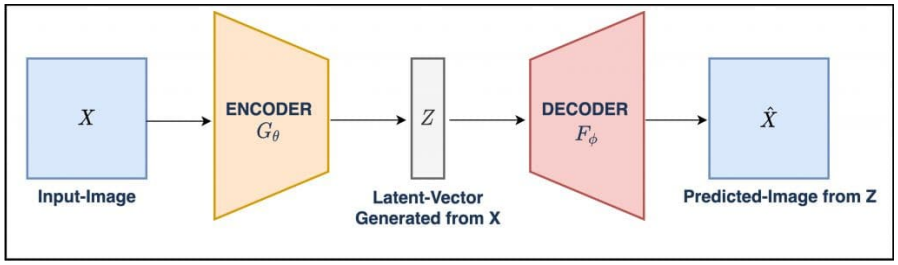

Figure 2.4: Illustration of an Autoencoder

First, the encoder produces a compressed latent representation z of the input images x, then the decoder uses this representation to output a reconstructed version $\hat{x}$ of this last one. In this case, the supervisor signal needed for the training of the network is the input image itself, indeed the loss function is defined as the difference between the original image x and the reconstructed version $\hat{x}$. To measure this difference there are different possible choices, like for example the

MSE loss:

$$L_{AE}(\theta, \phi) = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)} - F_{\phi}(G_{\theta}(x^{(i)}))^2 \tag{2.1}$$

During the training, the parameter (θ, ϕ) are learned together, in order to produce a reconstructed image as similar as possible to the input one.
This process provides some guarantees on the quality of the latent representation produced by the encoder. It will contain for sure, high-level details that are necessary, for the decoder, to produce a high-quality reconstructed image. The level of the details obtained seems to be correlated to the loss function used for the training of the model. Modern VAE and GAN mainly use an *adversarial loss* which gives more importance to small details of the images, using a different loss function could lead to worse performance on the reconstruction task.

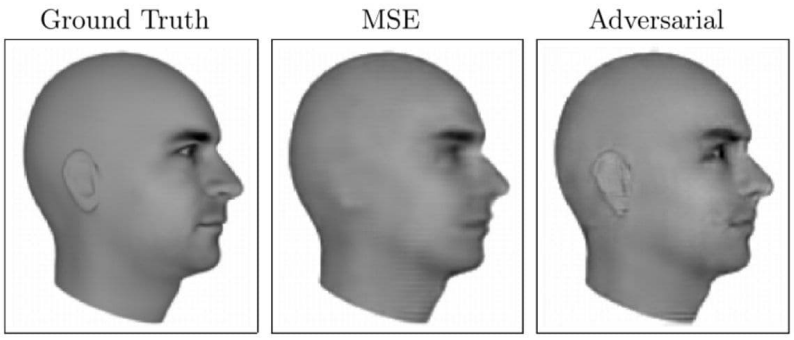

Figure 2.5: Illustration of the effect of the loss function on the a reconstruction task

In figure 2.5 it is possible to see the effect of the loss function on the reconstruction of a $3D$ face. The first image is the ground truth, the other two are produced by using the same model, but trained with different loss functions. The second image uses the MSE loss, its resolution quality is much lower than the original, especially near the nose and the mouth, moreover, the ear is missing at all, it does not cause enough difference in brightness with respect to the skin, so it was not sufficiently salient for the model to encode its representation. The last one instead, uses the standard adversarial loss, both nose and month are much more accurate, and the ear is correctly reconstructed. With this loss function, the ears become salient because it follows a regular pattern easily predictable.
The base idea of these methods is to use the encoder as a feature extractor to perform other downstream tasks on a similar image distribution. Unlikely, have been noted that the high-level information learned by these encoders, sometimes can be considered useless or even harmful when it comes to performing other

downstream tasks, like classification or object detection.
Now, will be shortly described some proposed frameworks, that use this self-supervised method.

2.2.1 Denoising autoencoder

This method, originally proposed by Vincent et al., is inspired by humans ability to recognize the object into an image even if it is partially corrupted, proving that is possible to extract key visual features also from a corrupted image.
This is a variation of a standard autoencoders where the input images are corrupted before being given as input to the NN. The corruption process usually includes the addition of some gaussian noise, masking noise, jittering, etc. which are controlled by a stochastic mapping $M_D(\tilde{x}|x)$. Given an image x, through the mapping M is obtained the image $\tilde{x}$ similar to x but less clear and more difficult to recognize.
The model will be trained to bring back the image to its initial condition, without noise or other problems. To perform optimally on this task, the encoder must be able to separate the noise from the useful visual information and then encode only these last ones on the latent representation. Finally, the decoder will use this representation, free from noise information, to reconstruct the initial image.

2.2.2 Context encoder

Similarly to the denoising autoencoder, this method is based on the corruption of the input image before being fed into the NN. In this case, rather than adding some noise, the corruption will consist of removes a part of the image. That section will be delimited by a mask that could have different possible shapes.
The objective of the model will be to reconstruct the missing part by having access to only the remaining portion of the image. To do that, the model must be able to understand the context of the image and the missing information that must be added to complete the image.

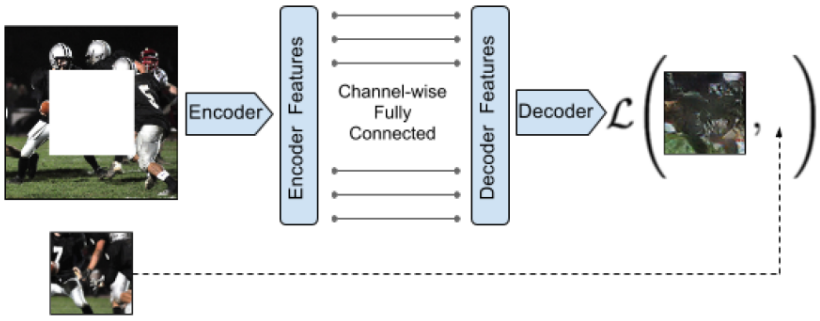

Figure 2.6: Illustration of a Context Autoencoder proposed by Pathak et al.

2.2.3 Split-Brain autoencoder

This method, like the others, follows the idea of removing some information from the initial images to make the task more difficult for the encoder, which must be able to extract good visual features from only a portion of the input data.
This time, Zhang et al. proposes to drop some colour channels from the input image and use the remaining one to reconstruct the part that was removed. As already said, the images can be visualised as a $3D$ matrix $x \in R^{h \times w \times c}$ where h and w indicate the height and the width of the image and c the number of colours channels. Given an input image x, it is split into two disjoint parts x_1 and x_2, with the same height and width of x but using a different set of colour channels.
To accommodate this new type of input, also the model will be internally split into two sub-network F_1 and F_2, to perform a complementary prediction task, where F_1 will use x_1 to predict x_2, while F_2 will use x_2 to predict x_1.

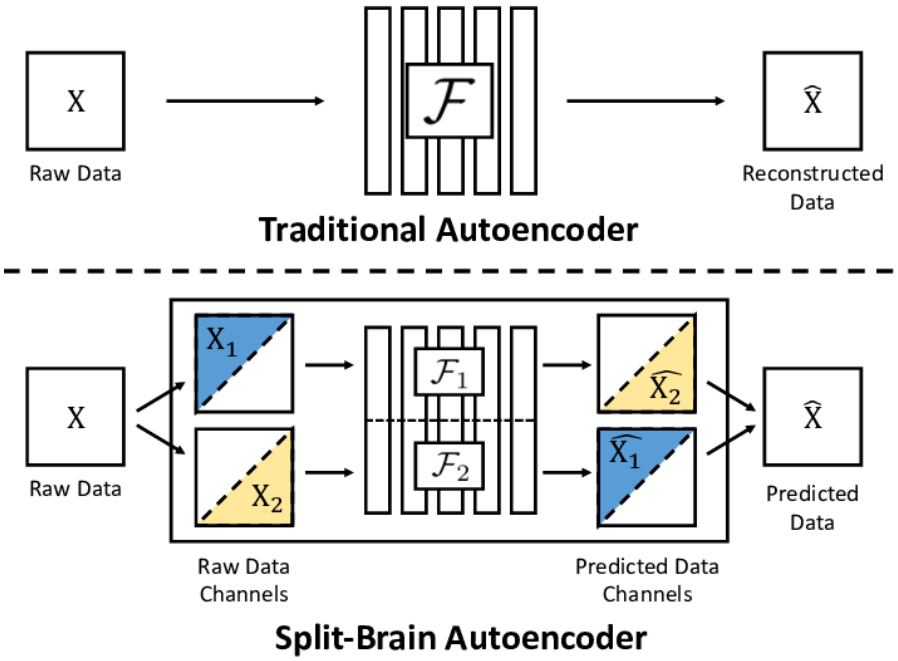

Figure 2.7: Illustration of the difference between a standard Autoencoder and the Split-Brain Autoencoder proposed by Zhang et al.

2.3 Contrastive learning methods

This category contains the actual *state-of-the-art* regarding the self-supervised training approach. The base idea of the contrastive learning methods is to teach

the network which data are similar, by pushing their representation near each other, and which ones are different by pulling their representation far away, in the latent space. The loss function, used by these methods, is called *contrastive loss* and comes from a variation on the *triplet loss*.
Given an image A called *anchor*, the triplet loss will assign a *positive* image defined by P, and a *negative* image defined by N, for that anchor. By means of the Euclidean distance function [3], this loss can be written as:

$$L(A, P, N) = max(\|f(A) - f(P)\|^2 - \|f(a) - f(N)\|^2 + \alpha, 0) \tag{2.2}$$

Where f is a function that projects the input image into the latent space, and α is a parameter that defines the minimum margin, that must be enforced between the positive and the negative images.
Then, through the learning process, the distance between the anchor and the positive will be minimized while at the same time the distance between the anchor and the negative will be maximized. As a consequence, since the anchor and the positive will be pushed near each other, also the distance between the positive and the negative will be maximized indirectly.

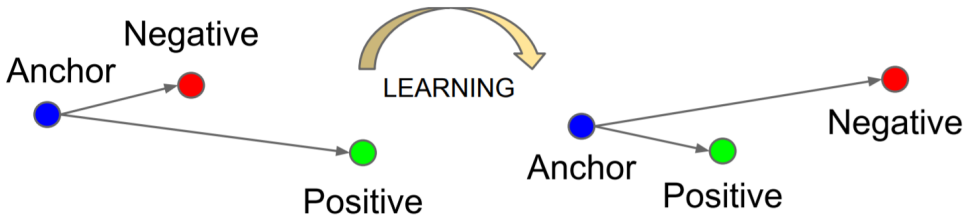

Figure 2.8: Illustration of the triplet loss

While training a model with this type of loss function, to observe an effective improvement in the performance of the model, is crucial the choice of challenging pairs. If the positive is already nearest to the anchor than the negative, the model will learn almost nothing since the requirement about the relative distance of the representation is already satisfied, while if are sampled a positive far away from the anchor or a negative very near to it, the improve obtained by the model, during each step of the training, is much greater.
Thanks to the use of both positive and negative samples, this method has proved to be very stable during the training phase.

The contrastive losses follow the same structure of this loss function, where instead of samples only a positive and a negative image for each anchor, are sampled more than one negative image for the same anchor-positive pair. So, when will be enforced the distance between the anchor and the negative image, it will be done

for all the negatives at the same time.
Thus, by extracting a random pair of images from a given dataset, it can be classified as:

- **Positive pair**, if both images contain the same, or at least similar, semantics. For example, can be considered as a positive pair, all the pairs, whose elements come from the same class. For this type of pair, it's reasonable to expect a similar embedding representation between the two elements, for this reason, during the training, the model will try to push that images near each other, on the latent space.
- **Negative pair**, if the images contain different semantics. Can be considered as a negative pair, all the couples, whose elements come from different classes. For this type of pair, it's desirable to have considerably different representations, in order to discriminate between them more easily. To achieve this, during the training, the model is forced to produce dissimilar representations for these two images.

Figure 2.9: Examples of positive and negative pairs

Figure 2.9 shows an example of positive and negative pairs, the positive pair consists of images of two cats, even if the races are different the semantic and the information contained remains similar since the animal is the same. The negative pair instead contain the images of a cat and a dog, as consequence, since the animals are different, also the semantic and the information contained is different, however *"similar"* that animals may be. So, the objective will be to obtain a model

able to produce representations that are near each other in the first case and be far away in the last case.

Among the different contrastive learning frameworks present in the literature, some of most the representatives in the computer vision are:

- **MoCo:** Momentum Contrast
- **SimCLR:** Simple Contrastive Learning
- **BYOL:** Bootstrap Your Own Latent
- **SwAV:** Swapping Assignments between Views
- **SimSiam:** Simple Siamese network

In the next chapter, these and other frameworks will be analyzed and compared on a set of common downstream tasks.

Chapter 3

Contrastive learning frameworks

3.1 Contrastive learning on images

This section will present the contrastive learning methods designed to operate on images data, so with a particular focus on extracting optimal spatial information. All these methods use some data augmentation techniques, which are essential to learning a useful and generalizable set of features. It adds some non-essential variation to the images, guiding the model to recognize the main details and patterns useful for producing a good representation. Moreover, these frameworks are mainly thought to operate on an object-centric datasets where the main subject is placed on the centre of the image.

3.1.1 MoCo

He et al. proposes an interpretation of the contrastive task as a dictionary lookup problem; it consists of finding, into a dictionary of keys, the one that matches the query. The main assumption of this method is that the query, should be similar to its matching key and dissimilar to all the other keys, likewise, in the contrastive task, an image should be similar to its positive and different from all the other negatives. Both the query and all the keys will be encoded views of images contained in the dataset.
So, maintaining the parallelism between the two tasks, the query and its matching key form a positive pair, while the query together with any other possible keys, compose a negative pair. In particular, the query and its matching key are encoded views of the same image but following different data augmentation pipelines. The data augmentation setup is tuned in a way that the images look different but the

semantic of the two augmented views remain the same.

On figure 3.1 is shown the architecture of this framework. It is composed of two encoders, the first one is the main encoder f_q and will produce the query q; while the second encoder f_k, called *momentum encoder*, will provide the keys $k_0, k_1, \ldots, k_n$ for the dictionary, among them, the only one that matches the query will be defined as $k+$. These two encoders will have the same CNN architecture but with different sets of weights.

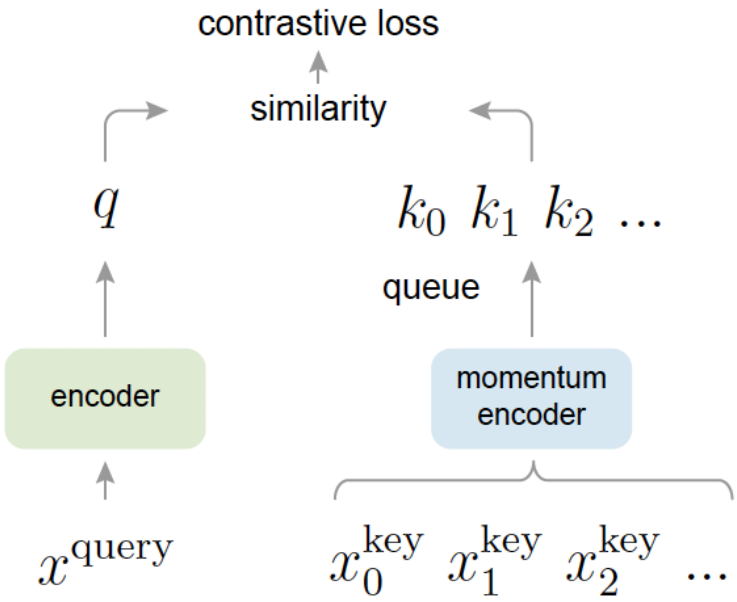

Figure 3.1: Illustration of MoCo

The dictionary is implemented as a queue, this allows to decouple the number of negative samples from the size of the mini-batch used for the training of the model. During each step of the training process, a new mini-batch of keys is produced by the momentum encoder and enqueued, while the oldest one is removed from the queue, in this way is possible to reuse the encoded key from the preceding mini-batch as a negative for the actual query. The idea is that a larger dictionary, which covers a wide set of negative samples, can help to learn a good representation, stabilize the training and speed up the convergence of the model. Moreover, removing the oldest mini-batch from the queue allows maintaining a dictionary of keys more consistent with the actual state of the encoders.

Unfortunately, this approach has some drawbacks, the main one is that updating the weights of the momentum encoder by back-propagation becomes infeasible,

since the gradient should be calculated by considering also all the samples in the queue. To address this issue He et al. propose a method called *momentum update*, from here the name momentum encoder. It consists of updating its parameters through a weighted average between its actual values and the values of the main encoder, the one which produces the query. More in detail, by denoting with θ_q the parameter of the first encoder, and with θ_k the parameters of the second one, θ_k will be updated as follow:

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q \tag{3.1}$$

Where $m \in [0,1)$ is a hyperparameter called *momentum coefficient* which regulates how much, from the actual parameters of the momentum encoder, must be maintained. If $m = 1$, the weights of the encoder will never be updated and will remain equal to their initial value, if $m = 0$, the values will be completely overwritten with the ones from the main encoder. From the experiment results that a relatively large momentum like 0.999, works better than smaller values, suggesting to keep slow the progression of the momentum encoder.
Once obtained both the query and the queue of keys, is possible to compute the contrastive loss, MoCo uses the *InfoNCE* loss:

$$L_q = -log\frac{exp(q \cdot k_+/\tau)}{\sum_{i=0}^{K} exp(q \cdot k_i/\tau)} \tag{3.2}$$

Where K is the number of negative samples, τ is a temperature hyper-parameter and the metric used to compute the similarity between the query and a key, is the dot product. Its value is low when q is similar to its positive key $k+$ and dissimilar to all other keys. This loss is the log loss of a (K+1)-way softmax-based classifier that tries to classify q as $k+$.

From some tests, He et al. noted that the model appears to cheat on the contrastive task and easily find a low loss solution that leads to a bad representation for the final downstream task. This seems caused by a leak of information between the samples on the same batch, inside the batch-normalization layer of the encoders. To avoid this problem they propose to shuffle the samples among the different GPUs used for the training of the model, before the batch-norm layer. Then the batch normalization is performed independently for each GPU, and finally, the samples are shuffled back to their initial position. This ensures that the batch-statistic used to compute the query and its positive key comes from different subsets.

He et al. propose also a subsequential variation of this framework called *MoCo v2* that implement some features, inspired by the *SimCLR* framework, to improve the quality of the final representations and the performance on the downstream task. In particular, they implement:

- An MLP projection head: before applying the contrastive loss, the representations produced by both ConvNets are projected on a smaller latent space through the use of small MLP nets.
- Stronger data augmentation: the data augmentation pipeline, used to produce the two views of the same image, is extended with more transformation functions like the blur augmentation proposed on SimCLR.

Once the training is ended, the momentum encoder and both the projection heads are discarded, only the main encoder will be kept and used as a backbone for the downstream task.

3.1.2 SimCLR

This framework proposes to perform the contrastive task without the use of an additional structure like the queue, but by exploiting all the information already contained on the current mini-batch processed by the model. Just like on MoCo, a positive pair will be obtained from two augmented views of the same image, while the negatives will be provided by all the other images included on the same mini-batch of the current positive pair. The objective is to maximize the agreement between the positive pairs and at the same time, repel all the other images from them. This is done through the projection of the images on a latent space, where the contrastive loss will be applied.

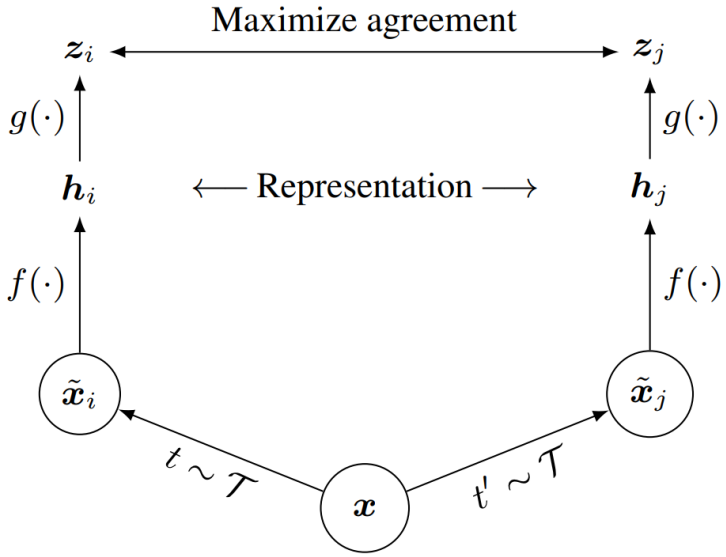

Figure 3.2: Illustration of SimCLR

This framework is composed of four main components:

1. stochastic data augmentation module, it consist of a pipeline of operations composed of multiple data augmentation functions. This process is crucial to ensure a challenging contrastive task between different augmented views of the same images. Figure 3.3 show the data augmentation operator that composes this module, and are:

- Crop, Resize and Flip: crop only a part of the image, reduce its resolution and randomly flip it along the y axis.
- Colour distortion: modify the intensity of each colour, permute the different channels among them or apply some jittering to obtain different effects on the final image.
- Rotation: randomly rotate the image of a given degree between 90°, 180° and 270°.
- Cutout: remove some parts of the image by covering it with some grey squares.
- Gaussian noise and blur: make the image less clear at the sight.
- Sobel filtering: is a filter that allows to accentuate the edge of the image, at the cost of losing colour and background information.

Each augmentation can be applied with a given probability and transform the images in a stochastic way with some internal parameters, like rotation angle, noise level, etc.

2. A base Neural Network encoder $f()$, that extracts the representation vector h from the input image. This is also the backbone that will be used on the downstream task. It can be any type of CNN, actually is used a ResNet model.

3. A small MLP network $g()$, it is a projection head that maps the representation h on a smaller space with a ReLU non-linear transformation. Applying the contrastive loss on these projections z rather than on h, results in improving the quality of the learned representation and speed up the computation of the loss.

4. A contrastive loss function for the contrastive task, Chen et al. propose to use a *NT-Xent loss* (Normalized temperature-scaled cross entropy loss) [8], it is a variation of the infoNCE loss used by MoCo, where the distance metric is the cosine similarity that, for a given pair (a, b) can be written as:

$$sim(a, b) = \frac{a^T b}{\|a\|\|b\|} \tag{3.3}$$

For a given positive pair of images (x_i, x_j), the loss can be defined as:

$$l_{i,j} = -log\frac{exp(sim(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbf{1}_{k\neq i} exp(sim(z_i, z_k)/\tau)} \tag{3.4}$$

Where $\mathbf{1}_{k\neq i} \in \{0,1\}$ is an indicator function which returns 1 if $k \neq i$ and 0 if $k = i$, to avoid the computation of the distance between a sample and itself, N is the size of the batch and τ is a temperature hyper-parameter. This final loss will be computed over all the positive pairs in a batch, including both the pairs with switched index (i, j) and (j, i).

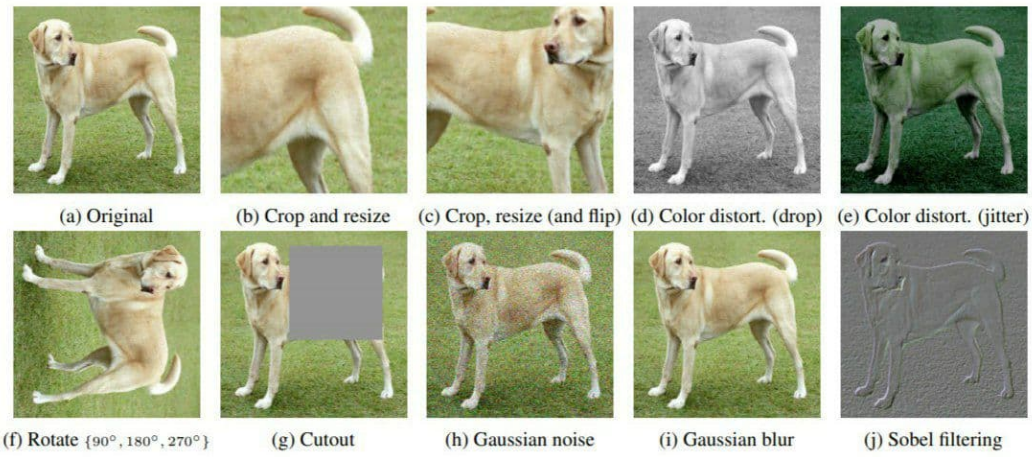

Figure 3.3: Illustration of the data augmentation operators

For the training of this model, first must be extracted a batch of N samples from the dataset which doubles its size thanks to the data augmentation module that will generate two views for each sample. Then, the batch is fed to the network to obtain the representation and the projection on the latent space where the contrastive loss is applied. Finally, the loss is calculated and the gradient is back-propagated to the whole model, both encoder and prediction head.

Like MoCo, also SimCLR, suffers from some local information leakage on the batch normalization layer, to avoid this Chen et al. propose to aggregate the batch normalization mean and variance over all the devices used for the training of the model.

From some tests carried out by the authors, have been observed some interesting characteristics of this framework, the main ones are:

- A bigger batch size allows improving the quality of the final representation and the consequent performance on the downstream tasks. With bigger batches,

the number of negative samples increases and as seen on MoCo, compare a positive pair with a large number of negatives leads to better representation.

- Wider and deeper nets lead to a smaller performance gap with the supervised training version of the same net. They tested the ResNet models with different widths 50, 101 and 200, besides the general performance on the downstream task, also the gap with the supervised counterpart tends to improve when the depth of the net increases.

- Longer training is beneficial for contrastive learning, with a small number of training epochs, the batch size has a great impact on the performance of the model, which will prefer a larger batch over smaller ones. Increasing the number of training epochs help to reduce or remove the gap. It can be seen as another way to provide more negative examples and improve the final representation.

The successive versions of this framework, called *SimCLR v2* [9], propose to execute a distillation process following the *teacher-student technique* [20] and introduce the memory queue mechanism used in MoCo, to reduce the effect of the batch size on the final performance of the model.

3.1.3 BYOL

Differently from the previous frameworks, BYOL does not require the negative pairs of images but relay only on the positive pairs, focusing on learning all the crucial information from them. Grill et al. propose to use two asymmetric networks, called *online* and *target*, and iteratively extract the output from the first net to enhance the representation produced by the other one. More in detail, the online network will be trained to predict the representation produced by the target net, of the same input image, but under a different augmented view.
Thanks to this continuous interaction, these networks are encouraged to learn from each other on how to produce ever better representations. As result, the full model will automatically *"bootstrap"*, in its idiomatic sense, its own optimal representation by using only the positive pair.

On figure 3.4 is show the architecture of the two networks that compose the complete model. The target network will consist of:

- An encoder f_ξ, which is a CNN like a ResNet or a VGC net. It will process the input image to produce a representation vector y_ξ.

- projection head g_ξ; it is an MLP net that maps the representation y_ξ on a smaller latent space with a ReLU non-linear transformation, similarly to what happens in SimCLR. This projection can be defined as z_ξ.

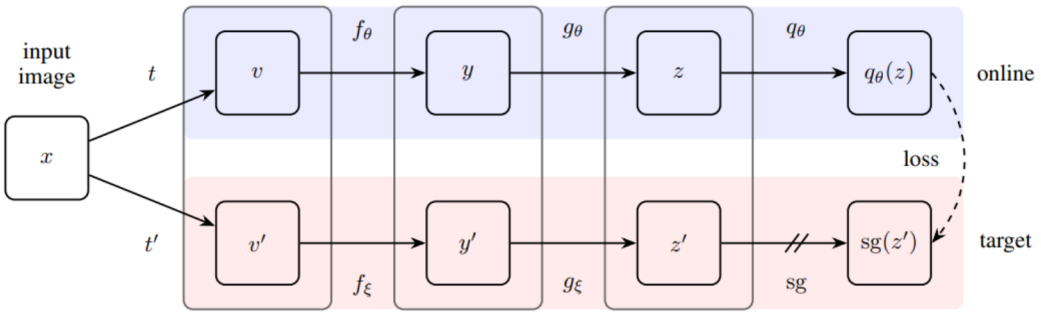

Figure 3.4: Illustration of the BYOL architectures

The online network follows the same structure as the target one but will use a different set of weights, defined by θ and adds a second MLP head q_θ, responsible for producing a prediction over the representation produced by the target network. Furthermore, like MoCo with its momentum encoder, also the weights of the target net are updated through a slow-moving average from the weights of the online one. The updating can be defined as follow:

$$\xi \leftarrow \tau\xi + (1-\tau)\theta \tag{3.5}$$

Where τ is the momentum hyperparameter.
These two networks receive as input two different batches of images v and v', obtained by applying different data augmentation operators, t and t', to the same initial batch x. Then, by using the batch v, the online network is trained to predict the target network's representation, obtained instead from the batch v'.
The loss function defined for this task is:

$$L_{\theta,\xi} = \left\| \bar{q}_\theta(z_\theta) - \bar{z}'_\xi \right\|_2^2 \tag{3.6}$$

It is the mean squared error loss between the normalized prediction from the online network and the projection produced by the target network. This loss is then symmetrized obtaining $\tilde{L}_{\theta,\xi}$. To do that, the training step is repeated but with the batches swapped between the two networks. So, the online network will receive the batch v' to output the predictions, and the target network will use the batch v to produce the projections. Thus, during a single training step, will be minimized the loss: $L^{BYOL}_{\theta,\xi} = L_{\theta,\xi} + \tilde{L}_{\theta,\xi}$.
This loss is minimized only with respect to θ, so the target network parameters' are not updated in the direction of $\nabla_\xi L^{BYOL}_{\theta,\xi}$, as consequences, there is no a priori reason for which the model should converge to a minimum of L, with respect to both set of parameters. Anyhow, the model is able to converge to an optimal solution, so the authors hypothesize that like on a GAN net, where there isn't a

loss to minimize both the generator and the discriminator, also for BYOL there may not be a loss that must be minimised with respect both θ and ξ. However, since BYOL does not use any explicit or implicit method to avoid a collapse, such as the use of negative pairs, its dynamics still admit unstable equilibrium points that may lead to collapsed or *degenerative solutions*.
With the objective of minimizing the loss function, the model could cheat on the prediction task, by producing the same representation, regardless of the input sample. In this way, the prediction task becomes easier and the training loss fastly decreases to zero, unluckily, the representations obtained are useless for any possible downstream task.
Even if, in theory, BYOL allows this degenerative solution, Grill et al. have never observed the convergence of the model to this situation, they hypothesize that the addition of the prediction head on the online network, and the use of a slow-moving average to update the target net parameters encourages the online projection to encode more and more information and avoids such a collapsed solution.

From the test conducted by Grill et al., BYOL results to be more robust to the choice of the image augmentation and of the batch size, by removing some transformation function from the transformation set or reducing the batch size, the model suffers less performance degradation than other methods, like SimCLR.

3.1.4 SimSiam

SimSiam is a contrastive learning framework that, like BYOL, does not use any negative pairs of data for the training of the model and differently from SimCLR does not need a large batch size to obtain competitive performance on the downstream task. Moreover, it uses only a single features encoder, removing so the necessity of a second network, updated by a slow-moving average of the weight from the main one.
This framework is based on the use of *Siamese Neural Networks* [4], this type of NN architecture includes two, or even more, identical sub-networks with the same set of weights, to process input pairs of data and produce comparable representation vectors. Usually, the goal is to perform a comparison between the two inputs to determine their similarity or belonging to the same class. If the inputs are extremely similar, then their representations will be too, and if the inputs are the same also their representation will be equal, because the two sub-networks are exactly identical and process the same input data in the same way.
Some applications of Siamese nets are face verification, tracking, signature verification, etc.
More than comparing or classify entities, in this case, this architecture is proposed to *"contrast"* among them to learn meaningful representations.

The figures 3.5 shows the architecture of the full model, it consists of:

- An encoder network f parametrized by θ, which is composed of a backbone plus a projection head. The backbone is a CNN model, while the projection head is an MLP net with a ReLU non-linearity. It is responsible for the production of the representation and its projection on a smaller latent space where is applied the training loss.
- A predictor h; It is an MLP network applied only on the left branch of the architecture, this makes the model asymmetric with respect to the possible path followed by the input data.

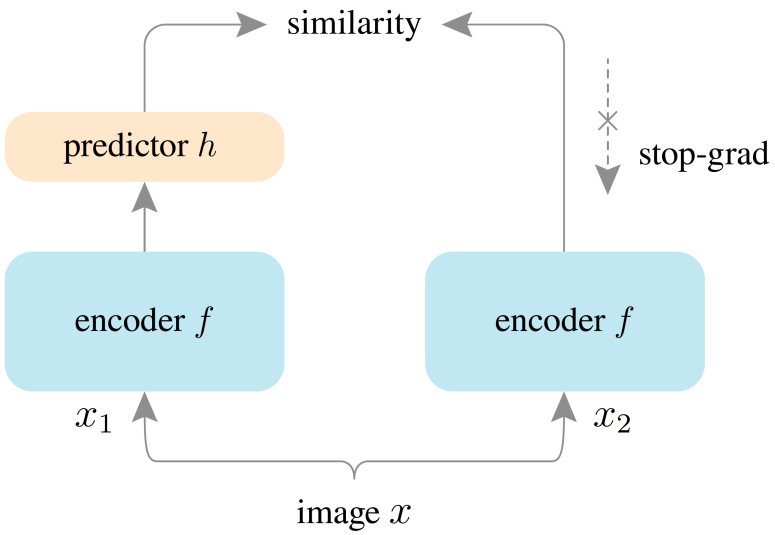

Figure 3.5: Illustration of the SimSiam architecture

Following the previous frameworks, also in this case the model will receive as input two randomly augmented views x_1 and x_2, obtained from the same initial image x. The task will consist of using the image x_1 to predict the representation produced by using the image x_2. To do that, x_1 will be processed by the left branch, on which is located the predictor h, while x_2 by the right branch where there is only the encored f. In doing so, a *stop-gradient operation* is applied on the right branch, making it possible to back-propagate the gradient only on the left part of the model, this operation results to be a crucial operation to avoid a collapsing of the model to a degenerative solution.
It is important to specify that even if the gradient will not be back-propagated in the left branch of the network, the model will use the same identical encoder f for both branches, consequently, at the end of each training step, both sections of the

network will end up with a new set of updated weights.
The objective is to minimize the negative cosine similarity between the representation provided by one branch, and the prediction produced by the other one:

$$D(p_1, z_2) = -\frac{p_1}{\|p_1\|_2} \cdot \frac{z_2}{\|z_2\|_2} \tag{3.7}$$

Where $z_2 = f(x_2)$ is the representation to predict, $p_1 = h(f(x_1))$ is the prediction and $\|\cdot\|_2$ is the $l2$-norm. Following the BYOL suggestion, the authors of this framework defined a symmetrized loss function as:

$$L = \frac{1}{2}D(p_1, z_2) + \frac{1}{2}D(p_2, z_1) \tag{3.8}$$

Where the input batch images x_1 and x_2 are swapped between the two branches to repeat the prediction task, thus will be used x_2 to predict the representation produced from x_1. Its minimum possible value is -1, and to make more explicit the stop-gradient operation, this loss can be rewritten as:

$$L = \frac{1}{2}D(p_1, stopgrad(z_2)) + \frac{1}{2}D(p_2, stopgrad(z_1)) \tag{3.9}$$

Since like BYOL also SimSiam does not use negative pairs, it could reach a degenerative solution where all the input data are mapped in the same way to minimize as fast as possible the training loss, but so obtaining useless representation. To avoid it, BYOL relies on a slow-moving average of the parameters from a network to another one, SimSiam instead relies on the stop-gradient operation applied on the right branch of the model.
As matter of fact, from some tests run by Chen and He, results that removing that operation leads the model to quickly finds a degenerative solution. They also propose a proof-of-concept that explains the reasons for which, according to them, the stop-gradient is a critical operation to stabilize the training and avoid degenerative solutions.

3.1.5 SwAV

It is an online *clustering-based* self-supervised contrastive method, which differently from MoCo or SimCLR does not contrast directly between the representations learned, but use a cluster label obtained through a clustering algorithm, performed on top of the actual representation produced by the model. SwAV enforces consistency between the cluster assignments, also called *codes*, generated by different augmented views of the same image. The authors propose a swapped prediction task where the code produced from a view is predicted by using the representation produced from another view. Moreover, this method does not require the mining

of explicit negative pairs of images but relies on the different cluster assignments to discriminate between different images.
The majority of the clustering-based methods are *offline*, in the sense that, they alternate cluster assignment step, where the data in the entire dataset are clustered using the actual representation produced by the model and an algorithm like *K-Means* or *DBSCAN*; and training step, where the cluster assignments are used to train the model and improve the quality of its representation. Unfortunately, this approach requires a huge amount of computational effort that with a big dataset and for a high number of training epochs, becomes unsustainable even with the availability of high-level GPUs.
SwAV instead is an online clustering method, it proposes to compute the code online directly during the training step and only for the current batch of the data. In this way, the clustering process becomes much faster and make it possible to scale this procedure also to a big dataset, however, on the clustering algorithm must be enforced a constrain for which the codes of different images in a batch size must be distinct as possible, to avoid trivial solution where all the images in the same batch have the same codes.

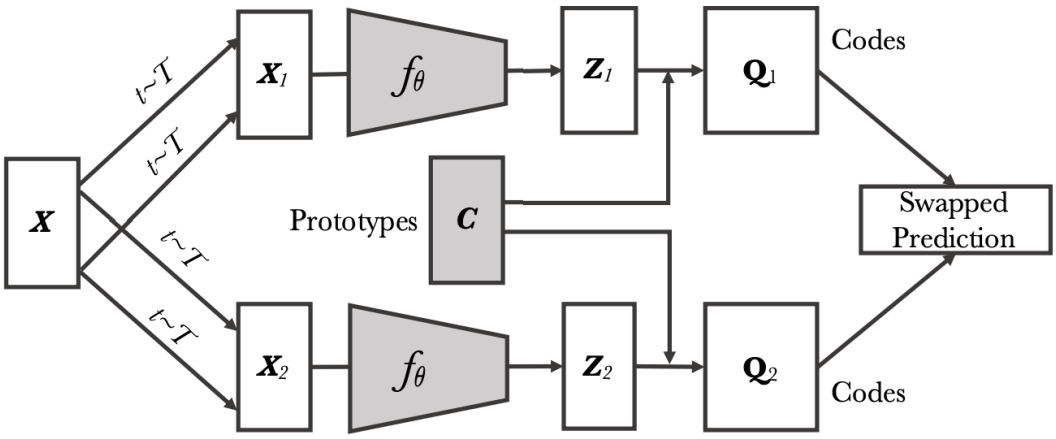

Figure 3.6: Illustration of the SwAV architecture

The architecture of the model consists of a feature extractor f_θ and a prototype matrix C. f_θ is ConvNet with an MLP projection head, while the matrix C contains a set of K prototypes vector $\{c_1, \ldots, c_k\}$ which are trained together with the entire network. Starting from the image x are extracted two augmented views x_1 and x_2 which are fed to the encoder f_θ to obtain the two representations vectors z_1 and z_2 of the input images. Then by using these representations and the prototype matrix C, are assigned the codes q_1 and q_2. The code q_i denote the mapping of the feature vector z_i to the prototypes C and is optimized to maximize their similarity,

by solving the optimization problem:

$$\max_{Q \in \mathbf{Q}} \; Tr(Q^T C^T Z) + \varepsilon H(Q) \tag{3.10}$$

Where ε is a hyper-parameter that regulate the smoothness of the mapping and H is the entropy function. Moreover, to ensure different codes for different images, the codes are constrained to be equally partitioned among the K prototypes vector. To do this, the matrix Q of the codes will be enforced to belong to the set of *transportation polytope* matrix:

$$\mathbf{Q} = \{Q \in \mathbb{R}_+^{KB} \mid Q1_B = \frac{1}{K}\mathbf{1}_K, \; Q^T\mathbf{1}_K = \frac{1}{B}\mathbf{1}_B\} \tag{3.11}$$

Where B denote the size of the batch and $\mathbf{1}_K$ is a vector of ones of dimension K. These constraints ensure that on each batch, each prototype vector c_i , is selected on average, B/K times. Finally, is performed the swapped prediction task, where the representation z_1 is used to predict the codes q_2, and the representation z_2 is used to predict the codes q_1, with the aim of minimizing the following loss function:

$$L(z_1, z_2) = l(z_1, q_2) + l(z_2, q_1) \tag{3.12}$$

Where $l(z, q)$ measure the fit between the code q and the features vector z, it is the cross-entropy loss between the codes and the softmax probability of the dot product between z_i and C.

$$l(z, q) = -\sum_K q^{(k)} \log p^{(k)} \qquad with \qquad p^{(k)} = \frac{exp(\frac{1}{\tau} z^T c_k)}{\sum_{K'} exp(\frac{1}{\tau} z^T c'_k)} \tag{3.13}$$

This loss function is jointly minimized with respect to both the parameters θ of the main encoder f_θ, used to produce the features z, and the prototypes vector in C.

3.2 Contrastive learning on video

Differently from the images, the videos contain an additional dimension provided by the time axis. Many works have shown that even if the spatial information of each frame is important, also the temporal information between frames is essential to operate on the video data in an optimal way.
The previously proposed self-supervised contrastive frameworks are quite general and do not assume any hypothesis regarding:

- The backbone of the model, which essentially could be any type of 2D-CNN.
- The domain, the property and the distribution of the data into the dataset.

- The downstream task, which also in principle must remain independent from the self-supervised pretraining.

However, since these methods are mainly thought to operate on image data, they are mainly concentrated on improving the spatial representation by extracting useful information from all the spatial areas of an image. This is also the reason why the use of some heavy data-augmentation functions is a key point of that frameworks to obtain competitive performance.
Thus, when it comes to operating on the video data with these methods, some changes are required. First of all the backbone must be replaced, passing from a 2D to a 3D ConvNet, able to operate on such a category of data. Then, the framework must focus on improving both the spatial and temporal features representation by proposing a new way of applying data-augmentation functions or by adding new components that help to improve the temporal representation.
Many frameworks have been proposed, and some of them come as a direct variation on the previous ones, like:

- **VideoMoCo:** Video Momentum Contrast
- **CVRL:** Contrastive Video Representation Learning

3.2.1 Video MoCo

Starting from MoCo, Pan et al. propose some variation to improve its temporal feature representation. VideoMoCo keeps from MoCo: the usage of a queue structure to store the negative representations, a momentum encoder and the InfoNCE as a contrastive loss to train the model.
Regarding instead the variation with respect to the original frameworks, the first one is the replacement of a ResNet-X architecture with a *convolutional-3D* (C3D) architecture, as a backbone for the encoders of the model. This change will affect, as expected, both the query encoder and the momentum encoder. As already said this variation is mandatory, to be able to operate on the video data-format, which is the new input type of the model. Similarly to MoCo with the images, this method applies randomly, some data augmentation function on each frame on the input clip, to produce two augmented versions of the same clip with which feed the model.
Then to optimize the usability of the representation contained in the queue, is proposed a *temporal decay system*. A long queue is a crucial element of MoCo to improve the quality of the final representations since a longer queue means more negative samples to compare with the actual positive pair. The older keys in the queue are based on previous states of the momentum encoder, and this may cause a huge discrepancy between them and the current representation. To mitigate this

problem, Pan et al. proposed a temporal decay to model the degradation of the keys in the queue. For each key k_i is set a temporal decay $t^i \in (0, 1)$, where i is the position of the key in the queue.
The loss function 3.2 can be rewritten as:

$$L_q = -log\frac{exp(q \cdot k_+/\tau)}{\sum_{i=0}^{K} t^i \cdot exp(q \cdot k_i/\tau)} \tag{3.14}$$

Greater is i, the more is the time already spent in the queue by that sample, and lower will be the value of t^i. In this way, each key will contribute, to the loss, in relation to its time in the queue, more recent keys will have a greater contribution than older keys.

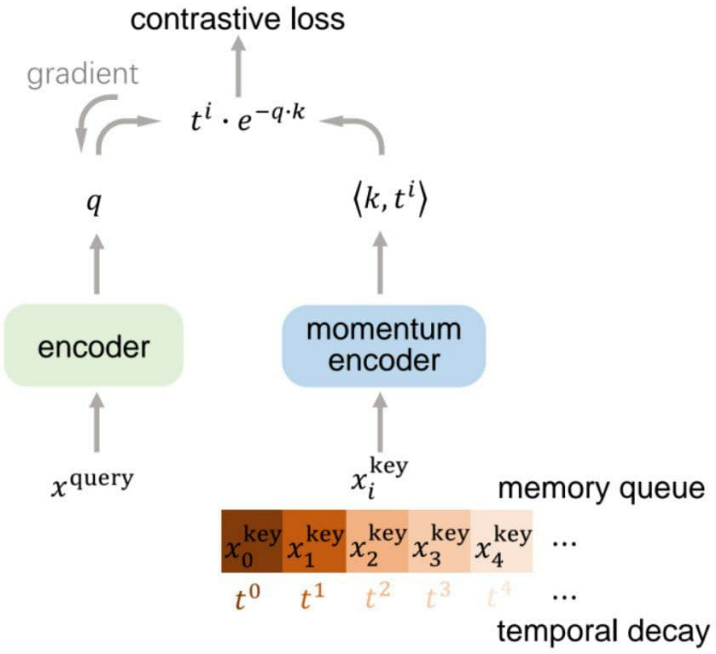

Figure 3.7: Illustration of the temporal decay system

Finally, to improve the temporal features representation, they propose a temporal data augmentation method on which some frames of the input clip are removed to make the contrastive task more challenging for the encoders, whose now have fewer frames to recognize the label of the clip.
To accomplish this, before the query encoder is placed a generator G. It consists of a *ConvLSTM* model whose task is to output a prediction of the importance of each frame in the clip. Then, the 25% of the most important frames are dropped out by using a temporal mask. So, the input sample x^{query} received by the encoder,

to produce the query q, can be written as:

$$x^{query} = G(x) \otimes x \tag{3.15}$$

Where x is the initial input and $\otimes$ indicates the dropout operation. x^{query} is equal to x but contains fewer frames. To train G is used the L_1-norm loss, between the representations of x and x^{query} produced by a discriminator encoder D, those representations are expected to be similar. This loss can be written as:

$$\max_G L(G) = \mathbb{E}_{x \sim P_x}(|D(x^{query}) - D(x)|_1) \tag{3.16}$$

Where P_x is the data distribution of x and as discriminator D is used the query encoder. To avoid convergence problems of the whole model, the generator G is added into the training pipeline only when the encoder D has reached a semi-stable state. So, first, the query encoder and the momentum encoder are trained by using only the contrastive InfoNCE loss 3.14 and no drop-out of the frames, then the generator G is involved with its drop-out operations of the frames and its own training loss. During each training step, G and D will be trained alternately.

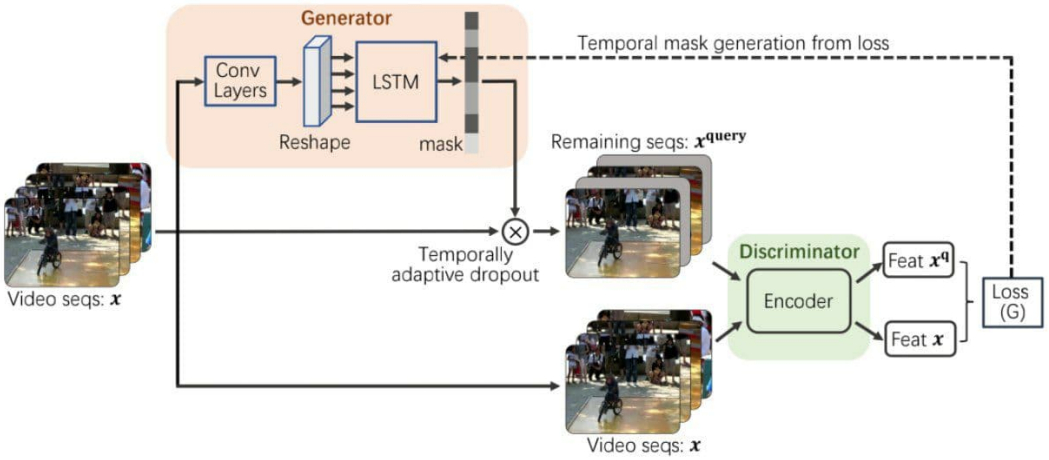

Figure 3.8: Illustration of the VideoMoCo architecture

3.2.2 CVRL

This framework proposes a contrastive learning method to learn spatiotemporal features from unlabelled video, based on the SimCLR architecture. Here, the positive pairs will consist of two augmented clips extracted from the same initial video, while the set of possible negatives samples could contain any other clip sampled from the other videos contained in the dataset. Just like on SimCLR, the

network will be trained to push near each other the positive pairs representations and at the same time pull away, the representations of the negatives samples.
Like SimCLR, CVRL does not use a momentum encoder or a queue to store the negative samples, and moreover, it too relies on the use of a heavy data augmentation module, that will be adapted to ensure the presence of temporal consistent information inside the clips.
As the main encoder for this architecture is used a *3D ResNet*, whose design follows the slow path of the *SlowFast* network [27] but with some modification on the temporal stride and on the temporal kernel of the initial convolutional layers.

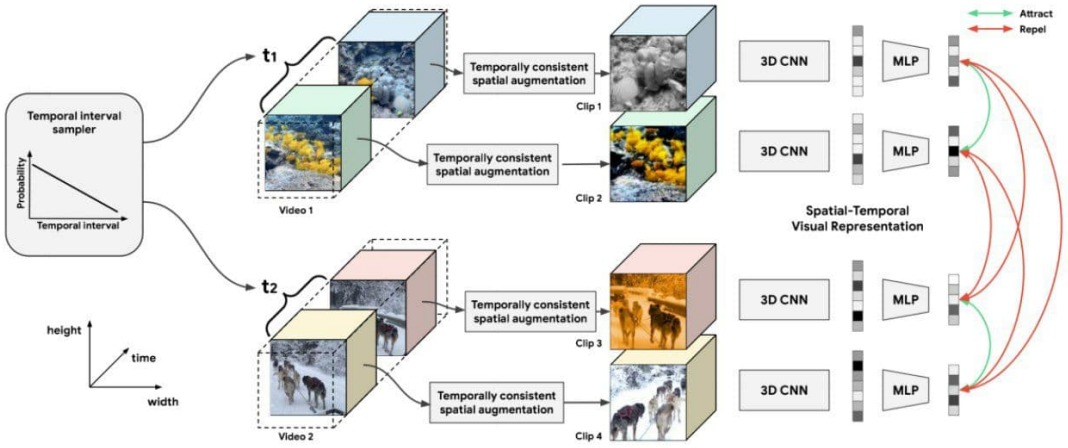

Figure 3.9: Illustration of the CVRL architecture

The main variation proposed by Qian et al. to improve the quality of the spatiotemporal features representation learned by the model are:

- *A sampling-based temporal augmentation*: to obtain a positive pair, are sampled two random clips from the same initial video, however, the way in which these two clips are sampled is not irrelevant. In general, two clips are more similar if their temporal interval is smaller, while are more differents if the interval is larger. The idea is to sample with greater probability, clips that are closer in time, in this way the contrastive loss would focus more on those types of clips and will enforce a smaller penalty for the clips with higher temporal distance. To do that, given an input video of length T, is sampled an interval t from a distribution $P(t)$ over $[0, T]$, then from the interval $[0, T - t]$ is sampled, uniformly, a clip. Finally delayed by a time t from the first clip, is sampled the second clip.

- *A temporally consistent spatial augmentation*: randomness on the data augmentations between consecutive frames of the same clip, could harm the temporal information and make it more difficult to recognize the temporal

pattern. Avoiding randomness on the data augmentation transformation between consecutive frames helps to maintain temporal consistent information. Thus, when a given transformation must be applied on a clip, first are defined its parameters, like size and position for a random crop or the angle of rotation for the random rotation, then they are maintained fixed for all the frames on the clip. In this way, it is possible to impose strong spatial augmentation on each frame while maintaining temporal consistency across frames.

3.3 Performance comparison

Once the self-supervised pre-train ends, it is possible to use the learned representation to perform the downstream task. The performances of these contrastive methods are now compared on the classification, object detection and instance segmentation tasks.

3.3.1 Classification on ImageNet

The table 3.1 shows the performance of the representation obtained with the previous contrastive methods, under the linear evaluation protocol on the *ImageNet* dataset [23]. It is a really huge dataset composed of 1000 different categories and about 2 millions images, nowadays it is considered the main benchmark to evaluate the performance of a model on the image classification task.
To test these models will be followed the linear evaluation protocol, which already said consists of training a linear classifier upon the representation produced by a backbone pre-trained with the self-supervised method. The linear classifier will consist of a fully connected layer, with a number of output neurons equal to the number of classes contained in the ImageNet dataset. Since the weights of the backbone are frozen, the performance of the classifier will mainly depend on the quality of the representation learned during the pre-training. The backbone instead, will consist of a ResNet-50 for all the methods.

Is possible to see that the performance obtained with supervised training is still higher than the performances obtained by the self-supervised methods, 76.6% of accuracy against the 75.3% of SwAV which result to have the best performance among the contrastive methods. Is known that supervised training benefits from a large train dataset (just like ImageNet), however, datasets of this size are rare and very difficult to obtain.
Thus, to make a more realistic test, the training set is reduced by letting only 10% of it available, while the test set size is maintained fixed. In this way, the classification task becomes more difficult, since there are fewer samples available for the training. Under these conditions the situation is reversed, the accuracy of

Method	Top-1 accuracy on ImageNet (Full dataset)	Top-1 accuracy on ImageNet (10% of the dataset)
Supervised	76.6%	56.4%
MoCo	60.6%	-
MoCo v2	71.1%	-
SimCLR	69.3%	65.6%
SimCLR v2	71.7%	68.4%
BYOL	74.3%	68.8%
SimSiam	71.3%	-
SwAV	75.3%	-

Table 3.1: Linear evaluation on the ImageNet dataset

the supervised training becomes 56.4%, a drop of about 20%, while SimCLR and BYOL score respectively 65.6% and 68.8% of accuracy, with a drop of 3.7% and 5.5%, much smaller than 20%.
This show that, at a low labelled data regime, the contrastive methods are able to outperform the supervised training.

3.3.2 Transfer learning for classification on other dataset

The same representations are now used to perform the same task but on other datasets, to test if the learned representations are generally enough to fit a data distribution different from the training one, or if they are overfitted on the ImageNet and thus not so useful to perform the same tasks on other datasets. If the representations are truly generalizable, is possible to expect a boost in the performance of the model, with respect to the standard features obtained from the supervised training on ImageNet. Thus, the backbone is first pre-trained on ImageNet, with the contrastive methods, and then is re-trained on a different dataset in a supervised way.

Method	Food101	CIFAR100	Birdsnap	SUN397	Cars	DTD	Aircraft	VOC2007
ImageNet	72.3%	78.3%	53.7%	61.9%	66.7%	74.9%	61.0%	82.8%
SimCLR	72.8%	74.4%	42.4%	60.6%	50.3%	75.7%	50.3%	81.4%
BYOL	75.3%	78.4%	57.2%	62.2%	67.8%	75.5%	60.6%	82.5%

Table 3.2: Transfer learning from ImageNet: Linear evaluation

Method	Food101	CIFAR100	Birdsnap	SUN397	Cars	DTD	Aircraft	VOC2007
ImageNet	88.3%	86.4%	75.8%	64.3%	92.1%	74.6%	86.0%	85.0%
SimCLR	88.2%	85.9%	75.9%	63.9%	91.4%	75.4%	88.1%	84.5%
BYOL	88.5%	86.1%	76.3%	63.7%	91.6%	76.2%	88.1%	85.4%
Random init	86.9%	80.2%	76.1%	53.6%	91.4%	64.8%	85.9%	67.3%

Table 3.3: Transfer learning from ImageNet: Fine-tuned

On the tables 3.3 and 3.2 are shown the performance obtained under two different evaluation protocols, the first is the, already described, linear evaluation while the second one is the *fine-tuning evaluation*. It consists of re-training also the weights of the full backbone together with the classifier.
The objective, in this last case, is to understand if the self-supervised pre-training can be a good initializer, that helps the model to go deeper and reach better performance, with respect to the case in which the model's weights are initialized randomly or from the supervised training on another dataset.
The row *"supervised"* refers to the case in which the weights of the backbone are the values obtained from full supervised training on ImageNet, while the row *"random init"* indicate that the initial values of the backbone are generated randomly.

Under the linear evaluation protocol, BYOL and SimCLR have shown that the contrastive methods are able to produce optimal results by equaling or outperforming the supervised representation on almost all the datasets. Thus, the representation produced by these methods can be considered more general than the one obtained from the supervised training.
Even in the fine-tuning case, these contrastive methods have shown competitive results that are also able to slightly improve the final performance. Furthermore, it is interesting to note that their results are always better than those obtained with random initialization of the weights. Consequently, those methods can be also considered useful as initialization techniques for supervised training.

3.3.3 Transfer learning on other tasks

The learned representations are now employed to perform the object detection and instance segmentation tasks. These tasks are different from the classification one, so also the features necessary to perform them in an optimal way are different. With these tests, in addition to analyzing the generalizability of the representation on different data distributions, is also analyzed their reusability on different categories of downstream tasks.
The table 3.4 shows the results obtained on the *COCO (Common Object in Context)* dataset [24]. It is a large-scale dataset for object detection, instance segmentation

and captioning tasks, containing more than 200 thousand labelled images, 80 object categories and 91 stuff categories.
The architecture used is a *Mask R-CNN* where the backbone consists of a ResNet-50. As before, the *"supervised"* row, refers to the weights obtained from a pre-training in a supervised way on the ImageNet dataset.

Method	Object Detection		Instance Segmentation	
	AP_{50}	AP_{75}	AP_{50}	AP_{75}
Supervised	58.2	41.2	54.7	35.2
SimCLR	57.7	40.9	54.6	35.3
MoCo v2	58.8	42.5	55.5	36.6
BYOL	57.8	40.9	54.3	35.0
SwAV	57.6	40.3	54.2	35.1
SimSiam	59.3	42.1	56.0	36.7

Table 3.4: Transfer learning from ImageNet to COCO

The evaluation metric used for both tasks is the *Average Precision (AP)*, with different thresholds for the *Intersection over Union (IoU)*. Also on these tasks, the contrastive methods have shown competitive performance, even able to outperform the supervised counterpart, this confirms the transferability of the representation obtained by those methods, beyond the only classification tasks.

Chapter 4

Downstream task, materials and architectures

In this chapter, first of all, is described what is the *online action detection* task, which problems it aims to solve, in which contexts can be applied and how it can be accomplished by modern machine learning models. Then, are presented and analyzed the main architectures, suggested in the literature, that are able to perform this task optimally, together with the videos dataset upon which those models will be trained and tested in this work. Finally, it's described how the implementation of some contrastive learning frameworks, among those already described, have been adapted to operate on a single GPU device and with a wider range of backbone architectures.
The purpose of this work is to study the performance of these contrastive methods when it comes to operating on a videos dataset and performing the downstream task of the online action detection task. As said before, the contrastive methods are mainly thought for image data, where actually represent the state-of-the-art, while the corresponding proposals for the video data are still relatively new and not so widely used.

4.1 Action recognition task

Given a dataset of videos, it can be used to train a deep learning model on different possible tasks like video classification, object tracking, pose estimation, video object detection etc. Among them, there is the *action detection* task, which consists of recognizing a specific action or move inside the video, that could contain the execution of several different moves. To detect an action, a neural network must be able to recognize all the different parts that compose it. Usually, an action is not defined by a single specific movement, but by a set of following movements

that constitute the complete action. For example, if it's considered the action of running, it is composed of the movements of legs, arms, and hips, at the same time with coordination between the different parts of the body.

When usually, someone thinks about this task, he is manly led to think about the detection of activities done by adults, children, animals or vehicles like cars, buses, planes etc., however, these are not the only entities capable of moving in the environment. In principle, everything can be both motionless or on movement, all depends on the choice of the point of reference, changing it could lead to a modification of the state of all the entities in the analyzed environment. Anyhow, by avoiding philosophizing too much, just think about a ball hit by a kick, or a door closed by the wind, neither of the two is a living being, but both have performed an action, even if indirectly through an external force. For these reasons, this task can be easily extended to any type of action performed by any possible entity in the environment. Given that, it is immediately clear that on a single video, could happen that more actions are performed at the same time but on different locations of the camera, either near to each other or far away. The presence of different actions uncorrelated action, performed by different entities at the same time is one of the crucial factors that make this task challenging.

A deep learning model for action detection must be able to identify the salient factors that are useful to identify the correct action, and on doing this, it must face a huge amount of noise information coming from the other irrelevant actions on the video and from the background on which the activity takes place. Another difficulty that must be faced when a model is trained to perform this task, reside in the labelling process of the datasets. As already said, these datasets are composed of videos, each video is made of many frames that when seen on rapidly sequences show the development of the recorded scene. The labelling process for this type of dataset consists of assigning to each frame a label that indicates the class of the action executed on that frame, moreover, if a single frame contains more than one action, it will have a number of possible labels equal to the number of actions in that frame. These types of frames can belong to two different categories:

- Frames that are positioned between two successive actions and contain information relating to the transition from an action to the other one. Assigning them to a specific action result be difficult since the two actions are a bit overlapped, as shown in the figure 4.1. However, sometimes it must be done to avoid further problems of clarity into the dataset.

- Frames that actually contains different actions on different locations of the scene recorded, usually in this case the frame and the label are followed by a vector of coordinates for a *bounding box* that indicates the spatial location

where the action is being performed.

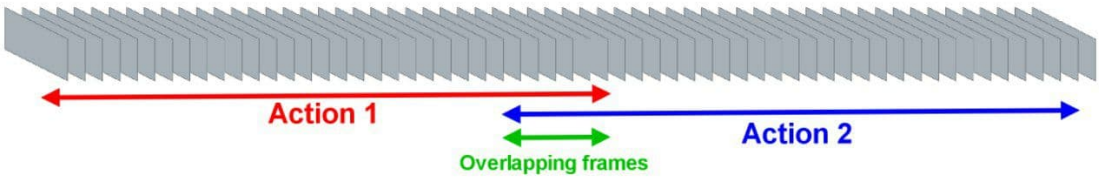

Figure 4.1: Illustration of overlapping frames between different action

Following this labelling procedure could arise two main problems: the first one is related to the identification of the correct initial frame of the action, we live in a *continuous-time* world while the video cameras, used to record the videos, operate in the *discrete-time*. To approximate the real world as well as possible, it tries to register a huge number of frames per second, higher is that frame rate more information are captured from the real world into the recorded video. Thus, the frames around the beginning of the action are almost equal to each other and this makes more difficult the choice of a single initial frame, seldom a correct frame does not exist at all and the initial frame is decided in an approximate manner, even if at the cost of making that information a little dirty and nasty to operate with. The same reasoning can be done for the identification of the final frame of the action.
The other problem instead, could arise only if there are frames with more than one label, when operating on those frames, the model will extract a set of salient information that refers to more possible actions, by making really difficult to discriminate between which is the actual correct one. In these cases, the model will heavily rely on the following and preceding frames in order to recognize the correct class label.

The action detection task can be divided into two sub-categories that mainly differ in the possibility of accessing all frames of the video, and are:

- Offline action detection.
- Online action detection.

4.1.1 Offline action detection

In the offline action detection task, the model could have access to the entire video, so to all the frames contained on it. Given a video, the goal is to identify the start, the end and the class label of all actions contained on it. Sometimes could be also

asked to indicate the spatial location on which the activities take place, by using some bounding box to define it. To achieve this, the model must learn to produce representations that encode the full evolution in time of the action, from its start to the end.

Even if in principle the offline methods could use all the frames contained in the registered video, usually they prefer to sample only some of them with a given frequency to reduce the amount of redundant information received from frames too much similar to each other, and at the same time analyze a bigger time interval with less computational effort.
For example, suppose that a camera records a $10s$ video at $30fps$ *(frame per second)* and is available a deep learning model able to receive as input 30 frames at a time. The final video will contain 300 frames and to process it the model must perform 10 steps, on each step, it will be able to see just 1s of the full video, which depending on the action may not be enough to detect it, since the frames are too much similar to each other becomes difficult to recognize a clear movement.
But if now, before feeding the model with the video, it is sampled with a *frame-step* of 2 frames, the produced video will have 150 frames and can be processed by the model with only 5 steps. In this case, the model will see $2s$ of the video on each step, this temporal interval is greater than before, and could facilitate the detection task for the model, because the temporal distance and the differences between consecutive frames are much more pronounced than before, and that helps a lot to recognize the movement in the video.

Besides this sampling approach, in the literature, there are various proposed methods that exploit different mechanisms to improve the detection ability of the model or reduce its execution time by avoiding useless computations. Some methods propose to combine the *RGB* input frames with some other that comes from a different modal, like for example the *RGB-D* frames which are obtained by using a specific type of camera that on each pixel of the frame instead of the colour, encode the distance of the target from the camera. Other methods instead, propose to insert a proposal stage before the action detection stage, to decide if is it's necessary to check the presence of action or not, to avoid useless computations.

The offline action detection can be mainly used on different activities and in different contexts, like:

- *Video summarization*: reduce the duration video by selecting only the most important parts.

- *Content-based video retrieval*: research of a specific action in a video.

- *Healthcare*: for example to monitor the improvements on the moves of a patient in the rehabilitation phase.
- *Ecological monitoring*: analyze and study the behaviour of the animals in their natural environments.
- Etc...

4.1.2 Online action detection

The online action detection task can be described as a video classification task executed in real-time, thus without access to future or past frames of the video, but by relying only on the current input frames. In this case, the neural network must be able to recognize an action even if it is available only a small part of it, and as consequence, a model trained for this task will face three possible states of the same action:

- *Start of the action*: it consists of the very first movements performed by the analyzed entities and contains the transition frames between the previous action and the current one, its duration is very short.
- *End of the action*: These are the final moments of the action, on which it is running out. It contains the transition frames between the current action and the next one. Like the start, also the end has a very short duration.
- *Development of the action*: this is the central and most important part of the action that is being performed. This state, differently from the other two, can have a variable duration that depends on the type of action, it could be both very short or very long.

The goal of the online action detection task is to recognize an action as soon as it starts, follow it during its entire development and indicate when it is ended; all that by using only the current input frames that usually are just been generated by a video camera. While the model is employed to operate on those frames, that video camera will continue to record new frames that will define the next input for the model. As consequence, that the model must be able to work in *"real-time"* by directly operating on the new frames just recorded, thus without long inference time or excessive delays. It must be light and fast enough to match the recording speed of the camera.

The sets of input frames with which is fed the model are called *chunks*, since those frames are the only ones available to perform the detection task, sampling methods like on the offline detection task are not viable. A chunk consists of frames

contiguously to each other, without so much temporal distance between them. Each chunk corresponds to a single action, however as already said, the labelling of the dataset used for the training is done frame by frame, and do not take into account the presence of these chunks. As consequence, given that dataset and extracted a random chunk from one of its videos, could happen that it contains frames with different class label, in these cases, the label of the chunk is decided by a majority voting from the labels of the frames that constitutes it.

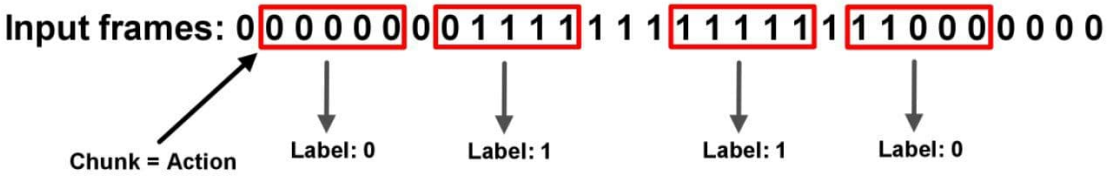

Figure 4.2: Example of chunk labeling

The online action detection task can be implemented into a wide range of activities of different works environments, like:

- *Autonomous driving*: this is one of the most recent sectors in which are being applied deep learning methods. An autonomous car is a special category of vehicle able to moves on the road without the necessity of a human pilot, it is able to sense the nearing environment thanks to the use of a huge number of sensors and video cameras. The online action detection could help to improve the performance of the car by also making it more secure, for example, if a child is chasing a ball in the middle of the road, an autonomous car should immediately detect him and promptly press the brakes to avoid an accident.

- *Automated surveillance* and *homeland security*: give a security camera, it is able to constantly register the surrounding environments, by combining it with an online detection model becomes possible to improve the security levels. For example, if a thief breaks into an apartment, the camera could be able to understand what is going on and call the authority as soon as possible, and not when the theft has already ended and is already too late.

- *Automatic sign-language recognition*: detect all the gestures of that language to convert them into their respective written form as soon as possible, to facilitate communication with this type of language.

- Etc. . . .

4.2 Dataset

The dataset used in this thesis project is a *Judoka-training-moves* dataset, it is composed of small clips registered in *full-HD* (1080×1920) at $30fps$ and of duration between $1s$ and $4s$, on which two judokas, on top of workout mat, perform a specific judo move, also called *Waza*. More in detail, the judokas that will execute the Waza in an active way is called *Tori* which means *"Taker"*, while the other one that will suffer the move in a passive manner, is called *Uke* which means *"receiver"*. The Waza's categories registered in this dataset are:

- *Ashi Waza*: foot and leg techniques
- *Te Waza*: hand techniques
- *Koshi Waza*: hip techniques
- *Masutemi Waza*: rear sacrifice techniques
- *Yoko-sutemi Waza*: side sacrifice techniques

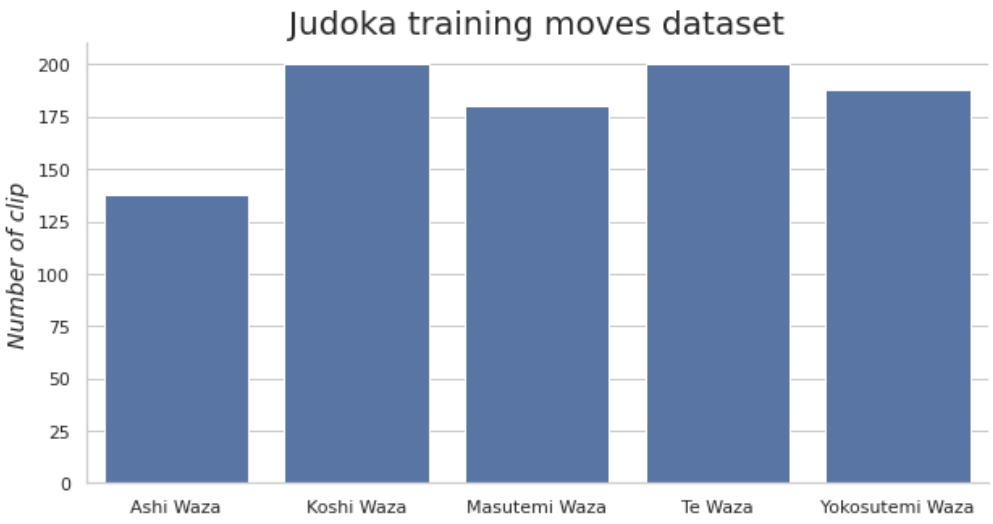

Figure 4.3: Clips distribution of the Judoka training moves dataset

All those moves fall under the *Nage-waza* class, which contains standings and sacrifice techniques. On figure 4.3 is shown the distribution of the clips among the different classes, seems that the dataset is balanced well enough and only the Ashi waza class is a bit less represented than the other, however, this discrepancy is quite small and does not cause any problem during the training of the model.
Before using this dataset to train a deep learning model for online action detection,

it is divided into three different, not overlapping subsets: *training set*, *validation set* and *test set* with the respective proportion of 70%, 20% and 10% of the entire dataset. Moreover, while splitting the dataset into the different sets, is preserved the relative distribution of the samples among the different classes. The figures 4.4, 4.5 and 4.6 show the clips distribution of all three sets.

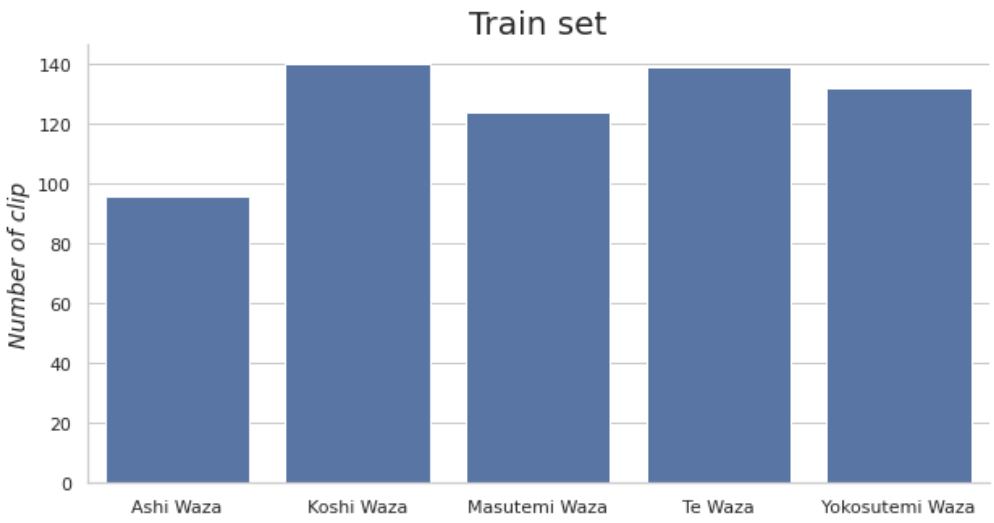

Figure 4.4: Clips distribution of the training set

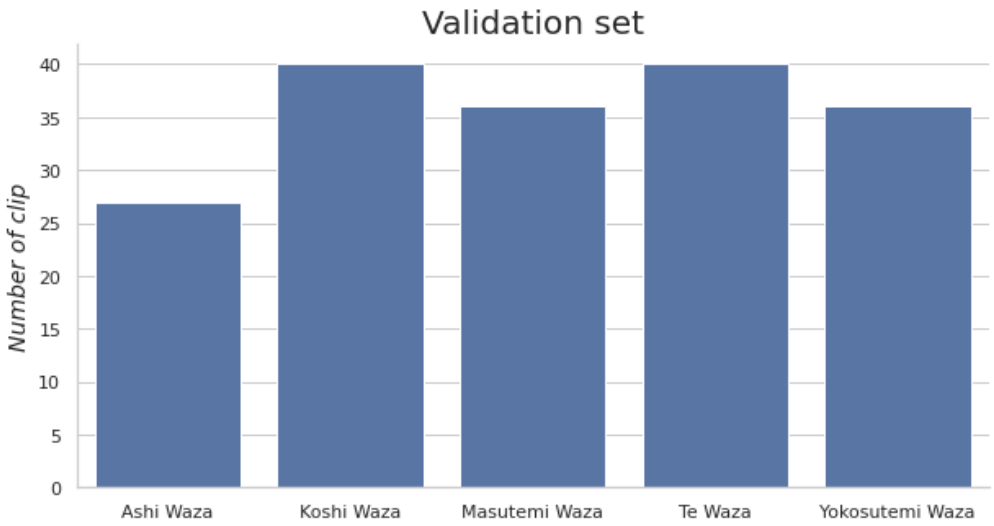

Figure 4.5: Clips distribution of the validation set

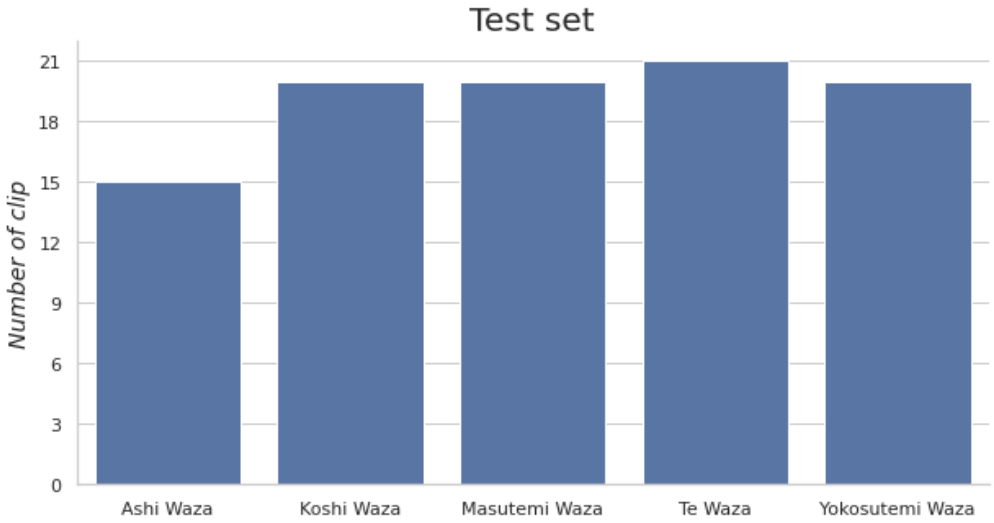

Figure 4.6: Clips distribution of the test set

From some analysis on the clips of the dataset, it has been noted that the main movements are concentrated on the centre of the clip where the judoka effectively performs the technique, while during the initial and final frames almost no movement is performed.
So, a new class is added to the previous five. It is called *background* and does not represent any type of judo moves, but contains the temporal intervals that belong to the initial moments where the two judokas are stand up facing each other just before the start to perform the Waza, and the final moments where, as a result of the judo moves, one or both judokas are landed on the ground. More in detail, the 15% of the initials and finals frames will be labelled as background class.

A particular aspect of this dataset is that the different moves have been recorded through the use of four different *GoPro* cameras placed on the four corners of the workout mat. So the same action in the same exact time instant has been registered from different points of view.
Figures 4.7, 4.8, 4.9 and 4.10 shows an example of the same temporal instant recorded from the four different angulations.

Figure 4.7: View: GoPro number 1

Figure 4.8: View: GoPro number 2

Figure 4.9: View: GoPro number 3

Figure 4.10: View: GoPro number 4

4.3 Architectures for the online action detection task

In the last years, together with the increasing availability of video data, have been proposed many types of deep neural network architectures able to operate on them adequately. Those NN can be mainly divided into two main categories, each one following a different path to produces good spatial-temporal representations for the input video.

The first category proposes to combine a 2D convolutional neural network with a recurrent neural network. The idea is to use first the 2D ConvNet to process the frames of the clip independently, by producing representations that contain only the spatial information within each frame. Then, these representations are feed into the RNN one after the other, following their respective temporal succession, in order to extract the temporal relationship between consecutive frames and produce the final representation that contains both the spatial and temporal features contained in the clip.
Following this type of architecture, into this work will be implemented a neural network that combines a *ResNet-18* as 2D-CNN, with a *GRU* net as RNN. This model is finally ended by a linear classifier, which in this case is a fully connected layer, to perform the classification of the current chunk.

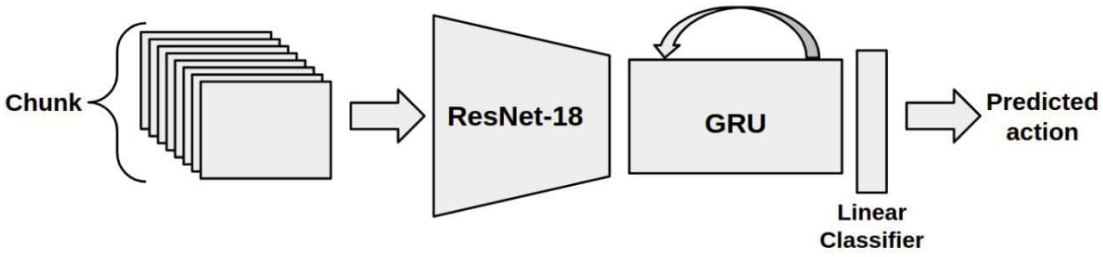

Figure 4.11: Illustration of the first model: ResNet-18 + GRU

The other category of architecture consists of the already named 3D-CNN, this type of neural network proposes to extends the convolutional layers along the time axis, by inflating and moving their filters along also the temporal axis while performing the convolutional product. The objective of these models is to improve their performances by exploiting all the well-known properties of local connectivity and hierarchical patterns, of the convolutional layer to extract and combine both spatial and temporal features into a unique representation.
To observe how the contrastive learning methods work with different types of convolutional backbone, in this work will be used also a 3D-CNN architecture which performance will be compared to the previous one. The model chosen is the *X3D* net [28], it is obtained from a small 2D-CNN model on which are applied a series of expansion and contractions on the size of its filters, to improve the final performances or reduce its complexity. The objective is to find the optimal trade-off between computational cost and final performances.
As an initial architecture from which start this process, Feichtenhofer choose the *MobileNet* [29]. It belongs to a special category of ConvNets that are mainly thought to operate on mobile and embedded systems, on which the computational power is quite limited. It uses the *Depthwise Separable Convolution layers*, which allow computing an *"approximate"* version of the result given by a standard convolutional layer but using much less computational effort.

There are different versions of the X3D nets which mainly differ for the size of the temporal activation, the depth of the model, the size of the channel for all the layers, the size of the inner convolution filter and as consequences also for the inferences time and training computational cost. To accommodate the available GPU and still maintain a fair batch size during the training, is chosen the *X3D-XS*, among the different versions it is one of the more small and light, that however is able to ensure good performance. Similarly to before, this net is terminated by a fully connected layer to perform the classification task.

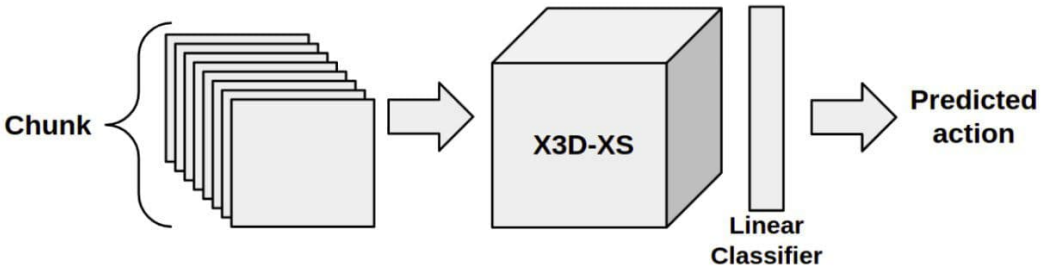

Figure 4.12: Illustration of the second model: X3D-XS

To be able to perform the online detection task in an optimal way, these models must learn how to produce good spatial-temporal representations, that encode the crucial movements of both judokas.

4.4 Contrastive frameworks implementation

As already said, the goal of this work is to analyze the performance of some contrastive methods on the downstream task of online action detection. Thus, starting from the examination of the source code published by the authors of the various methods, and from the information provided in their respective scientific papers, some of the previously described frameworks have been re-implemented. Specifically, have been chosen:

- MoCo v2
- SimCLR
- BYOL
- SimSiam

Initially, all those four contrastive methods will be re-implemented and tested, however, for the final experiments and the fine-tuning phase, some of them are dropped to focusing more on the methods that had shown the most promising results. During the re-implementations, all these frameworks have received some modifications to adapt them to the new datasets, backbone and device with which is carried out the training phase. The main variation consists of:

- *The data augmentation modules*: each method use a different set of data augmentation function, that however are quite similar to each other. Thus, it was modified to better fit the characteristics of the actual dataset, and standardized among all the different frameworks, to also makes fairer the comparison between them.
- *The architecture of the main encoder*: each method contains one main feature encoded which is the backbone that will be transferred and used for the different downstream tasks. All these methods mainly use as the default backbone the ResNet-50, 101 or 200. To match the architecture of the model used on the actual downstream task, the main feature encoder has been replaced with a ResNet-18 or with an X3D-XS net, depending on which is the chosen architecture.
- *The batch-normalization layer and the GPUs usage*: Initially, all these frameworks were projected to operate on a multi GPU device, or even on the TPUs,

to parallelize the computation and reduce the training time, that on big datasets like ImageNet can be very long. To execute them on the actually available machines, they have been adapted to operate on a single GPU device. For some methods like BYOL and SimSiam, this conversion has almost no influence on their general structure and behaviours, the only possible drawback is the increase of the training time. However, the same is not true for MoCo v2 and SimCLR, they both exploit the use of multiple GPUs to avoids leaks of information on the batch-normalization layer, by shuffling or aggregating the different samples or parameters among the different devices. On these implementations, that part was completely removed, and no additional operation is done on the data when they reach the batch-norm layer.

4.5 Training and test details

In this section will be described how the training and the testing phases, for both the online action detection task and the self-supervised contrastive task, are carried out with the different architectures.

4.5.1 Training phase

In this work, different parts of the models for the online detection task will be trained by using different types of training methods, more in detail, the convolutional networks, which are the backbone of those architectures, will follow a self-supervised contrastive method, while the GRU and the linear classifier will be trained in a supervised manner.
The training pipeline is the following one: first, the backbone of the model is injected inside a contrastive learning framework as its main feature encoder and the whole framework is trained by using the contrastive method. Once this training ended, the weights of the backbone are transferred into a second model and frozen by blocking the back-propagation of their gradient. Finally, this new model is trained in a supervised way.

For what concerns the supervised training, depending on the model's architecture, are used different parameter's configurations. If the model consists of the couple ResNet-18 plus GRU, it will be trained for 560 steps with the *cross-entropy loss* function. As the optimizer is used the *stochastic gradient descent* (*SGD*) with the momentum equal to 0.9 and an initial learning rate of $5 \cdot 10^{-2}$. Moreover, is implemented a *step scheduling* method that divides the actual learning rate by a factor 10, on steps 210 and 420. If instead is used the model based on the X3D-XS net, it will be trained for 600 steps with the cross-entropy loss function and the SGD optimizer with a momentum of 0.9 and an initial learning rate equal to 1.2 .

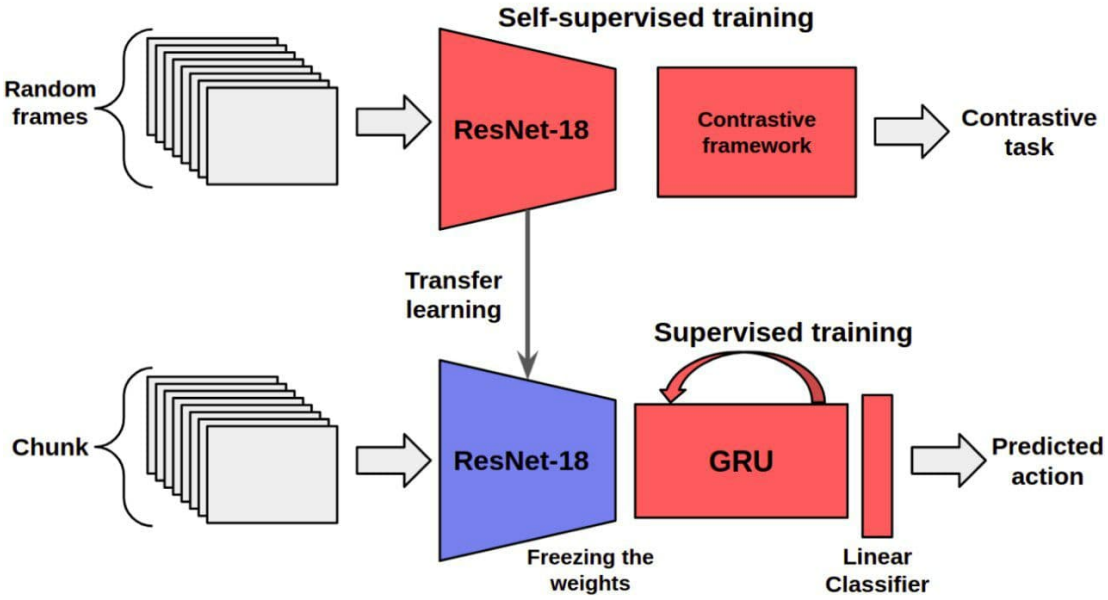

Figure 4.13: Training pipeline for ResNet-18, GRU and linear classifier

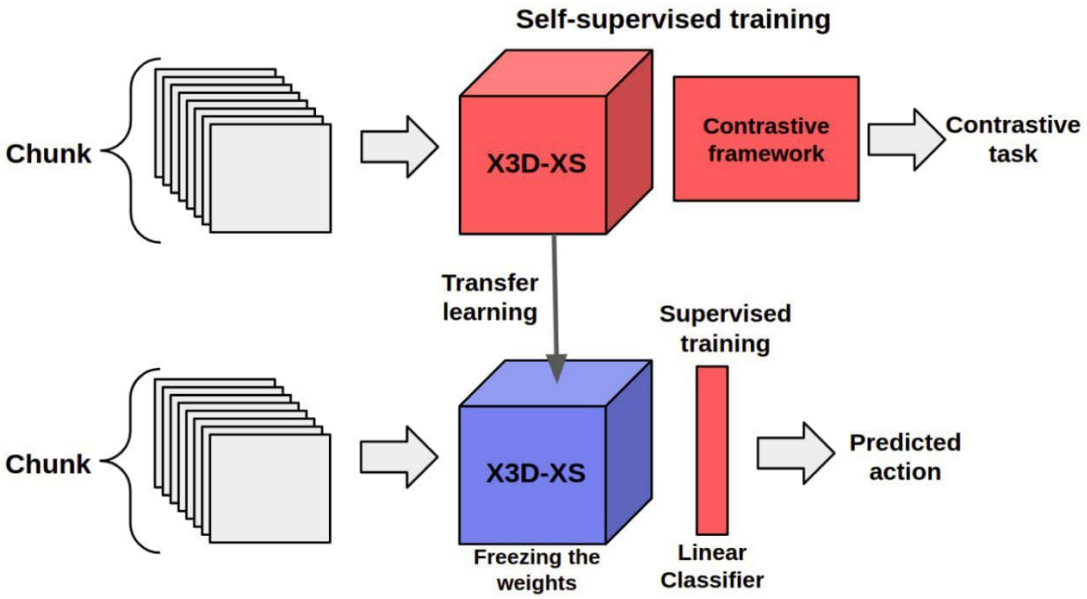

Figure 4.14: Training pipeline for X3D-XS and linear classifier

Also in this case is used a scheduler to modify the learning rate during the training, more in detail are implemented a *linear warm-up* for the initial 60 steps and a *cosine decay* during all the remaining steps.

In this training, a *"step"* corresponds to the process of feeding the model with a batch of chunks sampled from the clips contained in the dataset. For each clip extracted from the dataset on that step, one chunk is sampled following a uniform distribution. The size of the chunk can be variable but this work will use chunks of two possible lengths: 8 frames or 4 frames, while the size of the batch varies in relation to the length of the chunk and the resolution of the input video. These training set-ups will be used for the training of both the baseline and the backbone pre-trained with the contrastive methods, in this way is possible to obtain a more fair comparison between the representation produced by the different frameworks.

The baseline, necessary to estimate how effectively good are the representation obtained by the contrastive methods, is obtained by considering the performance reached by these models when the backbone is pre-trained in a supervised way on, ImageNet if it is the ResNet-18, or on *kinetics-400* if it is the X3D XS net.

Regarding instead the self-supervised training, all the implemented methods will follow the parameters' configuration proposed on their respective paper, exception done for the learning rate, the batch size and the number of training steps.
Moreover on this training method, must be noted that the concept of "step" will change depending on the backbone used. If the backbone is the X3D-XS, a step will correspond to the already given definition, if instead is used the ResNet-18 a step will consist of a slightly different process. In this last case, the backbone is a 2D-CNN that is not able to operate directly on the chunks of the clips but only on one frame at a time and in an independent way from each other. Thus, after extracting a batch of clips from the dataset, instead of sample a single chunk, from each clip are sample N random frames following a uniform distribution.

An important component of the contrastive methods is the data-augmentation module, and as already said, it was modified and standardized among all the frameworks. More in detail the data-augmentation function will be applied on all the input frames, and they are the following:

- *Central crop and resize*: The judoka are positioned on the centre of the workout mat, so a central crop of 1000×1000 is useful to remove useless information contained into the border, while the resize helps to reduce the resolution of each frame to 500×500.
- *Random crop and final resize*: after the first central crop is executed a further crop, in a random position of the frame and with a variable size between 350×350 and 450×450. It is then resized to its final resolution.
- *Random horizontal flip*: The frame is horizontally flipped, with a probability of 0.5 .

- *Random grayscaling*: Convert the RGB channel of the frame into a grayscale one, with a probability equal to 0.2 .
- *Random jittering*: The jittering process consist of modifying the brightness, contrast, saturation and hue of a given image. This will be applied to the input frames with a probability of 0.6 .
- *Random rotation*: The frame is rotated by a random degree within the range $(-10°, +10°)$, with a probability of 0.3 .
- *Random erasing*: This function consists of select a random rectangular region of the frame, and erase all the pixels in that region by setting their values to 0 for all the three colour channels. This function is applied with a probability of 0.5 .

All the probabilities were tuned once to regulate the difficulty of the contrastive task and then fixed for all the frameworks.
Some data augmentation functions are used also during the supervised training of GRU and linear classifier. More in detail are maintained the following data augmentations functions: the initial central crop, the resize, the random crop with a fixed size of 400×400, and the random horizontal flip with a probability equal to 0.4 .

Following the CVRL idea, all the transformations have been applied by forcing the usage of the same stochastic parameters, for all the frames in the same chunk. This allows maintaining consistency of the temporal information contained on them.
However, during the self-supervised training of the ResNet-18, the frames sampled from the clips are random and not correlated to each other. Thus, also the data augmentation functions will be applied independently between those frames.

4.5.2 Testing phase

As already said, on the online action detection task, the model works on a stream of frames that comes from a camera that is recording a video in real-time. To obtain a reliable testing procedure is necessary to replicate this situation, as similar as possible, by using the pre-recorded videos contained in the test set.

This can be achieved by using a *sliding window* technique. First of all, is fixed the size of the window, which in this case will coincide with the length of the chunk, then this window is slid along the entire clip to extract the various chunks. On doing that, two successive windows will be a bit overlapped and as consequences chunks near to each other will share some frames. The size of the overlapping will be regulated in relation to the length of the chunk, if it is equal to 8 frames, then

the overlapping will be of 3 frames, otherwise, if the chunk size is equal to 4 frames, the overlapping is reduced to 2 frames.
Once defined this approach to testing the model, becomes clear that the distributions of the samples into the test set can be rethought in terms of chunks, and not of clips.

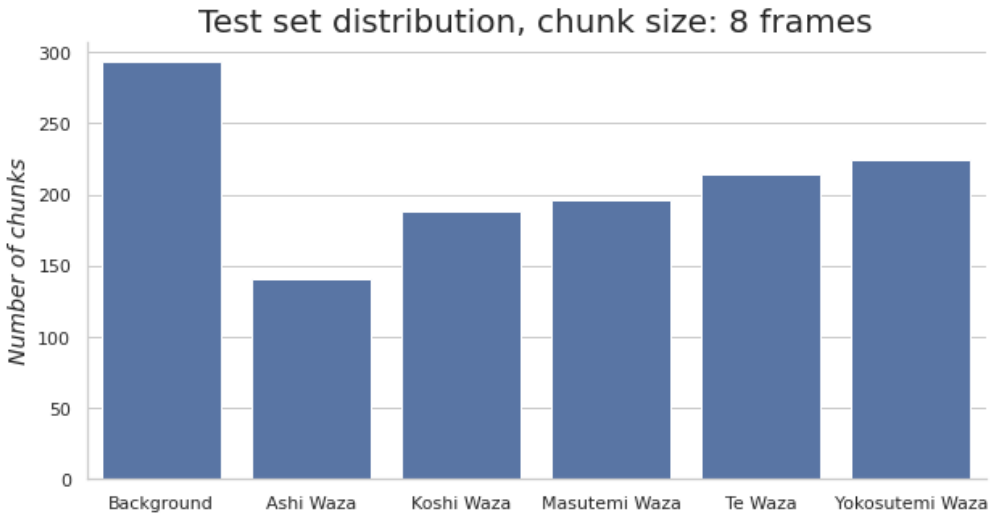

Figure 4.15: Distribution of the chunks on the test set with a chunk size equal to 8 frames

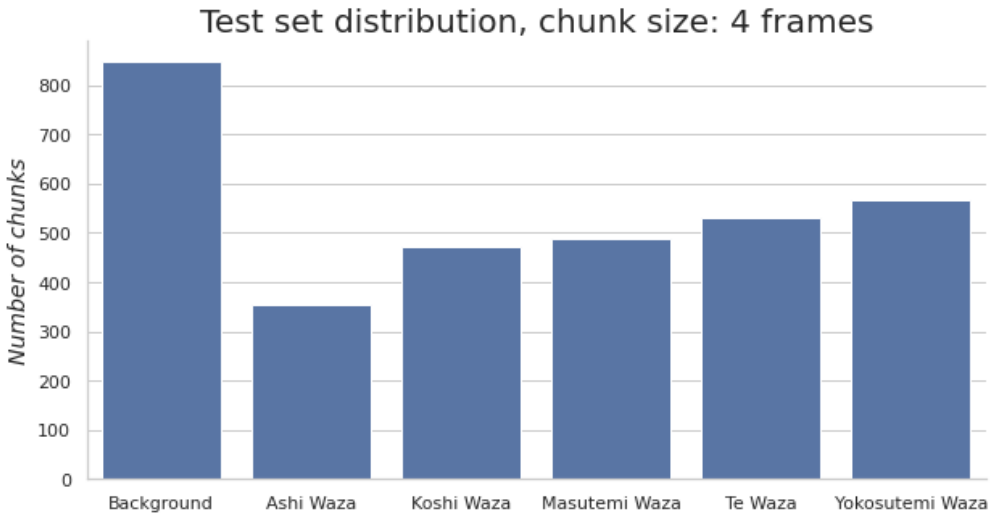

Figure 4.16: Distribution of the chunks on the test set with a chunk size equal to 4 frames

The figures 4.15 and 4.16 shown the new considered distribution of the test set, it is slightly biased towards the background class, which results to be the most numerous. Moreover as expected, by reducing the length of the chunk, the number of samples increases, unfortunately, seems that this makes the test set a bit more unbalanced.

Some data transformation functions are used also during the testing phase, however, in this case, the objective is not to perform an effective data augmentation of the clips, but just to pre-process them before being fed to the model. More in detail, they will consist of a series of central crops and resizes of the frames, to adapt them to the input resolution used during the training.

Chapter 5

Experiments and results

In this chapter will be reported and analyzed all the results obtained during the entire path of thesis development. First are shown the initial results obtained on the baseline models and with the default version of the contrastive methods. Then, the contrastive learning frameworks are modified by introducing a particular *hard-mining* method for the positive pair, to further improve the quality of the representation produced. Finally, are shown the new results obtained after the introduction of this variation.

5.1 Baseline and contrastive methods

Here will be shown the results relative to the baseline and to the base version of the contrastive frameworks, followed by an analysis of the training trends.

5.1.1 Results

Starting from the baseline, on the table 5.1, are shown the results obtained with both the architectures. For simplicity, the first model, which is composed of the couple ResNet-18 plus GRU will be named as the GRU model, while the other one, will be indicated as the X3D model.
During the training, these models will use different chunk-size and different resolutions for the final crop, trying to maintain the same input format used in their corresponding papers. The GRU model will receive frames of resolution 224×224 and chunks of length 8 frames, to ideally obtain a prediction after having seen less than 1/3 of a second of the action. While the X3D model will use chunks of only 4 frames with an input resolution of 182×182 pixels.
These configurations for the chunk and crop size will be maintained also during the self-supervised training of the corresponding backbone, in order to keep a coherent

input format during the whole training and make the comparison fairer.

Model	Chunk size	Frame size	Accuracy
GRU	8 frames	224×224	66.32%
X3D	4 frames	182×182	66.64%

Table 5.1: Baseline accuracy

The X3D model seems to provide a slightly better performance, however, since they adopt chunks of different lengths, the test set used for the evaluation will contain a different number of total samples and a different distribution of them over the class labels. Under this test condition, these two models are not directly comparable to each other, and the use of percentage values become not so reliable to understand the real ability of the models.

Therefore, to understand how good these values actually are, it may be useful to plot a *confusion matrix*. It is a technique used to review the actual performance of a classification algorithm. Given a confusion matrix $M_{i,j} \in R^{NxN}$, where N is the number of distinct classes, an element will be added into the position (i, j) if its true class label is i and the predicted class is j. Thus, the row i will include all the samples that belong to the class i, while the column j will contain all the samples for which the algorithm have predicted the class j. It helps to recognize where the model is performing well and in which cases it does more mistakes. If the samples are all correctly classified, they will be distributed along the main diagonal of the matrix.

On the figures 5.1 and 5.2 are plot the two confusion matrices for the baseline models. Can be observed that the major difficulty for both models, stands on the classification of the background chunks by predicting them as action, or predicting an action as background. This difficulty could come from the not so rigorous labelling process of the background class and from the already said problem with the identification of the correct initial frame for the action. Moreover, must be noted that some chunks could contain frames that come from both the background and the action class, making even more difficult its correct classification.
Another mistake that is sometimes done by both models, is on the discrimination between chunks of actions that belongs to the same *macro-category* of the judo moves. Indeed, can be seen that the mistakes are mainly related to the discrimination between Masutemi Waza and Yokosutemi Waza which are both sacrifice techniques, or Koshi Waza and Te Waza which are both standing techniques.
From a further analysis has been noted that all the chunks extracted from the same clip, are usually predicted on the same action class. Thus, when the model

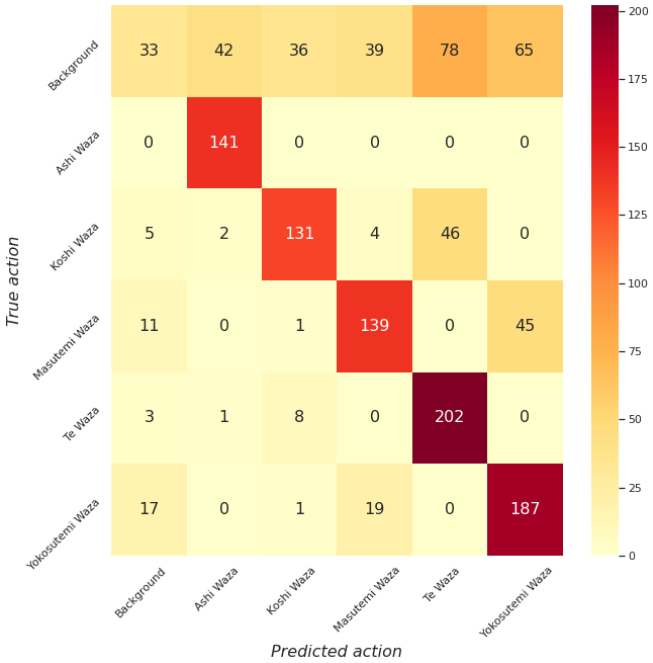

Figure 5.1: Baseline: GRU model

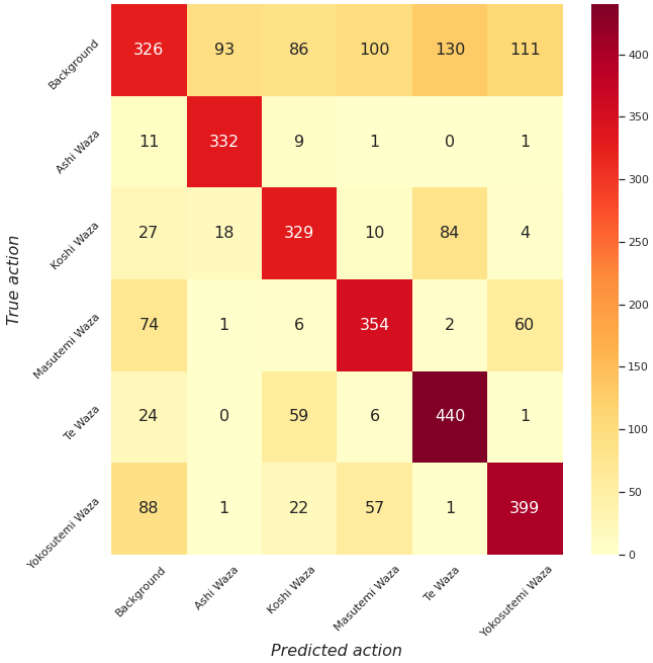

Figure 5.2: Baseline: X3D model

mistakes a class label, it does the exact same error for almost all the chunks that belong to the same clip.

In the table 5.2 are shown the results obtained by the GRU model, with the backbone pre-trained on the different contrastive frameworks. All the backbones have been trained for 3000 steps, with a batch size of 210 random frames.

Contrastive framework	Accuracy
MoCo v2	75.95%
SimCLR	60.83%
BYOL	—
SimSiam	64.25%

Table 5.2: Comparison of the accuracy reached by the GRU model with the self-supervised pretraining of the backbone

Among these methods, MoCo was the only one able to outperform the baseline by a +9.63%, while SimCLR and SimSiam perform poorly by remaining below the baseline of a −5.49% and −2.07% respectively, and BYOL does not converge at all. It reaches a degenerative solution, so the obtained representations are useless for any possible downstream task.
A possible explanation of the reason for which SimCLR perform so poorly can be related to the batch size used during the training, that framework in fact heavily rely on a huge batch size to obtain a high number of negative samples. Actually is used a batch of 210 samples, which is much lower than the batch of 2048 samples proposed on its paper.
This problem instead, seems do not affect MoCo which use a queue to increase its number of negative samples, and SimSiam that was presented as a framework able to produce competitive results even with a small batch size.

The confusion matrix in the figure 5.3, is produced by using the results obtained with MoCo, can be noted that in general, the model is now able to better recognize the background class, and also the errors within the macro-class are lower.

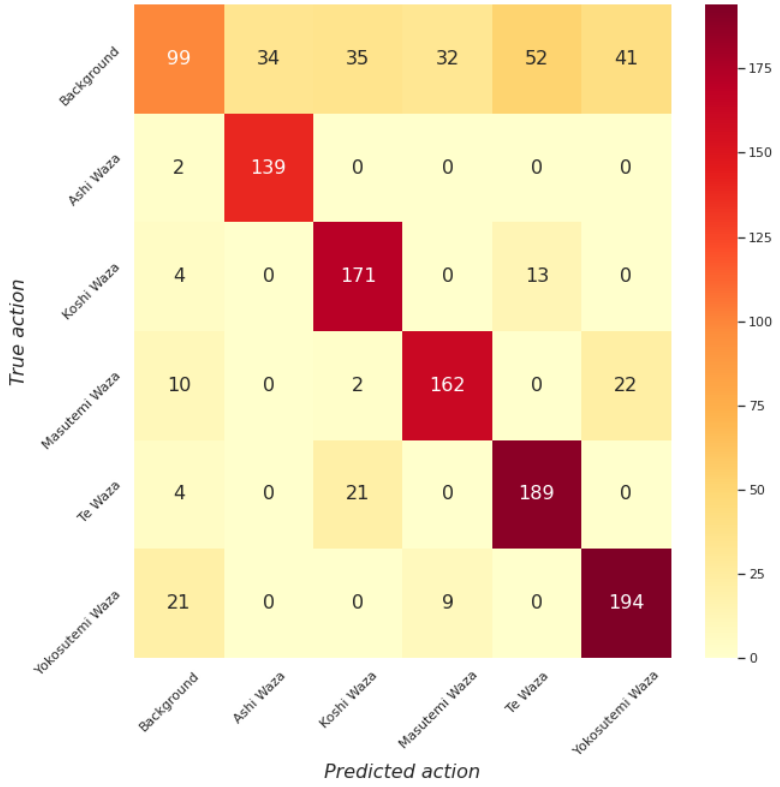

Figure 5.3: GRU model pre-trained with MoCo v2

Now, in the table 5.3 are shown the results obtained by the X3D model, with the backbone pre-trained in a self-supervised way. In this case, the backbone will be pre-trained for 4500 steps with a batch size of 40 chunks. Starting from the knowledge of the results just obtained with the ResNet-18 backbone, for the X3D model will be tested only MoCo v2 and SimSiam, which are the only two methods that in the previous case showed the most promising results.

Contrastive framework	Accuracy
MoCo v2	72.25%
SimSiam	67.71%

Table 5.3: Comparison of the accuracy reached by the X3D model with the self-supervised pretraining of the backbone

In this case, both methods were able to outperform the baseline, by a +5.61% for MoCo and a +1.07% for SimSiam so, with MoCo still performing better than SimSiam.

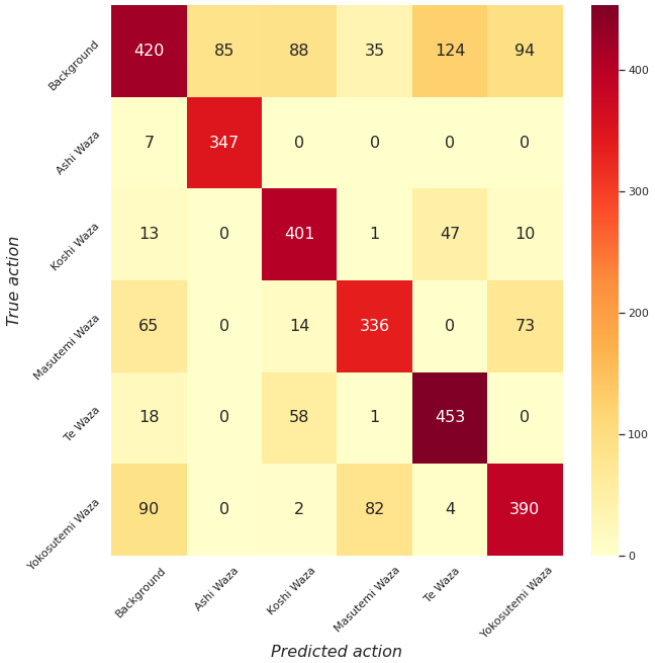

Figure 5.4: X3D model pre-trained with MoCo v2

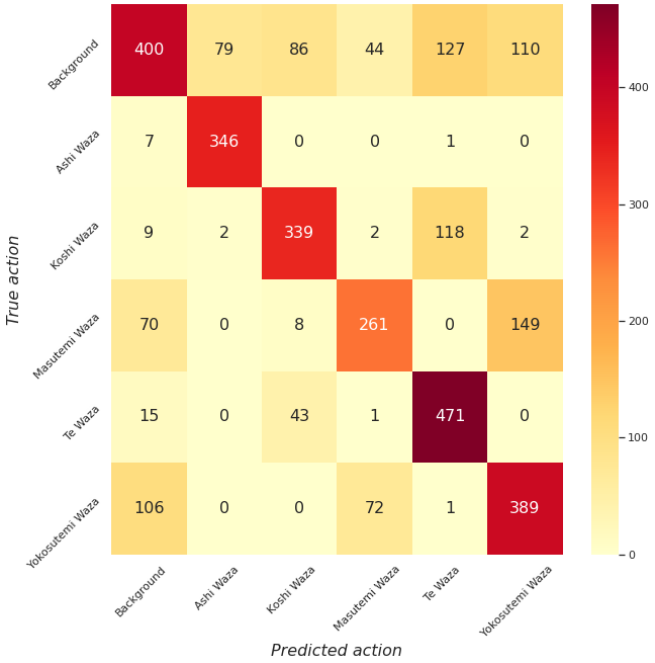

Figure 5.5: X3D model pre-trained with SimSiam

As can be seen in the figures 5.4 and 5.5, MoCo was able to improve on both the classification of the background chunks and the discrimination within the macro-class of the moves. Unfortunately, the same thing is not true for SimSiam, which improve on the recognition of the background, but presents more difficulty with the distinction of the different class actions.

5.1.2 Analysis of the training behaviour

All these self-supervised trainings have been carried out by using a number of training steps much greater than the one used by the corresponding supervised training for the GRU and the linear classifier. Usually, these contrastive methods gain more from longer training than the standard supervised method. However, the progress of the training of the models have been monitored and analysed to understand how really useful is to train them for 3000/4500 steps, or if the training can be stopped before, by reducing the training time and so the computational cost.

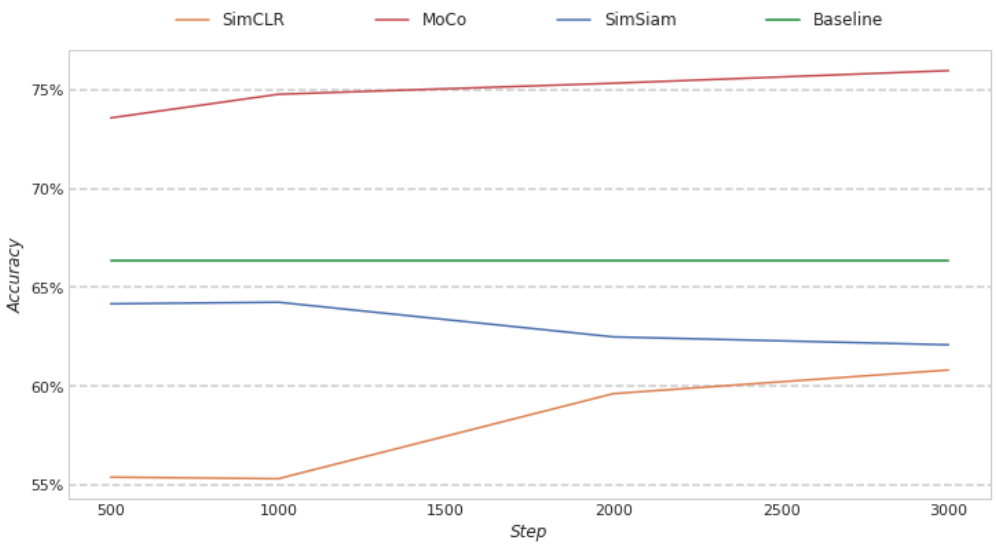

Figure 5.6: Accuracies of the GRU models

Regarding the GRU model, both MoCo and SimCLR show a clear improvement on the downstream task, as the number of training steps increases. For these two methods, a longer training seems to be beneficial for the quality of the final obtained representations. SimSiam instead, reaches its better performance with 1000 steps, and further training of the backbone results to be only harmful to the performance on the downstream task.

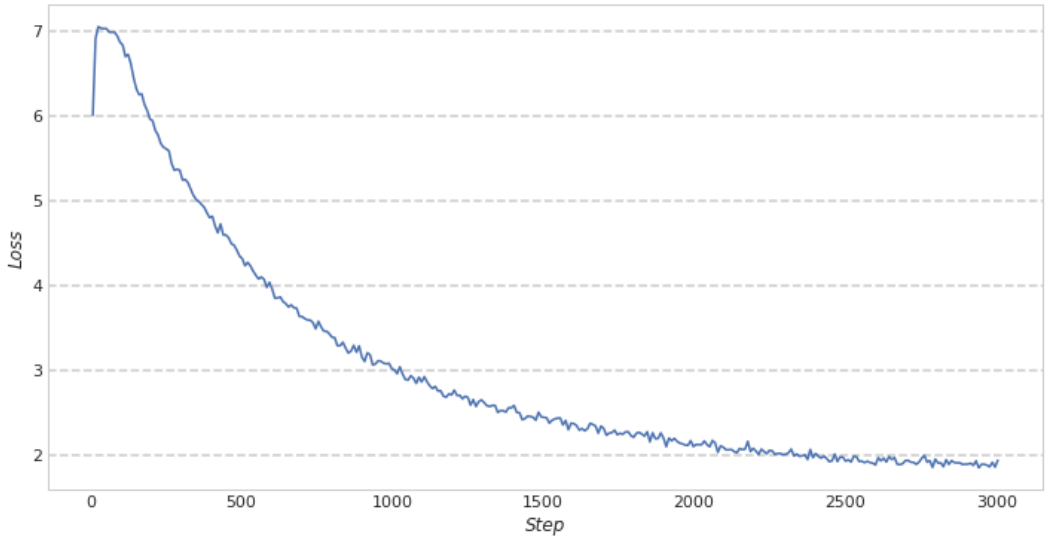

Figure 5.7: Training loss of MoCo v2: ResNet-18 as backbone

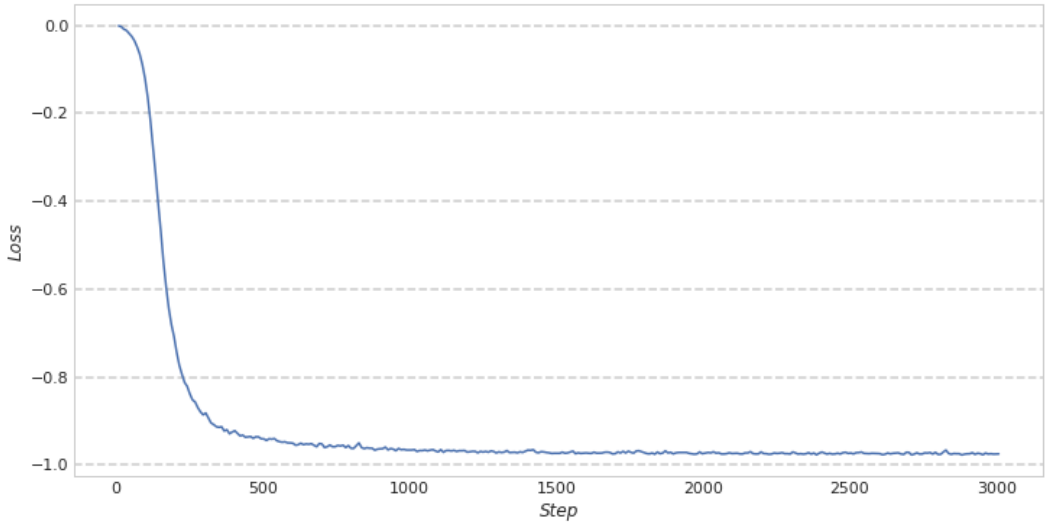

Figure 5.8: Training loss of SimSiam: ResNet-18 as backbone

The graphs on the figures 5.7, 5.8 and 5.9 show instead, the trend of the various training loss used by the different methods. Can be noted that in general, the losses tend to decrease principally during the initial 500 steps, then start a slow decrease until it reaches a plateau during the final steps. In particular, SimSiam reaches this plateau much before that the other method, while both MoCo and

SimCLR show a fast increase of their loss during the very few initial steps.

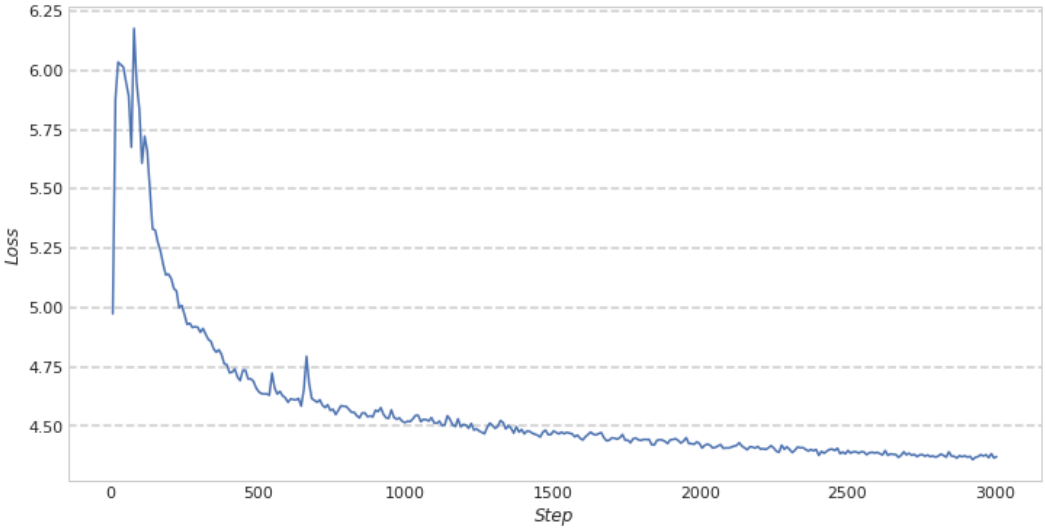

Figure 5.9: Training loss of SimCLR: ResNet-18 as backbone

For MoCo, this behaviour could be related to the process of filling the queue, during the first step the queue will be empty so the contrastive task is easier because there are fewer negative samples with which compare the actual positive pairs. From the second step onward the queue begins to be filled, the negative samples start to increase and the model has more difficulty on discriminate which is the correct positive sample between them, as consequences also the loss starts to increase. As soon as the queue reaches its maximum capacity, the model could face a more stable condition where the number of negative samples remains fixed and the loss could start to decrease.

For what concerns instead, the self-supervised training of the X3D models, the achieved accuracies are shown in the figure 5.10, while the graphs on 5.11 and 5.12 are reported the training losses of the contrastive method.
Moco exhibits a behaviour quite similar to the previous case, with the downstream accuracy that increases with the number of training steps. Surprisingly, the same behaviour is shown by SimSiam which, differently from before, gain from a longer training like MoCo.
The training losses instead have both a trend similar to the previous case, they seem to be characteristic of the contrastive method and independent from the backbone used during the training.

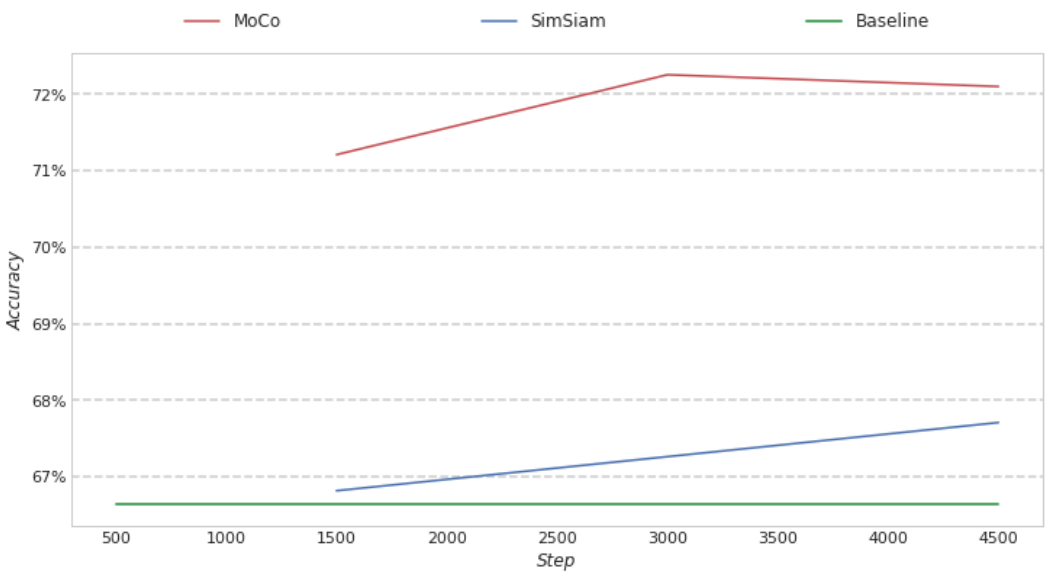

Figure 5.10: Accuracies of the X3D models

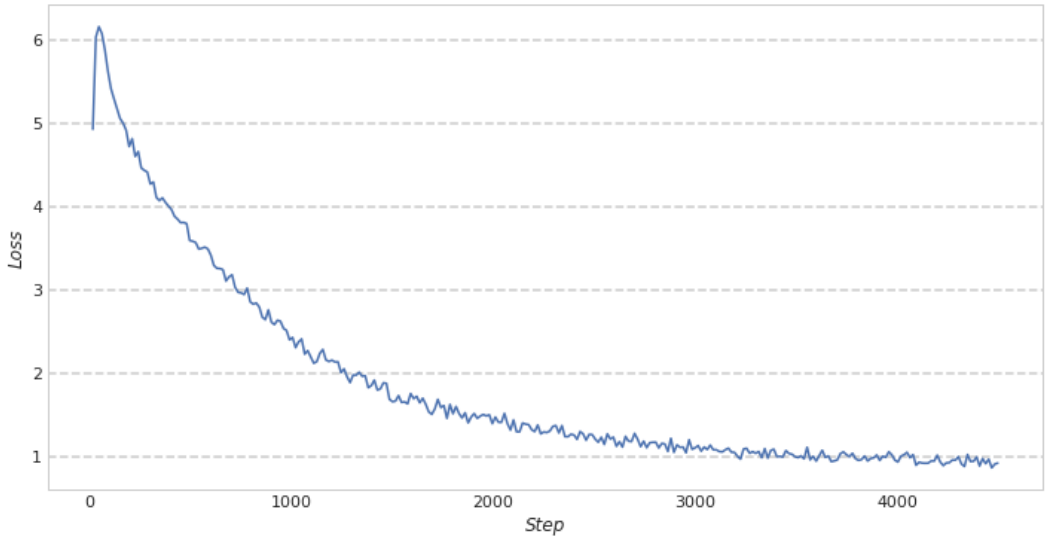

Figure 5.11: Training loss of MoCo: X3D-XS as backbone

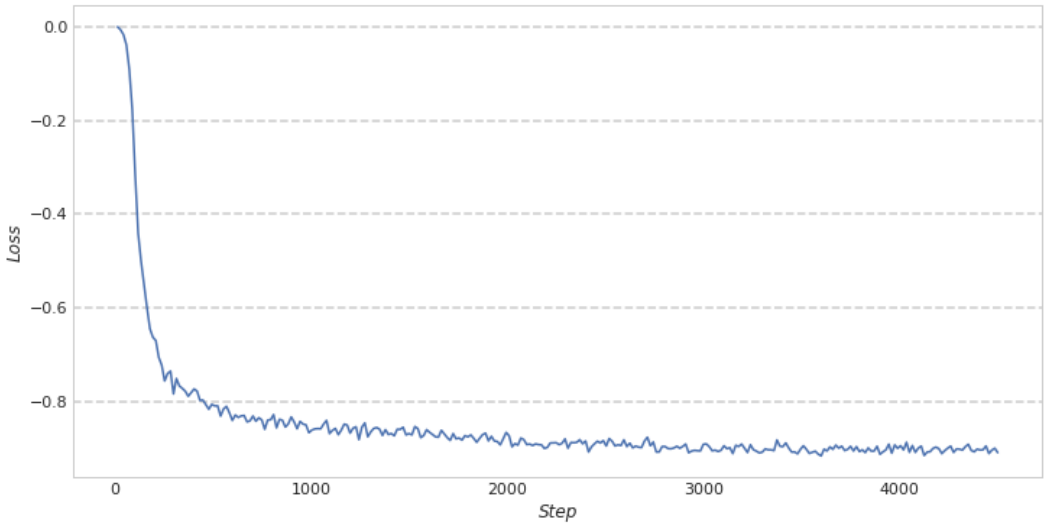

Figure 5.12: Training loss of SimSiam: X3D-XS as backbone

5.2 Hard positive mining

Was already said that the contrastive methods gain a lot from receiving input pairs of data on which the distance between the positive and the anchor is much greater than the distance between the negative and the anchor. Usually, these pairs are defined as *hard positive* or *hard negative* pairs.
A hard positive pair is a particular positive pair composed of images that look quite different to each other but contain the same information. Similarly, a hard negative pair is composed of images that look similar to each other but have different semantic information. These two categories of pairs, are particularly tough for the models and make the contrastive task more challenging to solve. By increasing the difficulty of the task, the model is forced to develop higher-quality features to overcome that difficulty. It must learn to recognize the similarity between images that looks different and the difference within similar images.

Starting from this assumption, it is possible to improve the quality of the representations produced with these contrastive methods by enforcing the use of the *hard pairs*. In particular, this work proposes to enforce the use of the hard positive pairs by mining them from the dataset. The main idea is to exploit the characteristic of the dataset of having records of the same time instant from a different point of view, thanks to the use of multiple video cameras at the same time. Two videos that have registered the same action, but from two different points of view, can be considered as a good example of a hard positive pair. Even if the action is

exactly the same, the different angulation of the camera makes them look quite different to each other, because some movements that look clear from a point of view can become more difficult to see and recognize from another angulation. For example, if a camera record a judoka from its back, the movement of its hands are not so visible since are mainly covered by its body, if instead the camera is positioned in front of him, that action becomes much more clear to see and recognize.

Thus, during the self-supervised training of the model, the positive pairs will be composed of frames or chunks, sampled from two different recorded videos of the same action. Moreover, these samples will correspond to the same temporal instant, in order to maximize the similarity of the semantic information contained. In this way is possible to enforce the model to produce representations invariant to the point of view of the video camera.

A similar method is used on [30], in this paper Sermanet et al. propose to use a self-supervised approach to produce representation useful on some robotic imitation tasks, like the *imitation of human pose* and *imitation of object interaction*. They use a triplet loss where the anchor and the positive comes from the same timestamp of two different views, and the negative is sampled from the same view of the anchor but is temporary distant enough to contain a different set of semantic information.

5.2.1 Results

The table 5.4, shows the new results reached by the GRU model after having added the hard positive mining into the contrastive methods. Like in the previous case, the backbone will be pre-trained for 3000 steps with a batch size of 210 random frames.

Contrastive framework	Accuracy
MoCo v2	80.65%
SimCLR	55.89%
SimSiam	70.94%

Table 5.4: Comparison of the accuracy reached by the GRU model with the self-supervised pretraining of the backbone and with the hard positive mining

In this case, both MoCo and SimSiam outperform the baseline by respectively a +14.33% and a +4.62%, with MoCo still performing better than SimSiam. They show also a clear improvement with respect to their corresponding versions without the hard positive mining, with a further increase of the accuracy by a +4.7% for MoCo and a +6.69% for SimSiam.

SimCLR instead, still performs poorly and remain below the baseline by a -10.43%, moreover seems that for this method the hard positive mining does not help at all, causing a further drop of the accuracy by a -4.94%, maybe the contrastive task has become too much difficult for this method, making also the convergence to an optimal point more challenging.

To better understand the new performance reached by these models, in the figures 5.15 and 5.16 are plotted the confusion matrices obtained from MoCo and SimSiam.
MoCo is the one that provides the best downstream task performance, the error on the distinction within the different macro-class are almost gone, and also the accuracy on the background class is increased. Maybe, the new representations produced by MoCo are able to focus more on the positions of all the body parts, especially on arms and legs which are the initial parts that usually a judoka moves to start the action, and that could greatly help to recognize which action is going to be performed.
SimSiam instead, shows a clear improvement on the distinction of the different actions, but unfortunately, becomes unable to correctly classify any possible background chunks, by making the standard version without hard positive mining, or also the baseline, somehow preferable to this last one.

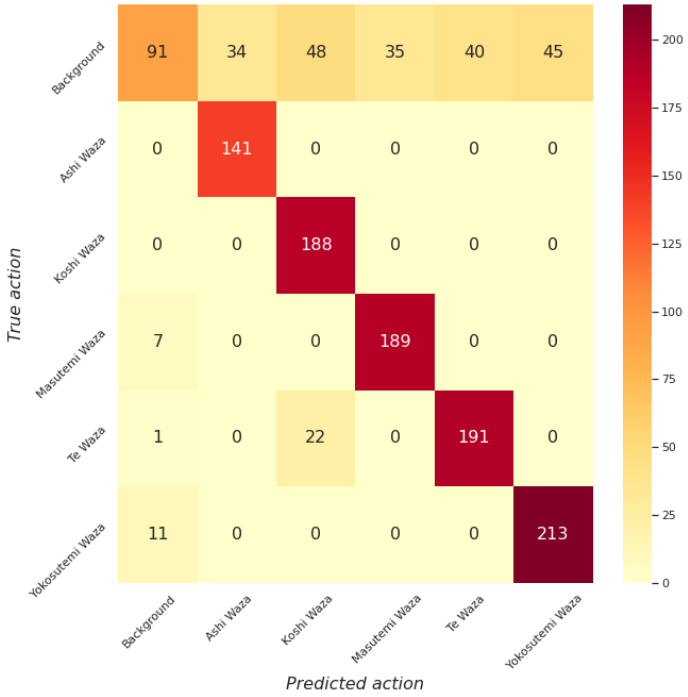

Figure 5.13: GRU model pre-trained with MoCo v2 and hard positive mining

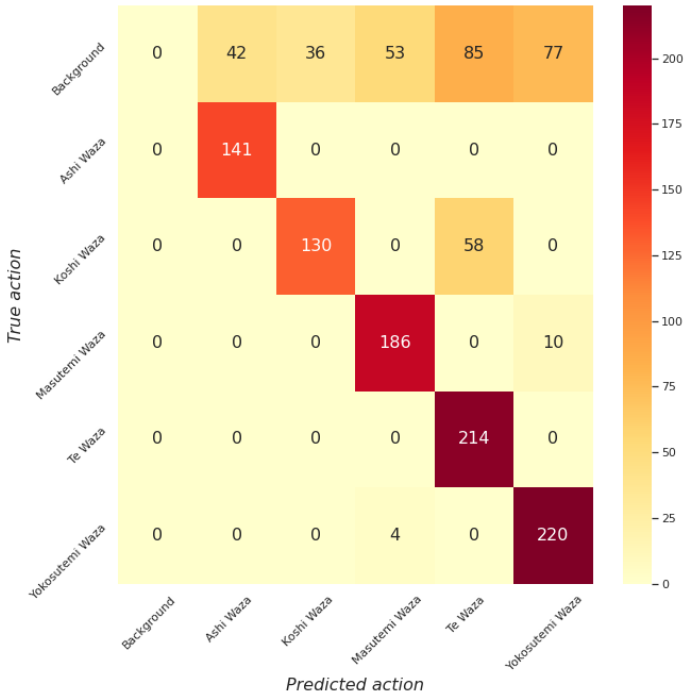

Figure 5.14: GRU model pre-trained with SimSiam and hard positive mining

In the table 5.5 instead, are reported the results obtained by the X3D models, after the introduction of the hard positive mining. The self-supervised pre-training of the backbone will last for 4500 steps and with a batch of 40 chunks.

Contrastive framework	Accuracy
MoCo v2	73.79%
SimSiam	71.79%

Table 5.5: Comparison of the accuracy reached by the X3D model with the self-supervised pretraining of the backbone and with the hard positive mining

Also with the X3D models, both MoCo and SimSiam are able to outperform the baseline by a +7.15% and a +5.14% respectively. Moreover, like for the GRU models, the use of hard positive mining has led to a further increase of the accuracy in the downstream task by a +1.54% for MoCo and a +4.08% or SimSiam.

Following the previous flow, will be plotted the confusion matrices of these two models to better understand their true performance.

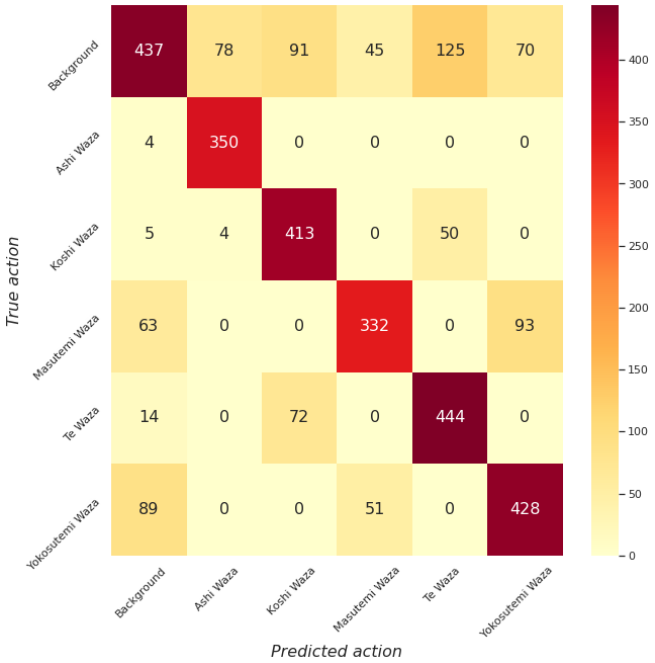

Figure 5.15: X3D model pre-trained with MoCo v2 and hard positive mining

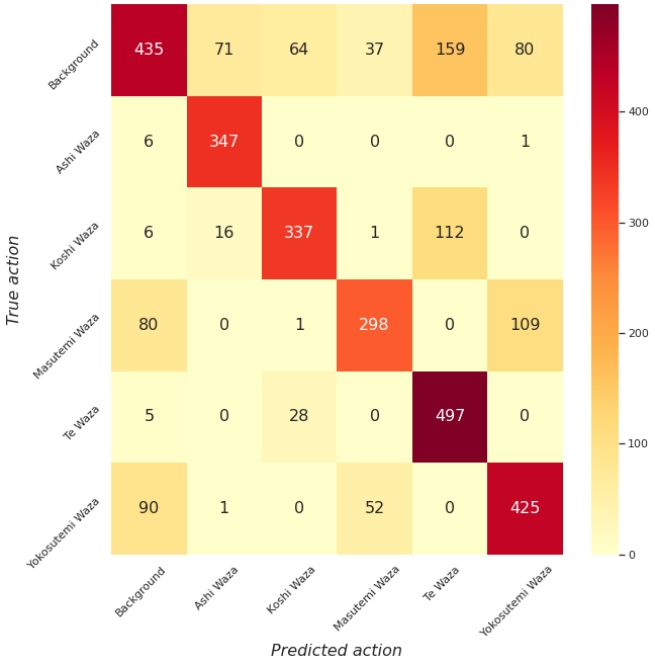

Figure 5.16: X3D model pre-trained with SimSiam and hard positive mining

Both methods show similar overall behaviour, with some mistakes on the distinction of the chunks within the two macro-class, and a small improvement on the recognitions of the background chunks.
However, MoCo still performs better than SimSiam, especially on the recognition of the different actions. Indeed this latter matches MoCo on the recognition of the background class, but do more mistakes on the classification of all the other actions.

5.2.2 Analysis of the training behaviour

In the same way as before, the behaviour of the model during the whole self-supervised training have been monitored and analyzed, to discover if also in this case a longer training is beneficial or not.

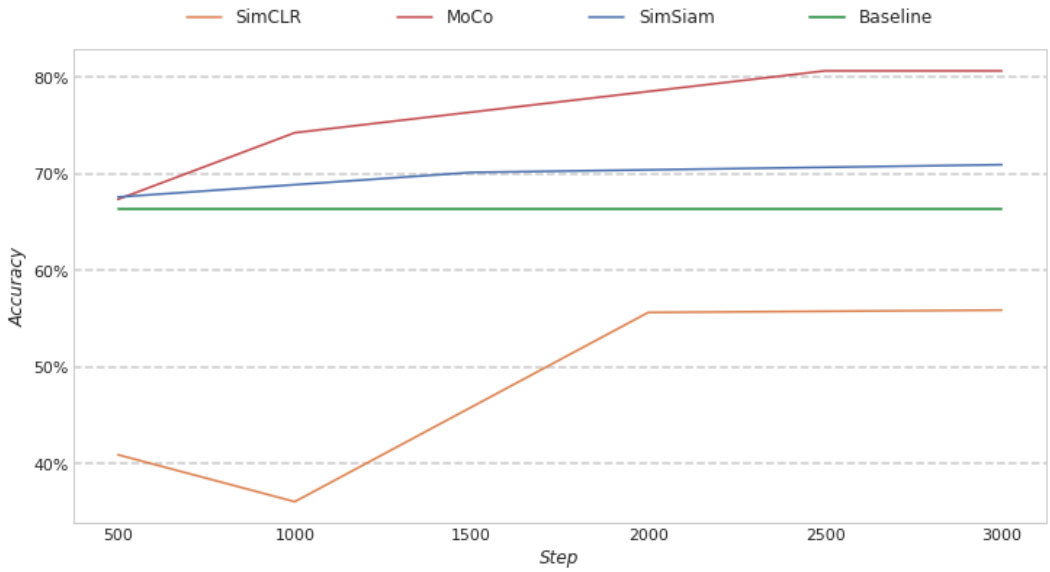

Figure 5.17: Accuracies of the GRU models with hard positive mining

From the graphs on the figures 5.17, 5.18, 5.19 and 5.20, can be observed that longer training results to be beneficial for all the methods, since their downstream accuracy increase with the number of training steps.

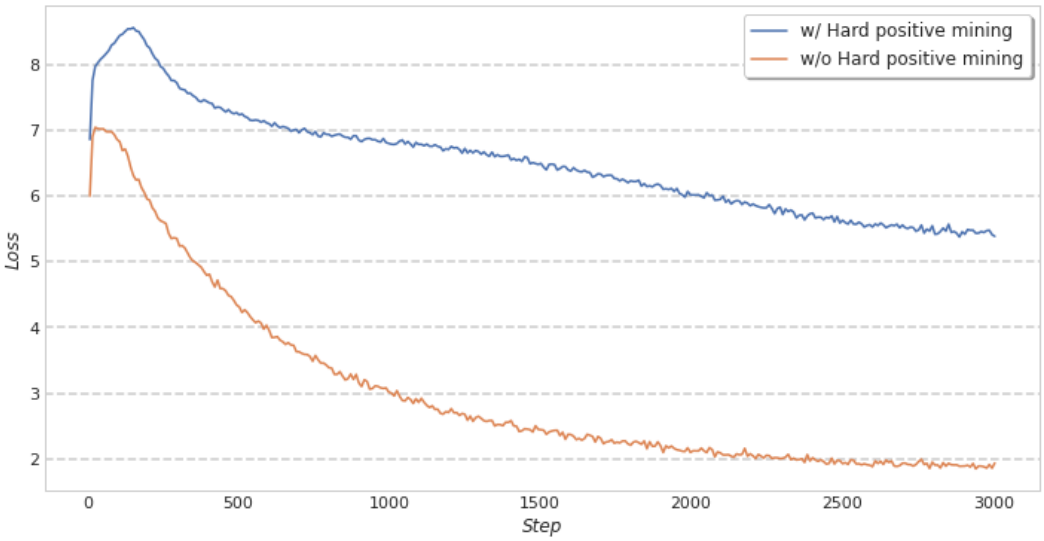

Figure 5.18: Training loss of MoCo: ResNet-18 as backbone

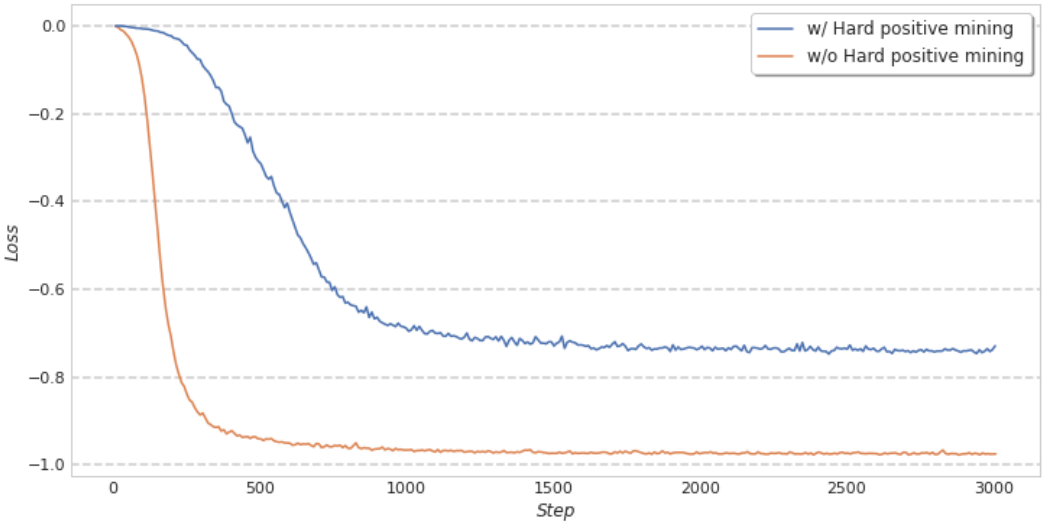

Figure 5.19: Training loss of SimSiam: ResNet-18 as backbone

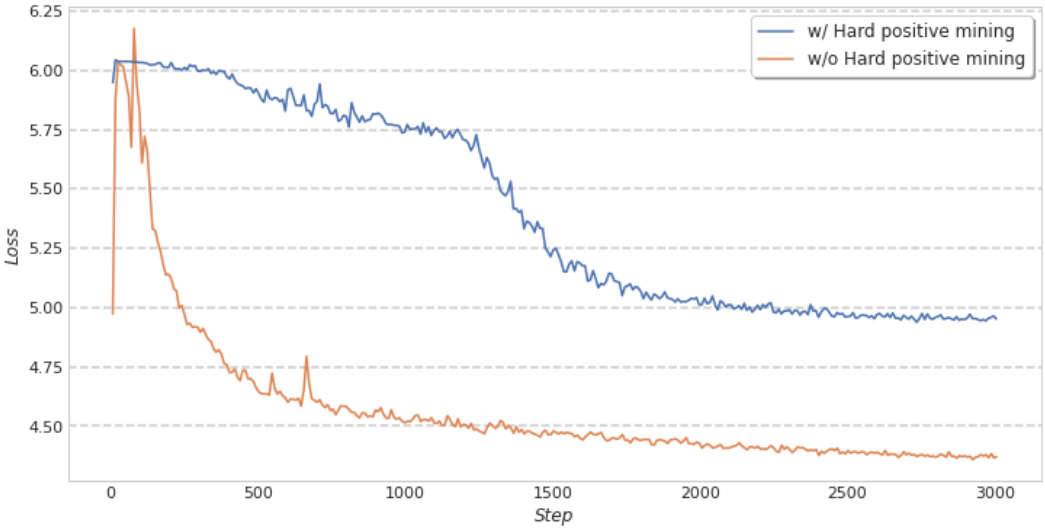

Figure 5.20: Training loss of SimCLR: ResNet-18 as backbone

In particular, MoCo follows the same behaviour that has already been recorded in all previous cases. It has a fast initial increase of the training loss, followed by an equally quick decrease, ended by a slow decrease and a final plateau. The same consideration can be applied also to SimSiam. SimCLR instead shows a trend quite different than before, its training loss has an initial plateau followed by a fast drop and a subsequential plateau. Is interesting to note that for all three methods, the training loss remain always higher than the loss reached by their corresponding version without the hard positive mining.

Regarding instead the X3D models, from the graphs on the figures 5.21, 5.22 and 5.23 can be seen that, also in this case MoCo and SimSiam maintain their characteristic behaviour, both in terms of loss trends and accuracy.
Like on the GRU models, also for the X3D, the sampling of the hard positive pairs leads to an overall higher train loss, both at the beginning and at the end of the training.

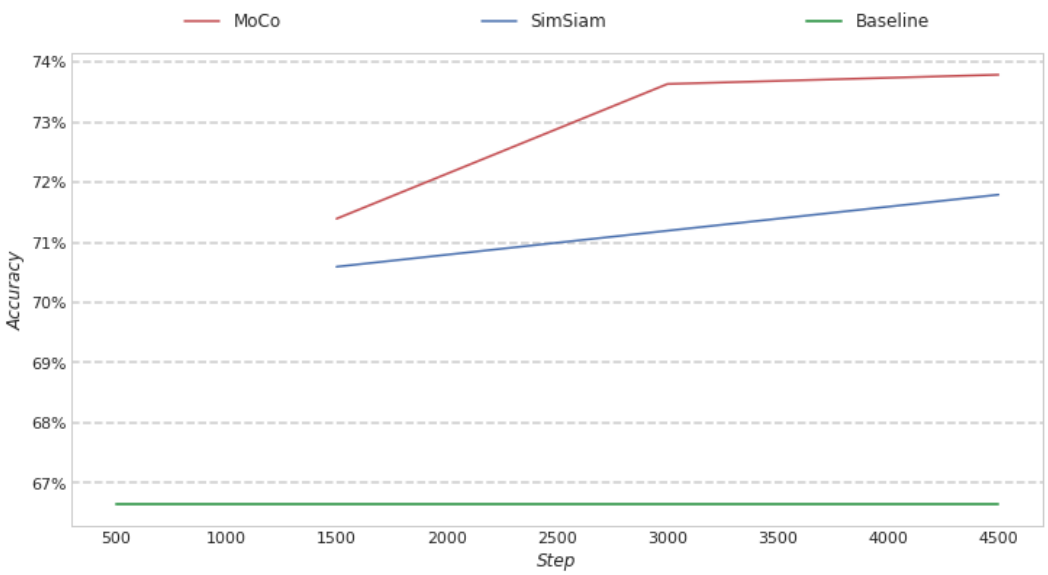

Figure 5.21: Accuracies of the X3D models with hard positive mining

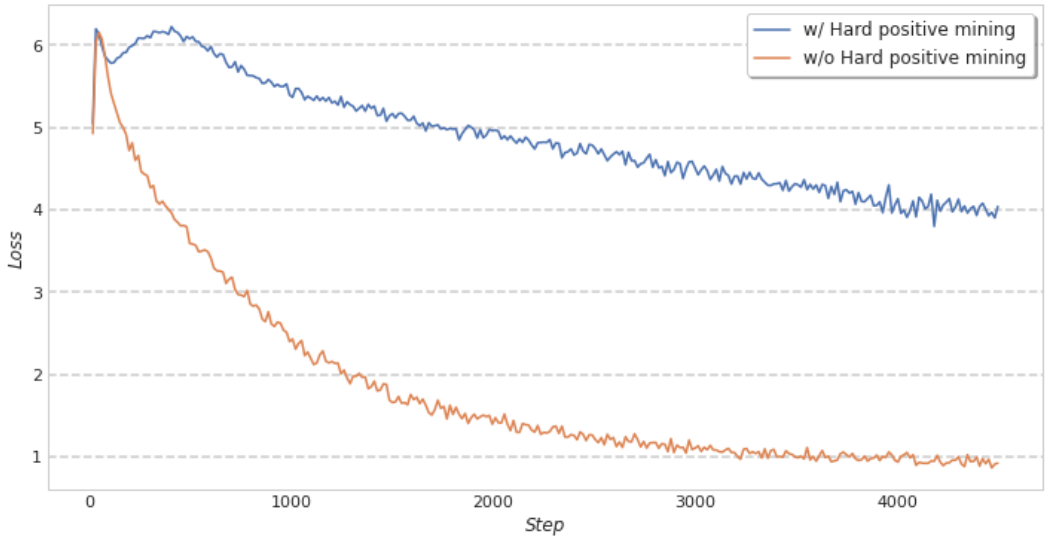

Figure 5.22: Training loss of MoCo: X3D-XS as backbone

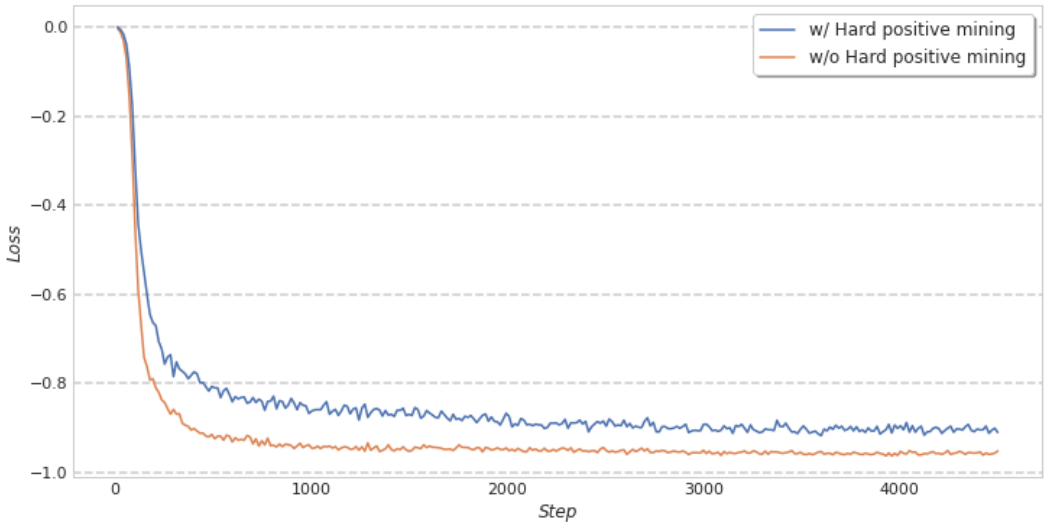

Figure 5.23: Training loss of SimSiam: X3D-XS as backbone

Usually, a higher loss means that the model does many mistakes, and so its outcome is quite different from what is expected. As consequence, also its performance on the task and the quality of the learned representation are expected to be poorly. However in this case, even if the contrastive loss is higher, the performances on the downstream task are always better than before. This confirms that a lower loss on the self-supervised contrastive task does not necessarily mean a better quality on the final representation.

The reason for which the contrastive loss increase so much, can be easily related to the mining of the hard positive pairs. As matter of fact, those new pairs really makes the contrastive task more difficult than before, since they are composed, no more by the same frame with just some variation given by a different data augmentation pipeline, but by two different frames recorded from a different point of view, and with also a different data augmentation pipeline.
Thus, even if the model becomes able to produce a really good representation for the input data, their relative distance into the latent space will remain greater than before and as consequence, also the loss will tend to increase.

However, these new representations contain an invariance to the point of view of the camera that results to be a very useful property to achieve optimal results on the downstream task.

5.3 Further changes and improvements

MoCo results to be the contrastive framework with the best performance among all, both before and after the addition of the hard positive mining. To further improve its downstream task accuracy, some hyperparameters of the training process or some elements of its architecture are modified to understand how they could influence the behaviour of the model. More in detail are tested the following variations:

- Longer training process.
- Wider output size for the projection head.

5.3.1 Longer training process

It is already known, and have also been shown that contrastive methods gain a lot from a longer training, more than usually do the standard supervised methods. This is particularly noticeable when is pre-trained the backbone of the GRU model since in that case the plateau on the accuracy is reached later on than in all the other cases. Thus, the number of training steps is increased from 3000 to 10000. The accuracy achieved is 83.04%, the model obtains a further increase on the downstream performance by a +2.47%.
With the increment of the training steps, there are two factors that mainly changes during the training, the first is the number of possible different samples seen by the model, the other one is the trend of the learning rate. This implementation of MoCo uses a cosine decay of the learning rate, as a consequence when the number of training steps changes, the variations of the learning rate at the end of each step, will be automatically adapted to the new number of steps.

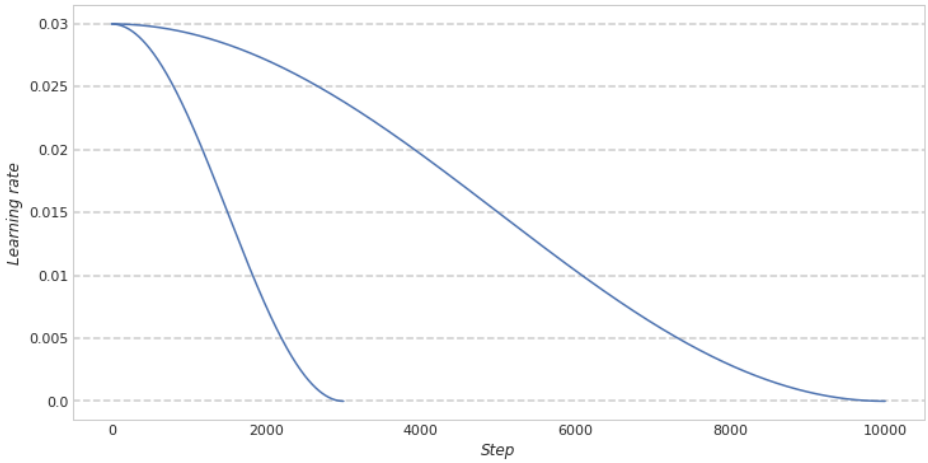

Figure 5.24: Cosine decay of the learning rate

In the figure 5.24 is reported the trend of the learning rate for both training conditions. Can be noted that with a higher number of steps, the learning rate will decrease slowly than before, allowing also the net to learn more and for a longer time.

5.3.2 Wider output for the projection head

All the proposed contrastive frameworks have some architectures choice in common, the main one is the use of a projection head to map the representation on a smaller latent space where the contrastive loss is applied.
On the actual implementation, MoCo uses a projection head with 128 output neurons and a hidden dimension of 2048 neurons. Now, the output size of the head will be progressively increased until it matches the size of its hidden layer, as consequence, the contrastive loss will be computed on a bigger latent space, where is also possible to encode more information. In this way, the model should be encouraged to produce higher quality representations, to better perform on the contrastive task.

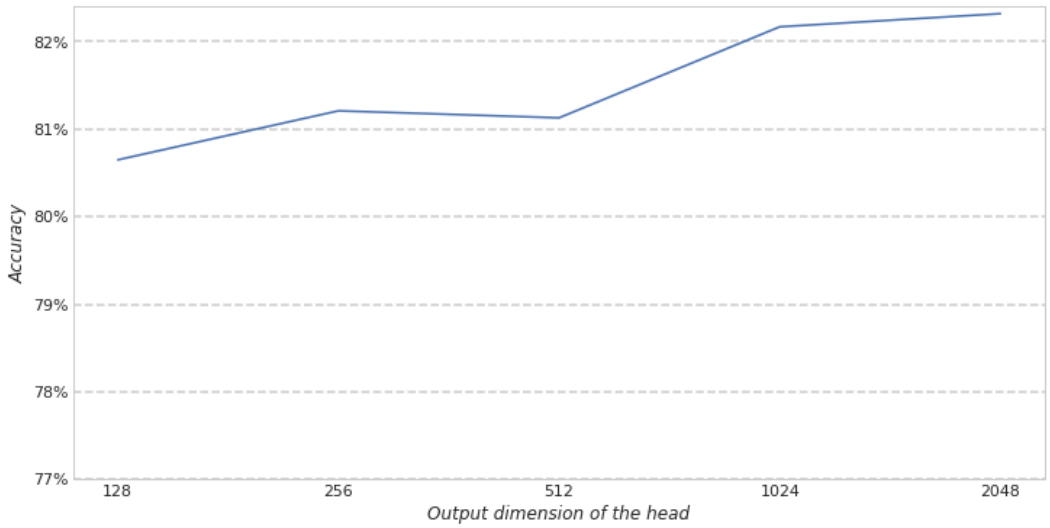

Figure 5.25: Accuracy of the GRU model for different size of the prediction head on MoCo v2

The figure 5.25 shows the accuracies obtained by the GRU model under the different settings, the highest one is a 82.33% and is reached with a prediction head of 2048 output neurons. It has an increment of a 1.68% with respect to the default setting.

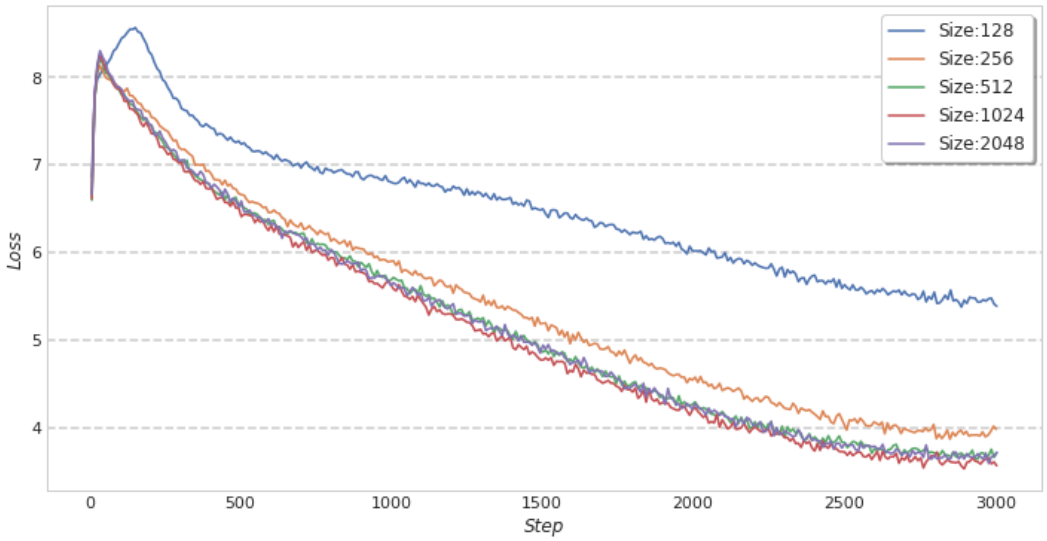

Figure 5.26: Training loss of MoCo v2 with: ResNet-18 as backbone, hard positive mining and different size for the output head

From the graph on the figure 5.26 can be observed that Independently from the output size of the head, the training behaviour of MoCo remains almost the same. The only difference consists of the value of the contrastive loss, that with a greater is the output size tends to decrease quickly.

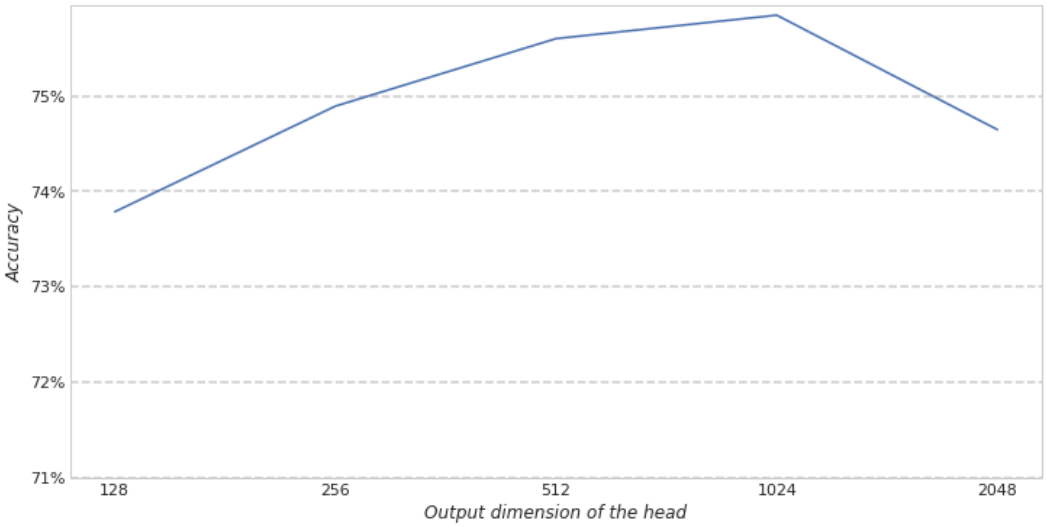

Figure 5.27: Accuracy of the X3D model for different size of the prediction head on MoCo v2

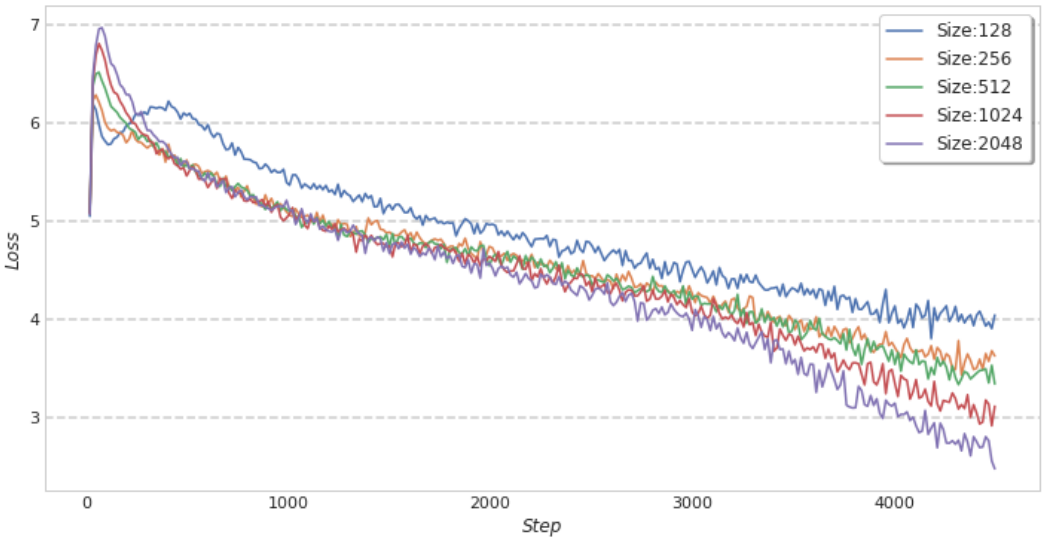

Figure 5.28: Training loss of MoCo v2 with: X3D-XS as backbone, hard positive mining and different size for the output head

Regarding instead the X3D model, figure 5.27 shows the obtained accuracies. The highest one is a 75.84% and is obtained with a prediction head of 1024 output neurons. It shows an increment of a +2.05% with respect to the default setting. Differently from the GRU model, a further increase of the prediction head leads to a deterioration of the downstream task performance.

The figure 5.28 show the trends of the contrastive loss for different output dimension of the projection head. With a greater output size can be observed two particular behaviours, the first is that the initial peak reached by the loss tends to become higher, while the other one is that the loss decrease more quickly during the last 1000 steps than it usually does with smaller output size of the head.

Thus overall, a bigger projection head could help the model to rapidly decrease the contrastive loss function while maintaining a high quality of the representations.

Chapter 6

Models comparison and conclusion

In this last chapter, first will be done some further comparison between the different models, by taking into account the different possible settings and methods with which they can be trained. Then, will be drawn the final conclusions concerning what was obtained till now, from the different experiments and analyses that have been done.

6.1 Models comparison

The first comparison will be between the GRU and the X3D model. Actually, on all the different experiments done, they have always been examined independently, and the reason for this choice is mainly related to the different chunk's lengths used by the models. It leads to different numbers of samples and a different distribution on the test set, making the obtained results not directly comparable to each other. Now, to compare them directly and in the fairest way, the models will be re-trained with the configurations of the chunks swapped to each other.
Thus, the GRU will use chunks of 4 frames with a resolution of 182×182, while the X3D will use chunks of length 8 and a resolution of 224×224.

Table 6.1, shows the new baseline results that must be taken into account during this test. For completeness, are reported also the previous results to have a more direct comparison between the different baseline accuracies. Can be noted that, when trained under the same input settings, the X3D model performs overall better than the GRU, especially with a smaller chunk size.

Model	Chunk size	Frame size	Accuracy
GRU	8 frames	224×224	66.32%
X3D	8 frames	224×224	66.40%
GRU	4 frames	182×182	61.54%
X3D	4 frames	182×182	66.64%

Table 6.1: Baseline accuracy

Model	Chunk size	Frame size	Accuracy
GRU	8 frames	224×224	82.33%
X3D	8 frames	224×224	70.54%
GRU	4 frames	182×182	76.92%
X3D	4 frames	182×182	75.84%

Table 6.2: MoCo v2 accuracy

In the table 6.2 instead are shown the results obtained by pre-training the backbones with MoCo. During this self-supervised training, the output size of its projection head will be equal to the optimal one found in the previous chapter, that is, 2048 for the GRU backbone and 1024 for the X3D.
Can be seen that all the baseline models have been outperformed. In particular is interesting to note that, the situation has been reversed, with the GRU models that performs always better than the X3D. This could indicate that the contrastive framework is still too focused on the spatial features and not enough on the temporal features representation.

The GRU model obtains its spatial-temporal representation through a two-step process, first, it extracts the spatial features from each frame and then, combines them to produce the final representation. In this architecture, the component that is pre-trained with the contrastive method is only the convolutional network which is responsible for the production of the spatial features. In this case, the pre-training helps only to improve the quality of the spatial representation, the improvement in the final spatial-temporal representations is a consequence of the availability of better spatial information from which to start to obtain the final one. The X3D model instead, produce its spatial-temporal representation directly on a single step, thanks to the operation of 3D convolution, so the whole network will be pre-trained with the contrastive method. In this case, the pre-training must help to learn both good spatial and temporal representations.

The lower gain obtained by the X3D model, with respect to the corresponding GRU, could be a signal that the temporal features still lacking in quality.

Finally, the last comparison will be between the baseline models, their corresponding version pre-trained with MoCo and the baseline models trained in a fully supervised way. So, in this last case also the backbone, which till now was always maintained frozen during the supervised train, will be trained together with the GRU and the linear classifier.

Model	Chunk size	Frame size	Accuracy
GRU	8 frames	224×224	78.18%
X3D	8 frames	224×224	70.86%
GRU	4 frames	182×182	77.22%
X3D	4 frames	182×182	71.15%

Table 6.3: Baseline accuracy: fully supervised training

The table 6.3 report the accuracies obtained from a full supervised training of both models with different input size of the chunks. For these training, the batch size has necessarily been reduced to load the whole architecture, and not only the classifier, on the GPU memories. Moreover, for the X3D have been reduced also the learning rate to stabilize the training and make the model converge more easily.

Can be observed that the accuracies reached by these latter are almost always better than the baseline, proving that in this way the convolutional networks are learning a set of features, more adequate to the new data distribution, which is different from ImageNet or Kinetics. It's interesting to note that also in this case, differently from the baseline, the GRU models perform better than the X3D.
Regarding instead the models pre-trained with MoCo, their performance are able to match or even outperform the full supervised training models, and without the necessity to make the detection task more difficult, like done on the SimCLR and BYOL papers, where the size of the ImageNet training set was reduced to make the task harder for the supervised training.

This confirms that the self-supervised pre-train with the contrastive methods can effectively help to produce representations better than the one achievable with the standard supervised training.

6.2 Conclusion

In this work have been tried to prove that the most recent self-supervised methods, in particular, the contrastive learning methods, have become a very powerful technique that can be used to reach different possible results, like:

- Produce representations with some specific property, like the invariance to a given transformation.
- Avoid overfitting when training on small datasets, similar to the one used in this works, which contains less than 1000 short clips.
- Obtain more general representations that could be used on a wide range of downstream tasks.

More in detail, have been shown how is possible to produce representations invariant to the point of view of the camera that records the video, and how this characteristic could help on the downstream online action detection task.

In principle, this proposed variation can be applied to any possible contrastive methods, since it is quite uncorrelated to the frameworks themselves. It requires *"only"* the availability of datasets on which the same object, environment or action, have been recorded from different points of view. A dataset with this property can be obtained in a relatively easy way, through the use of multiple cameras at the same time, positioned with a given distance from each other and with the lens facing toward the same spot. In general, producing this type of dataset can be considered easier and less expensive than obtaining a standard labelled dataset.

The promising results obtained with this approach could encourage to implement it on an increasing number of possible tasks or applications, like for example:

- *Autonomous driving*: The cars designed to implement autonomous driving are equipped with a high number of sensors and cameras that helps the vehicle to understand its relative position with the other entities in the surrounding environment. The data collected by these cars usually satisfied the just mentioned requirement to apply this hard positive mining method, since those cameras are positioned in different places of the car but look the same road from different points of view.
- *Automated surveillance*: A surveillance system is composed of several sensors and video cameras distributed over the area that must be monitored, that are able to recognize what is currently happening. Usually, those cameras are positioned in a way that is possible to see the same location from different

angles. This allows increasing the security level because makes it more difficult to do something sneaky without being seen at all. By using the data recorded from those cameras is possible to apply the proposed method to increase the ability of the cameras to recognize what is happening in a given location.

In general, further research on self-supervised contrastive learning, could help to reduce the actual strong correlation between the performances of the deep learning models and the availability of a large amount of labelled data.

Bibliography

[1] Wikipedia contributors. *Machine learning — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=1046345973. [Online; accessed 25-September-2021]. 2021 (cit. on p. 3).

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep Residual Learning for Image Recognition». In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (cit. on p. 8).

[3] Wikipedia contributors. *Euclidean distance — Wikipedia, The Free Encyclopedia*. [Online; accessed 1-October-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Euclidean_distance&oldid=1038865400 (cit. on p. 22).

[4] Wikipedia contributors. *Siamese neural network — Wikipedia, The Free Encyclopedia*. [Online; accessed 4-October-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Siamese_neural_network&oldid=1020522415 (cit. on p. 33).

[5] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. *Momentum Contrast for Unsupervised Visual Representation Learning*. 2020. arXiv: 1911.05722 [cs.CV] (cit. on pp. 25, 27).

[6] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. *Improved Baselines with Momentum Contrastive Learning*. 2020. arXiv: 2003.04297 [cs.CV].

[7] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. *A Simple Framework for Contrastive Learning of Visual Representations*. 2020. arXiv: 2002.05709 [cs.LG] (cit. on pp. 29, 30).

[8] Kihyuk Sohn. «Improved Deep Metric Learning with Multi-class N-pair Loss Objective». In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper/2016/file/6b180037abbebea991d8b1232f8a8ca9-Paper.pdf (cit. on p. 29).

[9] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton. *Big Self-Supervised Models are Strong Semi-Supervised Learners*. 2020. arXiv: 2006.10029 [cs.LG] (cit. on p. 31).

[10] Jean-Bastien Grill et al. «Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning». In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 21271–21284. URL: https://proceedings.neurips.cc/paper/2020/file/f3ada80d5c4ee70142b17b8192b2958e-Paper.pdf (cit. on pp. 31, 33).

[11] Xinlei Chen and Kaiming He. «Exploring Simple Siamese Representation Learning». In: *CoRR* abs/2011.10566 (2020). arXiv: 2011.10566. URL: https://arxiv.org/abs/2011.10566 (cit. on p. 35).

[12] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. *Unsupervised Learning of Visual Features by Contrasting Cluster Assignments*. 2021. arXiv: 2006.09882 [cs.CV].

[13] Mehdi Noroozi and Paolo Favaro. *Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles*. 2017. arXiv: 1603.09246 [cs.CV] (cit. on p. 18).

[14] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. «Unsupervised Representation Learning by Predicting Image Rotations». In: *CoRR* abs/1803.07728 (2018). arXiv: 1803.07728. URL: http://arxiv.org/abs/1803.07728 (cit. on p. 16).

[15] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. «Unsupervised Visual Representation Learning by Context Prediction». In: *CoRR* abs/1505.05192 (2015). arXiv: 1505.05192. URL: http://arxiv.org/abs/1505.05192 (cit. on p. 16).

[16] Wikipedia contributors. *Chromatic aberration — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Chromatic_aberration&oldid=1028300634. [Online; accessed 29-September-2021]. 2021 (cit. on p. 17).

[17] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. «Extracting and Composing Robust Features with Denoising Autoencoders». In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: Association for Computing Machinery, 2008, pp. 1096–1103. ISBN: 9781605582054. DOI: 10.1145/1390156.1390294. URL: https://doi.org/10.1145/1390156.1390294 (cit. on p. 20).

[18] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. «Context Encoders: Feature Learning by Inpainting». In: *CoRR* abs/1604.07379 (2016). arXiv: 1604.07379. URL: http://arxiv.org/abs/1604.07379 (cit. on p. 20).

[19] Richard Zhang, Phillip Isola, and Alexei A. Efros. «Split-Brain Autoencoders: Unsupervised Learning by Cross-Channel Prediction». In: *CoRR* abs/1611.09842 (2016). arXiv: 1611.09842. URL: http://arxiv.org/abs/1611.09842 (cit. on p. 21).

[20] Sajjad Abbasi, Mohsen Hajabdollahi, Nader Karimi, and Shadrokh Samavi. *Modeling Teacher-Student Techniques in Deep Neural Networks for Knowledge Distillation*. 2019. arXiv: 1912.13179 [cs.CV] (cit. on p. 31).

[21] Florian Schroff, Dmitry Kalenichenko, and James Philbin. «FaceNet: A unified embedding for face recognition and clustering». In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015). DOI: 10.1109/cvpr.2015.7298682. URL: http://dx.doi.org/10.1109/CVPR.2015.7298682.

[22] Senthil Purushwalkam and Abhinav Gupta. «Demystifying Contrastive Self-Supervised Learning: Invariances, Augmentations and Dataset Biases». In: *CoRR* abs/2007.13916 (2020). arXiv: 2007.13916. URL: https://arxiv.org/abs/2007.13916 (cit. on p. 13).

[23] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. «Imagenet: A large-scale hierarchical image database». In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255 (cit. on p. 42).

[24] Tsung-Yi Lin et al. «Microsoft COCO: Common Objects in Context». In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312 (cit. on p. 44).

[25] Tian Pan, Yibing Song, Tianyu Yang, Wenhao Jiang, and Wei Liu. «VideoMoCo: Contrastive Video Representation Learning with Temporally Adversarial Examples». In: *CoRR* abs/2103.05905 (2021). arXiv: 2103.05905. URL: https://arxiv.org/abs/2103.05905 (cit. on pp. 38, 39).

[26] Rui Qian, Tianjian Meng, Boqing Gong, Ming-Hsuan Yang, Huisheng Wang, Serge J. Belongie, and Yin Cui. «Spatiotemporal Contrastive Video Representation Learning». In: *CoRR* abs/2008.03800 (2020). arXiv: 2008.03800. URL: https://arxiv.org/abs/2008.03800 (cit. on p. 41).

[27] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. «SlowFast Networks for Video Recognition». In: *CoRR* abs/1812.03982 (2018). arXiv: 1812.03982. URL: http://arxiv.org/abs/1812.03982 (cit. on p. 41).

[28] Christoph Feichtenhofer. «X3D: Expanding Architectures for Efficient Video Recognition». In: *CoRR* abs/2004.04730 (2020). arXiv: 2004.04730. URL: https://arxiv.org/abs/2004.04730 (cit. on p. 57).

[29] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. «MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications». In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861 (cit. on p. 57).

[30] Pierre Sermanet, Corey Lynch, Jasmine Hsu, and Sergey Levine. «Time-Contrastive Networks: Self-Supervised Learning from Multi-View Observation». In: *CoRR* abs/1704.06888 (2017). arXiv: 1704.06888. URL: http://arxiv.org/abs/1704.06888 (cit. on p. 76).

[31] Wikipedia contributors. *Turing test — Wikipedia, The Free Encyclopedia*. [Online; accessed 1-November-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Turing_test&oldid=1050856464.

[32] Philip Bachman, R. Devon Hjelm, and William Buchwalter. «Learning Representations by Maximizing Mutual Information Across Views». In: *CoRR* abs/1906.00910 (2019). arXiv: 1906.00910. URL: http://arxiv.org/abs/1906.00910.

[33] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. «A Closer Look at Spatiotemporal Convolutions for Action Recognition». In: *CoRR* abs/1711.11248 (2017). arXiv: 1711.11248. URL: http://arxiv.org/abs/1711.11248.

[34] Elijah Cole, Xuan Yang, Kimberly Wilber, Oisin Mac Aodha, and Serge J. Belongie. «When Does Contrastive Visual Representation Learning Work?» In: *CoRR* abs/2105.05837 (2021). arXiv: 2105.05837. URL: https://arxiv.org/abs/2105.05837.

[35] Longlong Jing and Yingli Tian. «Self-supervised Visual Feature Learning with Deep Neural Networks: A Survey». In: *CoRR* abs/1902.06162 (2019). arXiv: 1902.06162. URL: http://arxiv.org/abs/1902.06162.

[36] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. *Learning deep representations by mutual information estimation and maximization*. 2019. arXiv: 1808.06670 [stat.ML].

[37] Mike Wu, Chengxu Zhuang, Milan Mosse, Daniel Yamins, and Noah D. Goodman. «On Mutual Information in Contrastive Learning for Visual Representations». In: *CoRR* abs/2005.13149 (2020). arXiv: 2005.13149. URL: https://arxiv.org/abs/2005.13149.