

Comparison between Self-Supervised Learning and Classical Supervised Learning in the classification task

David Caprari, Francesca Zitoli

Abstract—Since the last two years (2019-2020), deep learning is shifting its attention to the possibility of using large amounts of data for training neural networks without labels and information provided by human agent. In self-learning we train the network with unlabeled data and then we make a fine tuning with a lesser amount of labeled data. In this work we want to compare the results of the usual classification and self-supervised learning. We used four different types of self-supervised methods: MoCo, SimSiam, SimCLR and BYOL.

Finally, we focus, for each method, on which details the classifier uses to carry out the classification using Captum, a Python suite for network interpretability, and we compare the results with a more standard classification network, a ResNet-50.

I. INTRODUCTION

One of the biggest problems in image classification is that datasets used to train neural networks for deep learning approaches usually need to be labeled by human and this usually requires a lot of time and resources.

For these reasons, self-supervised deep-learning approaches are being developed in the field of deep learning. This type of structures uses large quantities of non-labeled images to train themselves. They are, based on their specific implementation, capable of a sort of self-clustering and self-labeling of the output and, for these reasons, they usually perform well in spurious datasets. Usually, they can be completed with a fully-connected layer, then classification is carried out with smaller and labeled datasets as a classification-layer training set. The purpose of this project is to compare the standard classification results of a standard supervised classifier with these state-of-the-art technologies, try to interpret part of their learning techniques and convolutional layers and provide a report regarding the state of their public usability and implementation.

Regarding self-supervised structures, we used four methods: Moco, SimSiam, SimCLR and BYOL. Initially we were tasked to code them using Lightly.ai library, but due to some problems, lesser documented and lesser upgraded coding, we switched to the use of the Pytorch-Lightning's Lightning-Bolts library. We then trained a classical Torchvision defined ResNet-50 for the aforementioned comparison.

Finally, we tried to understand in which areas of the image the network focuses on the upcoming classification using Captum.ai, a well-known library for network interpretability.

Then we compared the results obtained with self-supervised learning and results obtained with the supervised classification method reported before via an analysis of the pure classification results and via a comparison of the outputs generated by the interpretability section.

In the next paragraph we introduce self-supervised learning and the technologies behind the MoCo, SimCLR, SimSiam and BYOL methods. Paragraph 3 explains the experiments done and paragraph 4 compares the results.

II. SELF-SUPERVISED APPROACHES AND LEARNING

In fig.1 is reported the general pipeline of self-supervised learning. In general, a visual feature is learned through the process of training convolutional neural networks to solve a pre-defined pretext task. After self-supervised pretext task training is concluded, the learned parameters serve as a pre-trained model and are transferred to other downstream computer vision tasks by finetuning. The performance on these downstream tasks is used to evaluate the quality of the learned features. During the knowledge transfer for downstream tasks, the general features from only the first several layers are unusually transferred to downstream tasks.

The basic idea of Self-Supervised Learning (SSL)[1] is to automatically generate some kind of supervisory signal to solve some task (typically, to learn representations of the data or to automatically label a dataset). So, in general, we use the term SSL to represent every non human-labeling approach to deep learning and in this work we use several of these techniques.

A. Momentum Contrast - MoCo

Momentum Contrast (MoCo)[2] trains a visual representation encoder by matching an encoded query q to a dictionary of encoded keys using a contrastive loss. The dictionary keys are defined on-the-fly by a set of data samples. The dictionary is built as a queue, with the current mini-batch enqueued and the oldest mini-batch dequeued, decoupling it from the mini-batch size. The keys are encoded by a slowly progressing encoder, driven by a momentum update with the query encoder. This method enables a large and consistent dictionary for learning visual representations. In this work, encoder and momentum encoder are defined as ResNet-50 architectures stripped of their fully connected tail.

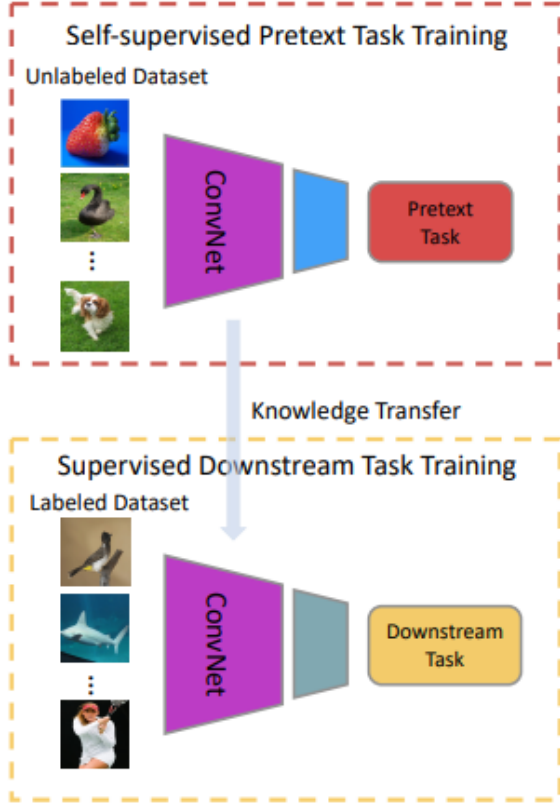


Fig. 1: A general self-supervised approach: first a ConvNet is trained on unlabeled data, then via transfer-learning, the status of the ConvNet is passed to a second ConvNet for a downstream task and is fine-tuned on labeled data.

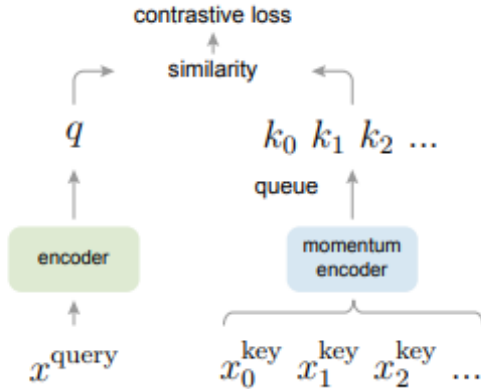


Fig. 2: A brief scheme of Momentum Contrast - MoCo.

B. A Simple Framework for Contrastive Learning of Visual Representation - SimCLR

SimCLR[3] stands for a *Simple framework for Contrastive Learning of visual Representations*. For this architecture, two separate data augmentation operators are sampled from the same family of augmentations and applied to each data example to obtain two correlated views. A base encoder network and a projection head are trained to maximize agreement using a contrastive loss. After training is completed, we throw away

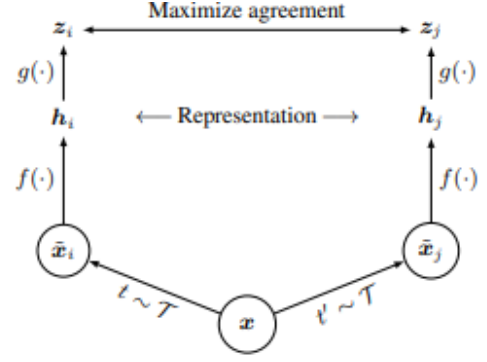


Fig. 3: SimCLR: we define $f(\cdot)$ as the encoder, $g(\cdot)$ as a projection head and h as their representations. t and t' are two augmentations.

the projection head and use encoder and its representation for downstream tasks. We use a ResNet-50 for the codification of the encoder. The source code for the projection head and its implementation are provided by the used libraries.

C. Simple Siamese - SimSiam

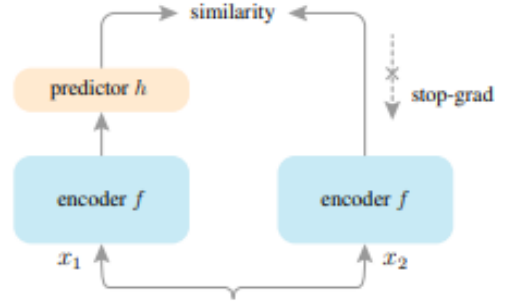


Fig. 4: Simple Siamese - SimSiam: two encoders and a predictor.

In SimSiam[4] two augmented views of one image are processed by the same encoder network (a backbone plus a projection MLP). In this work, the backbone of the encoder is coded from the architecture of a ResNet50, just for the previous networks, stripped of its fully connected layer. Then a prediction MLP is applied on one side, and a stop-gradient operation is applied on the other side. The model maximizes the similarity between both sides. It uses neither negative pairs nor a momentum encoder.

D. Bootstrap Your Own Latent - BYOL

BYOL[5] uses two neural networks, referred to as online and target networks, that interact and learn from each other. In this work they are both generated from a ResNet-50 architecture and then correctly placed with the use of the methods provided by Lightning-Bolts library. Starting from an augmented view of an image, BYOL trains its online network to predict the target network's representation of another augmented view of the same image. While this objective admits

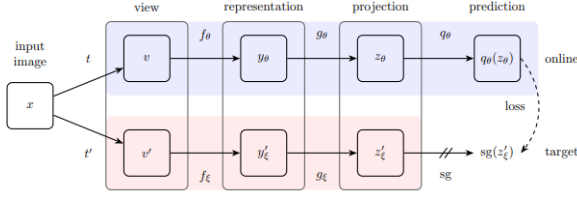


Fig. 5: Bootstrap Your Own Latent - BYOL: two structures with representation and projection interact on base of a prediction layer to the tail of the online network.

collapsed solutions, e.g., outputting the same vector for all images, its renowned that BYOL does not converge to such states.

III. MATERIALS AND METHODS

The implementation of the solutions is realized using Python3.9 in addition to a variety of its libraries generally employed for deep learning tasks. More clearly, we based our work on Pytorch and its subpackages like Torchvision, Pytorch-Lightning, Lightning-Bolts and Captum, obviously correlated by other general tools like SciKitLearn, Matplotlib, OpenCV for Python, Numpy and others. At first, we were tasked the use of Lightly.ai, a library for self-supervised approaches, learning techniques and datasets based on Pytorch-Lightning. Due to subsequent implementation bugs and other difficulties like several incompatibilities with other modules or errors in networks' source code, we successfully tried basing our scripts on Lightning-Bolts, an official subpacket of Pytorch-Lightning. The tools and kits provided by the latter, at the time of this work's realization, seemed more powerful and more faithfully realized on their relative article publications. So, we decided to start again from the beginning and rewrite our code with these solutions. It's not excluded that in future this might change, being the easier-to-use dataloader's implementation of Lightly a probable game-changer.

We thus implemented MoCo, SimSiam, SimCLR and BYOL networks and tested their training on a small debugging dataset. We managed the training of all the aforementioned networks on one of our personal machines.

Then, based on the work provided by the Information Engineering Department (DII) of our Alma Mater, we stated a dataset of different kinds of fashion elements. The first section of the dataset is an unlabeled set of 32,642 fashion themed images of different shapes; these images obviously concern themes with similarities to the counterpart's classes. The second section consists of a smaller dataset of labeled images. The dataset consists of a total of 6,466 images divided into 9 classes:

- Bauletto: 445
- Clutch: 1.285
- Hobo: 360
- Marsupio: 1.097
- Sacca: 325
- Secchiello: 351
- Shopping: 108
- Tracolla: 1.812

- Zaino: 636

With the defined dataset, we implemented a complete training algorithm based on the code supported by Pytorch and Pytorch-Lightning: we pass the architectures 32-sized batches of specifically custom-augmented images loaded from the unlabeled section of the dataset in form of torch-tensors of shape [3, 256, 256] (Pytorch's image sizes convention). We defined the last convolutional-encoder layer of the 4 different solutions to be of 2048 features, itself based on a pretrained ResNet-50 architecture. First training of fine-tuning completed, this encoder is extracted and we tailed it with a fully connected layer ending in 9 neurons, which is the exact number of labels we have in the dataset. In a second lesser time-consuming training, we then trained our newly born classifiers on the remaining part of the dataset consisting of humanly labeled data, without data augmentations.

Obtained our fine-tuned models we test them on the labeled section of the dataset, generating proper confusion matrices and classification reports. We then provided an interpretability dedicated section of our code based on the tools provided by Captum.ai for Python library.

Specifically, we used the Saliency: a simple approach for computing input attribution, returning the gradient of the output with respect to the input. The absolute value of these generated coefficients can be taken to represent feature importance; we also implemented the use of Integrated Gradients: an algorithm that assigns an importance score to each input feature. Both these tools focus their relevance on the analysis of the forward pass of the various networks and give outputs based on the gradient variations provoked by a noisy or defined input on subsequent layers.

The output images resulting from the previous passes on the different networks are then given to a new script designed for image comparison. The script utilizes a couple of well-known image similarities comparison techniques: Root Mean Square Error (RMSE) and Structural Similarity Index (SSIM).

In the end we implemented a classifier on a ResNet-50, passing it only the labeled dataset and then compared the results with those obtained with self-supervised learning.

IV. EXPERIMENTS AND RESULTS

The experiments are propagated by the execution of our defined software.

Regarding the MoCo architecture, unfortunately, passing to the structure the whole dataset, and not just its small debug realization, an error occurs. It can depend on the queuing and unqueuing of some batches due to Momentum Contrast specific implementation. It cannot be determined what is the actual problem in the dataset loading procedure, and the whole error is due to a proper incompatibility of the source code with custom datasets. At the present state, the bug is publicly renowned and it is being taken care of by the software maintainers.

Training BYOL did not give such problems, but we experienced the zeroing of the loss after the first third of the process. This gave us some problems in the extraction of the interpretability data provided by Captum; the extraction based

on the analysis of the learning gradient is zeroed by the loss zeroing. We cannot define if the described anomaly gave rise to further errors and inaccuracies in the results obtained. At the present state, this bug is not publicly renowned.

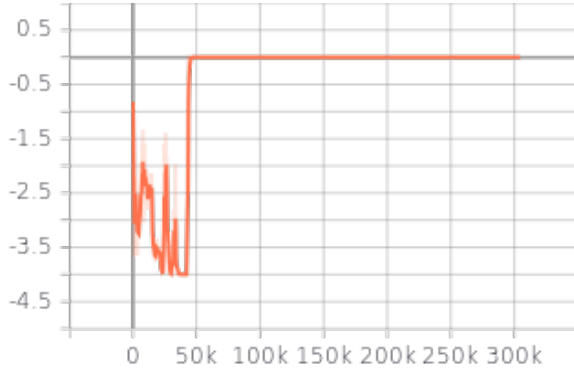


Fig. 6: Tensorboard log image extraction of BYOL's train-loss parameter showing the zeroing of the loss at near 50k training steps.

The 3 methods BYOL, SimCLR, SimSiam, are trained for 1500 epochs on the aforementioned dataset. Then, the classifiers built upon these networks are trained for 150 epochs (reduction in the epochs due to the lesser number of layers and trainable parameters). The results obtained are the following:

From the BYOL's confusion matrix on Fig.7 we can observe that BYOL classifies everything as *Tracolla*. This slightly poor performance can be due to aforementioned zeroing of the loss.

From SimCLR's confusion matrix on Fig.8 we can observe that SimCLR makes a more accurate classification than BYOL. We can see that SimCLR has classified with good performance the *Tracolla* class. We get good results also for *Clutch* and *Marsupio*.

Even with SimSiam's confusion matrix on Fig.10 we can observe that this net has classified ratherly good the *Tracolla* class and we have good results for 'Clutch' and *Marsupio*.¹

¹For more detailed reports, we refer to the ones in the project's repository (link provided in the first page).

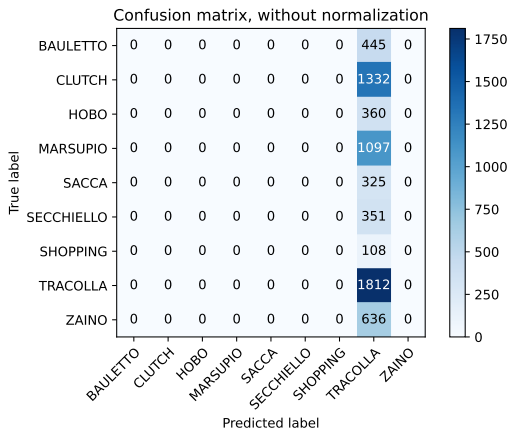


Fig. 7: BYOL Confusion Matrix.

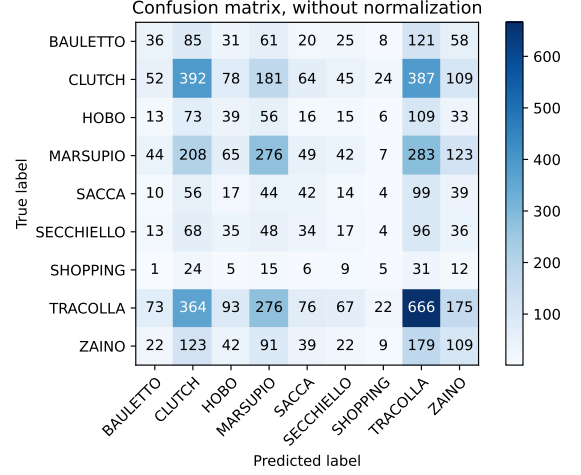


Fig. 8: SimCLR Confusion Matrix.

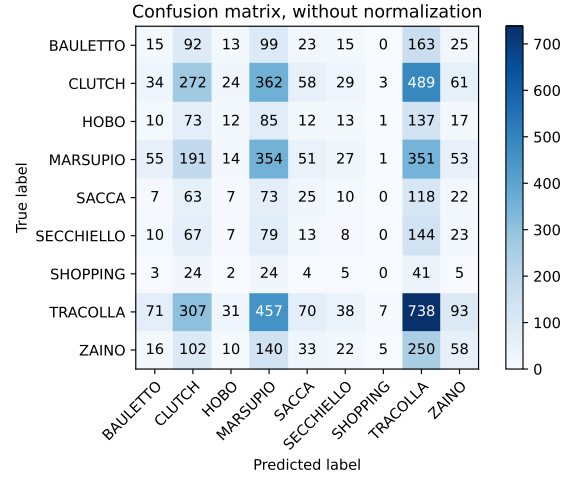


Fig. 9: SimSiam Confusion Matrix.

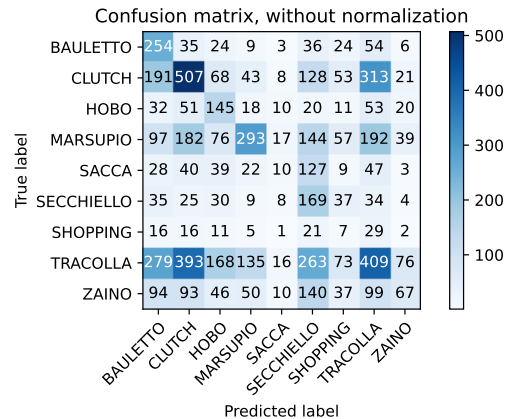


Fig. 10: Supervised approach Confusion Matrix.

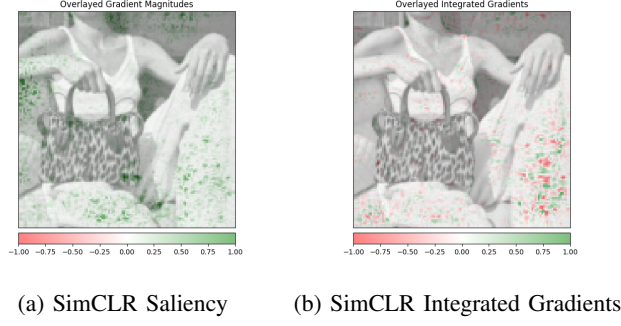


Fig. 11: SimCLR Saliency and Integrated Gradients

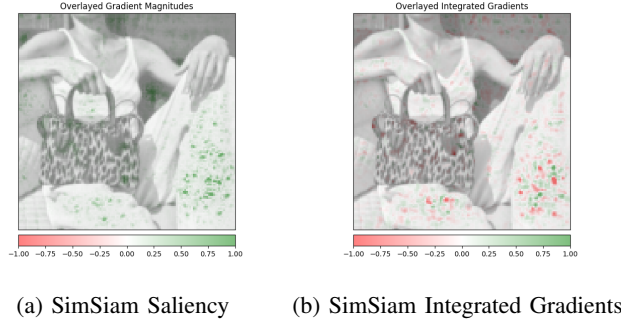


Fig. 12: SimSiam Saliency and Integrated Gradients

Then we extract the Saliency and Integrated Gradients using Captum. Due to the fact that during the training step of the self-supervised model, BYOL experienced a zeroing of its training loss and its gradient, we did not obtain relevant data out of the interpretability analysis. Regarding the data we obtained from SimCLR and SimSiam, we hoped in more comprehensible results: it is quite hard to define used patterns for classification. This sort of complexity is probably due to the not uttermost great precision and accuracy of the networks and their state of gradient descent at the end of the training. Gladly, we experienced the same results even with the supervised training of the ResNet. We concluded that this phenomenon may depend almost entirely on the noise of the dataset and its relative small dimension, given the fact that the supervised learning gains almost the same results obtained with the self-supervised learning; these results indicated a poor performance for the well-known and well described ResNet-50, that usually performs better on given datasets like Cifar10 or ImageNET.

From the images Fig.11 and Fig.12 we can see that the architectures do not try to identify the bag basing themselves on bag's features. The structures focus their learning on the images' background. This may be due to the fact that at this point the architectures passed through different trainings and their descent of gradient is irrelevant. We then compared the presented images with the Root Mean Squared error and Structural Similarity Index. Both the RMSE value (0.00247) and SSIM value (0.99633) given from the comparison indicated a strong similarity of the two outputs, not providing further informations.

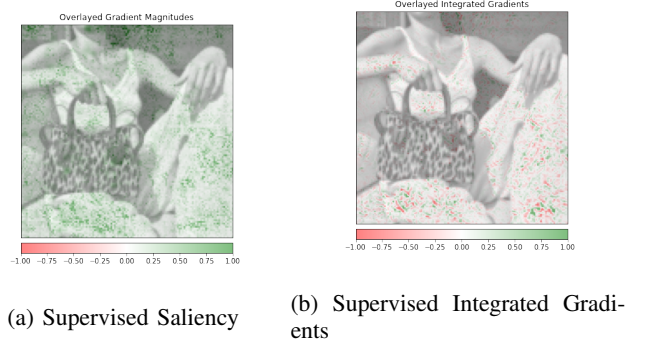


Fig. 13: Supervised Saliency and Integrated Gradients

V. CONCLUSION

In a short recap, in this work we made a comparison between the results obtained with self-supervised approaches and the classic supervised classification. We started by implementing 4 methods: MoCo, SimCLR, SimSiam and BYOL with Lighly. Following some problems with Lighly we decided to do it all again with Lightning-Bolts. During the training of the 4 methods we had some problems with MoCo, passing the method the whole dataset, and not just its small debug realization, an error occurs. Then we implemented a classifier for each method and verified the results with the confusion matrices. We have noticed that all methods perform quite well with the exception of BYOL: This may be due to aforementioned zeroing of the loss. Then we provided an interpretability dedicated section of our code based on the tools provided by Captum, we used Saliency and Integrated Gradients for the valuation. In the end we implemented a traditional supervised classifier to compare results using Saliency and Integrated Gradient. Gladly, we experienced the same results even with the supervised training of the ResNet.

We concluded that the classification performance provided by the selfsupervised methods tested in this work it's not utterly poor in terms of accuracy and precision and may actually be comparable to the ones obtained by supervised methods. What we did not foresaw was all the struggle with the implementation of the source code, starting with the bugs raised by Lighly.ai to the limitations of Lightning-Bolts. Struggle obviously caused by the state of innovation of these solutions. We hope that in the future such struggles might be overtaken.

REFERENCES

- [1] L. J. . Y. Tian, "Selfsupervised visual feature learning with deep neural networks: A survey."
- [2] Y. W. S. X. Kaiming He, Haoqi Fan and R. Girshick, "Momentum contrast for unsupervised visual representation learning."
- [3] G. H. Simon Kornblith, Mohammad Norouzi, "A simple framework for contrastive learning of visual representations."
- [4] K. H. Xinlei Chen, "Exploring simple siamese representation learning."
- [5] F. A. C. T. P. R. E. B. C. D. B. P. Z. G. M. A. B. P. K. K. R. M. M. V. Jean-Bastien Frill, Florian Strub, "Bootstrap your own latent: A new approach to self-supervised learning."