EXPERIENCE
FRUIT QUALITY

Master Thesis

Self-Supervised Learning in Avocado Quality Prediction

Visual feature extraction through pretraining

Sjors Peerdeman

June 15, 2022

MSc Business Analytics

VU VRIJE UNIVERSITEIT AMSTERDAM

Master Thesis

Self-Supervised Learning in Avocado Quality Prediction

Visual feature extraction through pretraining

Sjors Peerdeman

Student number: 2628398

Thesis committee:

First supervisor	Jakub Tomczak
Company supervisor	Martijn van Wezel
Second reader	Sandjai Bhulai

VU VRIJE UNIVERSITEIT AMSTERDAM

Faculty of Science
Business Analytics
De Boelelaan 1081a
1081 HV Amsterdam

Host organization:

EXPERIENCE FRUIT QUALITY

Europalaan 400 - 2 Noord
3526 KS Utrecht

June 15, 2022

Preface

This thesis is written in fulfillment of the requirements for the Master of Science degree in Business Analytics at the Vrije Universiteit Amsterdam (VU). It focuses on applying self-supervised machine learning to the context of avocado quality prediction. The research for this thesis has been conducted during a six-month period at Experience Fruit Quality (EFQ). EFQ is a company that focuses on predicting fruit quality with its digital assistants, using a physical machine in combination with a machine learning model.

I would like to thank Martijn van Wezel, my supervisor at EFQ, for his enthusiasm, helpful comments, support and guidance. Our weekly meetings helped me to stay on track, and Martijn's detailed comments resulted in a thesis that is more streamlined than it would have been without him. Additionally, a big thanks goes out to Jakub Tomczak, my external supervisor at the VU, for his invaluable advice and guidance. Furthermore, I'd like to express my gratitude to Sandjai Bhulai for being the second reader. Last but not least, the EFQ management has always been very trusting towards me and my goals, both as a working student and as a graduation intern. Without their support, this thesis would not have been possible.

Executive summary

Problem statement - REDACTED

Methodology - REDACTED

Results and recommendations - REDACTED

Table of Contents

1 Introduction **1**
1.1 Company background . . . 1
1.1.1 Avocado supply chain . . . 1
1.1.2 EFQ Machines . . . 1
1.2 Problem statement . . . 2
1.3 Motivation . . . 2
1.4 Outline . . . 3

2 Background **5**
2.1 Supervised Learning in computer vision . . . 5
2.2 Self-Supervised Learning in computer vision . . . 6
2.2.1 A note on semi-supervised learning . . . 6
2.3 SSL methods . . . 7
2.3.1 SimCLR . . . 8
2.3.2 MoCo . . . 9
2.3.3 BYOL . . . 10
2.3.4 SimSiam . . . 10
2.3.5 Barlow Twins . . . 11
2.4 Datasets used in SSL . . . 12
2.5 SSL image augmentation . . . 12
2.6 Matthew's Correlation Coefficient . . . 13
2.7 Grad-CAM . . . 14
2.8 t-SNE . . . 14

3 Methods **17**
3.1 Limitations and restrictions . . . 17
3.2 Baseline model . . . 17
3.2.1 Architecture . . . 17
3.2.2 Image augmentation . . . 18
3.2.3 Hyperparameters . . . 18
3.3 SSL model . . . 19
3.3.1 Method and architecture . . . 19
3.3.2 Image augmentation . . . 21
3.3.3 Hyperparameters . . . 21
3.4 Research procedure . . . 22
3.4.1 Evaluation metrics . . . 22
3.4.2 Baseline Experiments . . . 22
3.4.3 SSL Experiments . . . 22
3.4.4 Model analyses . . . 24

4 Data analysis **25**
4.1 Data gathering . . . 25
4.2 Data preparation . . . 25
4.3 Defects and classes . . . 26
4.3.1 Internal defects . . . 26
4.3.2 External defects . . . 27

4.3.3 Defects in dataset 27
4.4 Dataset construction 28
4.4.1 Supervised Learning datasets 28
4.4.2 SSL pretraining datasets 29
4.4.3 Fine-tuning datasets 30

5 Results **33**
5.1 Baseline model 33
5.1.1 Training set 33
5.1.2 Hyperparameter tuning 33
5.1.3 Results 34
5.2 Self-Supervised Learning 35
5.2.1 Pretraining 35
5.2.2 Fine-tuning 36
5.2.3 Hyperparameter tuning 38
5.2.4 Results 42
5.3 Analysis 44
5.3.1 Defects 44
5.3.2 Input ablation 45
5.3.3 Grad-CAM 46
5.3.4 t-SNE 49

6 Conclusions **51**
6.1 Discussion and recommendations 51

References **53**

Appendix A: Underlying data per fold for cumulative confusion matrices **57**

Appendix B: Results of extra test with augmentations for SSL pretraining **59**

1 Introduction

In this chapter, the problem that this work aims to solve is stated. The background of the problem within the company is also explained, and the structure of the rest of this work is laid out.

1.1 Company background

Experience Fruit Quality (EFQ) is a company that focuses on predicting fruit quality (currently mainly avocados and mangos) with its assistant AVOS, using a physical machine in combination with a machine learning model. The unique selling point of AVOS is the ability to perform fruit quality assessment on the basis of pictures (Red-Green-Blue (RGB) and Near Infrared (NIR)) and pressure measurements quickly, without producing waste by having to cut open any fruit (the way quality assessment is normally done). This work focuses on avocado quality prediction, as models for the quality of other fruits are still in development.

1.1.1 Avocado supply chain

A condensed version of the avocado supply-chain process from farming to consumer is as follows:

- Growth and harvest
- Shipping to ripening facility
- Cold storage to slow fruit changes
- Ripening (controlling the metabolic process of fruit ripening)
- Sorting and shipment to clients

The role of AVOS in this process is at the ripening facilities, where it can be used to assess the quality of avocados both before and after cold storage or ripening.

1.1.2 EFQ Machines

EFQ currently has two different machines in use: the AVOS Mini and the AVOS Maxi.

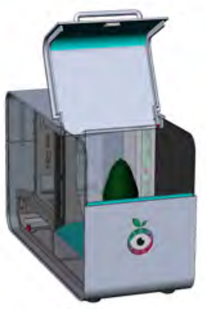


Figure 1: A technical drawing of the AVOS Mini machine.

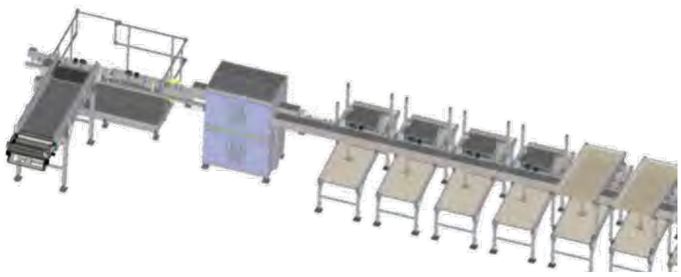


Figure 2: A technical drawing of the AVOS Maxi machine.

The AVOS Mini (see Figure 1) is a small portable machine that allows avocado quality inspectors to put an avocado in, after which the pressure is measured by tapping the avocado and the RGB and NIR photos are taken from four different sides. This allows the quality inspector to assess the quality of a sample of avocados from a pallet.

The AVOS Maxi (see Figure 2) is a large sorting machine that allows for the sorting of full pallets of avocados. The avocados are put on an assembly line, after which their RGB and NIR photos are taken and their pressures are measured by tapping them. The quality model then predicts whether the avocado is of good quality ('a-quality', also known as class I) or if it has one or more defects (internally, externally or both). Avocados with external defects might still be sellable, while avocados with internal defects are not. The avocados are then sorted into different exits based on their predicted quality. This works focuses on the AVOS Maxi exclusively.

Both machines also predict avocado color and ripening stage with seperate machine learning models. This work focuses on the quality model exclusively, as this is the main focus of EFQ.

1.2 Problem statement

The problem that this work aims to solve is stated as follows:

- How can pretraining with self-supervised learning methods, using unlabeled data, result in an improvement over the existing avocado quality prediction model?

To expand upon this problem we formulate the following research questions:

- Which self-supervised pretraining method is most promising for the EFQ context?
- What does the labeled dataset look like, and how does this influence the existing supervised model?
- What are the quality prediction results per type of avocado defect using the existing model and how can these be explained?
- Which data augmentations perform best in the chosen self-supervised pretraining method?
- What do a well performing pretraining and/or fine-tuning dataset look like?

1.3 Motivation

REDACTED

1.4 Outline

The rest of this work is structured as follows:

- **Chapter 2** discusses the background and literature on self-supervised learning and any other methods that we will touch upon in this work.
- **Chapter 3** focuses on choosing the methods that will be used in our research, keeping in mind any practical restrictions.
- **Chapter 4** analyses the particulars of the avocado data: how it was gathered and which defects exist. We also construct our datasets here.
- **Chapter 5** shows the results of our research for both the baseline model and the self-supervised learning methods.
- **Chapter 6** will discuss the conclusions and recommendations.

2 Background

This chapter first briefly discusses supervised learning in computer vision, after which we will discuss the existing literature on self-supervised learning and any other methods that we will touch upon in this work.

2.1 Supervised Learning in computer vision

The most common form of machine learning is supervised learning. Supervised learning in computer vision is generally done by collecting a large dataset of images, each labeled with the category that the model should learn to recognize (LeCun, Bengio, & Hinton, 2015). In supervised learning, given a dataset $\{X_i\}_{i=0}^{N}$ and corresponding labels $\{Y_i\}_{i=0}^{N}$, the general training loss function for a model with parameters θ is defined as in Formula 1 (Jing & Tian, 2020).

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} loss(X_i, Y_i). \tag{1}$$

In practice, the loss function is used to optimize the weights of the network using stochastic gradient descent (SGD) (LeCun et al., 2015).

One of the most common deep learning network structures is the Convolutional Neural Network (CNN). Since the early 2000s, CNNs have been applied with great success to the detection, segmentation and recognition of objects and regions in images (LeCun et al., 2015). The main feature of CNN architectures is the convolutional layer, which aims to learn feature representations of the inputs.

A schematic representation of a convolutional layer is shown in Figure 3. Such a layer is composed of several convolution kernels, each of which is used to compute a different feature map. Each neuron of a feature map is connected to a region of neighboring neurons in the previous layer (the input). The new feature map can be obtained by first convolving the input with a learned kernel and then applying an element-wise non-linear activation function (usually the ReLu function) on the results. Mathematically, the feature value at location (i, j) in the k-th feature map of the l-th layer (assuming a ReLu activation function) is calculated using Formula 2 (Gu et al., 2018).

$$a_{i,j,k}^{l} = ReLu\left(z_{i,j,k}^{l}\right) = ReLu\left((\mathbf{w}_k^l)^T \mathbf{x}_{i,j}^{l} + b_k^l\right). \tag{2}$$

Where large amounts of images with accurate human-annotated labels are available, supervised learning methods have achieved break-through results on different computer vision applications (Jing & Tian, 2020). However, obtaining large volumes of high-quality data often is quite costly, time-consuming, or both (Liu et al., 2021).

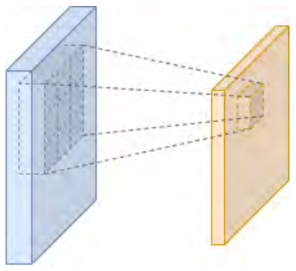


Figure 3: In a convolutional layer, each neuron of a feature map (orange) is calculated using the values of a region of neighboring neurons in the previous layer (blue).

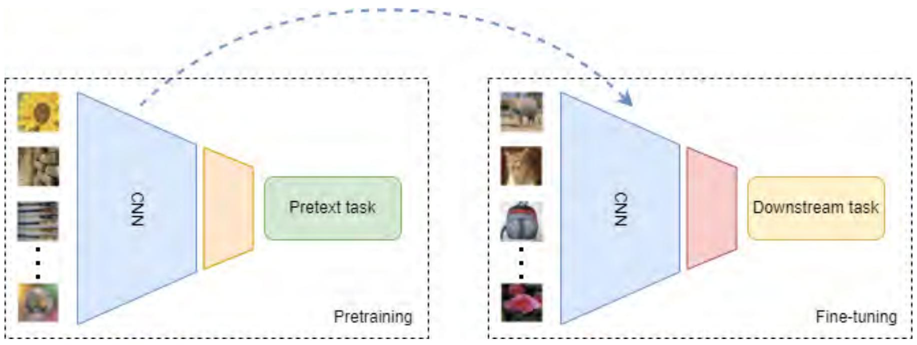


Figure 4: The general pipeline of self-supervised learning, adapted from Jing and Tian (2020). After training on unlabeled data using a self-supervised pretext task (pretraining), the learned weights of the CNN are transferred for training on labeled data in a supervised downstream task (fine-tuning).

2.2 Self-Supervised Learning in computer vision

Self-supervised learning (SSL) is a subset of unsupervised learning (Jing & Tian, 2020). SSL refers to learning methods in which neural networks are explicitly trained with automatically generated labels. The general pipeline of SSL in computer vision is schematically shown in Figure 4. This pipeline consists of first solving a pre-defined so-called pretext task on unlabeled data, where general visual features are learned by a convolutional neural network. After this, the learned model parameters (sometimes only those of the first several convolutional layers) are transferred to the actual task at hand, called the downstream task (Jing & Tian, 2020). Then, the downstream task is solved using 'regular' supervised learning. SSL, in this way, is a form of transfer learning. Solving the pretext and downstream task is also known by the terms pretraining and fine-tuning, respectively (Jaiswal, Babu, Zadeh, Banerjee, & Makedon, 2021).

In Formula 3, the general training loss function for a model with parameters θ is defined for SSL, given an unlabeled dataset $\{X_i\}_{i=0}^{M}$ and pseudo-labels $\{P_i\}_{i=0}^{M}$.

$$\min_{\theta} \frac{1}{M} \sum_{i=1}^{M} loss(X_i, P_i). \tag{3}$$

The pseudo-labels P_i are automatically generated for a predefined pretext task, without involving any human annotation (Jing & Tian, 2020). It is important to note that these pseudo-labels can take many different forms. For instance, in generative SSL approaches all or parts of the training data are part of the model's produced output, and the pseudo-labels are typically in the same feature space as the training data (Mao, 2020).

Models pretrained with SSL methods are able to learn with less labeled data (Newell & Deng, 2020), and are more stable and robust to adversarial examples and input or label corruptions (Hendrycks, Mazeika, Kadavath, & Song, 2019).

2.2.1 A note on semi-supervised learning

A term that is closely related to SSL is semi-supervised learning. Just like SSL, semi-supervised learning uses large amounts of unlabeled data during training, but also uses labeled data at the same time (Zhu, 2005). Given a small labeled dataset $\{X_i\}_{i=0}^{N}$, corresponding labels $\{Y_i\}_{i=0}^{N}$ and a large unlabeled dataset $\{Z_i\}_{i=0}^{M}$, its general training loss function for a model with parameters θ

is defined as in Formula 4.

$$\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} loss(X_i, Y_i) + \frac{1}{M}\sum_{j=1} Mloss(Z_j, R(Z_j, X)), \tag{4}$$

where $R(Z_j, X)$ is a task-specific function to represent the relation between each unlabeled training data point Z_j with the labeled dataset X (Jing & Tian, 2020). As semi-supervised learning methods result in more hardware requirements than self-supervised pretraining methods, this work will not look further into semi-supervised learning methods.

2.3 SSL methods

Many variations of self-supervised learning methods exist, ranging from the Masked Language Model pretraining objective used in BERT training (Devlin, Chang, Lee, & Toutanova, 2018) to the image reconstruction objective used in training Variational Auto-Encoders (VAEs) (Kingma & Welling, 2013). Liu et al. (2021) use the following three categories to divide existing SSL methods for representation learning. A conceptual comparison between these categories is shown in Figure 5.

- **Generative** methods train an encoder to encode input x into an explicit vector z and a decoder to reconstruct x from z.
- **Contrastive** methods train an encoder to encode input x into an explicit vector z to measure similarity.
- **Generative-Contrastive** (Adversarial) methods train an encoder-decoder to generate fake samples and a discriminator to distinguish them from real samples.

Generative SSL methods, such as the VAE (Kingma & Welling, 2013), are able to recover the original data from the learned representations without assumptions for downstream tasks (Liu et al., 2021). However, pixel-level generation in the decoder is computationally expensive (T. Chen, Kornblith, Norouzi, & Hinton, 2020). Additionally, generative SSL methods have recently been found far less competitive than contrastive SSL methods in some classification scenarios (Liu et al., 2021).

Generative-Contrastive SSL methods, such as Generative Adversarial Networks (GANs) (Radford, Metz, & Chintala, 2015), are particularly successful in image generation, transformation and ma-

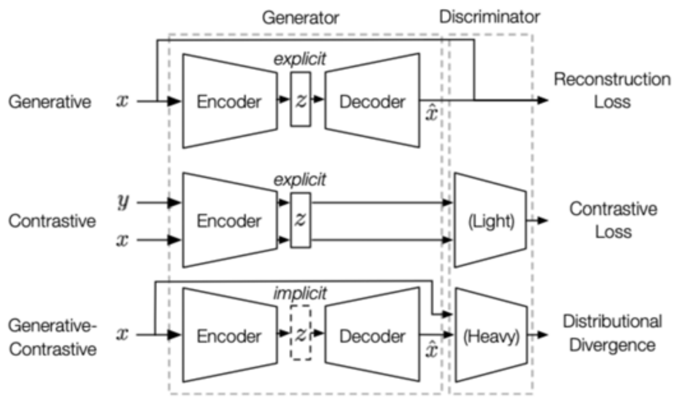


Figure 5: Conceptual comparison between Generative, Contrastive, and Generative-Contrastive self-supervised methods, from Liu et al. (2021).

nipulation, but do not output an explicit learned representation, and are therefore not as suitable for transfer learning as contrastive SSL methods (Liu et al., 2021).

Contrastive methods aim to group similar samples close to and diverse samples far from each other. To achieve this, a similarity metric is used to measure how close two embeddings (also called representations), extracted from an encoder network, are. This encoder network is trained in such a way that it learns to differentiate augmented versions of the same image (positive samples) from the augmented versions of different images (negative samples). In this way, the encoder network learns good representations of the samples, which can later be used for transferring to downstream tasks. Recent self-supervised contrastive methods have produced results comparable to state-of-the-art supervised methods on the ImageNet (Russakovsky et al., 2015) dataset (Jaiswal et al., 2021).

Research into the limits of contrastive SSL indicates that these methods do not seem to be able to improve accuracy past supervised methods (Newell & Deng, 2020), though note that as mentioned in Chapter 2.2, pretrained SSL models are able to learn better with less labeled data.

Five recent well-performing contrastive SSL methods are explained in the following subchapters. A summary of their properties is shown in Table 1.

Table 1: Summary of the properties of five recent well-performing contrastive SSL methods.

Method	Date	Low b.s.	Datasets	CNN Structure	Augmentations
MoCo (v2)	Nov 2019 (v2: Mar 2020)	256	ImageNet	ResNet	Resized crop, Horiz. flip, Color distortion, (v2: plus Gaussian blur)
SimCLR	Feb 2020	512	ImageNet, CIFAR-10	ResNet	Resized crop, Horiz. flip, Color distortion, Gaussian blur
BYOL	Jun 2020	256	ImageNet	ResNet	Resized crop, Horiz. flip, Color distortion, Gaussian blur, Solarization
SimSiam	Nov 2020	128	ImageNet	ResNet	Resized crop, Horiz. flip, Color distortion, Gaussian blur
Barlow Twins	Mar 2021	256	ImageNet	ResNet	Resized crop, Horiz. flip, Color distortion, Gaussian blur, Solarization

Low b.s. = Lowest well-performing batch size, Horiz. = Horizontal

2.3.1 SimCLR

T. Chen et al. (2020) introduced a simple framework for contrastive learning of visual representations (SimCLR). A visual representation of this method is shown in Figure 6. In SimCLR, each image x is augmented stochastically from a family of augmentations $\mathcal{T}$, resulting in two correlated images $\tilde{x}_i$ and $\tilde{x}_j$. An encoder network $f(\cdot)$ is then used to encode both images into vectors in representation space (h_i and h_j). The encoder network used is ResNet from He, Zhang, Ren, and Sun (2016). An MLP $g(\cdot)$ with one hidden layer, called the projection head, is then used to get $z_i = g(h_i)$ and $z_j = g(h_j)$.

SimCLR then uses a contrastive loss function to identify $\tilde{x}_j$ in $\{\tilde{x}_k\}_{k\neq i}$ for a given $\tilde{x}_i$ (using their corresponding z_i and z_j). The loss function for a positive pair of examples (i, j) is defined as in Formula 5.

$$l_{i,j} = -\log \frac{\exp(sim(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{k\neq i} \exp(sim(z_i, z_k)/\tau)}, \tag{5}$$

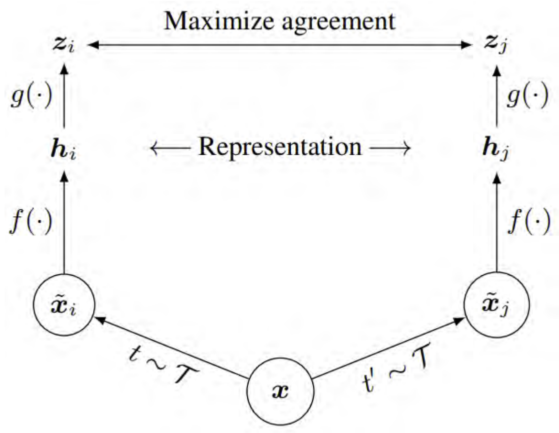


Figure 6: Schematic representation of the structure of the SimCLR method (T. Chen et al., 2020).

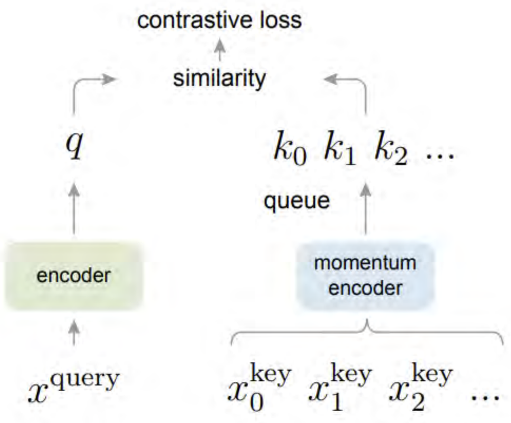


Figure 7: Schematic representation of the structure of the MoCo method (He et al., 2020).

where τ is a temperature parameter (a value of 0.1 has the best performance in the original paper) and N is the mini-batch size before augmentation. Using Formula 6, the final loss is computed across all positive pairs in a mini-batch, twice for each pair (i, j).

$$L = \frac{1}{2N} \sum_{k=1}^{N} [l(2k-1, 2k) + l(2k, 2k-1)] \tag{6}$$

Due to the importance of negative samples in the loss function, the lowest batch size that performs well in the paper introducing SimCLR is 512.

2.3.2 MoCo

He et al. (2020) introduced Momentum Contrast (MoCo), of which the structure is shown visually in Figure 7. Using this SSL method, each image x is again augmented stochastically to arrive at x_i and x_j. Each augmented image in a mini-batch is then put through two different encoder networks (ResNets), a query network $q = f_q(x)$ and a key network $k = f_k(x)$. The final loss for one resulting query (q) is then defined as in Formula 7.

$$L_q = -\log \frac{\exp(q \cdot k_+ / \tau)}{\sum_{i=0}^{K} \exp(q \cdot k_i / \tau)}, \tag{7}$$

where k_+ is the key k that resulted from the same (non-augmented) image as q, τ is a temperature parameter (set to 0.07) and K is the number of negative augmented images used. This loss function is called InfoNCE (Van den Oord, Li, & Vinyals, 2018).

The keys outside of the mini-batch currently being processed are not calculated every time, but come from a dictionary that remembers the keys from a certain previous number of batches. The weights θ_q of the query network are updated using a normal optimizer. To avoid collapse, the weights θ_k of the key network are calculated from θ_q with a momentum update, using Formula 8. Thanks to this momentum encoder, the encoding of all negative samples does not have to be calculated each batch, resulting in lower hardware requirements.

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q. \tag{8}$$

X. Chen, Fan, Girshick, and He (2020) show that MoCo's performance improves when adopting some aspects of SimCLR; namely, the projection head and better data augmentation (see Chapter 2.5). They call this improved method MoCo v2.

Due to the importance of negative samples in the loss function, the lowest batch size that performs well in the papers introducing both MoCo and MoCo v2 is 256.

2.3.3 BYOL

Grill et al. (2020), in their method Bootstrap Your Own Latent (BYOL), do away with the negative samples, and only use positive samples to calculate the loss. A schematic representation of its structure is shown in Figure 8. BYOL uses two networks: one online network consisting of a backbone (f_θ, a ResNet), projector (g_θ) and predictor (q_θ), while the target network only consists of a backbone (f_ξ) and projector (g_ξ), with the same structure as (but different weights than) their counterparts in the online network.

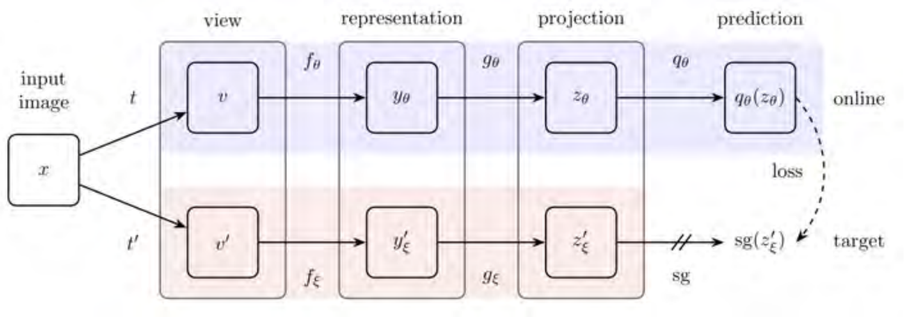


Figure 8: Schematic representation of the structure of the BYOL method (Grill et al., 2020). sg = stop-gradient.

Each image x is again augmented stochastically to arrive at the two augmentations ν and ν'. One of these is put through the online network, while the other is put through the target network. The resulting prediction $p_\theta = q_\theta(g_\theta(f_\theta(\nu)))$ and projection $z_\xi = g_\xi(f_\xi(\nu'))$ are used in the loss function (Formula 9), which is a negative cosine similarity metric.

$$l(\nu, \nu') = 2 - 2\frac{p_\theta \cdot z_\xi}{||p_\theta||_2 \cdot ||z_\xi||_2}, \tag{9}$$

where z_ξ is seen as a constant, and is therefore not backpropagated through (also known as stop-gradient: $sg(z_\xi)$). This loss function is calculated twice for each augmented image pair in the final loss function using Formula 10.

$$L = l(\nu, \nu') + l(\nu', \nu). \tag{10}$$

Only the weights θ of the online network are updated using a normal optimizer and the final loss function. The weights ξ of the target network are an exponential moving average of θ to avoid collapse, calculated with Formula 11.

$$\xi \leftarrow \tau\xi + (1 - \tau)\theta, \tag{11}$$

where τ is a target decay rate $\in [0, 1]$, with a value of 0.99 performing best.

Although BYOL only uses positive examples in its loss calculation, the lowest batch size that performs well in the paper introducing the method is 256. 128 is tried as well, but resulted in a decrease in fine-tuned classification accuracy on ImageNet of 2.6 percentage points.

2.3.4 SimSiam

X. Chen and He (2021) go even further, and reduce the SSL contrastive learning problem to a simple Siamese network (SimSiam). A schematic representation of this structure is shown in

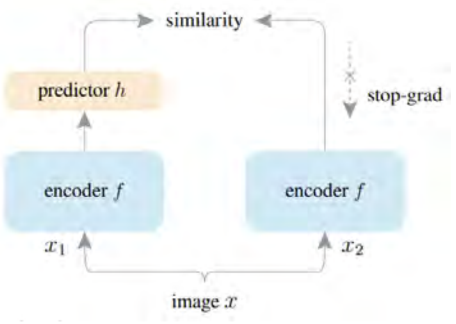


Figure 9: Schematic representation of the structure of the SimSiam method (X. Chen & He, 2021).

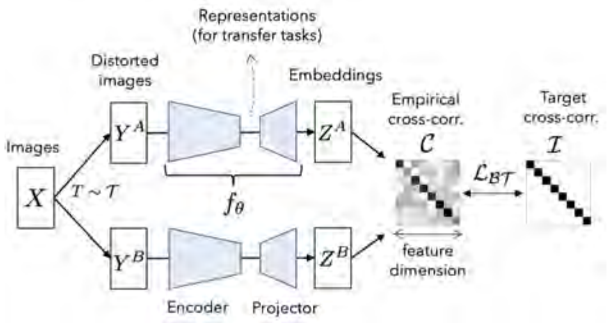


Figure 10: Schematic representation of the structure of the Barlow Twins method (Zbontar et al., 2021).

Figure 9. Two stochatically augmented images x_1 and x_2 originating from the same base image x are put through the same encoder network f consisting of a backbone (ResNet) and projection MLP, resulting in the representations z_1 and z_2. These representations are then separately used as input for a prediction MLP h with a bottleneck structure, resulting in the predictions p_1 and p_2. The representations and the predictions are used in the negative cosine similarity loss, as in Formula 12.

$$l(p_1, z_2) = -\frac{p_1}{||p_1||_2} \cdot \frac{z_2}{||z_2||_2}. \tag{12}$$

The final loss is then symmetrized using Formula 13.

$$L = \frac{1}{2} l(p_1, \textit{stop-grad}(z_2)) + \frac{1}{2} l(p_2, \textit{stop-grad}(z_1)), \tag{13}$$

where *stop-grad* stands for stop-gradient, meaning the representations z_1 and z_2 are treated as constants in the loss and are therefore not backpropagated through.

The authors show that this structure in combination with the stop-gradient and batch normalization of projection and prediction layers is enough to avoid collapse and to allow the network to learn meaningful representations. Furthermore, the authors show that SimSiam performs well with a batch size of 128, and even works reasonably well with a batch size of 64 (-2 percentage points in ImageNet linear evaluation accuracy).

2.3.5 Barlow Twins

The method called Barlow Twins was introduced by Zbontar et al. (2021). A schematic representation of this method's structure is shown in Figure 10. In Barlow Twins, all images in a mini-batch are stochastically augmented twice (resulting in two mini-batches Y^A and Y^B) and put through a siamese encoder network (a ResNet plus a projector MLP) with identical weights, resulting in the batches of embeddings Z^A and Z^B. The loss function (Formula 14) directly uses the correlation between the outputs of the two networks along the batch dimension.

$$C_{ij} = \frac{\sum_b z^A_{b,i} z^B_{b,j}}{\sqrt{\sum_b (z^A_{b,i})^2} \sqrt{\sum_b (z^B_{b,j})^2}}, \tag{14}$$

where b indexes batch samples and i, j index the vector dimension of the network's outputs.

The resulting matrix C is the cross-correlation matrix between the two output batches. The final loss (Formula 15) is constructed as follows to push the matrix to being as close to the identity matrix as possible, indicating perfect correlation between representations resulting from the same image, and no correlation between representations resulting from different images.

$$L_{\mathcal{BT}} = \sum_i (1 - C_{ii})^2 + \lambda \sum_i \sum_{j \neq i} C_{ij}^2. \tag{15}$$

The authors show that Barlow Twins is robust to small batch sizes, with a performance almost unaffected for a batch as small as 256, and -2 percentage points in Top-1 accuracy on ImageNet for batches of size 128.

2.4 Datasets used in SSL

Most literature uses balanced datasets for self-supervised pretraining purposes. All methods mentioned in Subchapter 2.3.1-2.3.5 use ImageNet (Russakovsky et al., 2015) as their pretraining dataset. The paper introducing SimCLR (T. Chen et al., 2020) also performed some testing on the CIFAR-10 dataset (Krizhevsky, Hinton, et al., 2009).

Practical applications of self-supervised learning generally use datasets with less balanced classes. In the medical domain for example, Azizi et al. (2021) use SSL pretraining on two medical datasets. Both these datasets contain some classes that occur in less than 1% of images. One of the datasets contains only images that contain just one class, while the other dataset can contain images with multiple labels. Both datasets contain at least 100 images per class with only that class. H. Li et al. (2021) also use two imbalanced datasets, but the authors use an SSL method that constructs balanced mini-batches during pretraining, which requires a well-labeled pretraining dataset.

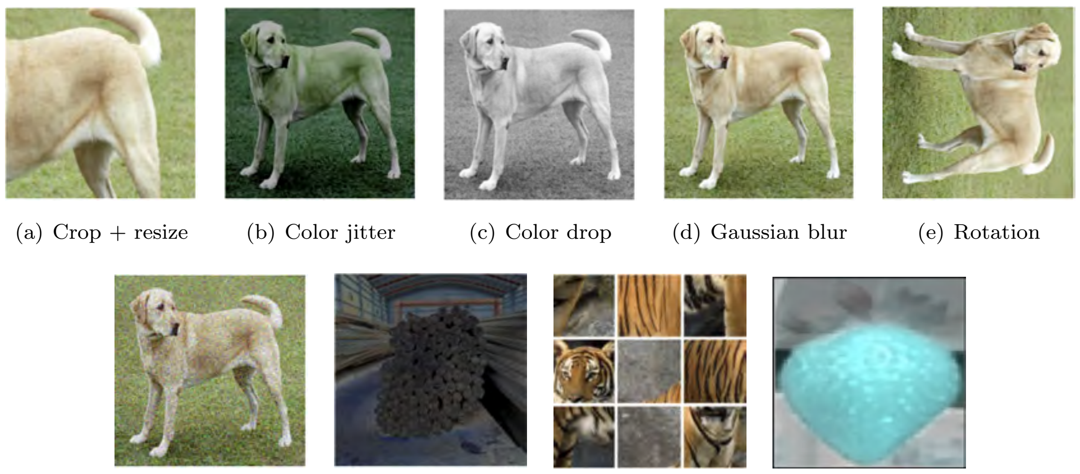
In the fruit quality domain, Choi, Would, Salazar-Gomez, and Cielniak (2021) use a dataset of strawberry images. The anomalous class in this dataset is just 4.3% of the data, though the image augmentation method during SSL pretraining is based on color channel randomization, which can be seen as creating new anomalous examples of strawberry images.

2.5 SSL image augmentation

As found by T. Chen et al. (2020), the composition of multiple image augmentation operations is crucial to yield effective representations in the SimCLR method. Furthermore, they found that contrastive SSL needs stronger data augmentation than supervised learning. The best combination of data augmentations that these authors found was a combination of random crop and resize, horizontal flip, color distortion and Gaussian blur. Examples of some of the augmentations discussed in the text below are shown in Figure 11, the letters of the particular augmentations in this Figure are referenced in the text using brackets.

In random cropping and resizing (a), a crop of random size is made, and resized to the original image size. Color distortion is the random application of color jitter (b) (multiplying the brightness, contrast, saturation and hue of an image by a random factor) and/or color drop (c) (converting the image to grayscale using the formula $0.3 * Red + 0.59 * Green + 0.11 * Blue$ (Padmavathi & Thangadurai, 2016)). Gaussian blur (d) is applied to 50% of the images, using a 2-dimensional Gaussian kernel with a random $\sigma \in [0.1, 2.0]$ to convolve over the original image.

Other image augmentations that were tried by the authors of SimCLR (T. Chen et al., 2020) include random image rotations (e) and Gaussian noise (f). Gaussian noise means adding a random number to each pixel, according to a Gaussian distribution centered around 0. Other contrastive SSL methods use the same image augmentations as SimCLR. BYOL (Grill et al., 2020) and Barlow



(a) Crop + resize (b) Color jitter (c) Color drop (d) Gaussian blur (e) Rotation

(f) Gaussian noise (g) Solarization (h) Jigsaw (i) CH-Rand

Figure 11: Examples of the image augmentations discussed, from T. Chen et al. (2020), Y. Li et al. (2021), Jaiswal et al. (2021) and Choi et al. (2021)

Twins (Zbontar et al., 2021) additionally use solarization (g), where pixels above a threshold are inverted (with a threshold of 0.5: $x \mapsto x \cdot \mathbf{1}_{\{x<0.5\}} + (1-x) \cdot \mathbf{1}_{\{x\geq 0.5\}}$). PIRL (Misra & Maaten, 2020) uses a jigsaw augmentation (h), where the original image is divided in a number of blocks, which are then shuffled.

According to Kolesnikov, Zhai, and Beyer (2019), the optimal combination of augmentations is dependent on which CNN structure is used in training. Additionally, the optimal augmentations for contrastive SSL seem to be task-dependent (Tian et al., 2020). For example, Choi et al. (2021) construct their own image augmentation CH-Rand (i), based on color channel randomization, to push their SSL pretraining towards detecting color anomalies in strawberries.

2.6 Matthew's Correlation Coefficient

When dealing with imbalanced multiclass datasets, using accuracy as an evaluation metric does not properly represent the true performance of the models, as it tends to hide strong classification errors for classes with few units (Grandini, Bagli, & Visani, 2020). A solution to this problem comes from Matthews Correlation Coefficient (MCC). Starting in the 2000s, MCC and its multiclass extension became widely employed metrics for machine learning (Chicco & Jurman, 2020).

MCC expresses the degree of correlation between two categorical random variables (predicted and true classification). It has a range of $[-1, +1]$ and is calculated for the multiclass case by Formula 16.

$$MCC = \frac{cs - \sum_k^K p_k t_k}{\sqrt{(s^2 - \sum_k^K p_k^2)(s^2 - \sum_k^K t_k^2)}}, \tag{16}$$

where the following intermediate variables are used:

- K = the total number of classes
- c = the total number of elements correctly predicted
- s = the total number of elements
- p_k = the number of times that class k was predicted
- t_k = the number of times that class k truly occurred

In contrast to accuracy and F1-score, MCC includes all entries of the confusion matrix, and is therefore generally regarded as a balanced measure which can be used in multiclass classification even if the classes are very different in size (Grandini et al., 2020).

2.7 Grad-CAM

While CNN-based models enable good performance in computer vision, their lack of decomposability into individually intuitive components makes them hard to interpret (Lipton, 2018). This generally makes it very hard to link the output of such a network to specific pixels of its input image. There have been a lot of methods proposed to visually attribute the predicted classes of a CNN to the pixels of an input image. In a comparison between many of these methods, Adebayo et al. (2018) show that the method Grad-CAM is very sensitive to randomization of both the model weights and the image labels, which indicates that this method is a good way to visually represent the areas of input images that are most responsible for model predictions.

Selvaraju et al. (2017) introduced the method called Gradient-weighted Class Activation Mapping (Grad-CAM), a generalization of an earlier visual attribution method called Class Activation Mapping (CAM) (Zhou, Khosla, Lapedriza, Oliva, & Torralba, 2016). Where CAM could only be used on neural networks that have no fully-connected layers, Grad-CAM can be used on any CNN-based network. The method is based around the idea that the last convolutional layer of a network both retains the spatial information of the original input image (as any convolutional layer does) and captures high-level concepts (as opposed to early layers that capture more fine-grained details).

Grad-CAM works by setting the prediction output of the class c that is to be visualized to 1. We then backpropagate up to the activated values A^k_{ij} of each channel k in the last convolutional layer, and average these values in each channel (as in Formula 17).

$$\alpha^c_k = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A^k_{ij}}, \tag{17}$$

where Z is equal to the total number of values in one channel of the last convolutional layer. We use these weights α in Formula 18 to obtain a linear combination of all feature maps, then use a ReLU on the resulting values (since we are only interested in the features that have a positive influence on the class of interest):

$$L^c_{Grad-CAM} = ReLU\left(\sum_k \alpha^c_k A^k\right). \tag{18}$$

Thus, we obtain a heatmap of the same size of the channels of the final convolutional layer. To obtain a heatmap of the same size of the input image, bilinear interpolation is used. Examples of resulting heatmaps for a particular image are shown in Figure 12.

2.8 t-SNE

T-distributed stochastic neighbor embedding (t-SNE) (Van der Maaten & Hinton, 2008) is a method to visualize high-dimensional data by giving each data point a location in a 2- or 3-dimensional map. It does so by minimizing the Kullback-Leibner divergence in Formula 19 (using gradient descent) between two probability distributions, one in high-dimensional space (P), and one in the target low-dimensional space (Q):

$$L = KL(P||Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{19}$$

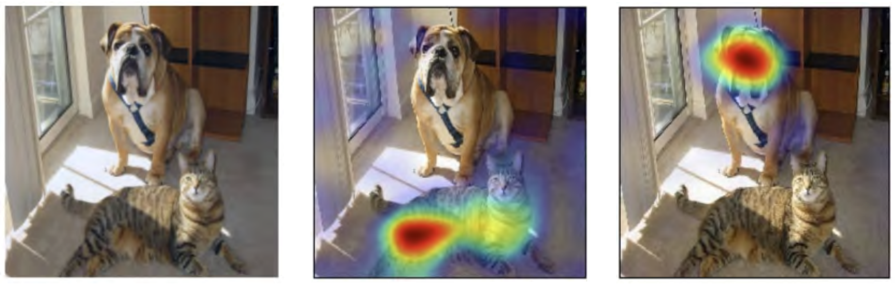

(a) Original (b) Grad-CAM for Cat class (c) Grad-CAM for Dog class

Figure 12: Support for classes 'Tiger cat' (Cat) and 'Boxer' (Dog) for a particular image according to Grad-CAM visualizations (Selvaraju et al., 2017).

Here, q_{ij} is calculated using Formula 20 and p_{ij} is calculated using Formula 21, which itself uses the variables $p_{j|i}$ as calculated by Formula 22.

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}}. \tag{20}$$

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}. \tag{21}$$

$$p_{j|i} = \frac{\exp(-||x_i - x_j||^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-||x_i - x_k||^2 / 2\sigma_i^2)}. \tag{22}$$

In these formulas, x_i are the data points in high-dimensional space, y_i are the corresponding data points in low-dimensional spance, n is the number of data points, and σ_i is set in such a way that it produces a fixed perplexity (see Formula 23) that is specified by the user.

$$Perplexity = 2^{-\sum_j p_{j|i} \log_2 p_{j|i}}. \tag{23}$$

Typical values of the perplexity parameter are between 5 and 50.

An example of a 2-dimensional t-SNE visualization of the MNIST dataset is shown in Figure 13.

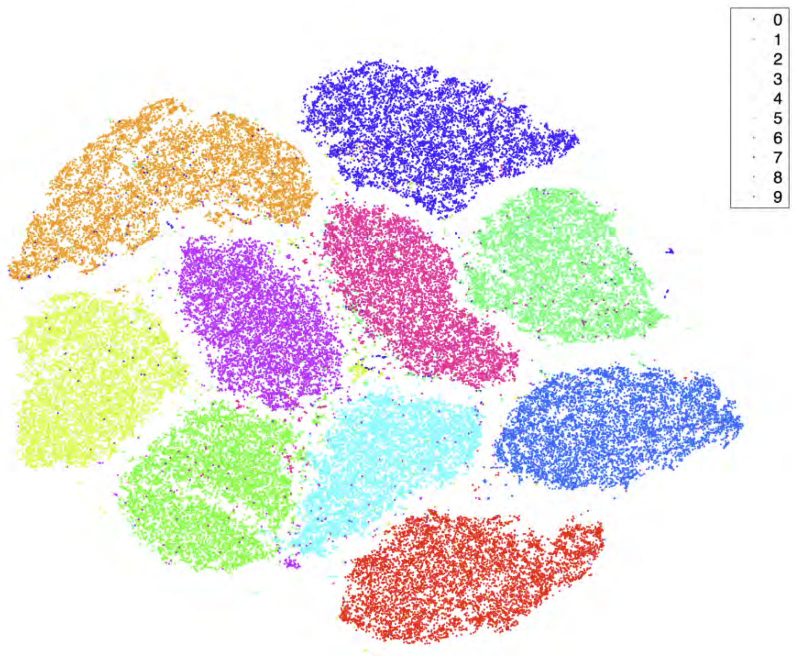


Figure 13: Example of a 2-dimensional t-SNE visualization of the MNIST dataset, with each of the 10 different numbers having a different color (Chan et al., 2018).