SIMSIAM: SIMPLE SIAMESE NETWORKS FOR PRETRAINING ON UNLABELED DATA

Till Ariel Aczel - s203216

ABSTRACT

It has not been well understood why Siamese networks without negative pairs don't collapse to a degenerate solution, but their simplicity and results make them compelling. This work implements SimSiam [1] and investigates the following: (1) model performance resistance against smaller labeled training data-set size (2) predictor optimization strategy, especially the relative learning rate compared to the embedding model (3) effect of optimizer weight decay (4) components critical to avoid mode collapse. Code is available at [2].

Index Terms— SimSiam, representation learning

1. INTRODUCTION

A common trend can be observed, that deep learning model performance improves with bigger models, which require more computing power and more data. For most applications data is limited, and collection and labeling of new data is too expensive. Natural language processing solves this by using massive unlabeled data-sets to train huge general models and fine tune them on a smaller labeled data-set. A similar idea can be adopted for computer vision, where general model is trained with a pretext task in an unsupervised fashion on a large data-set, then the model is fine tuned for a specific task on a smaller labeled data-set.

Siamese networks are a commonly used architecture for representation learning. Recently it has been shown that Siamese networks can be trained with contrastive loss without negative pairs [1, 3, 4, 5, 6, 7] and they do not collapse to a degenerate solution. Exclusion of negative pairs is beneficial as designing a mining strategy is a difficult task and it affects convergence greatly.

2. MODEL AND METHODOLOGY

Model training is done in two steps. First the model learns general representations via unsupervised pretraining on the unlabeled training-set. With these representations a task-specific model is trained on the labeled training-set. In this report for evaluation I am using kNN classification and the commonly used linear evaluating strategy by training a linear classifier on the representations extracted by the encoder.

Thanks to Tommy Sonne Alstrøm and Bo Li for guiding me trough this project and to DTU Compute for providing the GPU servers.

2.1. Unsupervised pretraining

The pre-training task is to create a model which is invariant in the embedding space to data augmentations. Model architecture is visualized in Figure 1. Two augmentations are sampled to generate two different views of an image, see Figure 2. The augmented images are fed trough two identical encoders and projectors creating two embeddings. One of the embeddings is processed by the predictor and it's distance is computed to the other embedding with a stop gradient applied. The loss is symmetrized by feeding both augmentations trough the online and target network as well.

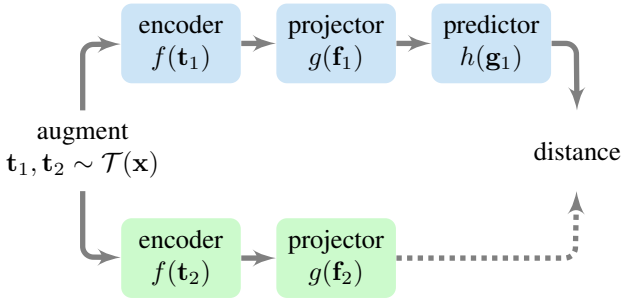


Fig. 1. SimSiam model architecture, blue: online network, green: target network. The dotted line represents the stop-gradient operation.

$$\mathbf{t}_1, \mathbf{t}_2 \sim \mathcal{T}(\mathbf{x}) \tag{1}$$

$$\mathbf{g}_i = g\big(f(\mathbf{t}_i)\big), i \in \{1, 2\} \tag{2}$$

$$\mathbf{h}_i = h(\mathbf{g}_i), i \in \{1, 2\} \tag{3}$$

$$\mathcal{L} = \frac{1}{2}\Big[\mathcal{D}\big(\mathbf{h}_1, \mathrm{sg}(\mathbf{g}_2)\big) + \mathcal{D}\big(\mathbf{h}_2, \mathrm{sg}(\mathbf{g}_1)\big)\Big] \tag{4}$$

Baseline settings. Unless specified, all experiments follow the baseline settings of SimSiam [1] CIFAR10 experiment except for the creation of a validation-set, and increasement of training epochs to have similar number of iterations:

Model architecture. The encoder $f(\cdot)$ is the CIFAR10 variant of ResNet18 [8], where the maxpooling layer is removed and the first convolutional layer is replaced by a kernel size of 3 and stride of 1. The classification layer is dropped and the features of the average pooling layer is used as $\mathbf{f}$. The projector $g(\cdot)$ is a 2 layer MLP with batch-norm at each layer and ReLU activation at hidden layers. The batch-norm of the

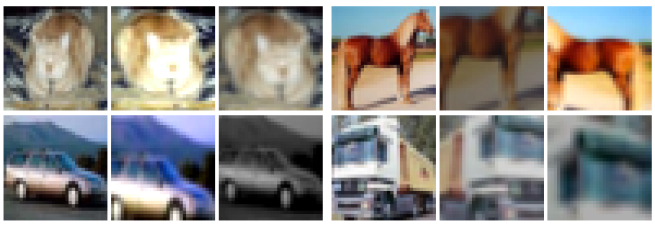


Fig. 2. Augmentation samples with the original and two samples for each image. Augmetations include: random resize crop, horizontal flip, color jitter and gray scale.

last layer has fixed mean ($\beta = 0$) and variance ($\gamma = 1$). The hidden layer dimension of the projector is equal to the output dimension of the average pooling layer (512 for 32×32 images). The projector output dimension is 2048. The predictor $h(\cdot)$ has 2 layers with batch-norm and ReLU activation on the hidden layer. The predictor has an auto-encoder like architecture where each hidden layer is one fourth of the size of the output layer, which matches the output dimension of the projector network.

Loss function. The loss is the negative symmetric cosine similarity between the embeddings, see Equation 4, where

$$\mathcal{D}(\mathbf{u}, \mathbf{v}) = -\frac{\mathbf{u}}{\|\mathbf{u}\|_2} \cdot \frac{\mathbf{v}}{\|\mathbf{v}\|_2} \tag{5}$$

Optimizer. For optimization I use the SGD optimizer with a batch size of 512. The base learning rate is $lr \times \text{BatchSize}/256$, where $lr = 0.03$. I use cosine learning rate decay for the embedding and projector network. The SGD momentum is 0.9 and weight decay is $5\text{e}{-4}$. SimSiam [1] trained for 800 epochs, but they did not use a validation-set for the CIFAR10 experiment, contrastingly I split the training-set 80%-20% and increase the number of epochs to 1000 to have similar amount of iterations. Parameter initialization follows the default PyTorch initialization.

Data. As mentioned above the CIFAR10 [9] training-set is split 80%-20% to create a validation-set. For augmentations I use a random resize crop with scale [0.2, 1], horizontal flip with a probability of 50%, color jitter with [brightness, contrast, saturation, hue] of [0.4, 0.4, 0.4, 0.1] with a probability of 80% and gray scale with a probability of 20%. All images are normalized with the mean and standard deviation of the training-set. For visualization of sample augmentations please refer to Figure 2.

With these settings one training run takes about 8 hours on a Tesla V100 PCIe 16GB GPU, and the evaluation calculations for each epoch in total about 2 hours.

2.2. Self-supervised

After the unsupervised pre-training, the projector and predictor are thrown out. The embedding $f(\cdot)$ model's representations can be used for a specific task with a small labeled data-set. In this work I evaluate the models with two metrics where I assume, that 100%, 10% and 1% of the training-set is labeled. I use kNN classification and I follow the commonly used linear evaluation. For both evaluations I normalized the images with the training-set mean and standard deviation, no further augmentations are used. I run each experiment 5 times with different seeds and report the mean and standard deviation. Note that how the embedding model is used after the pre-training is task and data-set specific. The final model can be further improved by using fine-tuning and distillation [4], but that was out of scope for this project.

kNN classification. I evaluate the representations by training a kNN classification model on the training-set with exponentially weighted cosine similarity. I use 200 nearest neighbours and a temperature of 0.1.

Linear evaluation. The images are embedded using $f(\cdot)$ and a linear classifier with cross-entropy loss is trained on the representations. I use the SGD optimizer with a batch size of 256 and learning rate of $lr \times \text{BatchSize}/256$, where $lr = 30$. The momentum is 0.9 and there is no weight decay. To achieve the same number of iterations for the three scenarios I duplicated the training-set 10 and 100 times for 10% and 1% respectively.

3. RESULTS

I compare my reproduction to the SimSiam paper [1], then I look at the k-nearest neighbours qualitatively, I highlight the advantage of self-supervised learning compared to supervised learning, I perform ablation on the predictor learning rate and optimizer weight decay and finally I explore how SimSiam avoids mode collapse.

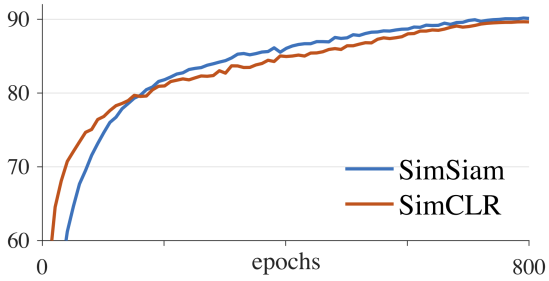
3.1. Reproduction

I reproduced the SimSiam [1] CIFAR10 experiment with the same hyper-parameters except for the linear evaluation optimizer. Besides not using a validation split and I training for 800 epochs, like in SimSiam [1], I also compare to doing the 80%-20% data-split with 1000 epochs. For the linear evaluation I use the SGD optimizer with the parameters described in the baseline settings in Section 2.2.

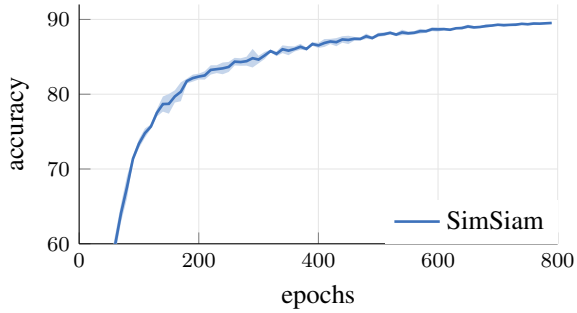
I achieved similar results as presented in the paper, see Table 1. It is hard to make an exact comparison as the kNN accuracy was only plotted, and did not report the linear evaluation with SGD optimizer.

	kNN acc.	linear acc.
SimSiam [1]	$\sim$90	91.8
reproduction	89.42±0.19	89.94±0.14
80%-20% data split	88.02±0.35	88.34±0.21

Table 1. Test accuracy in percentage. Note that the SimSiam [1] linear accuracy result is with the LARS optimizer.



(a) SimSiam CIFAR10 experiment [1].



(b) My reproduction (5 runs).

Fig. 3. kNN accuracy over training epochs. My reproduction visualizes the mean and standard deviation of 5 runs.

3.2. Qualitative assessment

In Figure 4 I plot some query images from the test-set and their nearest neighbours in the training-set. In general the nearest neighbours are similar images with just a few mistakes. The image of the owl is confused with dogs, as there are not many images in the data-set wit bird faces. The second to last image of a plane has many similar images with pets in front of green background. This suggest stronger color augmentations might be useful. You can see a boat on land in the last query, which is confused with planes. In general the model has problems with images which are rare for the given category. Note that the learned representations are not task specific. For example even though the labels do not have *white horse* or *horse head*, just *horse* it learned the difference between them.

3.3. Labeled training-set size

To highlight the strength of self-supervised learning I compare the model performance to a supervised benchmark for the three scenarios where we have access to 100%, 10% and 1% of the labels. The supervised benchmark was trained similarly to SimSiam, but with a cross entropy loss. The network is the CIFAR10 version of the ResNet18 model trained for 800 epochs with a batch size of 512, SGD optimizer, base learning rate $lr = 0.03$, cosine learning rate annealing and $5e{-}4$ weight decay.

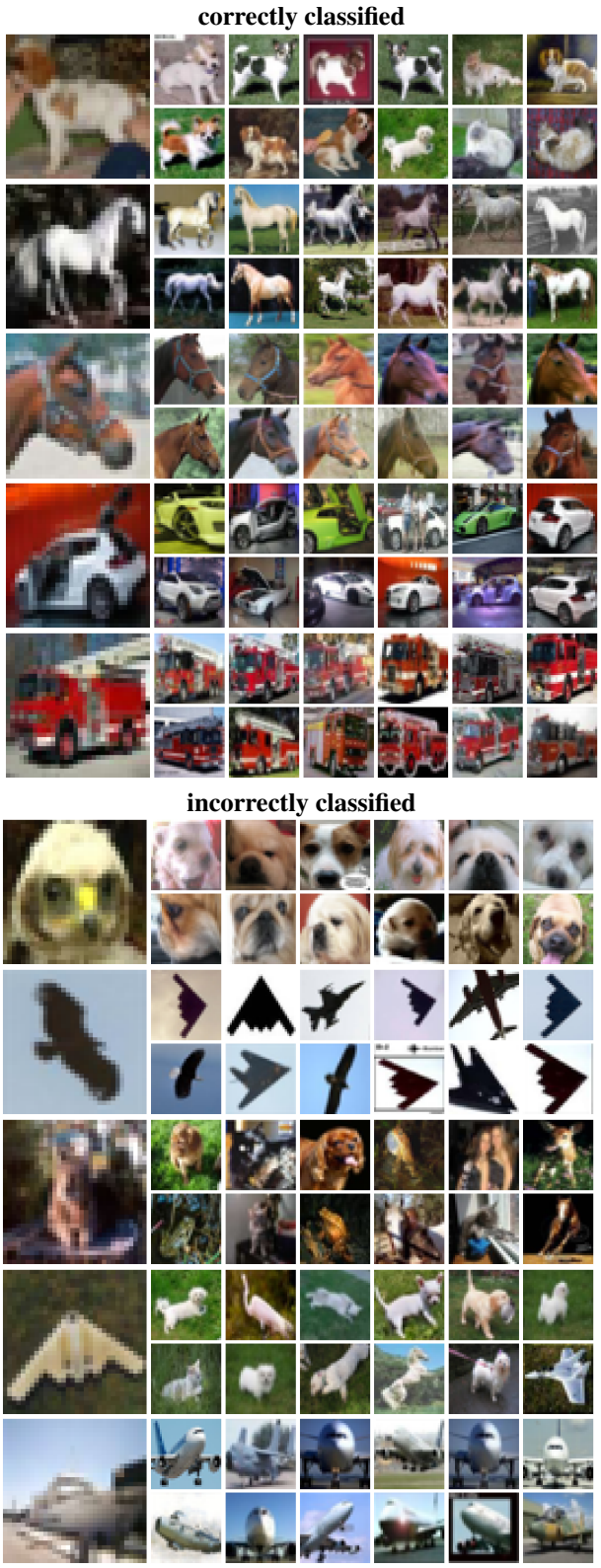


Fig. 4. Incorrectly classified test images and their closest training-set neighbours.

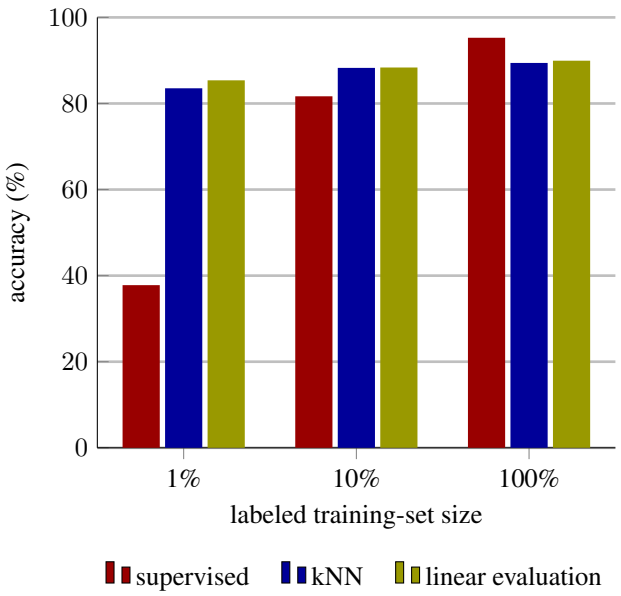


Fig. 5. Test accuracy in percentage of the supervised benchmark for different labeled training-set sizes.

For lower labeled training-set proportion the supervised model accuracy goes down more as the SimSiam accuracy, see Figure 5. The two self-supervised evaluation tasks, kNN and linear classifier perform similarly. For tasks, where collecting quality labels is expensive , self-supervised learning methods excel. With just 1% of the labels SimSiam outperforms the supervised learning model with 10% of the labels, even though the supervised learning model has access to ten times more labels.

3.4. Predictor learning rate

Wrong learning rate selection causes divergence in gradient based optimizers. As convergence of the SimSiam [1] model is not well understood it is worth exploring the effect of chosing a different learning rate for the asymmetric pat of the model, the predictor network. Let's define α_h as the ratio between the predictor learning rate and the learning rate of the encoder and projector: $lr_h = \alpha_h \cdot lr$.

Lowering the predictor learning rate slightly improves performance, what suggests that an imperfect predictor is beneficial for model training. However if the predictor learning rate is too small the model struggles to learn anything and we risk mode collapse, confirming the findings of [5]. It is not clear if there is any benefit from using cosine learning rate decay for the predictor, the performance difference is too small. Note that I did not explore α_h for different encoder and predictor learning rates. It could be that the higher accuracy is due to the whole model benefiting from a lower learning rate and not due to the learning rate ratio.

kNN

α_h	scheduler	1%	10%	100%
1/3	constant	41.95±4.00	64.03±3.45	69.25±3.55
	cosine	38.21±5.57	45.43±1.17	51.55±0.81
2/3	constant	82.59±0.56	**88.52**±0.36	**89.35**±0.29
	cosine	**83.51**±0.50	**88.41**±0.20	**89.09**±0.21
1	constant	81.73±0.69	**88.11**±0.31	88.91±0.23
	cosine	81.90±0.85	**88.10**±0.50	88.78±0.35
4/3	constant	80.83±0.57	87.65±0.55	88.66±0.28
	cosine	80.87±0.71	87.56±0.47	88.55±0.32

linear evaluation

α_h	scheduler	1%	10%	100%
1/3	constant	65.81±3.26	69.73±3.07	70.31±3.18
	cosine	48.08±2.97	51.33±4.29	57.98±2.59
2/3	constant	84.06±0.57	**88.54**±0.56	**90.06**±0.37
	cosine	**84.78**±0.30	**88.69**±0.37	**90.09**±0.37
1	constant	82.93±0.83	87.70±0.50	**89.58**±0.49
	cosine	83.17±0.76	87.92±0.52	**89.56**±0.50
4/3	constant	81.92±0.48	87.17±0.79	89.04±0.57
	cosine	81.79±0.44	87.22±0.59	89.25±0.55

Table 2. Validation accuracy in percentage for different predictor learning rates. For each column values are highlighted where the Welch's one-sided t-test for mean comparison with 5% significance can not be rejected.

3.5. Weight decay

Optimizer weight decay is a common form of regularization in neural networks. In these experiments I explore the effect of weight decay on model performance and convergence.

kNN

Weight decay	1%	10%	100%
0.00005	74.90±0.76	77.94±1.24	80.03±1.10
0.00015	**81.95**±0.28	86.93±0.38	88.04±0.39
0.0005	**81.73**±0.69	**88.11**±0.31	**88.91**±0.23
0.0015	45.04±2.90	50.00±3.14	54.55±3.54
0.005	9.86±0.00	9.86±0.00	9.86±0.00

linear evaluation

Weight decay	1%	10%	100%
0.00005	75.07±0.96	78.20±0.39	83.87±0.71
0.00015	**83.54**±0.22	86.82±0.25	**89.18**±0.37
0.0005	**82.93**±0.83	**87.70**±0.50	**89.58**±0.49
0.0015	51.86±3.70	56.52±4.84	57.24±5.04
0.005	10.13±0.09	9.90±0.00	9.85±0.03

Table 3. Validation accuracy in percentage for different optimizer weight decays. For each column values are highlighted where the Welch's one-sided t-test for mean comparison with 5% significance can not be rejected.

Performance is sensitive to weight decay. Not enough regularization and the model performs poorly, but too much regularization causes model collapse. Future work could include the exploration how the model behaves if we set the weight decay for the predictor and the rest of the model differently. It could give us better insight how SimSiam avoids mode collapse.

3.6. Improved model

The goal of this work was not to improve the model, but to get better insights how SimSiam works, but with the hyper-parameter search performed in Section 3.4 and 3.5 we can re-train a new model on the whole training-set to see if the performance boost holds on the test-set. From Section 3.4 we expect that $\alpha_h = 2/3$ and a cosine learning rate decay for the predictor model should improve performance. I did not find a better weight decay in Section 3.5. Compared to the reproduction only these hyper-parameters changed.

kNN			
	1%	10%	100%
baseline	83.53±0.55	88.27±0.26	89.42±0.19
improved	83.84±0.60	88.50±0.28	89.79±0.25
p-value	23.3739	12.4484	**2.3852**
linear evaluation			
	1%	10%	100%
baseline	85.38±0.31	88.36±0.23	89.94±0.14
improved	86.38±0.23	88.69±0.45	90.11±0.30
p-value	**0.0553**	11.7594	17.9813

Table 4. Test accuracy of two models and their associated Welch's one-sided t-test p-value (both in percentage). The two models are: the baseline hyper-parameters and the optimized relative predictor learning rate. P-values are highlighted where the null hypothesis could be rejected with 5% significance level.

The improved model significantly outperforms the baseline for kNN accuracy with 100% of the labels and linear evaluation strategy for 1% of the labels. As the family wise error rate increases with the number of tests performed I counteract it by applying the Bonferroni correction. We can accept at a significance level of 0.3318%, that in some evaluation scenarios a smaller predictor learning rate yields better results.

3.7. Mode collapse

To better understand how SimSiam is avoiding mode collapse I remove one at a time the stop gradient, predictor network and the projector last layer batch norm.

As showed in SimSiam [1] removing the stop-gradient or the predictor causes the model to collapse. It has been suggested by [10] that the projector last layer batch normalization prevents complete collapse. Interestingly, my empirical results show the opposite: by removing the projector last layer batch-norm, the correlation strength increased marginally, thus it is not an essential part of the model, contrary it improved the kNN accuracy.

removed	kNN acc. (%)	corr. strength
-	89.53	0.0521
no stop-gradient	**10.17**	**0.8917**
no predictor	**13.27**	**0.9977**
no batch-norm	90.26	0.0853

Table 5. Baseline model compared to models where a component is removed one at a time. Correlation strength denotes the average absolute value of the non-diagonal elements of the correlation matrix. The correlation is computed on the output of the embedding network: $\mathbf{h}$.

4. ENVIRONMENTAL IMPACT

Training big deep learning models is compute intensive, which produces a lot of greenhouse gasses. I estimated how much CO_2 equivalent gasses have been produced by just the GPUs used for training the models.

The reproduction with debugging took about 30 runs and this work contains about 20 experiment settings with 5 runs each, so in total about 130 runs. One run with 1000 epochs took on average about 10 hours with an average GPU power consumption of 205 W, what leads to about 2.05 kWh per run. In total I used up about 1300 GPU hours and 266.5 kWh. Average CO_2 emission in Denmark is around 150 g CO_2/kWh [11], therefore the experiments produced approximately 39.975 kg CO_2. To put it in relatable unit, an average new passenger car in Europe produces about 108 g CO_2/km [12], therefore all the runs CO_2 emission is equivalent to driving around 370 km with a new European passenger car.

5. DISCUSSION AND FUTURE WORK

Through this work I have explored the SimSiam model focusing on highlighting when self-supervised methods outperform supervised models and gaining a better understanding on how SimSiam works and avoids mode collapse. With these experiments I have demonstrated that for tasks with limited labels even a simple siamese model can outperform supervised models.

Self-supervised learning is a promising research direction for vision models. Big transformer models, like BERT [13] and GPT-3 [14] brought a breaktrough in natural language processing and made many consumer product possible. With self-supervised learning a similar leap can be achieved in computer vision.

Self-supervised learning methods need to optimize the right pretext task to get quality representations. In recent years a popular pretext task was to create models which are invariant to data augmentations. A hidden assumption in these self-supervised learning models is, that the data augmentation used to generate positive pairs does not change the semantic meaning of the image. Without the right augmentations the model solves the wrong pretext task and thus does not generate the optimal features. Also for the model to be able to learn good representations it needs dissimilar views of the same underlying information. It is challenging to find augmentations which do not alter the semantic meaning of the image, but have a diverse distribution of views. Future work could focus on how to define and select good augmentations.

Multi-modal models, like [15] are another promising research direction. By creating a common embedding space for models with different input data modality the number of use cases explode. As this task is even more compute intensive only big organizations with many resources can tackle it.

References

[1] Xinlei Chen and Kaiming He, "Exploring simple siamese representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 15750–15758.

[2] Till Ariel Aczel, "Simsiam," `https://github.com/tillaczel/simsiam`, 2021.

[3] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[4] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton, "Big self-supervised models are strong semi-supervised learners," *arXiv preprint arXiv:2006.10029*, 2020.

[5] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al., "Bootstrap your own latent: A new approach to self-supervised learning," *arXiv preprint arXiv:2006.07733*, 2020.

[6] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin, "Unsupervised learning of visual features by contrasting cluster assignments," *arXiv preprint arXiv:2006.09882*, 2020.

[7] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin, "Emerging properties in self-supervised vision transformers," *arXiv preprint arXiv:2104.14294*, 2021.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[9] Alex Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., University of Toronto, 2009.

[10] Tianyu Hua, Wenxiao Wang, Zihui Xue, Sucheng Ren, Yue Wang, and Hang Zhao, "On feature decorrelation in self-supervised learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 9598–9608.

[11] Energinet, "Environmental report 2018," Tech. Rep., Energinet, 2018.

[12] Uwe Tietge, Peter Mock, Sonsoles Díaz, and Jan Dornof, "Co2 emissions from new passenger cars in europe: Car manufacturers performance in 2020," *International Council on Clean Transportation*, 2021.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[14] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al., "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[15] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever, "Zero-shot text-to-image generation," *arXiv preprint arXiv:2102.12092*, 2021.