Implementation of Self-Supervised Model for Transfer Learning

Arvind Sriram

Implementation of Self-Supervised Model for Transfer Learning

Self-supervised learning is a subtype of unsupervised learning that generates the supervised learning problem by employing unlabeled input data. This type of learning is also known as semi-supervised learning. You can usually make an accurate guess about how the bottom half of an image will look based solely on the top half of the image if you look closely enough at it. It is feasible to reconstruct an RGB image from a grayscale one using the appropriate steps. The idea behind self-supervised learning is that the model's input data may be mined to provide supplementary pre-text tasks. This is the central tenet of the methodology. While it is performing its secondary function, the model is learning about the structure of the data (Chen, et al., 2020). Contrastive learning procedures have been found to be the most successful for computer vision, despite the fact that many researchers have investigated unsupervised learning strategies. In this section, we will investigate alternative learning strategies that might be used for self-monitoring.

The effectiveness of supervised learning is at its peak when there is a significant amount of data that has been labeled and organized. Because complicated tasks such as object detection and instance segmentation require more in-depth annotations, acquiring labelled data of a high quality can be a tough and expensive endeavor. An increased number of annotations are required in general for activities as complex as these (He et al., 2020). On the other hand, there is a plethora of data that has not been categorized for us to work with, and this presents a challenge. It is necessary to generate meaningful representations of data from a pool of unlabeled data via self-supervision, and then to fine-tune the representations with few labels for the supervised job farther down the pipeline. This need motivates the practice of self-supervised learning. This requirement was the impetus for the development of the self-supervised learning practice. The ultimate goal

might be anything from easy labeling to intricate semantic segmentation as well as object recognition.

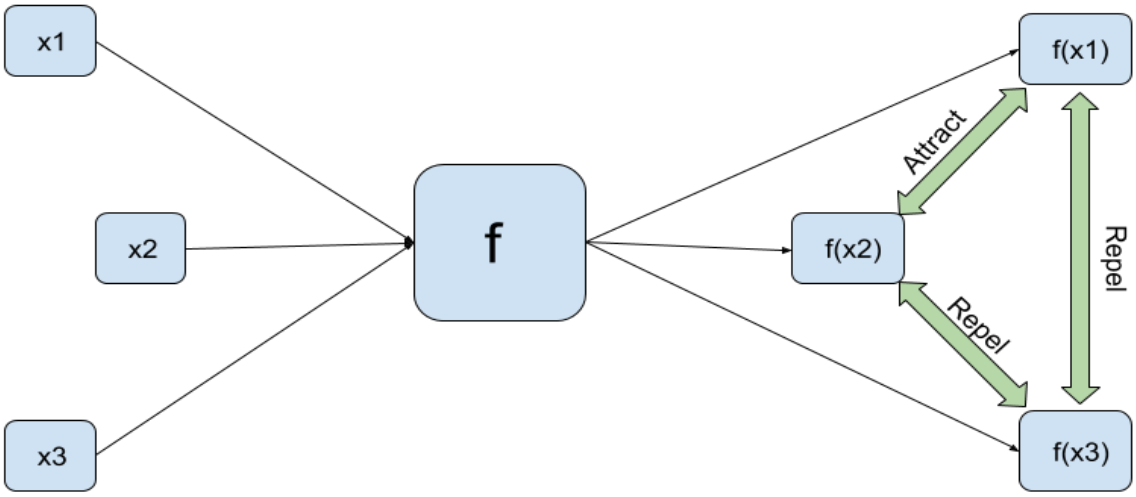
The utilization of transformer models is the driving force behind a sizeable portion of the natural language processing industry's recent achievements in terms of advancement. In the context of NLP operations, Bert[1, T5], along with other Transformers, has been known to make use of the term "self-supervision." [Note: First, it is educated using vast quantities of unlabeled data, and then, when it comes time to fine-tune, a very little quantity of data that has been labeled is used. The purpose of this work is to shed light on some of the self-supervised learning approaches that have been investigated for computer vision. My expectation is that we will be able to achieve this objective as a result of this paper.

A set of additional pre-text tasks can be mined from the model's input data in order to construct the tasks themselves, which is the concept that underpins the concept of self-supervised learning. As a result of these additional efforts, the model is able to acquire an understanding of the components that comprise the data (for instance the structure of the object in case of image data). Because contrastive learning methods give the impression of functioning more effectively than other methods for computer vision, I will focus this essay on self-supervised learning approaches that make use of contrastive learning because those are the methods that provide the impression of functioning more effectively than other methods.

What is Contrastive Learning?

Take any deep network, such as Resnet50, as an example of a function f that accepts input x and outputs features f. (x).

According to Contrastive Learning, if two inputs, x1 and x2, are both positive, then the functions f(x1) and f(x2) should be equivalent. However, when x3 is negative, f should not have the same results as f(x1) and f(x2) (x3). i.e., x1 x3 x2 f f(x2) Attract Repel f(x1) Repel f(x3)



The left- and right-hand cropping from the same image, two frames from the same video clip, two enhanced views (of which one is reversed horizontally), and so on are all examples of positive pairs. It's possible that one of the two negatives is a crop from another shot, a still from a movie, or an improved version of one of the photos that already exists. It wasn't until a paper written by DeepMind titled "Representation learning with contrastive predictive coding" [3] that the concept of "contrastive learning" was first presented to the public that the term was first coined. In the following section, we will demonstrate how the contrastive learning task laid the groundwork for understanding how to represent images in the most effective manner.

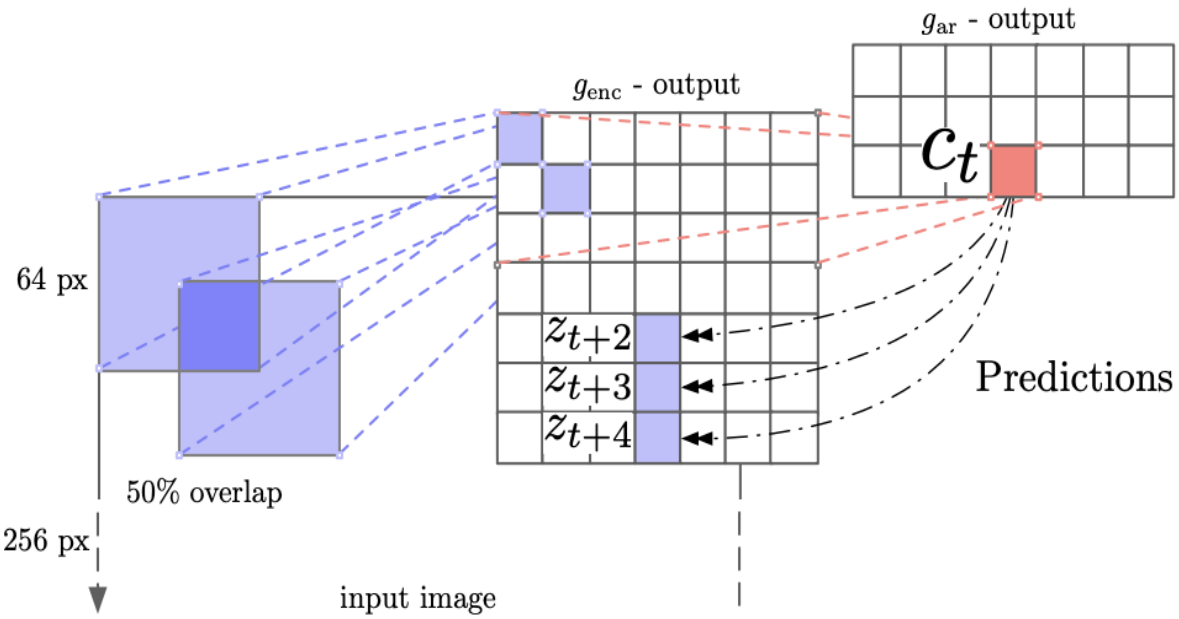
Contrastive Predictive Coding(CPC)

CPC is based on the idea of initially splitting the entire image into a rough grid. This is the first step in the process. The following step is to make educated guesses about the image's bottom

rows by referring to the rows that came before them. Because in order to complete the task, the model needs to have an understanding of the three-dimensional structure of the item that is displayed in the image. It is expected that the model will be able to deduce some characteristics from the data, such as the fact that a dog has four legs based solely on its face characteristics. Now that we have this representation in hand, we should find it much easier to complete the tasks that come after this one.

The entire supplementary task might be broken down into three steps.

Create a 7-by-7 grid with cells that are 64 pixels and spaces that are 32 pixels between each cell in an image that is 256 pixels by 256 pixels. To convert each grid cell into a vector with 1024 dimensions, you can make use of an encoder model (g enc) such as Resnet-50. The entirety of the picture has been converted into a tensor that has 7x7x1024 dimensions. If you are familiar with the first three rows of the modified grid (7x7x1024), you should be able to deduce the positions of the following three rows on your own (i.e., 3x3x1024 tensor). To make an accurate prediction of the bottom three rows, an auto-regressive generative model g-ar, similar to Pixel CNN, is utilized. Because Pixel CNN deserves its own article, it won't be discussed in this one because it doesn't fit the criteria. Pixel CNN builds a context vector c t from the top three rows in order to make predictions about the bottom two values (z t+2, z t+3, and z t+4 in the figure below). The drawing below shows the task pictorially.



50% overlap 64 × 256 px 11 picture genc input output 1 gar - output -Ct 2t+2 Zt+3 2t+4 Predictions

This model requires a loss function in order for it to be trained properly. The purpose of the loss function is to ensure that positive pairings (predictions that are exact) and negative pairings (predictions that are incorrect) are similar (incorrect patch). For the purpose of calculating the amount of loss, the set X of N patches, which consists of N-1 negative samples and one positive sample, is utilized (correct path) (Devlin et al., 2018). The N-1 negatives are selected at random from all of the different parts of the same image that can be seen (except for the part that is supposed to be right), as well as from all of the different photos that are included in the batch. This calamity is referred to as the InfoNCE loss, and a picture of it may be found farther down on this page

$$\mathcal{L}_{q,k^+,\{k^-\}} = -\log \frac{\exp(q \cdot k^+/\tau)}{\exp(q \cdot k^+/\tau) + \sum_{k^-} \exp(q \cdot k^-/\tau)}$$

.

In this instance, q denotes the prediction that was generated by the network, k+ indicates the patch that ought to be applied, and k- indicates a collection of N-1 patches that ought not to be applied. The "representation space" that is produced by g enc is distinct from the "original image space" in the sense that it contains the values k+, k-, and q, none of which are present in the original image space. This is because the "representation space" is produced by g enc.

Because of the formulation, the term "shortcut" is an abbreviation for the phrase "log SoftMax function." If you want to determine the degree of proximity that exists between two different things, use the dot product. To achieve this goal, one must first compute the log of SoftMax for the similarity score between the positive sample and the prediction q, then take the dot product of all N samples with the prediction q, and finally take the dot product of that with the positive sample. In particular, the first thing that needs to be done is to compute the log of SoftMax for the similarity score between the positive sample and the prediction q.

Utilizing linear assessment as a method for evaluation makes it possible to determine how complicated the representations that CPC has learned really are. The output of the frozen encoder model, which is referred to as g enc, is utilized to help in the training of linear classifiers by utilizing the Imagenet dataset as a source of data. After that has been completed, the linear classifier is tested on the Imagenet Val/Test dataset to determine how well it operates. The g-enc model is not actually constructed until much later, after the linear classifier has already undergone training (Raffel, et., 2020). Due to the fact that they have a top-1 accuracy of 48.7%, CPC representations are superior to other classification methods that have been used in the past. Classification performance of linear methods using ImageNet, with the top one percent achieving an accuracy of more than 95%. recognized the significance of the CPC signs in the text that was presented further down the page.

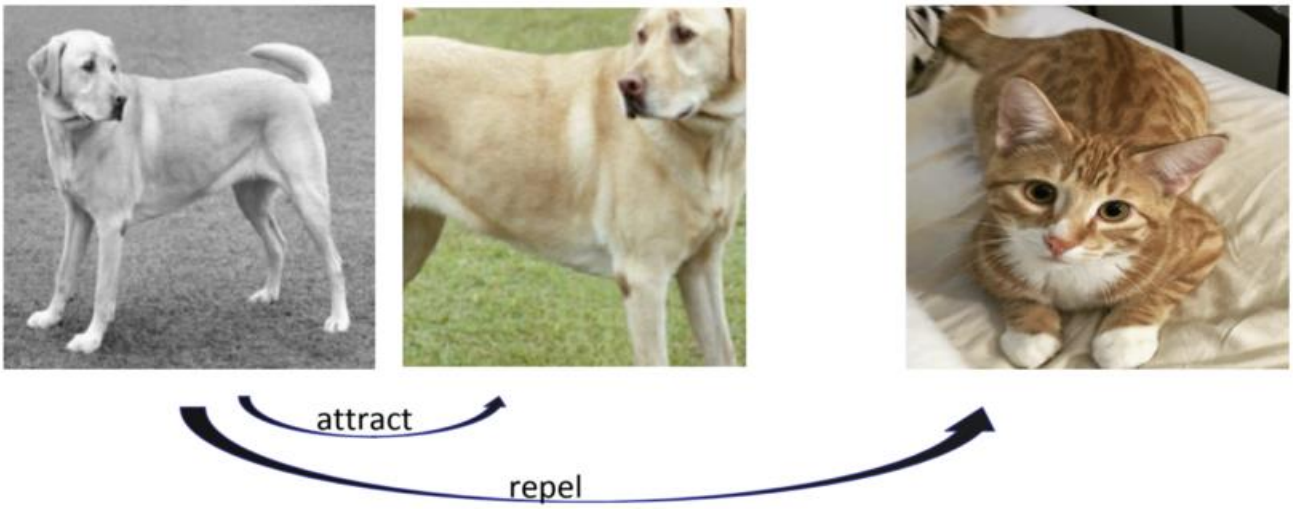
Motion Segmentation [36]	27.6
Exemplar [36]	31.5
Relative Position [36]	36.2
Colorization [36]	39.6
CPC	**48.7**

Although CPC outperformed other unsupervised learning methods for representation learning, its classification performance lagged far behind that of its supervised analogue (top-1 accuracy for Resnet-50 with 100% labels on ImageNet is 76.5%). We were able to make the process of self-supervised learning very similar to the process of supervised learning by applying the concept of image crop discrimination to the process of instance discrimination. This allowed us to make the two processes of learning very similar to one another.

Instance Discrimination Methods

In order to achieve the highest level of precision, the Instance Discrimination method employs contrastive learning throughout the entirety of the image instance. It is necessary for the Instance Discrimination technique that two augmented copies of the same image (referred to as a "positive pair") have the same representations (Chen et al., 2020). On the other hand, it is necessary for two augmented versions of different images (referred to as a "negative pair") to have different

representations.



There are two papers. Instance discrimination was a shared concept between MoCo and SimCLR about the same period. The primary focus is on maintaining the learnt representations even when specific changes are made to the images. A few examples of image manipulation techniques are the horizontal flip, the random crop to a specific size, the color channel swap, the gaussian blur, and so on. By all appearances, these alterations only affect the image's class and not the image itself. An image of a cat, for instance, would still be recognizable as a cat even if you flipped it and cropped it, so their representations shouldn't shift either.

The overall process can be broken down into the following steps:

i. Given an image, produce two new images that are randomly improved variants of the original image (x1, x2) A random selection of an improved version of any other image in the dataset is used to generate N-2 new negative instances. These new negative instances are then discarded.

ii. To get the representations, use the encoder model (g enc) on this pair of images. Get the representations of the N-2 negatives as well.

iii) Making sure that pairs with positive values are comparable, and pairs with negative values are dissimilar, by employing the InfoNCE loss algorithm, which is comparable to the CPC algorithm. SimCLR and MoCo have very different approaches, however, when it comes to the management of negative samples.

SimCLR

SimCLR has incorrectly recognized as examples of negatives each and every one of the images that are included in the current batch. When the linear assessment approach described in the CPC subsection is used to evaluate the accuracy of the SimCLR representations trained in this manner, the results obtain 69.3% accuracy on Imagenet.

Method	Architecture	Param (M)	Top 1	Top 5
Methods using ResNet-50:				
Local Agg.	ResNet-50	24	60.2	-
MoCo	ResNet-50	24	60.6	-
PIRL	ResNet-50	24	63.6	-
CPC v2	ResNet-50	24	63.8	85.3
SimCLR (ours)	ResNet-50	24	**69.3**	**89.0**

The results obtained by running Imagenet through a linear classifier referred to as SimCLR[5].

For the purpose of producing a correct calculation of the loss term, the performance of InfoNCE loss is evaluated based on the number of negatives, and this evaluation needs to take into account a sizeable number of negatives. Training simCLR with a large number of batches (up to 8,000) is computationally expensive, and for the greatest results, it requires training on a large number of GPUs. One could argue that this is the single most significant disadvantage of employing simCLR

On the other hand, Momentum Contrast (MoCo) maintains a separate negative buffer with a capacity of up to 8k and employs this buffer when calculating the InfoNCE loss. Because of this, they are able to instruct MoCo in a more precise manner in smaller groups.

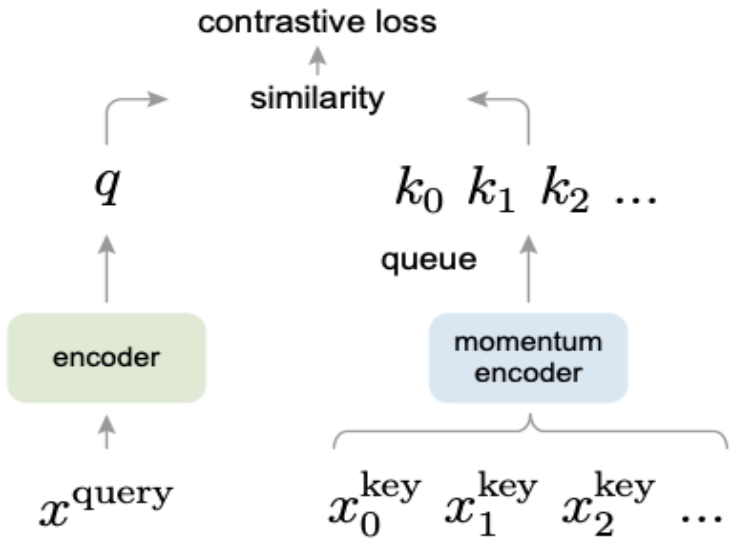
A buffer of a consistent size is used by MoCo to store all of the most recent mini-batch negatives (shown in the graphic below as x 0, x 1, x 2). A momentum encoder (_k) with the same design as the encoder (_q) is used, but the weights are gradually pushed towards the encoder. This is done so that the performance of the encoder can be enhanced (shown below in the image). The technique for updating the momentum encoder is outlined in the following text (4)

$$\theta_{\mathrm{k}} \leftarrow m\theta_{\mathrm{k}} + (1-m)\theta_{\mathrm{q}}.$$

The sole task of a momentum encoder is to construct representations out of negative samples (k 0, k 1, k 2, etc. in the picture below). Because the weights in a momentum encoder are not modified through the use of backpropagation, it is possible to keep a sizeable negatives buffer in memory with just a little effect on the performance of the system.

The initial input, x question, is sent to the encoder, and it produces the representations q. These representations are then compared to an enhanced version of the original image, x inquiry, as well as the N negatives that are produced by the momentum encoder (which are not shown in the image below) (Grill et al., 2020). After that, the loss from InfoNCE is applied to the CPC section so that the loss term can be calculated.

A table with the settings for the queue momentum encoder key xo x 1 x2 and the contrastive loss similarity ko k1 k2 of the q encoder can be found below.



There is a possibility that the readers of this post will not be persuaded of the significance of momentum encoders and MoCo applications by the information that has been provided. If you would like to read the paper in its original form, you can access it here. This idea is so vast that I've only just begun to scratch the surface.

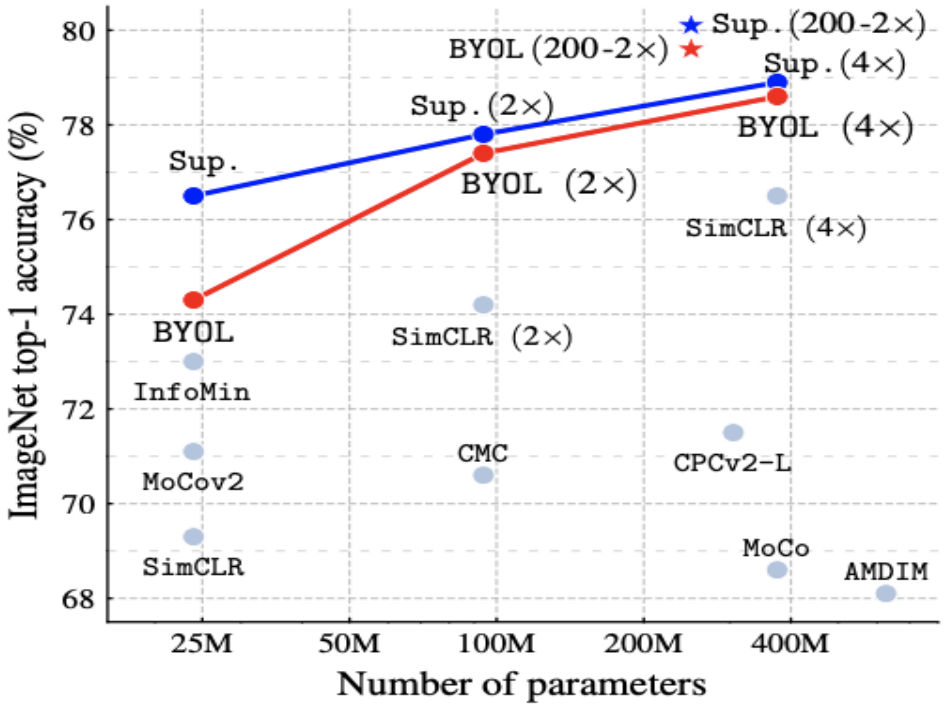
MoCo's second attempt at using linear evaluation methods achieved 71.1% accuracy when applied to ImageNet. This model achieved a higher level of accuracy (77.1%) when compared to the Resnet-50 model that was trained with human supervision. This was demonstrated by the classification of the Imagenet data by the MoCo linear classifier.

case	unsup. pre-train MLP	aug+	cos	epochs	batch	ImageNet acc.
MoCo v1 [6]				200	256	60.6
SimCLR [2]	✓	✓	✓	200	256	61.9
SimCLR [2]	✓	✓	✓	200	8192	66.6
MoCo v2	✓	✓	✓	200	256	**67.5**
*results of **longer** unsupervised training follow:*						
SimCLR [2]	✓	✓	✓	1000	4096	69.3
MoCo v2	✓	✓	✓	800	256	**71.1**

Table 2. **MoCo *vs*. SimCLR**: ImageNet linear classifier accuracy

Bootstrap your own Latent(BYOL)

MoCo's reliance on negative samples slowed down the process, even though the company had good results. BYOL[7] was just introduced, and it showed that even without negatives, stronger visual representations can be learned by using two networks similar to (MoCo Oord, Li,& Vinyals, 2018). With Resnet50 and a linear assessment procedure, their method gets 74.3% top-1 classification accuracy on ImageNet. This connects our method, which also uses a linear assessment procedure, to their unsupervised method, which uses more expansive and in-depth ResNets. Our method utilizes a linear assessment procedure. The pictures illustrate the occurrences that are described in the following paragraphs.

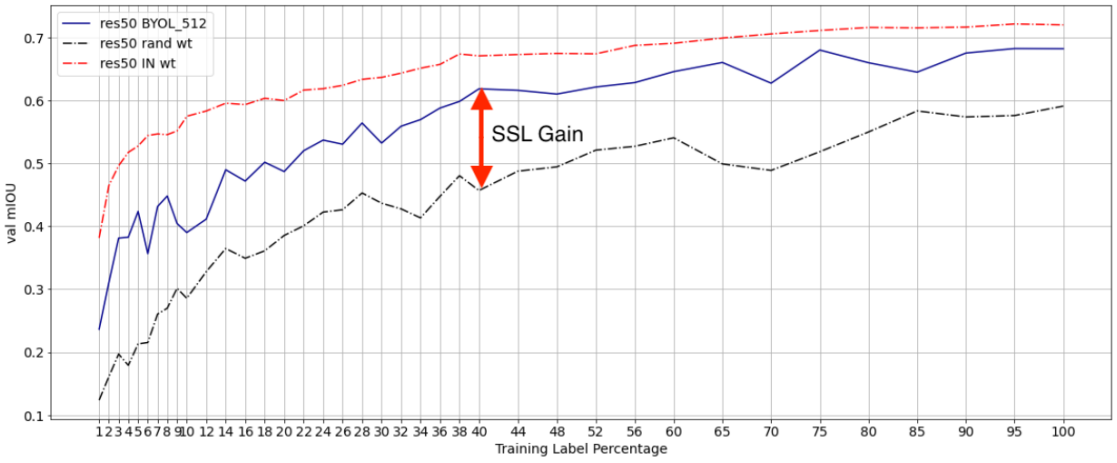


BYOL works best in dense prediction problems where only a few labels are available because the process of labeling data is hard and expensive. When BYOL is used for one of these tasks, semantic segmentation using the cityscapes dataset, FCN8s network, and Resnet50

backbone, it does better than the network trained from scratch, i.e., with random weights. Using the cityscape dataset, the graph below compares how well three important networks work.

1. Resnet50 trained from ImageNet weights and fine-tuned using 3k cityscapes labelled images(dotted red line).
2. Resnet50 trained from random weights using 3k cityscapes images only(dotted black line).
3. Resnet50 pre-trained on BYOL using 20k unlabeled cityscapes images, then fine-tuned using 3k cityscapes image(solid blue line).

The graph below demonstrates how helpful BYOL is in terms of learning relevant representations for the task at hand. In addition, the availability of large amounts of unlabeled data for unsupervised training suggests that it should be considered as a pre-training phase for other computer vision industrial applications where ImageNet weights cannot be utilized due to license limitations.



Implementation Details

In order to generate superior photographs, the following upgrades are applied to them. After the initial step of picking an area of the picture at random, the picture is cropped and then its dimensions are shrunk to 224 by 224 pixels. Following that, the image simultaneously undergoes a random horizontal flip, a random change in color, and a random transition from color to grayscale. The appearance of the colors of an image can be affected if the brightness, contrast, saturation, or hue settings are changed (Devlin et al., 2020). This can be done in any order that you like. The BYOL augmentation method is included in the following PyTorch code sample, which also has an integrated version of the function.

Resnet50 is frequently deployed as the encoder network in real-world BYOL deployments. [Case in point:] [Case in point:] Through the use of the Batch norm and the ReLU non-linear activation, the 2048-dimensional feature vector is transformed into a 4096-dimensional vector space. The dimensions are then decreased to 256 in order to make room for the projection MLP after that. The prediction network makes use of the same architectural foundation as the original network (Oord, Li, & Vinyals, 2018). The PyTorch code that follows implements the Resnet50-based BYOL network. However, it may be used with any encoder network, like as VGG, InceptionNet, and so on, without requiring any adjustments to be made.

References

Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020, November). A simple framework for contrastive learning of visual representations. In International conference on machine learning (pp. 1597-1607). PMLR. http://proceedings.mlr.press/v119/chen20j.html

Chen, X., Fan, H., Girshick, R., & He, K. (2020). Improved baselines with momentum contrastive learning. arXiv preprint arXiv:2003.04297. https://arxiv.org/abs/2003.04297

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805. https://arxiv.org/abs/1810.04805

Grill, J. B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., ... & Valko, M. (2020). Bootstrap your own latent-a new approach to self-supervised learning. Advances in neural information processing systems, 33, 21271-21284.https://proceedings.neurips.cc/paper/2020/hash/f3ada80d5c4ee70142b17b8192b2958e-Abstract.html

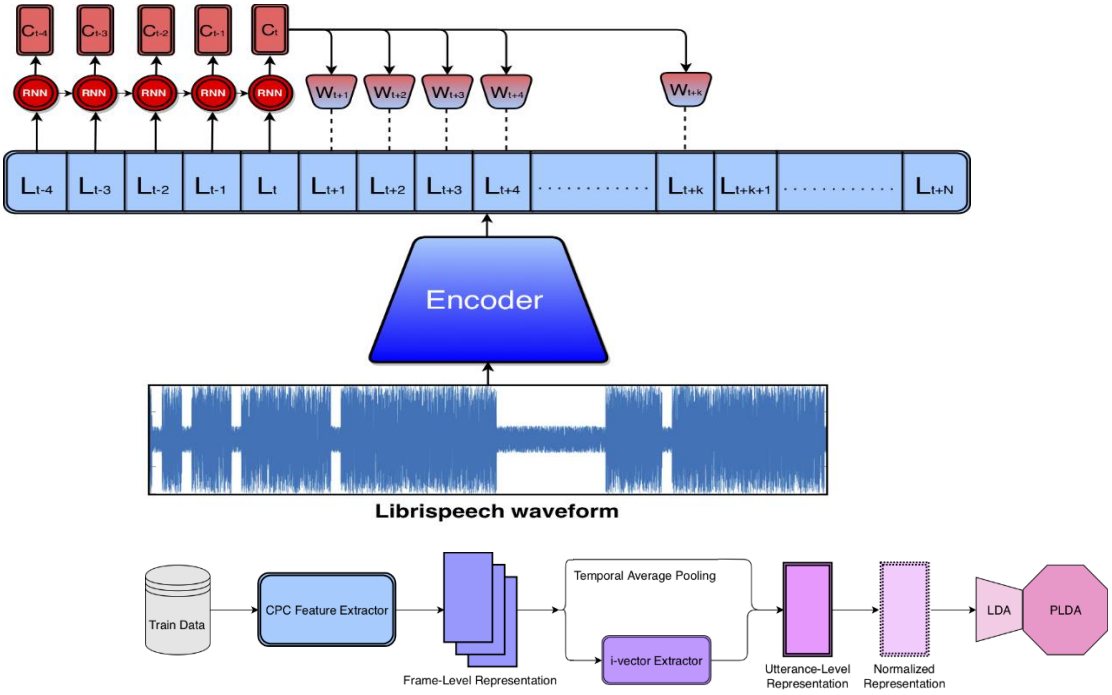
He, K., Fan, H., Wu, Y., Xie, S., & Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 9729-9738). http://openaccess.thecvf.com/content_CVPR_2020/html/He_Momentum_Contrast_for_Unsupervised_Visual_Representation_Learning_CVPR_2020_paper.html

Oord, A. V. D., Li, Y., & Vinyals, O. (2018). Representation learning with contrastive predictive coding. arXiv preprint arXiv:1807.03748. https://arxiv.org/abs/1807.03748

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res., 21(140), 1-67. https://www.jmlr.org/papers/volume21/20-074/20-074.pdf?ref=https://githubhelp.com

This repository contains the pertinent (PyTorch) code that can be used to reproduce the major results, which are as follows:

Automatic Speaker Verification Using Contrastive Predictive Coding's Contrastive Predictive Coding-Based Feature Representation Learning Using Contrastive Predictive Coding's



In order for you to recreate the primary results on your own, I have provided the following bit of code, which makes use of PyTorch to carry out the BYOL augmentation operation.

torch-vision is where it all started. tfmsbyol tfms = tfms is the form in which the transformations are introduced. We use the function compose([tfms.RandomResizedCrop(size=512, scale=(0.3, 1)), tfms.RandomResizedCrop(size=512, scale=(0.3, 1)) to generate a random image. RandomHorizontalFlip(), and thanks for using tfms. ToPILImage(), thank you very much. RandomApply([tfms.ColorJitter(0.4, 0.4, 0.1), p=0.8), tfms.Random([tfms.ColorJitter(0.4, 0.4, 0.1), p=0.8)])

within the orch) code of this repository: Learning of Representations for Automatic Speaker Verification Through the Application of Contrastive Predictive Coding