Semi-Supervised Object Detection

Matteo Rossi [1] **David Trakhtenberg** [2] **Guilherme Albertini** [2] **Huyen Nguyen** [2]

Abstract

In this paper, we present a semi-supervised model for object detection tasks. The goal is to accurately predict bounding boxes and their labels for objects inside images. Our dataset comprises 512,000 unlabeled 224×224 images and 50,000 labeled images of various sizes that we split 60:40 for training and validation purposes. The number of objects we train our model to recognize is 100. We approach the task by training a self-supervised model to learn visual representations from the unlabeled dataset and feed this trained model as a backbone to a downstream task that outputs bounding boxes and labels. We then fine-tune the whole model in a supervised fashion with the labeled dataset. We utilize SimCLR (Chen et al., 2020), a contrastive technique, as the framework to train the ResNet-18 backbone and Faster R-CNN (Ren et al., 2015) for the downstream object detection task. Overall, we achieved an Average Precision (AP) - Intersection of Union (IoU)=0.50:0.95 - of 0.093 on the validation set after training for ten epochs.

1. Introduction

There are many use cases where traditional ML techniques aren't enough—where the input space is highly dimensional, the data is unlabeled, or the training set is large. In these cases, self-supervised methods like energy-based models (EBM) prove to be incredibly useful. Namely, when dealing with unlabeled data, we are unable to classify an input to an output; however, EBMs introduce the concept of compatibility between an input and an output. Of which, there are two main classes of learning methods: contrastive and architectural (LeCun, 2020).

With contrastive learning, we push down on the energy of training data points while pulling up on the energy of the points outside the training data manifold. Within the manifold, where we push down, we define a positive pair as an (x, y) where x is an image and y is a transformation of x that preserves its information. Outside the manifold, we define a negative pair as any two images that have different information such as different labels so that the EBM can pull up on these examples.

For our purposes, we trained on 512,000 unlabeled images and 30,000 labeled images. To perform self-supervised learning on the unlabeled set, we performed data augmentations, used a CNN to learn feature representations from the transformed images, mapped those representations to a projection head, and finally maximized the similarity between positive pairs and minimized the negative pairs. These representations were then trained on a supervised task using Faster R-CNN.

2. Literature Review

2.1. Self-supervised Learning

Self-supervised and semi-supervised learning techniques for computer vision applications have become increasingly relevant in recent years. Supervised learning requires a significant amount of labeled data making it impractical due to cost and time constraints. In contrast, unlabeled data is readily available and abundant, which incentives the development of alternative learning methods. Novel techniques have emerged that try to learn useful visual representations from unlabeled data to improve the performance of downstream supervised tasks. Recent self-supervised learning techniques train models to learn invariant features by considering pairs of different versions of the same images obtained through image transformations. Models like MoCo (He et al., 2019) adopt this approach and have proven successful as they outperformed other prominent supervised models.

2.2. Object Detection

Object detection tasks consist in identifying objects within images or videos. A common approach is to use a bounding box to localize the object. The state-of-the-art techniques used in object detection make trade-offs between speed and

*Equal contribution [1]Courant Institute of Mathematical Sciences, University of New York, New York, U.S.A. [2]Department of Data Science, University of New York, New York, U.S.A.. Correspondence to: Alfredo Canziani <canziani@nyu.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

accuracy. YOLO (Redmon et al., 2015), which prioritizes speed, evaluates the images only once and employs techniques like residual layers and IoU to calculate the most appropriate bounding box. RetinaNet by Lin et al. also follows this approach and has demonstrated strong performance. Region-based Convolutional Neural Networks (R-CNN), introduced by Girshick et al., on the other hand, prioritize accuracy. The model searches for important regional proposals with the help of a selective search algorithm. After the search, the model extracts the critical features using a pre-trained convolutional neural network. The final step is to predict the bounding boxes and assign a label.

3. Methodology

3.1. Algorithm

Our adapted model is based on the Faster R-CNN network, which is an extension of the Fast R-CNN object detection model (Girshick, 2015), whereby the authors introduce a region proposal network (RPN) with a fully convolutional network that generates proposals with various scales and aspect ratios. This modification implements the so-called neural network with attention to guide the object detection search (developed by Fast R-CNN). The authors introduce the concept of "anchor boxes" instead of using image pyramids or pyramids of filters (i.e. multiple instances of a scaled image or multiple filters with different sizes) which are reference boxes of specific scales and aspect ratios. The architecture then moves to its second stage after the RPN phase. This is the aforementioned Fast R-CNN scheme for detecting objects in the proposed regions. Crucially, the computations in Faster R-CNN are shared across the RPN to reduce computational time. The reference anchors are given "objectness scores" and the region proposal is filtered based on this to finalize bounding boxes as "object proposals". After the regions are identified, feature vectors are extracted from each region using an ROI Pooling layer. This pooling layer takes each region, divides each into a fixed number of windows, and then performs max-pooling over the windows. Finally, the extracted feature vectors are classified using Fast R-CNN. The class scores are predicted by a Softmax layer and the bounding boxes are by a fully connected layer.

The chosen backbone model for RPN is SimCLR, which uses the contrastive learning method. The idea of SimCLR is to maximize the agreement between different augmented versions of the same image in order to learn the invariant visual representation. Each input image is transformed randomly to become a pair with distinct views, which is labeled as a positive example. Augmentation includes resizing, horizontal flipping, color distortion, etc. The base encoder is derived from the ResNet model; Lin et al. cite the ability to apply FPN on a single-scale feature map by assigning RoIs of different scales to the pyramid levels. Feature representation vectors from the input images are extracted and then passed through a projection head. This layer maps the representation to the space where NT-Xent loss, a normalized temperature-scaled cross-entropy loss, is applied. This contrastive loss pushes the model to minimize the distance between positive pairs and maximize the distance between negative pairs, which are different input images. In the paper, the authors also use the LARS optimizer, however, we opt to use the Adam optimizer.

3.2. Training

Our training strategy stemmed from the assumption that self-supervised learning benefits from bigger models (Chen et al., 2020), thus we started off by training a ResNet-50 backbone with a non-linear projection head as detailed in the SimCLR paper (Chen et al., 2020). This approach, however, soon turned out to be impractical due to resource constraints. The model would take too long to converge, and furthermore we were restricted to using a small batch size of 16 as the model and data wouldn't fit into GPU memory. The SimCLR paper doesn't recommend this practice since the model is exposed to fewer negative examples, thus inhibiting its contrastive convergence. It is also worth mentioning the disparity in hardware between our environment and the one used in the SimCLR paper, which employs 128 TPU v3 cores and allows for a batch size of 4096. This difference made it virtually impossible to replicate the results. For these reasons, we pivoted to using Pytorch Lightning (Falcon et al., 2019), as it allowed us to speed up training by parallelizing to 4 GPUs, and opted for a smaller ResNet-18 as the backbone but with the same projection head as before. We also employed the same image transformations (Chen et al., 2020), which are the ones that showed the most promising results in the SimCLR paper.

After settling on the architecture of the backbone, we trained the model with four different sets of hyperparameters for two epochs. The hyperparameters that provided the best performance were: batch size=128, learning rate=0.008, temperature=0.5, and optimzer=Adam. We used this setup to train the backbone for forty epochs. Figure 1 shows how the loss converges to a value of ~ 0.3 during training. The figure shows the loss starting from epoch 16 as the weights were loaded from a previous checkpoint due to time limitations in running jobs within the NYU HPC cluster. The initial contrastive loss at epoch 0 was ~ 4.5.

Once backbone training is complete, we discard the projection head and add a Feature Pyramid Network (FPN) (Lin et al., 2016) to the trained ResNet-18. We then use this architecture as the true backbone of the Faster-RCNN model. We employed various techniques to try to improve the performance of the supervised model. Below we list the most relevant and present a thorough analysis of their

performance in the Results section:

- Hyperparameter tuning (learning rate, optimizer, batch size, number of anchors, size of anchors).
- Transformations of labeled images.
- Freezing of ResNet-18 layers.

4. Results

4.1. Hyperparameter Tuning

Overall, the model that we found performed best consisted of training without freezing the backbone or applying image transformations. Regarding hyperparameter tuning, we noticed that using the Adam optimizer without decay provided better results than SGD or RMSProp. An interesting fact we observed, however, was that when using a learning rate of 0.01 we would encounter the phenomenon of exploding gradients. To remedy this issue we clipped the gradients, however, the resulting performance was sub-optimal. Reducing the learning rate to 0.008 resolved the problem and in the end, we settled for a learning rate of 0.005, as it provided a lower loss when training for longer epochs. The final batch size was two since the model wouldn't fit into memory when we experimented with a broader set of values. However, freezing some of the layers of the ResNet-18 would free up memory and allow us to test training with a batch size of four. Due to the large size of the model and images, training for one epoch took an average of ~ 1.5 hours which hindered our ability to prototype further. Finally, we tested different numbers of anchors with different sizes but we found the default Faster-RCNN parameters to be the optimal ones.

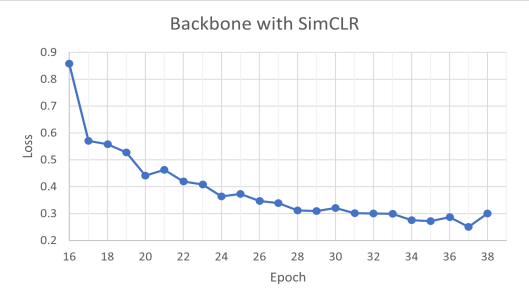

Figure 1. Backbone Loss over Training Epochs. The training was loaded from a previous checkpoint.

Figure 2 shows how the loss of our model decreases after each epoch. Due to time constraints, we were able to train for only 12 epochs, however, our results reveal significant room for improvement if the training steps were to be increased as the loss does not plateau. Furthermore, we run an evaluation script at every fifth epoch which allows us to estimate an average improvement of ~ 0.01 in the AP metric after each epoch. The score for the AP metric at epochs 0, 5, and 10 is 0.006, 0.063, and 0.093 respectively.

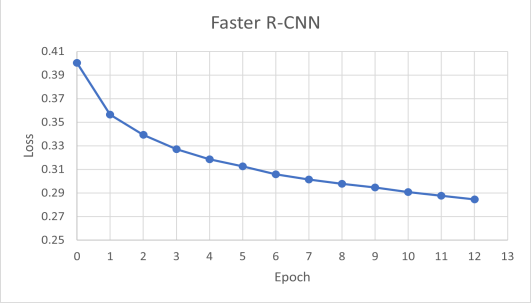

Figure 2. Faster R-CNN Loss over Training Epochs.

4.2. Image Transformations

As already mentioned, applying transformations to the labeled images and freezing the backbone layers proved unsuccessful. However, it is worth revealing our findings. We originally hypothesized that applying the appropriate image transformations would help the model avoid overfitting the training set. We utilized the Albumentations library (A. Buslaev & Kalinin, 2018) to perform the transformations, which include a geometric resize of the minimum side to 224px, a random crop of size 224x224px, and a random horizontal flip. The images were also rescaled to 800x800px and normalized before being fed to the Faster-RCNN backbone. Although the loss decreases similarly to our final model in Figure 2 when we evaluated it on non-transformed images after training for 12 epochs, the AP metric was 0.07. All the other hyperparameters were held constant. These results suggest that the transformations reduced the information content of the images and in turn, the model wasn't able to learn the same meaningful representations.

4.3. Freezing Backbone Layers

We compared the performance of our final model to a model with the same architecture but with a ResNet-18 backbone pre-trained on the COCO dataset (Lin et al., 2014). Overall, the performance after 10 epochs was similar, which indicated that our backbone was reasonably well-trained. An inevitable conclusion was thus to freeze some of the layers of our backbone and observe the performance. We trained in parallel two versions of the model, one with three frozen layers ["layer1", "conv1", "bn1"] and the other with all frozen layers ["layer4", "layer3", "layer2", "layer1", "conv1", "bn1"] of the ResNet-18 backbone. However, after 10 epochs, both models significantly underperformed our earlier results evaluating an AP metric of 0.32 and 0.27, respectively. The results suggest the superiority of the fine-tuning approach.

Figure 3. Redundant classification of a cat.

Figure 4. Correctly classified example of a ladybug.

5. Conclusion

We should mention that we did not attempt to tune other hyperparameters besides the ones already mentioned due to time constraints. However, further investigations should focus on adjusting the strides of the CNN and experimenting with different thresholds to generate a more precise number and size for the bounding boxes. Figures 3 and 4 show that our model outputs fairly accurate bounding boxes but produces too many redundant ones. Thus, a further improvement would be to take the top ROIs for each image, sort them by confidence scores, and remove those below a certain threshold. Furthermore, some models from our colleagues that involved only supervised methods outperformed hybrid approaches like ours. We believe this can also be attributed to a correct implementation of the data augmentation pipeline, which we lack and would certainly allow for better performance. Finally, our model is limited by the number of epochs trained and an inevitable next step would be to train for a longer period of time.

References

A. Buslaev, A. Parinov, E. K. V. I. I. and Kalinin, A. A. Albumentations: fast and flexible image augmentations. *ArXiv e-prints*, 2018.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. E. A simple framework for contrastive learning of visual representations. *CoRR*, abs/2002.05709, 2020. URL https://arxiv.org/abs/2002.05709.

Falcon et al., W. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

Girshick, R. B. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. URL http://arxiv.org/abs/1504.08083.

Girshick, R. B., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. URL http://arxiv.org/abs/1311.2524.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. Momentum contrast for unsupervised visual representation learning, 2019. URL https://arxiv.org/abs/1911.05722.

LeCun, Y. Energy-based models, Mar 2020. URL https://atcold.github.io/pytorch-Deep-Learning/en/week07/07-1/.

Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. URL http://arxiv.org/abs/1405.0312.

Lin, T., Dollár, P., Girshick, R. B., He, K., Hariharan, B., and Belongie, S. J. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016. URL http://arxiv.org/abs/1612.03144.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection, 2017. URL https://arxiv.org/abs/1708.02002.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection, 2015. URL https://arxiv.org/abs/1506.02640.

Ren, S., He, K., Girshick, R. B., and Sun, J. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. URL http://arxiv.org/abs/1506.01497.