Simple Siamese Representation Learning with RGB-D Images

Bryan Zhu
Department of Computer Science
Stanford University
bwzhu@stanford.edu

Abstract

Siamese neural networks have become a common mechanism for self-supervised representation learning on images in the past few years. Inspired by the simplicity and strong performance of these models, especially SimSiam, we experiment with using Siamese network structures on representation learning for 3-dimensional data in the form of RGB-D images. While our initial results find that our model does not perform as well as the baseline SimSiam, it achieves comparable performance, and we identify a few areas which could lead to model improvement.

1. Introduction

Learning good image representations is an important task in modern computer vision because it allows for better performance on a wide variety of downstream tasks. One class of self-supervised representation learning methods, known as contrastive learning, seeks to generate representations that minimize the distance between augmented views of the same object ("positive pairs") while maximizing the distance between views of different objects ("negative pairs"). Recently, various methods using Siamese neural networks have been developed to do this, such as SimCLR [1], BYOL [4], and SimSiam [2]. SimSiam in particular is interesting because it manages to achieve good comparable performance while being simpler than the other variants: SimCLR requires training on a large number of negative pairs while SimSiam doesn't require any, and BYOL uses an additional "momentum encoder" component which SimSiam doesn't need.

The Siamese network structure in general consists of one encoder model which runs on two different inputs and then compares the two outputs for similarity or dissimilarity. In the case of positive pairs, SimSiam takes one image, augments it with transforms in two different ways, and runs the two through its encoder. One of the encoded representations is passed through a predictor whose output is then compared to the other representation for similarity. The idea is that

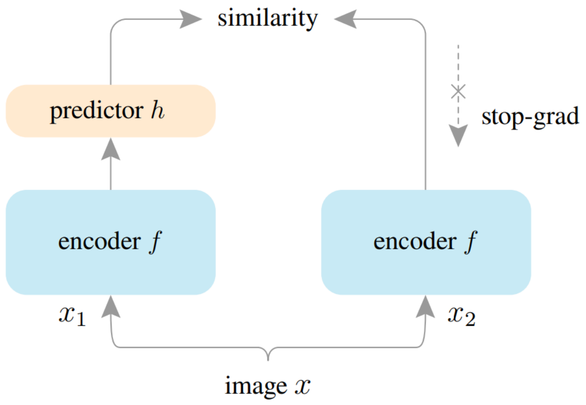


Figure 1: SimSiam architecture, an example of a Siamese neural network. Figure taken from [2].

even though the images look slightly different because of the augmentations, they contain the same semantic content so their representations should be similar. Without negative pairs, one issue with this structure is collapse to a constant: similarity can be maximized if all inputs produce the same encoding. However, Chen and He show that with a particular method of gradient backpropagation during training, known as stop-gradient, degenerate solutions like that are not produced.

This project aims to investigate whether SimSiam can produce even better representations when trained on images with additional 3-dimensional information through the use of RGB-D images. RGB-D images are images which, in addition to the standard three color channels, adds a fourth depth channel that encodes how far away each pixel was from the camera. Gupta et al. show that using the depth channel, one can calculate a 3-channel image that encodes at each pixel the horizontal disparity, the height above a ground plane, and the angle the pixel's local surface normal makes with the inferred gravity direction [6]. This is referred to as an HHA image. Convolutional neural networks which are designed for and pretrained on RGB images can be finetuned on HHA images, allowing us to use pre-existing structures on RGB-D data.

We find that in our experiments, representations created with RGB-D images using the HHA conversion did not perform as well on our downstream task of object classification, although their performance is not too far below that of representations created from standard RGB images. We hypothesize that this could have been caused by errors in the HHA images from where the depth map was missing data, and overfitting on the artifacts caused by those errors.

2. Related Work

Most work relating to image representation learning falls into one of two categories: generative and discriminative. Generative learning generally aims to learn image distributions through data and then generate images directly in pixel space, usually through autoencoding [9] or adversarial training [3]. However, full pixel-level image generation can be computationally expensive and this level of detail is often not necessary for representation learning.

Discriminative learning on the other hand has models learn a task using objective functions similar to those in supervised learning, often a classification or discrimination task, except that both the inputs and labels are obtained from an unlabeled dataset. Contrastive learning is a common example of such methods.

2.1. Contrastive learning and Siamese networks

There have been many variations on using Siamese network structures for contrastive representation learning. Chen et al. [1] propose a simple framework for this known as SimCLR that employs both positive and negative pairs. They experiment with different types of input augmentations and show that a combination of cropping and color jitter performs by far the best. They also show that these network structures require large batch sizes during training to be most effective. Instead of using negative pairs directly, He et al. [7] store them in a queue and turn one branch of the Siamese structure into a momentum encoder which improves consistency, creating a system they call MoCo. BYOL [4] also uses the Siamese structure and a momentum encoder but have one branch of the network directly try to predict the output of the other branch, and manage to avoid collapsing to degenerate solutions without negative pairs.

The most recent and simplest version of this is SimSiam [2], which has been referred to as "SimCLR without negative pairs" and "BYOL without the momentum encoder". Using SimSiam, Chen and He show that none of these methods are needed to prevent degenerate solutions as long as a "stop-gradient" technique is used, where gradients are backpropagated through one branch of the Siamese network but not through the other. They hypothesize that the stop-gradient actually turns the network's algorithm into alternating between optimizing two separate sets of variables in the fashion of expectation-maximization (EM) algorithms, which helps prevent collapse. Due to SimSiam's simplicity and ease of training, this is the model that we base our methods on.

2.2. RGB-D images and deep learning

Gupta et al. [6] devise the approach where HHA images are constructed from the depth channel of RGB-D images and then used in CNNs pretrained on RGB images. They choose these channels to add because they require extra information about the world that they believe models are unlikely to learn by themselves. They show that these CNNs can still perform decently on object detection and image segmentation tasks when only provided the HHA image, and they perform better when given both RGB and HHA than with just the RGB. We hoped to continue these results on representation learning.

3. Approach

An overview of the SimSiam architecture is given in Figure 1. The encoder, which produces the representations of input images, consists of a CNN backbone followed by a projection head which is a 3-layer multilayer perceptron (MLP) network. During each step of training, a batch of images is augmented in two different ways and passed through this encoder. Then, the output of one side is passed through a predictor, which is a 2-layer MLP, and the output of that is compared to the direct output of the other encoder for similarity. The loss function we use is cosine similarity: given projection output u and encoder output v we compute

$$\mathcal{L} = -\frac{u \cdot v}{\|u\|_2 \|v\|_2} \tag{1}$$

for each representation and then average loss across the batch. The starter code we used for this model came from `https://github.com/leaderj1001/SimSiam`.

We modify the encoder to handle RGB-D images by simply laying two CNN backbones side by side, one which learns from the RGB image and the other which learns from the corresponding HHA image constructed from the depth channel. Given a RGB-D image, we construct the HHA image from its depth channel and feed (augmented versions of) the two images into the encoder's CNNs. The outputs are then concatenated and this vector passes through the projection head to become the output of the modified encoder. This structure is shown in Figure 2.

3.1. Image augmentation

The key to robust representation learning is a good choice of augmentations. Chen et al. [1] show that crop-

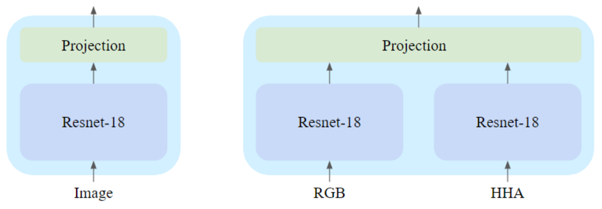


Figure 2: Comparison of SimSiam and SimSiam RGB-D encoders.

ping and color jitter are both almost absolutely necessary to prevent the network from learning spurious connections such as identifying objects solely by their color, so we follow their example in using these two transforms. In addition, we add a random chance to horizontal flip the image and a random chance to convert it to grayscale, which are both used in [2]. We take care that any random transforms applied to the RGB side of an image, such as the randomness in the choice of cropping window, must be applied in the same way to the HHA side.

3.2. Constructing HHA images

Recall that HHA stands for three channels:

- Horizontal disparity
- Height above ground plane
- Angle between surface normal and gravity direction

The horizontal disparity can be directly calculated from the depth channel, as disparity is directly proportional to depth. The other two values require the gravity direction $\mathbf{g}$ and the camera intrinsics K to calculate. In the general case, Gupta et al. [5] present an iterative algorithm for estimating $\mathbf{g}$ from a single RGB-D image by calculating the surface normals at each point and finding the direction that maximizes the number of points with normals parallel to that direction (corresponding to floor-like and tabletop-like surfaces) and perpendicular to that direction (corresponding to wall-like surfaces). It is important to note that this method was designed for indoor scenes and may not have the same accuracy elsewhere. In our case, the dataset we used consisted of images taken at specific angles above the horizon towards a turntable, so we directly knew the gravity direction of each image and did not need to estimate it. The camera intrinsic matrix was also provided to us from our dataset.

With these values, we calculate the second H and A channels as follows: First, using the camera intrinsics and the depth values D, we project the depth map into a 3D point cloud p representing the surface of our scene as follows:

$$p(x, y) = K^{-1}\begin{bmatrix} xD(x,y) \\ yD(x,y) \\ D(x,y) \end{bmatrix} \tag{2}$$

The ground planes are exactly those perpendicular to the gravity direction, so we take the dot product of the gravity direction with each point to calculate a relative height:

$$H(x,y) = \mathbf{g} \cdot p(x,y) \tag{3}$$

It is impossible to get any absolute height without more information, so we simply take the lowest value in our image to be ground height and scale the rest accordingly. The surface normals are calculated by taking the gradients in the row and column directions of the point cloud grid to obtain two tangent vectors at each point. The normal is then the cross product of these tangent vectors. We scale the normals to unit length and then calculate the angles between these and the gravity direction:

$$N(x,y) = \nabla_x p(x,y) \times \nabla_y p(x,y) \tag{4}$$

$$A(x,y) = \arccos\left(\mathbf{g} \cdot \frac{N(x,y)}{\|N(x,y)\|}\right) \tag{5}$$

Given disparity, relative height, and angles, we scale each channel to fit between 0 and 255 to match the format of RGB images. An example of a RGB image and its corresponding HHA image presented as an RGB can be seen in Figure 3. One can see that although color information is obviously lost, the HHA image captures the shape and contours of the object fairly well.

3.3. Model parameters

The specific parameters we use in our experiment are as follows. Most of them are the same as the ones Chen and He use in [2], although we reduced some of the sizes to be easier to train on the Google Cloud Platform VMs that were available to us.

- The CNN backbone structure we use for both RGB and HHA channels is ResNet-18 [8].
- The projection layer is a 3-layer MLP with ReLU and dropout. Its hidden layers have sizes of 2048.
- The prediction layer is a 2-layer MLP with ReLU and dropout. Its hidden layer has a size of 512. As its input and output have size 2048, this gives it a bottleneck structure.
- We use stochastic gradient descent for optimization with a learning rate of $lr\times$ BatchSize$/256$ for base learning rate $lr = 0.05$, and a momentum of 0.9. The learning rate uses a cosine decay schedule with weight decay 0.0001.

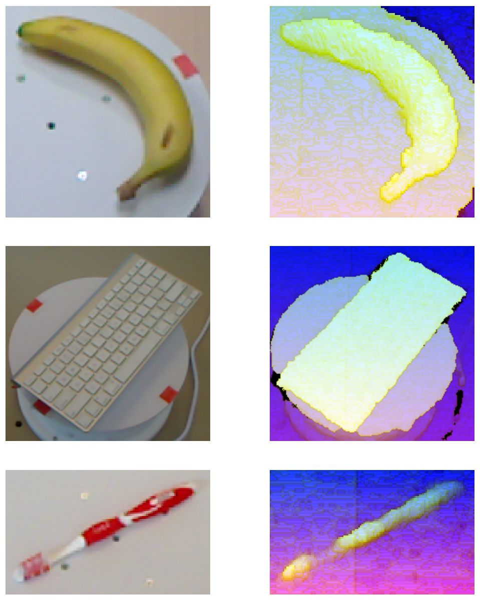


Figure 3: Example images from our dataset and their corresponding HHA images.

We pretrained the model using the Siamese structure on our training set, described below in section 4. Training ran for 100 epochs.

4. Experiments

4.1. Data

We trained our model with the Washington RGB-D Object Dataset [10], which consists of RGB-D images of 300 household objects arranged in 51 categories photographed on a turntable from various angles. Following Lai et al., we split the dataset into train/test sets by taking one instance of each category for the test set and using the rest for training. We sampled the turntable images by taking every fifth frame for each object and angle, leaving us with about 35000 images for training and 7000 for testing. To normalize for input into ResNet-18, we center cropped each image to a square image and rescaled to dimensions of 224×224.

4.2. Model evaluation

We compared three variants of our SimSiam model. The baseline model used the same structure as Chen and He's model, where we only fed in the RGB channels of each RGB-D image. The second was the modified model which used both RGB and HHA information.

The third variant of our experiment also used RGB and HHA information, but we only applied the spatial transformations (random crop and chance to horizontal flip) that we used on the RGB side to the HHA side, leaving out the color transformations (color jitter and random chance to grayscale). The reason for this experiment was because the HHA image is not truly a color image, but rather using color as a proxy to represent various facets of depth, and therefore we believed that color augmentation was distorting depth in ways that we did not intend. In addition, because the color profile of all the HHA images are similar (blue background, red towards the bottom, yellow-white object in the center), we believed that it was less likely that the model would falsely use HHA color as a proxy for object class.

After pre-training the models on the train set, the model's encoder was used to produce representations which are then evaluated on a downstream task. In this case, we used the representations on the task of object classification by training a supervised linear classifier to classify frozen representations of images in our training set and then evaluating it representations from our test set. We measured classification accuracy on our test set as a metric of representation performance.

4.3. Results

The downstream model accuracy on the evaluation task for our three model variants is given in Table 1.

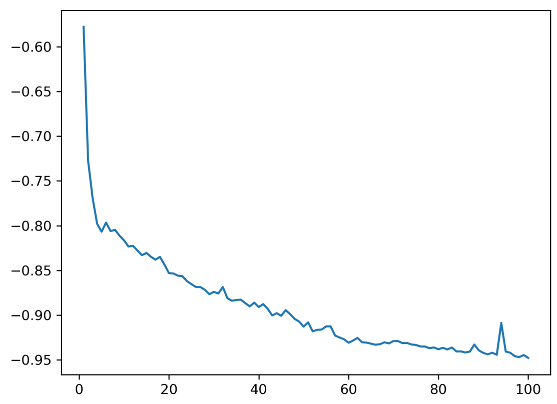
Model	Eval accuracy (%)
Baseline	70.62
RGB-D (all transforms)	66.38
RGB-D (HHA spatial only)	67.12

Table 1: Comparison of baseline and RGB-D models

For comparison, Chen and He [2] achieve accuracies of 66 to 68 percent (depending on the model parameters) on the 1000-class ImageNet training and validation sets, so our experiments result in similar results with a smaller dataset. However, we do notice that the RGB-D models did not perform as well as the baseline, and removing color transforms from the HHA encoder only improves the network slightly.

5. Analysis

We plot the training loss over time during pre-training for the baseline and the first RGB-D model in Figure 4. Looking at these graphs, we see that the loss for the RGB-D model dropped much faster, stabilizing at a loss of around -0.95 within fifty epochs. On the other hand, the baseline model didn't drop so sharply so quickly, but continued to slowly decrease in loss throughout the duration of training. (Note that in both cases, the stop-gradient technique did prevent collapse to degenerate solutions, which would look like the loss function dropping immediately to -1 and staying there. Thus we manage to reproduce this facet of Chen and



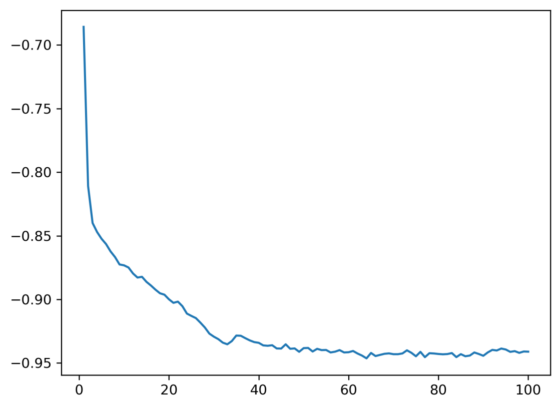


Figure 4: Training loss per epoch for representation learning. The top graph uses the baseline SimSiam model, the bottom uses SimSiam RBG-D.

He's work as well.) This, combined with the reduced performance on the evaluation set, suggests that there is some measure of overfitting happening in our RGB-D model.

One possible issue we could have had is missing depth data. Many of the images in the RGB-D Object Dataset had patches which were missing depth data, and they could be quite large in some cases. We decided to fill them in with black areas when reconstructing the HHA image, and the model could have attempted to learn some correlations based on these black patches which really shouldn't have any meaning. A few examples of these are shown in Figure 5.

We attempted to preprocess the depth maps by filling the missing areas in with simple interpolation; however, since most of the missing spots were on the borders between the object and the table, or worse, between the object and the far wall, the images resulting from this preprocessing did not turn out very well. Smarter machine learning techniques

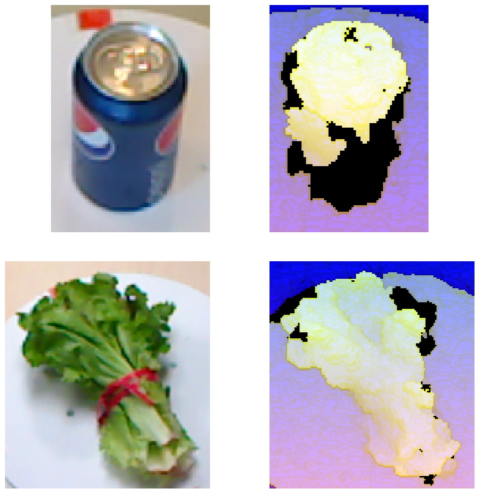


Figure 5: RGB images and their corresponding HHA images in a few cases where the depth channel was missing data.

would likely perform better, but that is beyond the scope of this project. If the depth maps could be filled in accurately, it would be interesting to experiment on the full unblemished RGB + HHA images.

Another issue is that we are likely adding too many parameters using our method. By placing two CNNs side by side, we basically double the number of parameters in our model, while there is not double the informational content going in: the HHA image adds clarity to the three-dimensional shape of the object, but many of the outline and edge filters that CNNs might want to take advantage of can already work on just the RGB side. Adding too many parameters can lead to overfitting. A better way to leverage depth data might be to use it or its corresponding point cloud map directly in a 3D CNN structure. For example, Zia et al. [11] have developed a kind of hybrid 2D/3D CNN designed for RGB-D data which might be more effective for this purpose.

One final factor which could have influenced our results is batch size during pretraining. Chen and He show in their experiments that SimSiam, while fairly robust to batch size and significantly more robust than methods like SimCLR that need large batches, is still somewhat affected. On their ImageNet data, lower batch sizes result in lowered performance on object classification by about two points. Because of memory constraints on available Google Cloud Platform resources, we were only able to use a batch size of 64, which is lower than recommended for SimSiam, and the smaller size could have disproportionately affected the larger RGB-D network.

6. Conclusion

In this project, we explored self-supervised representation learning with Siamese networks by modifying a Siamese network structure to work with 3-dimensional data in the form of RGB-D images. Our model achieves comparable performance to SimSiam on only RGB images, although some errors with overfitting and missing depth data prevent the model from achieving higher performance.

Future work in this area involves experimenting with different types of 3-dimensional CNN to see whether they can improve performance without the kind of overfitting we see here. In addition, it would be interesting to extend this functionality to tasks beyond object recognition and RGB-D images beyond object closeups (such as indoor or outdoor scenes), as the purpose of representation learning is to produce representations which are general and good for a variety of tasks. Finally, as we discussed monocular depth estimation in class, it would be interesting to see if we can leverage the outputs of monocular depth estimation to use RGB-D techniques on standard RGB data and compare how they perform.

References

[1] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations, 2020.

[2] X. Chen and K. He. Exploring simple siamese representation learning, 2020.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[4] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko. Bootstrap your own latent: A new approach to self-supervised learning, 2020.

[5] S. Gupta, P. Arbelaez, and J. Malik. Perceptual organization and recognition of indoor scenes from rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 564–571, 2013.

[6] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. Learning rich features from rgb-d images for object detection and segmentation, 2014.

[7] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning, 2020.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[9] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2014.

[10] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2011.

[11] S. Zia, B. Yuksel, D. Yuret, and Y. Yemez. Rgb-d object recognition using deep convolutional neural networks. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 896–903, 2017.

A. Appendix: Source Code

The source code for this project is located at `https://github.com/bwzhu352/cs231a-project`. It is a private repository, please contact Bryan at `bwzhu@stanford.edu` for access.