Self-Supervised Learning for Semantic Segmentation of Images

A thesis submitted to the
College of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Christopher Chamberlain

Abstract

Artificial Neural Networks (ANN) are powerful Machine Learning (ML) models that can help solve problems that are hard or even impossible to design solutions for by hand. These models learn to exploit information present in their target datasets to solve various problems. However, labelling data can be quite expensive, time-consuming and often requires a domain expert. Therefore it would be quite beneficial if one could train a model in such a way that exploits unlabeled data. Fortunately, Self-Supervised Learning (SSL) methods are a family of learning algorithms that attempt to do just that. Many SSL methods exist, but in this thesis, we explore Barlow Twins (BT) — a siamese network based on redundancy reduction, and Image Reconstruction (IR) — a method proposed in Karnam's thesis. In addition, we extend the Image Reconstruction method with both Coarse Cutout and Hide-and-Seek augmentations as they have been applied in similar supervised and weakly-supervised segmentation task scenarios. We apply these methods and investigate the results with the PASCAL VOC dataset.

Acknowledgements

I would like thank my supervisor Dr. Ian Stavness for the opportunity to undertake the Masters program and the continuous support over the years. I would also like to thank Jordan Ubbens, Danny Huang, Yuemin Wang, and Franklin Ogidi for being excellent discussion partners and helping straighten out my thoughts.

I would also like to acknowledge my dear, sweet Granny Lyla, who got multiple degrees before me, before the internet, all through remote correspondence. She is a wonderful woman and an excellent teacher.

Funding Notice

This research was undertaken thanks in part to funding from the Canada First Research Excellence Fund and Protein Industries Canada, Inc.

Contents

List of Tables

List of Figures

List of Abbreviations

ML	Machine Learning
ANN	Artifical Neural Networks
MLP	Multi-Layer Perceptron
CNN	Convolutional Neural Networks
SSL	Self-Supervised Learning
NLP	Natural Language Processing
AE	Auto-Encoder Network
LR	Learning Rate
SGD	Stochastic Gradient Descent
BT	Barlow Twins
IR	Image Reconstruction
HaS	Hide-and-Seek
IR+Orig	Image Reconstruction with the original augmentations
IR+HaS	Image Reconstruction with Hide-and-Seek augmentations
IR+CC	Image Reconstruction with Coarse Cutout augmentations
MSE	Mean Squared Error
MAE	Mean Absolute Error
SSIM	Structural Similarity Index Measure
MS-SSIM	Multi-Scale SSIM

1 Introduction

1.1 Motivation

Artificial Neural Networks (ANN) are powerful Machine Learning (ML) models that can help solve problems that are hard or even impossible to solve manually. One example of a challenging task is the task of semantic image segmentation (semantic image segmentation). This task requires the machine to output a per-pixel classification of relevant objects present in an input image. An everyday example of semantic image segmentation is background removal (i.e., green screen effect) in video conference software. In a commercial context, it may be used to help decision-making in a self-driving automobile. In a scientific context, semantic image segmentation may be used in a vision system to detect weeds and crops in agricultural images.

Neural Networks learn to perform these tasks by elucidating patterns of information contained in their training datasets. However, for ANNs to work effectively, a sufficient volume of data is necessary with respect to the problem's complexity. Traditionally, the training dataset requires human-provided labels to provide a supervisory signal from which the network can learn. However, human-generated labels can be prohibitively expensive for large datasets. Creating labels is particularly time-consuming for semantic image segmentation where labels contain pixel-level details and require a human annotator to precisely outline each object in an image. The cost of human-generated labels is compounded for task domains where identifying and outlining objects requires a domain-specific expert. For example, in medical imaging a trained Radiologist may be required to accurately annotate medical images. Likewise in the agricultural domain, trained plant scientists or plant pathologists may be needed to label plant organs or to identify lesions or symptoms specific to particular plant diseases. Because these skills are not commonplace, an expert annotator's time is often more expensive than their non-specialist counterpart.

Self-Supervised Learning (SSL) is a family of algorithms designed around the hopes of reducing human-provided labels (and thus the cost). Instead the algorithm operates upon a machine-provided supervisory signal that it creates itself.

1.2 Problem Definition

SSL is a recent and popular area of ML research. Therefore, there is a plethora of SSL papers in contemporary research, however, previous SSL papers largely focus on image classification rather than semantic image segmentation. Some papers do provide results for semantic segmentation but are often a secondary concern.

Image classification is a task that determines a category of the whole image — for example, a picture of a cat — whereas semantic image segmentation determines at a pixel level which category each pixel belongs to. The semantic image segmentation task is a harder challenge and thus affects the compatibility of the SSL pre-training with its downstream task.

Siamese Networks are a subset of SSL techniques that prescribe a network architecture that use multiple paths of information within the network to compose a supervisory signal. Examples include Barlow Twins (BT) [30], Simsiam [3], BYOL [9], MoCoV2 [2]. These example networks pre-train an encoder network by generating two augmented views of the input provided to the network to learn distinctions (or similarities) between the views. The resulting pre-trained encoder network can then be used for fine-tuning in a downstream task like semantic image segmentation.

Outside the realm of Siamese Networks, there are many other approaches for SSL. One important subset are reconstruction based approaches. These methods can be implemented across various domains. Conceptually, any data can have corruption (images, text, audio, etc) and removing that corrupted data is called reconstruction. A SSL approach can be derived from this by intentionally corrupting your data and repairing it in a controlled manner. The specific steps of corruption and reparation then affects the resulting qualities of pre-trained model. For example, in the context of Natural Language Processing (NLP), a common training procedure is to erase one or more words from a passage of text and then have the model either fill in the blanks or to learn in lieu of the missing words. Similarly, Image Reconstruction (IR) [12] is a SSL technique that learns by repairing intentionally removed pixels to pre-train for semantic image segmentation.

Most computer vision research is accompanied by public implementations and datasets for reproducibility reasons. However, some implementations are not available. We could not find a public implementation of IR, so as a contribution we will provide an open source implementation of the IR pipeline.

1.3 Proposed Solution

In this thesis, to address the problems describe above, we have chosen to evaluate and compare BT and IR methods in the context of semantic image segmentation. In addition, we further investigate the significance of the chosen corruption augmentations in the IR method by substituting them with alternative augmentations. Overall we will evaluate segmentation performance with 6 models: No Pre-Training, ImageNet Supervised Pre-Training, BT Pre-Training and 3 variations of IR Pre-Training. We use PASCAL VOC [7] dataset for training and evaluation. This dataset is also used for pre-training the BT and IR models.

1.4 Objectives

Our objectives for this thesis include:

1. Evaluate and compare IR and BT SSL methods for semantic image segmentation.

2. Evaluate and compare of 3 variations of the IR method for semantic image segmentation.
3. Provide an open source implementation of the IR method and its variations.

1.5 Thesis Structure

This thesis contains 5 chapters, including this introduction. Chapter 2 presents background knowledge and related work for semantic image segmentation. It includes a description of a ResNet based UNet (Section 2.2) as well as an overview of self-supervised learning with Siamese Networks (Section 2.3) and Image Reconstruction (Section 2.3.2). Chapter 3 presents the datasets and methodology used in our experiments. Chapter 4 presents and discusses the results of the experiments. Chapter 5 presents our conclusions and recommended future work. Finally, Appendix A and B contain supplementary diagrams, hardware and software specifications used in the project as well as links to source code and datasets.

2 Background and Related Work

2.1 Supervised Learning

Machine Learning

Machine Learning, in general terms, is the process of teaching a computer program to make predictions that approximate answers to various problems. More specifically, it involves tuning or fitting a model to a dataset to mitigate the need for hand-tuning and/or hand-crafting the model and its respective parameters. The ability to have the machine learn these parameters is quite useful as many problems are quite difficult to solve or even reason about by hand — especially when the parameter count can span into the order of thousands to millions. One of the simplest examples and often an introductory lesson to machine learning, is fitting a line or curve to one-dimensional data. The x-axis acting as the variable, with the y-axis representing the label. If the work is done correctly, this line or curve then approximates the data itself (see Figure 2.1). Thus the machine can make predictions about unknown inputs (the missing data points). However, some data is too complicated to fit a line or curve; more sophisticated algorithms are needed. This is especially true in the case where data points are whole images.

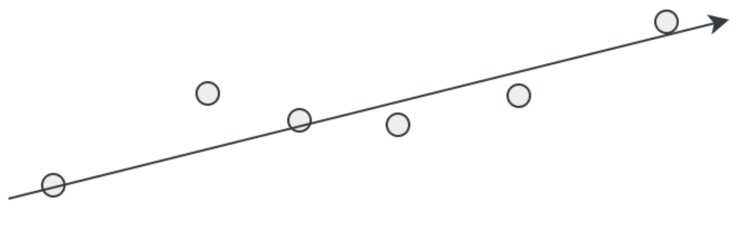


Figure 2.1: Example of fitting a line to data points.

There are many different methods of teaching a computer how to solve problems. However, one common and important approach is to simply tell the machine when it is right or wrong. This is known as *Supervised Learning*. The user prepares both the input to the model and the appropriate expected output. Dependent on the magnitude of error in the machine's answer, the learnable parameters are then adjusted in such a way that the next time the machine observes this input, it is less incorrect. We then repeat these steps until the error value fails to improve. This is an iterative approach to machine learning and is the standard optimization regime used in the field of Neural Networks.

2.1.1 Basics of Neural Networks

Artificial Neural Networks (ANN) are a family of ML models inspired by the human brain. They weakly model the structure of the human brain through an abstraction of the neurons and synapses. A neuron is approximated by a non-linear function (also referred to as an the activation function) while the synapses are each approximated by floating-point numbers referred to as a parameters or weights. The fundamental atomic piece of an ANN is called a perceptron (Figure 2.2). Stacking several of these perceptrons together is called a *layer*. In a basic ANN, these layers can be connected together to compose the overall network (Figure 2.3). Layers in the network are categorized into three opaque groups based on their position within the network: input, hidden, and output. The input "layer" is the input data itself and is not a traditional layer. The output layer contains the final aggregated values computed by the network. The remaining middle layers are known as hidden layers, because they are "hidden" from the user. The network as a whole is a mathematical function; only the input and output are visible to the user. Information flows in from the inputs, multiplied by the respective weights, followed by a summation and application of the activation function (Equation 2.1). Values are then passed along to the next layer until finally the output can be gathered by the program. This routine is known as a *forward pass*.

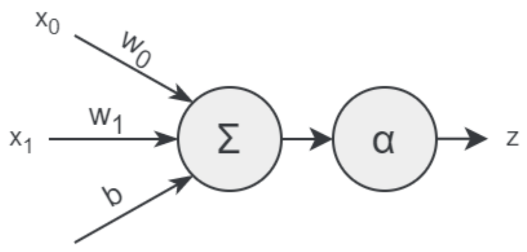


Figure 2.2: Visualization of a perceptron.

The following equation describes the mathematical form of a perceptron, where $\mathbf{x}$ is the input, $\mathbf{w}$ is the weights, $\mathbf{b}$ is the bias, and α is the activation function:

$$z = \alpha\left(\mathbf{w} \cdot \mathbf{x} + b\right) \tag{2.1}$$

Activation Functions

In the human brain, when the neuron receives a strong enough signal from the synapses, it triggers a response. In the context of ANNs, the activation function mimics this behavior by providing a non-linear response (ie. activation) to the input. It is important for activation functions to be non-linear, otherwise ANNs would be oversized linear classifiers. The non-linearity allows the network to learn non-linear decision boundaries, which enables the network to make more sophisticated predictions.

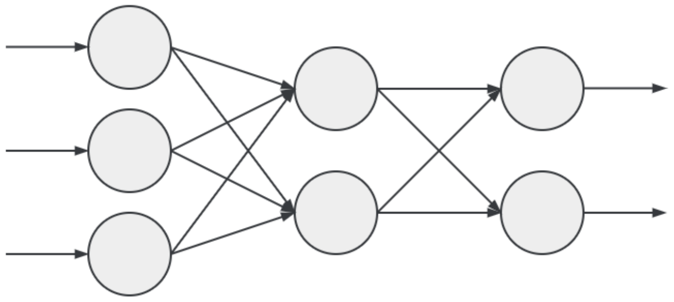


Figure 2.3: Example of a dense network composed of 3 layers (input, hidden, and output).

Sigmoid Logistic One of the most common activation functions is the sigmoid logistic function. It constrains the input x to the 0 to 1 range in a smooth s-curve shape. Defined in Equation 2.2 and visualized in Figure 2.4a.

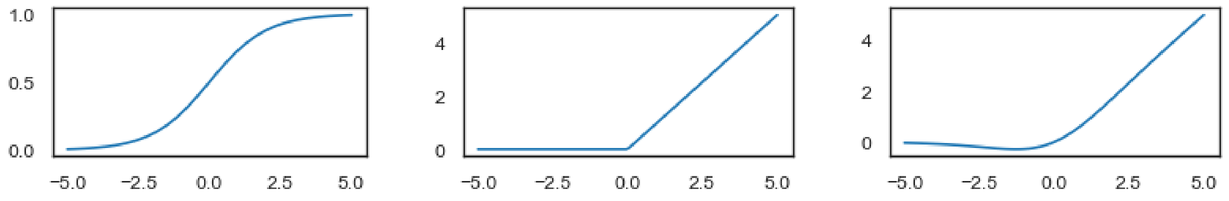
$$\alpha(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

ReLU Another very common activation function is ReLU (Rectified Linear Unit). Popularized by [19], this function applies non-linearity by simply preventing negative values. Defined in Equation 2.3 and visualized in Figure 2.4b.

$$\alpha(x) = \max(x, 0) \tag{2.3}$$

Swish This activation function [24] is similar to ReLU but smoother. Swish has a learnable parameter β, but common implementations just use $\beta = 1$ which is equivalent to SiLU [11]. Defined in Equation 2.4 and visualized in Figure 2.4c.

$$\alpha(x) = x * \text{sigmoid}(\beta x) \tag{2.4}$$



(a) Sigmoid Logistic **(b)** ReLU **(c)** Swish (SiLU)

Figure 2.4: Visualization of the various activation functions.

Loss Functions

As mentioned above, supervised learning requires a way to measure the magnitude of the error of a model's prediction for each input sample. Suppose we have a function $\mathcal{L}(z, y)$ that measures this error — where z is the prediction and y is the respective ground truth. We call this function the *loss function*. It is also sometimes known as the *objective function*. The output value of the loss function should be proportional to the error. As the error increases, the loss value should increase proportionally. Thus during training, the overall goal is to minimize the loss value. However, in some settings, this can be reversed where larger values indicate the network is performing better.

Any differentiable function can be used as a loss function, as long as it is suitable for the task. As such, there are many task-related loss functions to choose from. Sometimes, however, they are authored directly into the architecture of the network as seen in Section 2.3.1.

An intuitive example of a error metric is the commonly used variation of L_2 norm called mean squared error (MSE, Equation 2.5). It computes the average squared difference between prediction and ground truth. The square causes larger differences to be weighed more heavily during optimization.

$$\mathcal{L}_{\text{MSE}}(z, y) = \frac{\sum_{i=1}^{n} |\mathbf{z_i} - \mathbf{y_i}|^2}{n} \tag{2.5}$$

Optimization

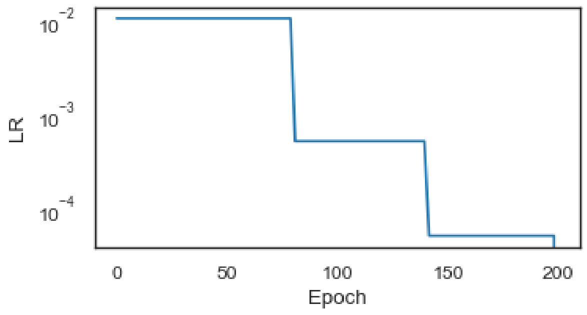
Optimization of ANNs often make use of the algorithm known as back-propagation. In essence, it takes advantage of the derivatives of the activation functions (and other computations present within the network) to propagate an error correction term to the network. The derivatives are computed during the forward pass with respect to each weight in the network, using the chain-rule from calculus. At the end of each training step, the network weights are adjusted with respect to the magnitude of the gradients. One complete pass of tarining steps through the dataset is known as an epoch. Various optimization implementations exist built upon this algorithm. *Stochastic Gradient Descent* (SGD) [27] is mechanically the simplest, taking a single step along the gradients in each iteration. Variants of SGD have been proposed, such as including momentum to avoid getting stuck in valleys or crawling along plateaus. Adam [14] is another widely-used optimizer. It exploits first-order information and adaptive estimates of lower-order moments for gradient updates. Empirically, it is known to converge significantly faster than other methods making it a prime choice for optimization. Another optimizer is LARS (Layer-wise Adaptive Rate Scaling) [29] which operates similarly to SGD but keeps the optimization parameters per layer instead of for the whole model.

Learning Rate Scheduling

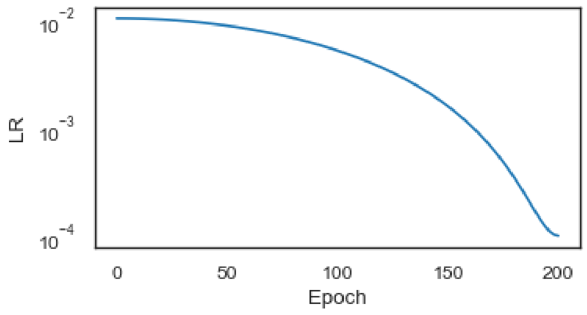
Schedulers control the Learning Rate (LR) over time, making the rate smaller (or possibly larger) when necessary to help the optimizer settle into an appropriate local minima. Many LR scheduling schemes exist, but two representative schemes are “Reduce on Plateau” and “Cosine Annealing”.

Reduce on Plateau When the tracked metric (e.g. loss or accuracy) fails to improve, the learning rate is adjusted by a constant multiplier (e.g. 0.1). The idea is that if the metric fails to improve, the optimizer may be bouncing around a local minima without settling. Lowering the learning rate then would allow it to take smaller steps and continue training. This scheduler is responsive to the training instead of being on a fixed schedule.

Cosine Annealing (or Cosine Decay) [16] Schedules the learning rate to migrate from an initial value (e.g. 0.01) to a lower bound (e.g. 0.0001) over many epoch’s through the use of a cosine curve. The general form of this scheduler also includes the ability to “warm restart” where after reaching the minimal rate, will jump back to the initial value and repeat but with a shorter epoch time.



(a) Visualization of a Reduce on Plateau LR scheduler.



(b) Visualization of a cosine decay LR scheduler.

Representation Learning

During the forward pass, the activation functions in each layer generate a set of values known as activations (these may also be referred to as representations or features). These values represent an abstraction of the input at various depths within the network. The intermediate representations are not often useful outside of the natural flow of the network itself. Instead it is common to use only the last layer of representations, which is an abstract summary of all representations, to compose the prediction. For example, using a dense layer as the output layer of the network to map the abstract representations to object classification.

2.1.2 Convolutional Neural Networks

Dense Layers can be computationally expensive when using them with image inputs. A (somewhat) small 128x128 RGB image can have nearly 50 thousand floating-point values. If each value in the image is *fully connected* to an equivalently sized layer, there would be a staggering 2.4 billion parameters. If there are many of these layers even after reducing their size deeper within the network, the number of parameters for the model and optimizer will result in significant memory and computation requirements. Increasing the image size may make things worse due to the quadratic relationship to image size (width by height). Additionally, a fully connected layer evaluates the global context of an image which may itself be an undesirable trait as it can be harder on the optimizer. Convolutional neural networks were introduced in order to reduce the number of parameters, allowing us to use larger images without explosive resource requirements and take advantage of local spatially-relevant patterns in the image.

Convolutional Layers

Convolutional Neural Networks (CNN) [20] are networks mainly composed of *Convolutional Layers*. These layers operate on images through a sliding window. They apply a convolutional filter within this sliding window, outputting a new *Feature Map* (Figure 2.6). A convolutional filter can be thought of as a weighted sum, which is very similar to the model of a simple perceptron. Like a typical layer, the resulting value from the filter is also passed through an activation function. In practice, the sliding window concept is achieved by weight sharing. For any particular unit in the layer, the same weights are used. The optimizer then only has to process the weights in the filter. For example, a 3x3 filter will only require nine weights which is significantly less than the entire image. This not only improves the memory footprint but also can shorten training time. However, a convolutional layer usually has more than one learnable filter. This allows the network to learn multiple image characteristics through the multiple filters.

Stride and *Padding* are two important parameters of a convolutional operation. They govern which patch of data on which the sliding window operates. Stride changes the distance (step size) by which the sliding window moves. For example, a 2x2 filter with a stride of 2 has no overlap. Padding, on the other hand, inflates the input with extra dummy values. This is used to compensate for mismatched sizes between the input tensor, patch size and stride. Different implementations exist, padding with zeroes, reflection padding, etc.

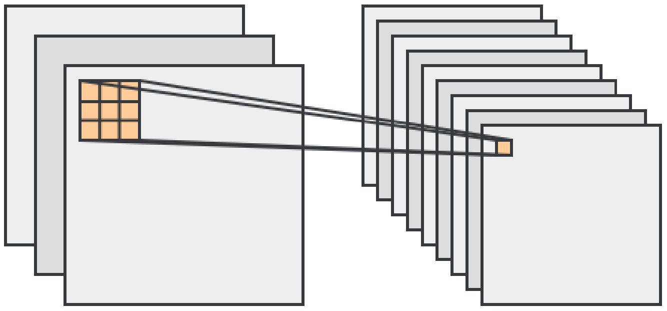


Figure 2.6: Visualization of a 2D convolution operation. In this example, the operation reads from a 3x3 sliding window on a 3-channel input tensor (e.g. RGB) and writes to a smaller 9-channel output tensor.

Pooling Layers

Another important category of layers for CNNs is *Pooling Layers*. Their job is to reduce the size of the feature maps. There are multiple types of pooling layers; *Max Pooling* and *Average Pooling* being some of the most common. Operating on a patch of the feature map — similar to a sliding window — the pooling layer outputs some aggregate value representing the input patch. Max Pooling, for example, might take in a 2x2 region of the input tensor and output the maximum value of the patch. Likewise, Average Pooling computes the average of the patch. These spatial reductions allow the network to learn from an aggregate of larger and larger spatial contexts as the network becomes deeper. Similar to convolutional layers, pooling layers also have stride and padding parameters.

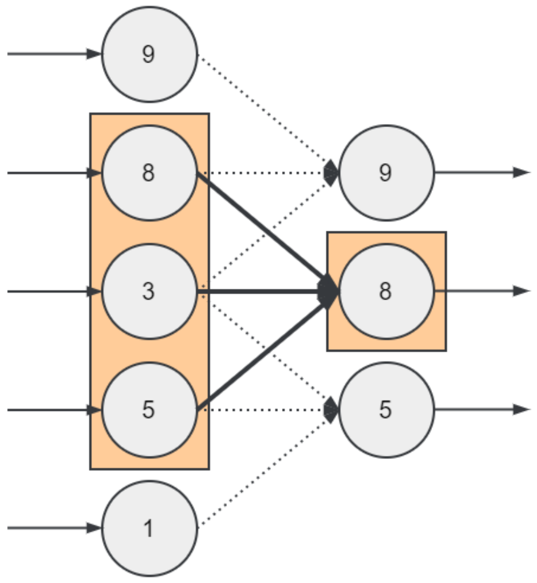


Figure 2.7: Visualization of a 1D max pooling layer. Here the max pooling operates on a sliding window with size of 3. Notice the layer size reduction from 5 to 3.

Receptive Fields

An important concept with deep convolutional neural is the receptive field. It is the consequence of applying multiple convolutional and max pooling layers together in sequence. The value of one unit deep in the model is the aggregate of the previous layer, that layers previous layer and so on. Each step backward increases the total area influencing the value (Figure 2.8). Understanding the shape of receptive fields can help explain behaviors of CNNs. For example, an image with a large object but small receptive fields may never “see” the entire image, and thus produce invalid classification. However, this may not be a problem with segmentation that would may only care to find object boundaries.

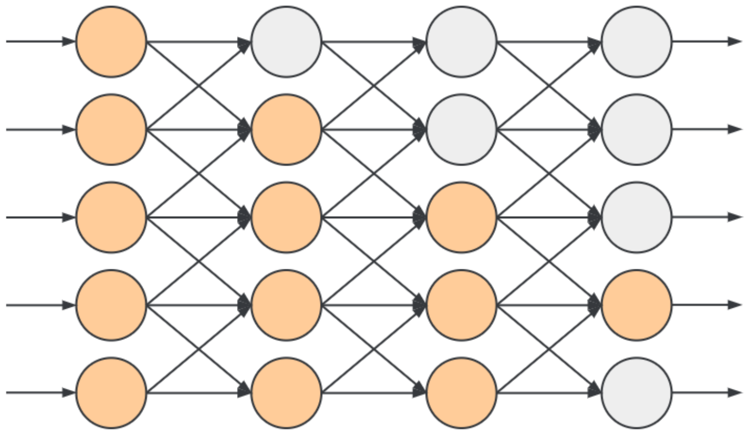


Figure 2.8: Simplified visualization of a receptive fields within CNNs. Here the layers are 1-dimensional, connected to neighboring 3 units in the subsequent layers. The colored unit show the information aggregation from left-to-right.

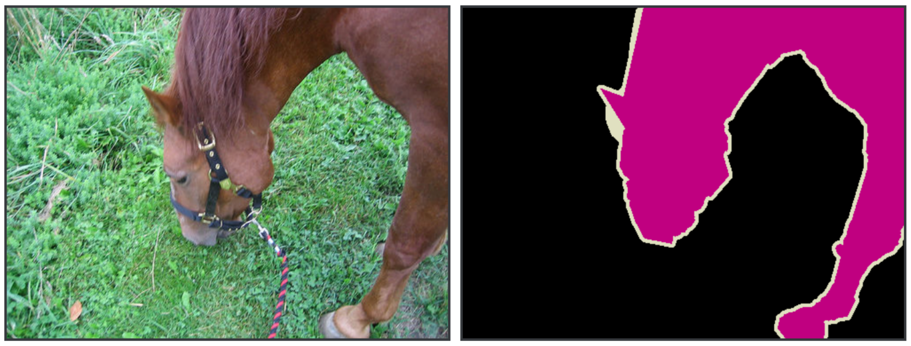


Figure 2.9: Example of semantic image segmentation from [7].

Typical CNN Architecture

A typical CNN architecture consists of one or more blocks. One of these blocks is likely composed of a one or more convolutional layers with an activation function (e.g. ReLU) and then a pooling layer. Several of these blocks are composed together possibly followed by, for example, a fully-connected layer for classification. The exact composition of a block and/or the final layer can be different depending on the task and overall architectural design.

2.2 Semantic Image Segmentation

This thesis is focused on semantic image segmentation; a task where the network where each pixel can be classified into two or more of the known object types. This has the overall effect of generating a binary mask of each class of object in the image (Figure 2.9).

There are at least two forms of the image segmentation task. These are generally referred to as "semantic segmentation" and "instance segmentation". For semantic segmentation, the model will classify pixels with only an object type. On the other hand, instance segmentation (as the name suggests) classifies the object type and the instance number (e.g. first vehicle, second vehicle, etc).

2.2.1 Jaccard Loss

The Jaccard Index measures the similarity of two sets by computing the sizes of the intersection and union of the sets. As such, it is often referred to as Intersection-over-Union (IOU) (or mIOU when representing an average of classes).

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{2.6}$$

The computed value ranges from 0.0 (disjoint) to 1.0 (exact overlap). We can construct a loss function by inverting this range.

$$\mathcal{L}_{\text{IOU}} = 1 - J(A, B) \tag{2.7}$$

To adapt this to semantic image segmentation, one can operate on each set of pixels corresponding to a class separately. Comparing each set in the prediction against the respective set in the label, one can use the average IOU score (mIOU) as a measure of the accuracy for the whole prediction. When combined with an optimizer (ex. Adam), the loss value will be minimized, thus encouraging predictions that overlap the label.

2.2.2 U-Net

U-Net is a commonly used CNN architecture for semantic image segmentation, and was original proposed for segmenting medical images [26]. At the time of release, it out performed contemporary models. It gets its name from the overall "U" shape the architecture provides. It is essentially a convolutional auto-encoder architecture but connects encoder blocks to the respective blocks in the decoder portion, allowing for a richer set of semantic information to produce the output.

U-Net, as described in the original paper, consists of basic convolutional blocks, max pooling and concatenation layers. The encoder portion consists of a sequence of convolutional blocks each consisting of two 3x3 convolutional layers with ReLU activation followed by a 2x2 max pool layer. This lowers the image dimensionality of each block while increasing the feature depth. After the the final encoder block, the decoder portion begins. In essence, this reverses the encoder's design with two 3x3 convolutional layers with ReLU activation followed by 2x2 up-scaling convolutional blocks. Each of the blocks are connected with its symmetrical partner block from the encoder. This concatenation combines image feature information (shallow layers) with semantic information (deep layers). The output size matches the input size. One important

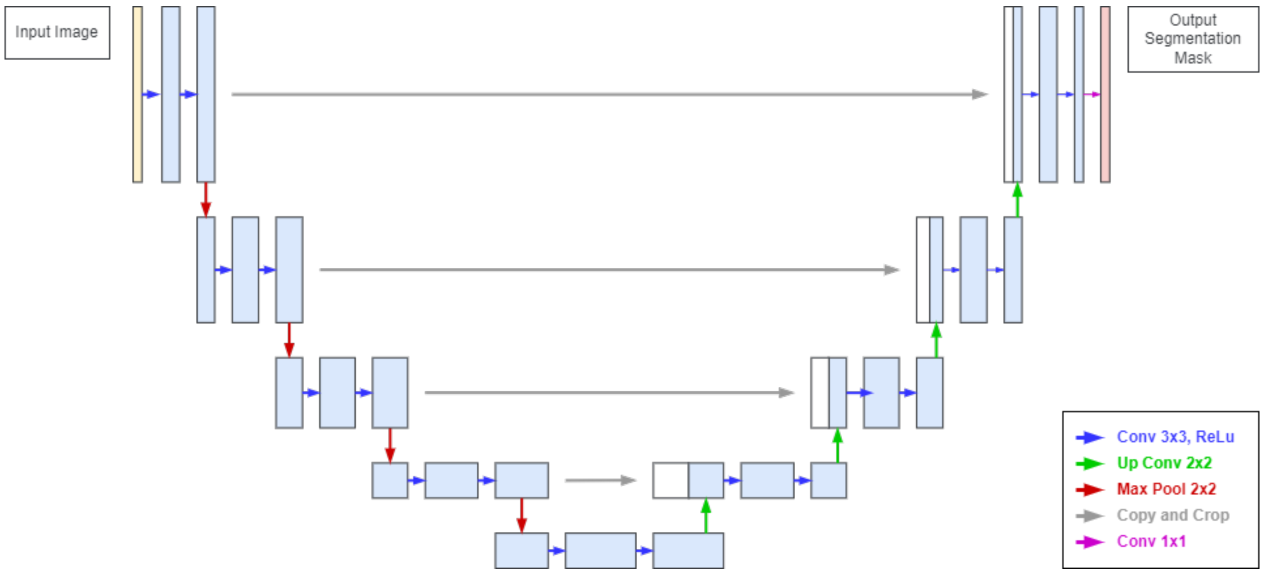


Figure 2.10: Diagram of U-Net architecture

aspect to take note of is the down-scaling and up-scaling operations enforce that the dimensionality of the input image must be a multiple of 32 (2^5, for the five 2x2 max pools).

ResNet34 Encoder

Due to the U-Net architecture having a fairly distinct and separable encoder and decoder sections, variations of the original architecture emerged with modifications, such as the use of alternative encoders [23] [18]. For example, it is fairly common to use an encoder from the family of residual networks (ResNet) to replace the basic convolutional encoder. Switching out the basic U-Net encoder for ResNet34 allows the network to learn better representations and to shorten training time.

ResNet's [10] are popular due to their ability to easily train deeper networks by reformulating stacks of layers (i.e. blocks) into residual functions. They do this with something called a shortcut connection. It is an identity mapping from the start of a block to sum with its output. This helps propagate the effect the input has to the entire network to combat the vanishing gradient problem. See Figure A.1 for architecture details.

2.3 Self-Supervised Learning

Supervised learning learns representations from many labelled images. Each labelled image contributes to the overall gradient for optimization. The more labels you have, the better it will learn. However, producing correctly labelled data is one of the most expensive parts of training a neural network. Creating these labels often requires one or more domain experts to painstakingly label (possibly hundreds of) thousands of images. Thus, it would be beneficial if the model could learn appropriate representations without as many human-provided labels. This raises the question: *How can we get the model to learn representations directly from the data itself?* This question is the primary motivation for Self-Supervised Learning (SSL) based methods.

SSL methods construct a label-pair from the data itself (in contrast to unsupervised learning). For example, a data augmentation process where the label-pair consists of an augmented image as the input and the original non-augmented image as the label. Some methods use multiple augmented views of the input instead of the original image [3] [2] [30]. These augmented views challenge the model to learn how to circumvent the effect of the augmentations and thus implicitly learn representations of the original image content. Once an SSL model has been trained, it is common to make use of this pre-trained model as an weight initialization scheme for the downstream task. This is sometimes referred to as transfer learning; the SSL task (aka. pretext task) kickstarts the model parameters for a supervised "downstream" task. Three common SSL approaches include Siamese Networks, Barlow Twins, and Image Reconstruction, which are described in the sections below.

Siamese Networks

Siamese Networks (sometimes referred to as "Twin Networks") are a popular SSL architecture. They are composed of multiple "arms" where each arm is composed of a sub-architecture. At a high level, each arm wraps an encoder network and (optionally) augments the input so that it operates on a slightly different view of the input. Some siamese network designs may choose to not augment their inputs but instead are given multiple inputs from the dataset directly. The output tensors through each arm then can be used by some loss function for optimization. Overall, the general idea is to either discriminate or find similarities between two or more related inputs via their output representations (Figure 2.11).

The exact steps involved can change significantly depending on specific design of the network. For example, MoCoV2 [2] establishes a queue of prior representations; allowing the model to distinguish features by contrast of positive and negative samples. In SimSiam [3] they avoid the use of negative samples. Instead they make use of a stop-gradient operation to promote learning feature similarity of views of the input image. In BYOL [9], the encoder in one arm has its weights updated via an exponential average from weights in the other arm, guiding itself in a teacher-student like manner.

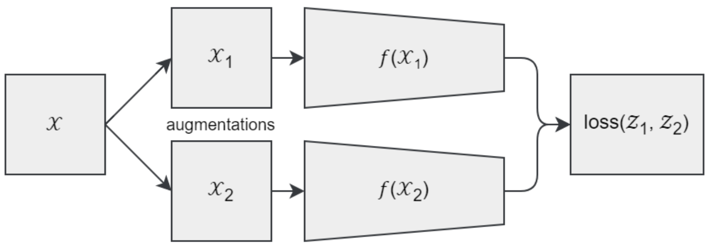


Figure 2.11: Generalized view of siamese network architectures.

2.3.1 Barlow Twins

Barlow Twins (BT) [30] is a siamese network SSL architecture designed around the concept of redundancy reduction. It gets its name from the work of neuroscientist H. Barlow. His work highlighted how the goal of sensory processing is to process complex redundant information into simpler statistically independent information. Based on this, the authors designed the objective function to encourage the cross-correlation matrix computed from the embeddings to become similar to the identity matrix. This way, the learned representations (from each augmented input) are not only both similar but also avoid containing redundant information.

We chose to employ this architecture in our experiments (vs other siamese methods) due to it being simple to implement as well as the authors claim to being robust under smaller batch sizes. The authors also claim this method provided state-of-the-art in downstream fine-tuning localization tasks compared to other contemporary SSL methods.

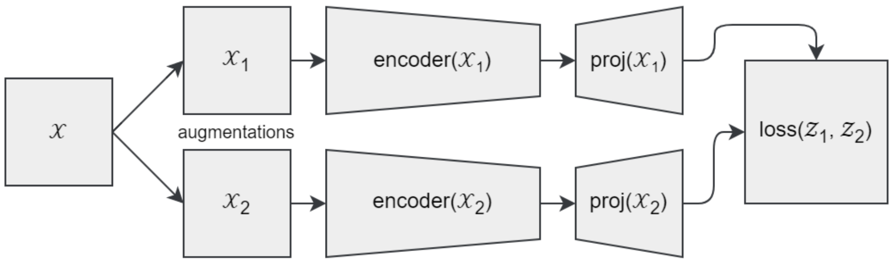


Figure 2.12: Diagram of the Barlow Twins architecture.

Augmentations

The augmentations used by Barlow Twins are the following: Random Resized Crop, Horizontal Flip, Color Jitter, Grayscale, Gaussian Blur, and Solarization. The first two augmentations are always applied while the remaining augmentations are randomly applied. The probability of each augmentation is dependant on the arm of the network.

Encoder and Projector

There are three main components of the network architecture: the encoder, projector, and loss function. The encoder can be any network that accepts the augmented images as input and outputs a feature vector. The original authors used ResNet50, removing the final classification layer. In our case, we similarly used the more lightweight ResNet34. The representations from the encoder are fed into a projector to produce "embeddings" for the loss function. The projector is composed of a few dense layers that project from one representation space to another. This helps the network "untangle" for the loss function. The same encoder instance is shared between the arms, while the projector is unique to each arm.

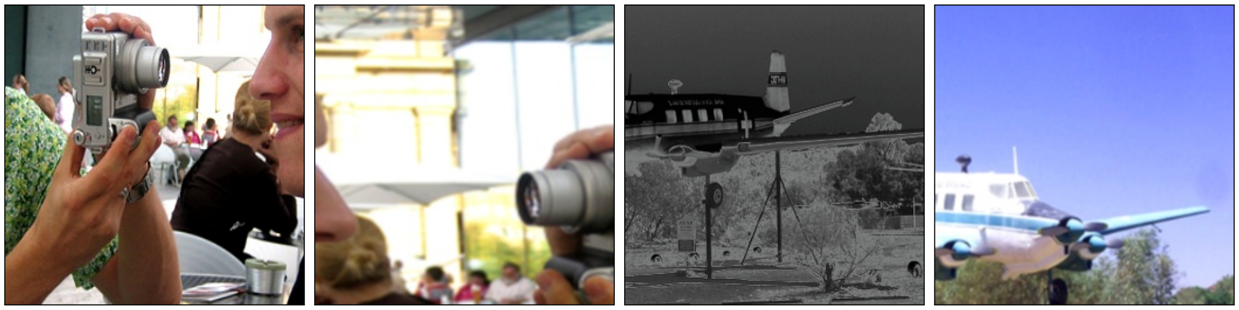


Figure 2.13: Visualization of a small set of randomly selected examples of the BT augmented images. Note the first two images are the augmentation pair of one source image, and last two are another. Image from [7].

Barlow Twins Loss

The BT loss function is computed from an empirical cross-correlation matrix formed by the two output embeddings in the network. It is then decomposed into two terms, the on-diagonal elements (invariance term) and the off-diagonal elements (redundancy reduction term). The loss function attempts to encourage the on-diagonal elements to equal one while the off-diagonal elements are encouraged to be zero. The intuition for the BT loss function is that the invariance term makes the embeddings invariant to the augmentations while the redundancy reduction term makes the embeddings contain non-redundant information about the input. The two terms are summed together with a trade-off parameter λ on the second term (Equation 2.8). The authors did not provide any insight on selecting a value for λ but did provide the value they chose of $\lambda = 5 \cdot 10^{-3}$ through algorithmic search.

$$\mathcal{L}_{\mathrm{BT}} = \sum_i (1 - C_{ii})^2 + \lambda \sum_i \sum_{i \neq j} {C_{ij}}^2 \tag{2.8}$$

Recent Extensions to Barlow Twins

In this subsection we briefly cover two related works that were developed concurrently to this thesis. They could be used as future work as a point of comparison. The first work evaluates alternative flavors of UNet with BT pre-training and evaluation on a two-class biomedical dataset. The second work evaluates BT pre-training with a modified augmentation stack for application on an agricultural dataset.

BT-Unet: A self-supervised learning framework for biomedical image segmentation using Barlow Twins with U-Net models The work presented in [23] evaluates the use of BT with various flavors of the U-Net architecture. At a high level, their approach is similar to the BT work in this thesis, however, they focused on two-class image segmentation of biomedical images. Our work is differentiated by the use of a multi-class image segmentation dataset.

On Domain-Specific Pre-Training for Effective Semantic Perception in Agricultural Robotics The work presented in [25] modifies the augmentation set used with two major changes. The first change is the addition of a background normalization pass suitable for agricultural images taking advantage of the green-brown separation of background to plant. The second change is the addition of coarse cutouts to further strengthen learning spatial features. At the time of writing, this work has not been published, but accepted into the IEEE ICRA conference.

2.3.2 Image Reconstruction

Image Reconstruction (IR) is a SSL technique described and explored in [12]. They hypothesize that a suitable pretext task needs to learn features in a spatial context for it to be used effectively with downstream segmentation tasks. The method itself is inspired by image interpolation. Similar to the way in which up-scaling an image requires inferring information between pixels, they propose an augmentation step to generate "corrupt image" inputs, and task the model to reconstruct the corrupted portions of the original clean image. In simple terms, the pretext task trains the network to generate an output image with high similarity to the input image before augmentations were applied. The theory is that these augmentations will encourage the model to learn spatially relevant features to achieve the reconstruction and thus be naturally suitable for segmentation. The work follows their evaluation and exploration of the choice of backbone model and variations of the loss function. They produce results for both image classification and semantic image segmentation. Additionally, this method pre-trains the whole segmentation model instead of only the encoder portion as seen in BT and many other SSL methods. The work in this thesis is differentiated by the use of an alternative backbone model (Section 3.2) and our alternative corruption augmentations (Section 3.3).

Corruption Methods

The following subsection describes the original corruption method presented for the IR task plus two related works of similar nature. In this thesis, we refer to a subset of image augmentations that damage or otherwise replace pixel data as "corruption". For example, setting a pixel or one of its channels to black is considered a corruption augmentation, whereas setting the pixel to its grayscale equivalent is a normal augmentation.

Original Method The original corruption method consists of one of three different augmentations (with equal probability) that are randomly applied to the input image before passing it through the network. They use two forms of dropout-like noise and a method they refer to as "Bed of Nails". The first augmentation is a probabilistic 50% whole-pixel dropout. The second augmentation is similar but instead operates on a per-channel basis. Finally, the third augmentation (Bed of Nails) operates in a 2x2 pixel pattern that only retains the top-left pixel (Figure 2.14). Pixels that are dropped out are set to a discard color (e.g. black or middle grey). Examples of these augmentations applied to real images can be seen in Figure A.7.

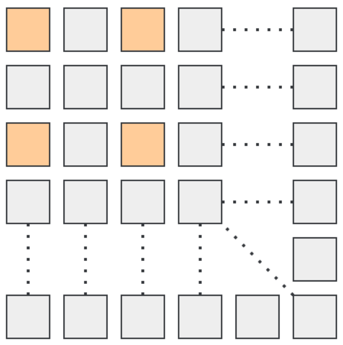


Figure 2.14: Visualization of the 2x2 "Bed of Nails" pattern. Orange pixels represent unmodified pixels. Grey pixels are set to the discard color.

Inpainting (ie. Coarse Cutout) This method was originally proposed to improve spatially relevant features in supervised and weakly-supervised settings. It has been viewed as either an augmentation step in [22] or a regularization method in [6]. It works by cutting out random rectangles and requiring the network to make predictions in lieu of the missing content. The authors of the respective papers found the method to improve spatial tasks like segmentation and localization. We extend their work by integrating the augmentation into an SSL pre-training scheme. Examples of these augmentations applied to real images can be seen in Figure A.8.

Hide-and-Seek The work in [15] is similar to the coarse-cutout method described above, except instead of randomly sized and positioned rectangles they propose cutting out regular squares from a grid overlaid on the image, where each cell in the grid has a probability of becoming a cutout. They further describe what value the cut pixels should be set to, recommending the use of the dataset-wide mean color. The authors motivate the effectiveness of the method by describing how the cutouts act like occlusion, forcing the network to look elsewhere for features for the task. Similarly, as mentioned above, we extend their work by integrating the augmentation into an SSL pre-training scheme. Examples of these augmentations applied to real images can be seen in Figure A.9.

Loss Function

In addition to the augmentations, the author explores the effect of L_1, L_2, MS-SSIM [28] and MS-SSIM+L_1 loss functions (see Equations 2.9, 2.5, 2.11, and 2.12, respectively). The author noted that L_1 loss would preserve the reconstructed image's color contrast but would not necessarily preserve the image clarity. On the other hand, MS-SSIM would preserve the image clarity but the results would have lower color contrast. After empirical evaluation, they recommend the use of an MS-SSIM and L_1 combination loss as it appeared to maximize both image clarity and color contrast.

Mean Absolute Error (L_1, MAE) Also known as L_1 loss, this measure is essentially the same as MSE (Equation 2.9) but without the exponent. It provides linear error to the optimizer unlike MSE which provides squared error.

$$\mathcal{L}_{\text{MAE}} = \frac{\sum_{i=1}^{n} |\mathbf{z_i} - \mathbf{y_i}|}{n} \tag{2.9}$$

Multi-Scale Structure Similarity Index (MS-SSIM) This image similarity measure — ranging from zero to one – is composed of three component functions: luminance (l), contrast (c) and structure (s). The three components are multiplied together in a weighted product as shown in Equation 2.10. Each component is composed of the mean (μ_x or μ_y), deviation (σ_x or σ_y), and covariance (σ_{xy}) of a sliding window (ex. An 8x8 window). The constants C_1, C_2, and C_3 are smoothing terms defined as $C_1 = (K_1 R)^2$, $C_2 = (K_2 R)^2$ and $C_3 = C_2/2$ where R is the dynamic range of the pixel values (ie. 255). Parameters α_j, β_j, γ_j, K_1, and K_2 are to be chosen by hand but [28] recommends to simplify their use by setting $\alpha_j = \beta_j = \gamma_j$ such that $\sum_{j=1}^{M} \gamma_j = 1$, and $K_1 = 0.01$, and $K_2 = 0.03$. The sliding window is passed over all pixels in the input image and the resulting quality map is then averaged to produce the overall similarity index value.

$$\begin{gathered} l(\mathbf{x},\mathbf{y}) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \qquad c(\mathbf{x},\mathbf{y}) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \qquad s(\mathbf{x},\mathbf{y}) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \\ \text{MS-SSIM}(\mathbf{x},\mathbf{y}) = [l_M(\mathbf{x},\mathbf{y})]^{\alpha_M} \cdot \prod_{j=1}^{M} [c_j(\mathbf{x},\mathbf{y})]^{\beta_j} [s_j(\mathbf{x},\mathbf{y})]^{\gamma_j} \end{gathered} \tag{2.10}$$

Reconstruction Loss The reconstruction loss is defined by the combination of MAE and MS-SSIM (Equation 2.12). However, you have to invert the zero-to-one range of the similarity measure so the optimizer will minimize the dissimilarity instead (Equation 2.11).

$$\mathcal{L}_{\text{MS-SSIM}} = 1 - \text{MS-SSIM}(\mathbf{x},\mathbf{y}) \tag{2.11}$$

$$\mathcal{L}_{\text{IR}} = \mathcal{L}_{\text{MAE}} + \mathcal{L}_{\text{MS-SSIM}} \tag{2.12}$$

Open Questions

In this chapter, we have described the relevant background and a few closely related works we believed promising for use in an SSL context. In addition, we highlight two recent related works that were released during the authoring of this thesis that both involve variations of Barlow Twins with different variations of U-Net. However, even with the multitude of existing SSL research, it still isn't clear which SSL technique is the best approach in the context of semantic image segmentation. In this thesis, we will compare two different SSL methods in the context of semantic segmentation: Barlow Twins and Image Reconstruction. Both of these approaches make use of image augmentation to produce supervisory signals for a SSL pre-training.

3 Methodology

This chapter describes the datasets, models, and setup of the experiments conducted for this thesis. These experiments compare and evaluate BT and IR pre-training methods for semantic image segmentation with the PASCAL VOC dataset. In addition, the SSL results are compared to two baseline methods: no pre-training and supervised pre-training.

3.1 Datasets

3.1.1 PASCAL VOC

PASCAL VOC [7] is a moderate-scale dataset that contains classification, object detection, segmentation, and action context tasks. It was run as a series of challenges from 2005 to 2012. The 2012 edition expanded the number of segmentation labels and is the version we used in this thesis. However, we used the testing set from 2007 as the 2012 testing set is not publicly available.

This dataset contains a total of 33,260 images of general real-world scenes (Figure 3.1). A subset of 2,913 images have segmentation labels and 30,347 images do not. The images not associated with segmentation labels are considered "unlabelled" and are used as the training set for SSL tasks. The 2007 test data is also excluded from SSL tasks. The images range in size from 112x174 to 500x500 with an average size around 390x465. The segmentation labels contain 21 different classes of objects plus an additional "void" class that is intended to allow some imprecision between object boundaries and ambiguous labelling cases. The "void" pixels are omitted when computing accuracy measures and such do not affect the results.

3.1.2 ImageNet

ImageNet [4] is the most widely used dataset in contemporary research for object classification, localization, and detection. The 2017 release of the classification and localization dataset contains 1000 classes of objects contained in 1.2 million images of general scenes (Figure 3.2). Images vary in resolution with an average size of around 469x387. We did not use this dataset directly, but instead used a pre-trained checkpoint provided by PyTorch for fine-tuning to act as a baseline. See Appendix B.3 for more details.

Figure 3.1: Five randomly selected example images from PASCAL VOC [7].

3.1.3 Standard Augmentations

Basic Augmentations

All images are augmented first via these three augmentations in IR pre-training and downstream tasks. Additional augmentation steps may be applied depending on the method and are described in their respective sections.

- Random Resize Crop – A random rectangle crop 25% to 100% of the image and then resized to the target size.
- Horizontal Flip – 50% probability to apply.
- Color Jitter – Randomly perturbs brightness, contrast, saturation, and hue slightly and has a 50% probability to apply.

Normalization

Before images are passed to the network they are normalized with mean and standard deviation values computed from the PASCAL VOC dataset. The normalization values are:

$$\mu = \begin{pmatrix} 0.4529 & 0.4317 & 0.3999 \end{pmatrix} \qquad \sigma = \begin{pmatrix} 0.4443 & 0.4665 & 0.4887 \end{pmatrix} \qquad z = \frac{x-\mu}{\sigma} \tag{3.1}$$

Tile Splitting

In preliminary experiments, as an attempt to accommodate higher resolution images, we split the images into smaller square tiles. This was especially important for the memory requirements for a sufficiently large

Figure 3.2: Five randomly selected example images from ImageNet [4].

batch size when training a Barlow Twins model. Before splitting the images into tiles, they are first resized to the nearest multiple of the tile size in each axis. To help avoid unwanted stretching and maintain the same aspect ratio, the images are uniformly scaled to fit the maximal axis and then reflection padded to fit the minimal axis to achieve the target size. The images are then sliced into tiles via a sliding window with a half tile step size in rasterization order. For example, if one had a 500x450 image and a tile size of 128, the image would be resized to 512x512 and then sliced into 9 tiles, as depicted in Figure 3.3.

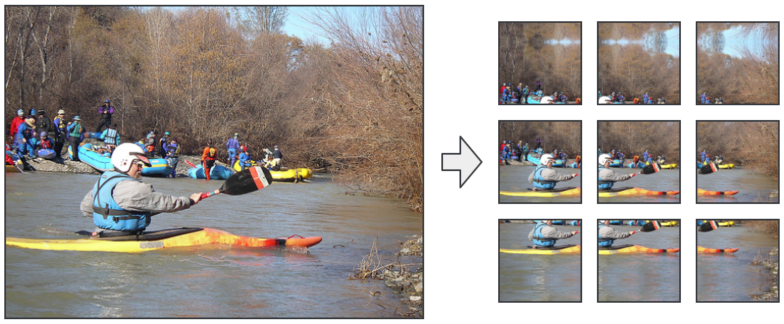


Figure 3.3: Example of the image resizing and tile splitting mechanism. Image from [7].

For our final experiments, our tile splitting scheme was superseded by random resized cropping. We switch to random resized cropping because our implementation of the tile splitting and caching mechanism occasional caused the training code to halt. Random resized crop also made the code simpler and is more consistent with contemporary works.

3.1.4 Dataset Splits

The datasets were split into training, validation, and testing splits. For PASCAL VOC, we used the pre-existing splits. With 1464 training images, 1449 validation images, and 1469 testing images. For the "unlabelled" PASCAL VOC data, we split the data 70/15/15, resulting in 21243 training images, 4552 validation images, and 4552 testing images.

The training set is used to optimize the model. The validation set is used at the end of each epoch to keep the best performing checkpoint, but otherwise does not affect optimization. The testing set is witheld until after training is completed. This allows us to evaluate the model on unseen data, and more genuinely reflects the general behaviour of the model. The loss metric is computed for each split at the end of each respective step.

3.2 Experimental Overview

For all experiments, we opted to use U-Net with a ResNet34 based encoder (ResUNet) as our segmentation model. The experiments are split into two phases, a "pre-training" phase and a "downstream" phase. The pre-training phase uses either BT or IR to pre-train the segmentation model. We evaluate three variations of the IR method by switching out the corruption augmentations with an alternative. The downstream phase trains the model following a typical supervised learning scheme. However, the weights may be initialized from a pre-trained state (i.e. weight transfer). Multiple runs of each model are trained using different random seeds to observe the effect of stochastics. Due to time constraints, only one run of SSL pre-training was performed. The downstream task was run on three different seeds for each respective weight initialization method.

3.2.1 Hardware & Software Configuration

The hardware, software, and other implementation details are available in Appendix B.

3.3 Pre-Training Experiments

In this section, we describe the pre-training experiments. The experiment is setup to pre-train either the whole segmentation model (via IR) or only the encoder portion (via BT). The resulting pre-trained model will be used to for weight initialization in Section 3.4. We pre-trained for 1000 epochs.

Barlow Twins

We used the original authors implementation with no architecture alterations. We used the minimum described effective batch size of 256 (64 per GPU), a projector width of 4096 and depth of 3. We chose to use these reduced parameters to due to VRAM limitations. Also, for compatibility with ResNet34, the output vector size was resized 512 (instead of the original 2048 for ResNet50). We used the paper recommended trade-off value of $\lambda = 5 \cdot 10^{-3}$. To keep consistency with the original paper, we also trained with LARS optimizer. The input images are augmented as described in Section 2.3.1.

Image Reconstruction

We implemented the IR model ourselves, because we could not find or acquire an implementation from the original author. In essence, the architecture can be any segmentation model with 3 class outputs for RGB color prediction. The input images are augmented as described below. The original non-augmented image then acts as the label. We use an effective batch size of 128 (32 per-GPU). Unlike [12], we do not use weight-decay because we found it to harm its ability to learn, possibly due to a 20% dropout regularization already existing in our segmentation model. We trained the IR models with the Adam optimizer [14]. The SSIM+L1 combo loss function is described in Section 2.3.2. We used the library default MS-SSIM parameters: kernel size = 11, $\sigma = 1.5$ and $K_1, K_2 = 0.01, 0.03$.

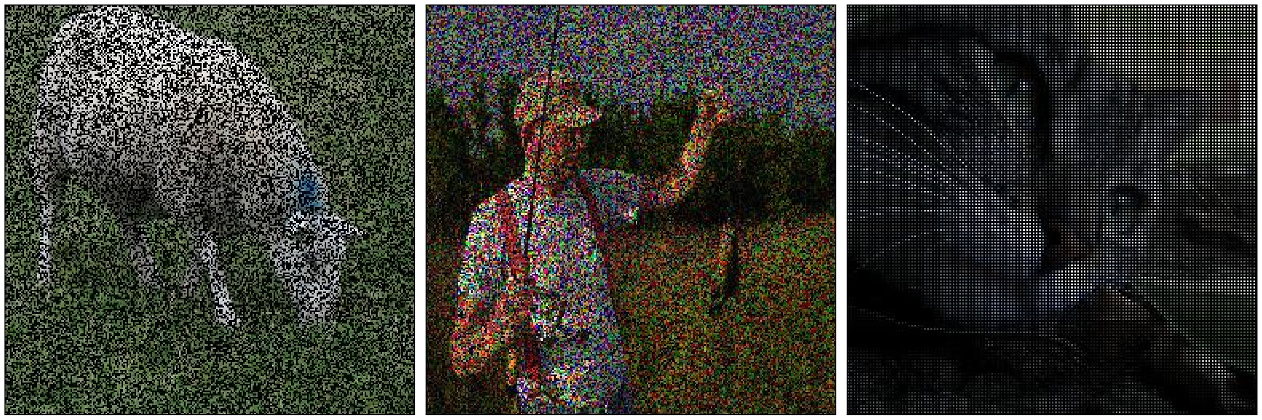
Corruption Method Variations

The IR task, as the name suggests, requires the network to reconstruct the pixels "corrupted" by a set of augmentations. The author proposed a set of augmentations that we refer to as the "Original" augmentations. As an extension to the work, we replaced this corruption step with two different sets of augmentations of similar nature. The intuition was to corrupt pixels in the image with a larger local-scale effect than per-pixel dropout. Agreeing with the argument proposed in [15] that larger cutout regions obscure the image, forcing the network to learn the relevant representations based on the non-obscured portions of the image.

Original (IR+Orig) [12] This corruption method uses a set of 3 augmentations with equal probability of occurring: a random whole-pixel dropout, a random pixel-channel dropout and "bed of nails". The method is fully described in Section 2.3.2 and visualized in Figures 3.4 and A.7.

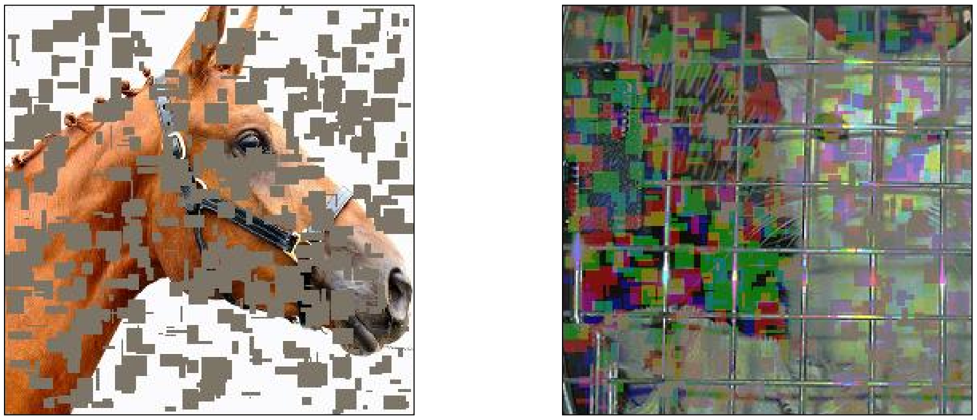
Coarse Cutout (IR+CC) [22], [6] This corruption method works by cutting out small rectangles from the source image. The cutout pixels are set to a constant color (as defined in Section 2.3.2). The augmentation has a 50% probability that it will either affect whole-pixel or per-channel. In the per-channel case, a random channel index is selected and the appropriate component of the constant color is used. These augmentations are visualized in Figures 3.5 and A.8.

Hide-and-Seek (IR+HaS) [15] This corruption method is nearly identical to the Coarse Cutout method above but instead of relying on randomly sized, randomly positioned rectangles, this method selects and cuts out squares on a randomly sized, randomly offset grid. These augmentations are visualized in Figures 3.6 and A.9.



(a) Whole-Pixel Dropout **(b)** Per-Channel Dropout **(c)** Bed of Nails

Figure 3.4: Two examples of input images generated during IR pre-training with the "Coarse Cutout" augmentations.



(a) Whole-Pixel Cutout **(b)** Per-Channel Cutout

Figure 3.5: Two examples of input images generated during IR pre-training with the "Coarse Cutout" augmentations.

3.4 Downstream Experiments

The downstream experiments consist of multiple training runs of the segmentation model with differing random seeds under each weight initialization scheme described below. We trained each edition of the model 3 times for 500 epochs. We optimize the segmentation model for Jaccard Loss (Section 2.2.1) using the Adam optimizer. The overall results are described in Section 4.1, for the following conditions:

No Pre-Training

In this run, we initialize the segmentation model without transfer learning. We simply create the model with PyTorch default randomized weights. This initialization acts a baseline for comparison with pre-training.

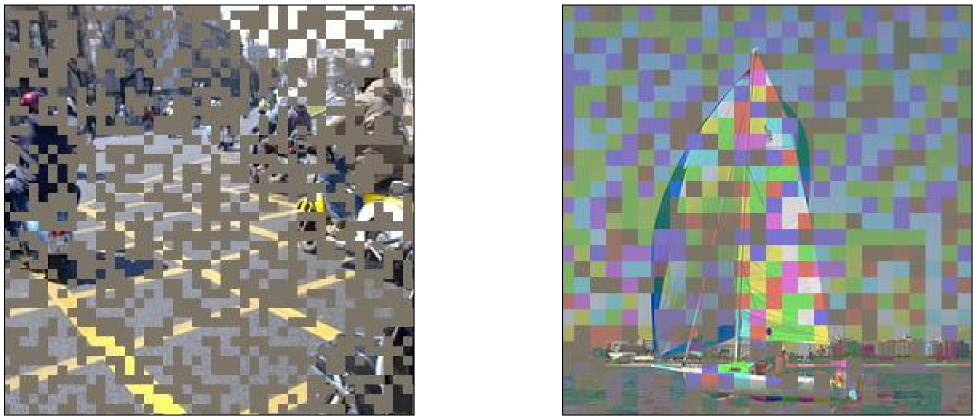

(a) Whole-Pixel Cutout **(b)** Per-Channel Cutout

Figure 3.6: Two examples of input images generated during IR pre-training with the "Hide-and-Seek" augmentations.

ImageNet Supervised Pre-Training

For ImageNet based weight initialization, we use an ImageNet pre-trained encoder provided by PyTorch. The decoder portion of the network is still randomly initialized. This initialized baseline helps demonstrates the how competitive SSL pre-training can be with a well-known supervised pre-training target.

Barlow Twins SSL Pre-Training

For BT based weight initialization, we load the BT SSL PyTorch checkpoint and only copy the encoder weights into the segmentation model. BT does not train the decoder, so these weights will still have random initialization.

Image Reconstruction SSL Pre-Training

For IR based weight initialization, we load the IR PyTorch checkpoint and copy over all weights except the final layer. The final layer will likely have mismatched dimensions as the IR checkpoint has 3 "classes" for RGB and the segmentation task will have a different count (21 classes).

4 Results and Discussion

In this chapter, we present and discuss the results of our experiments.

4.1 Overall Experiment Results

Overall, IR+HaS provides the most benefit for SSL transfer learning. All SSL methods but IR+Orig show improvement over the no pre-training baseline. However, all SSL methods still fall short of the supervised pre-training baseline. See Table 4.1 for complete details. See Figures A.10, and A.11 for examples of segmentation output.

Table 4.1: PASCAL VOC 2007 Segmentation Results

Pre-Training Type	Overall mIOU ± Std.Dev.	Run 1 mIOU	Run 2 mIOU	Run 3 mIOU
IR+HaS	**0.4855 ± 0.0278**	**0.4715**	**0.4606**	**0.5243**
IR+CC	0.4550 ± 0.0366	0.4411	0.4188	0.5052
BT	0.4320 ± 0.0109	0.4285	0.4208	0.4469
IR+Orig	0.3396 ± 0.0333	0.3151	0.3170	0.3868
ImageNet Supervised	**0.5768 ± 0.0059**	**0.5714**	**0.5740**	**0.5850**
None	0.3605 ± 0.0491	0.3285	0.3231	0.4299

On our hardware, BT pre-training took approximately 12.5 hours, IR+Orig pre-training took approximately 7.2 hours, IR+CC pre-training took approximately 7.2 hours, IR+HaS pre-training took approximately 42.5 hours. Training the downstream segmentation model with any of initialization methods took approximately 32.0 minutes. It is likely IR+HaS took significantly longer than the other IR variants due to a poor implementation of the augmentation rather than failure of the method itself.

4.2 Barlow Twins

Referencing Figure A.6, it can be seen this method struggled to train the model well. The training and validation loss were separated by multiple orders of magnitude, suggesting a failure to generalize. Downstream training with BT initialization showed a notable early improvement in mIOU metrics over no pre-training, however quickly plateaued also suggesting poor generalization.

We believe this poor performance is a consequence of two reasons. The first reason is that the BT Pre-Training task isn't specifically constructed for spatial learning. It is designed to learn distinguishing characteristics between images as a whole. This produces strong results for image classification (according to [30]) but it is not necessarily beneficial to learn the spatial features needed for semantic image segmentation. The second reason is that our BT pre-training was done with minimal resources. We used a much smaller dataset and reduced hardware configuration than the original paper. The small dataset size, the low batch size, and possibly a poor loss trade-off value caused the model to struggle while pre-training. In Section 5.3 we discuss potential research directions that may improve BT pre-training for semantic image segmentation.

4.3 Image Reconstruction

Referring to Table 4.1, we can see that both IR+CC and IR+HaS variants provide an improvement over the no pre-training baseline. The IR+Orig variation did not provide an improvement out of the IR variants, even if the mIOU metrics were higher in early epochs. The IR+HaS variation performed the best and is our overall recommendation for SSL pre-training for segmentation. See Figure A.2 and A.3 for loss curves during pre-training.

We believe both IR+CC and IR+HaS variations performed well due to them making larger cutouts of the image. The corruption step in the IR method destroys a portion of the original information in the image, this in turn forces the model to attempt to infer the missing information using the surrounding context. We believe that designing augmentations that encourage the model to "look" at the surrounding context will cause the model to learn spatial features relevant for semantic image segmentation. The authors of HaS [15] describe a similar sentiment. The original corruption augmentations described in the IR method involved dropping out pixels at a per-pixel level. We believe that reconstructing only a few pixels (i.e. 1-4) is only sufficient enough to teach the model to interpolate colors and will be an accurate enough reconstruction without learning the larger-scale features of an image. This is also why we removed the use of the "Bed of Nails" augmentation from the proposed alternative augmentations. The 2x2 pattern in "Bed of Nails" likely only teaches the model how to up-sample. These pixel-scale effects can only encourage the model to learn small-scale spatial features. Thus we believe it important to cause reconstruction to occur at a larger scale (e.g. 8x8 pixels). However, we believe destroying too much information be detrimental. In the original work [12], the authors found that 50% of information was the sweet-spot before learning became too difficult. We discuss potential extensions of this work in Section 5.3.

4.4 Ease of Use

Another important topic of discussion is the ease-of-use of each method. The BT method requires significantly more resources than the IR method, both in terms of hardware and data. Developing segmentation models already has a higher bar-of-entry due to the nature of the annotations, so a lightweight, easier to use SSL pre-training model is an important research direction. The IR method is significantly less demanding of computational resources when compared to BT. We chose our hardware configuration to meet the minimum described requirements for BT, however during IR training we never saturated the VRAM. We believe it possible to pre-train on a single GPU rather than four, unlike how the BT paper originally described. In addition, the IR pre-training time can be substantially faster (e.g. IR+CC training times). This would allow faster training iterations for tweaking and tuning the models.

5 Conclusion and Future Work

In this chapter, we present our conclusion, describe our contributions, and provide our thoughts for potential future work.

5.1 Conclusion

In this thesis, we presented two SSL pre-training approaches: Barlow Twins and Image Reconstruction. These methods make use of unlabelled image data for pre-training to save on annotation costs. Additionally, we proposed two alternative corruption augmentation stages based on Coarse Cutout and Hide-and-Seek. We demonstrate that the IR method can successfully improve the training quality of a segmentation model. With sufficient tweaking, we believe the IR method could be tuned to perform even better. The BT method proved difficult to use and was less effective than IR at improving our segmentation results. However, both methods failed to improve upon supervised ImageNet pre-training. Our results lead us to believe that image corruption exposes an an important characteristic in SSL pre-training for semantic image segmentation as both IR+CC and IR+HaS perform better with large corruption regions compared to IR+Orig and the a complete lack of image corruption in BT.

5.2 Contributions

This thesis provides the following contributions:

1. An evaluation and comparison of IR and BT methods in semantic image segmentation. The cutout variations of the IR method performed better than the BT method (0.023 to 0.054 mIOU difference on average), however the original IR method did not provide benefit over BT pre-training. Therefore, we recommend the cutout variations of the IR methods over BT.
2. Evaluation and comparison of three variations of the IR method. Of the alternative corruption augmentation stages (IR+Orig, IR+CC, and IR+HaS), the IR+HaS variation performed the best (0.125 mIOU above baseline), and is our recommendation.
3. An open-source implementation of the IR method and its variations. This is to help experiment with reproduction and confidence for any future work involving the IR method. The link to the repository can be found in Appendix B.

5.3 Future Work

In future work, we believe it would be pertinent to consider the following research directions:

- Further evaluate the best performing approach with different sizes of the cutouts. For example, making the cutouts a percentage size of the image may help the method to be invariant to resolution. It may also be beneficial to place the rectangle cutouts more selectively in correspondence to regions with high frequency information or other hard to learn image characteristics, or both.
- Investigate the effect of the cutout shape. For example, evaluate the effect of using non-rectangular cutouts (such as superpixels or image masks).
- Investigate the effect of both pre-training and downstream dataset size (ie. PASCAL VOC 2012 vs ImageNet 1K) on the quality of improvement of the SSL methods. The general sentiment in the field of ML is that more data results in a better model, but it would be good to test and determine at what point does dataset size provide diminishing returns for the SSL methods?
- Investigate the methods with different datasets. For example, an agricultural dataset, mainly consisting of green and brown. Distinction between desirable crops and undesirable weeds may be less clear due to the classes sharing similar color and shape characteristics.
- Extend the BT method to include the cutout-style augmentations. Following the intuition mentioned in [15], the corruption forces the model to focus on different parts of the input to solve the task and assists learning for semantic image segmentation. In a recent work, the authors have applied rectangular cutouts into BT augmentations among other modifications [25].
- Evaluate the methods using alternative common loss functions such as cross-entropy.

References

[1] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020.

[2] Xinlei Chen, Haoqi Fan, Ross B. Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *CoRR*, abs/2003.04297, 2020.

[3] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15750–15758, 2021.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255. IEEE, 2009.

[5] Nicki Skafte Detlefsen, Jiri Borovec, Justus Schock, Ananya Harsh, Teddy Koker, Luca Di Liello, Daniel Stancl, Changsheng Quan, Maxim Grechkin, and William Falcon. TorchMetrics - measuring reproducibility in PyTorch, February 2022.

[6] Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017.

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[8] William Falcon. PyTorch Lightning, 2019.

[9] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

[10] 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 (CVPR)*, June 2016.

[11] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs), 2016.

[12] Srivallabha Karnam. Self-supervised learning for segmentation using image reconstruction, 2020.

[13] Eugene Khvedchenya. PyTorch Toolbelt. `https://github.com/BloodAxe/pytorch-toolbelt`, 2019.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (Poster)*, 2015.

[15] Krishna Kumar Singh and Yong Jae Lee. Hide-and-seek: Forcing a network to be meticulous for weakly-supervised object and action localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3524–3533, 2017.

[16] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016.

[17] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, page 1485–1488, New York, NY, USA, 2010. Association for Computing Machinery.

[18] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 565–571, 2016.

[19] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning 2010*, pages 807–814, 2010.

[20] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[22] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2536–2544, 2016.

[23] Narinder Singh Punn and Sonali Agarwal. Bt-unet: A self-supervised learning framework for biomedical image segmentation using barlow twins with U-Net models. *Machine Learning*, 111(12):4585–4600, 2022.

[24] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017.

[25] G. Roggiolani, F. Magistri, T. Guadagnino, G. Grisetti, C. Stachniss, and J. Behley. On Domain-Specific Pre-Training for Effective Semantic Perception in Agricultural Robotics. In *International Conference on Robotics and Automation*, 2023.

[26] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[27] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[28] Zhou Wang, Eero P. Simoncelli, and Alan C. Bovik. Multi-scale structural similarity for image quality assessment. In *Conference Record of the Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 1398–1402, 2003.

[29] Yang You, Igor Gitman, and Boris Ginsburg. Scaling SGD batch size to 32k for ImageNet training. *CoRR*, abs/1708.03888, 2017.

[30] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stephane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12310–12320. PMLR, 18–24 Jul 2021.

Appendix A

Supplementary Diagrams

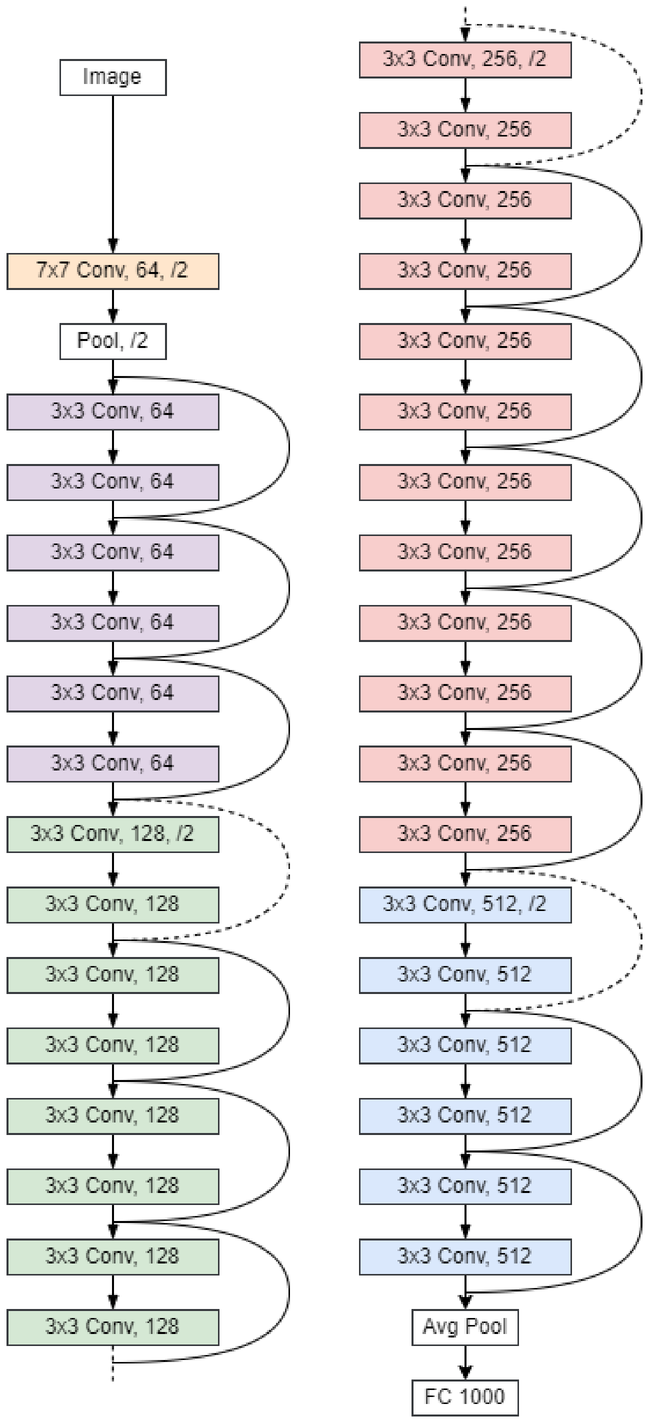


Figure A.1: Visualization of the ResNet34 architecture.

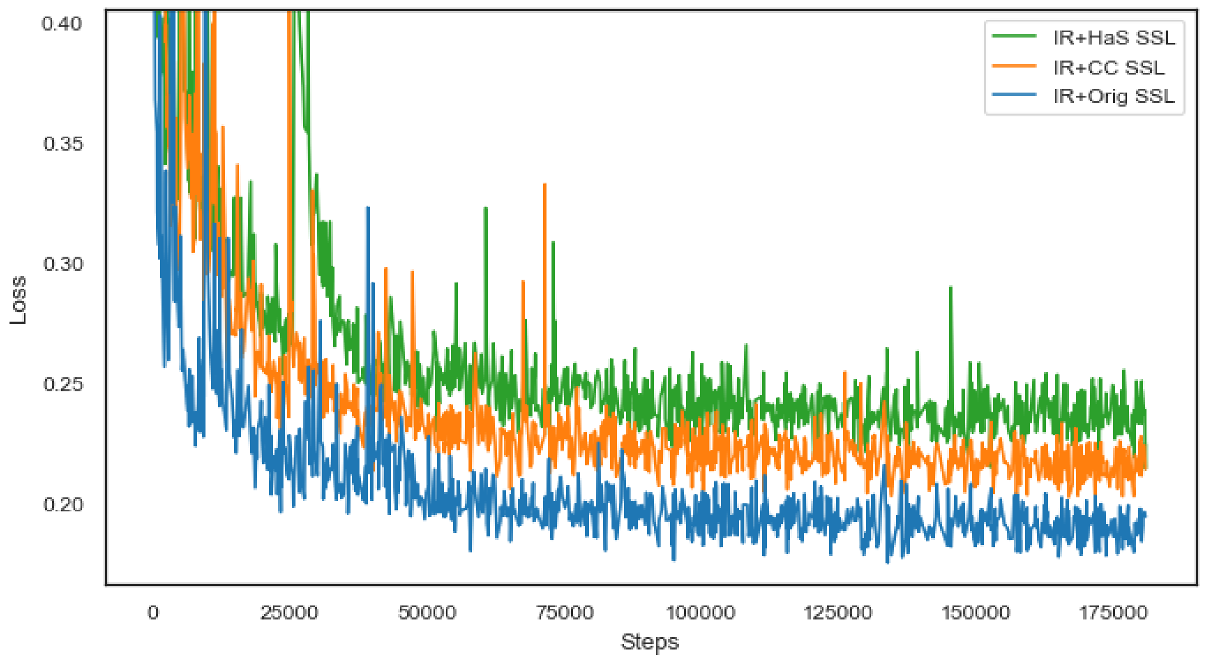


Figure A.2: Visualization of the training loss of the Image Reconstruction SSL proxy tasks.

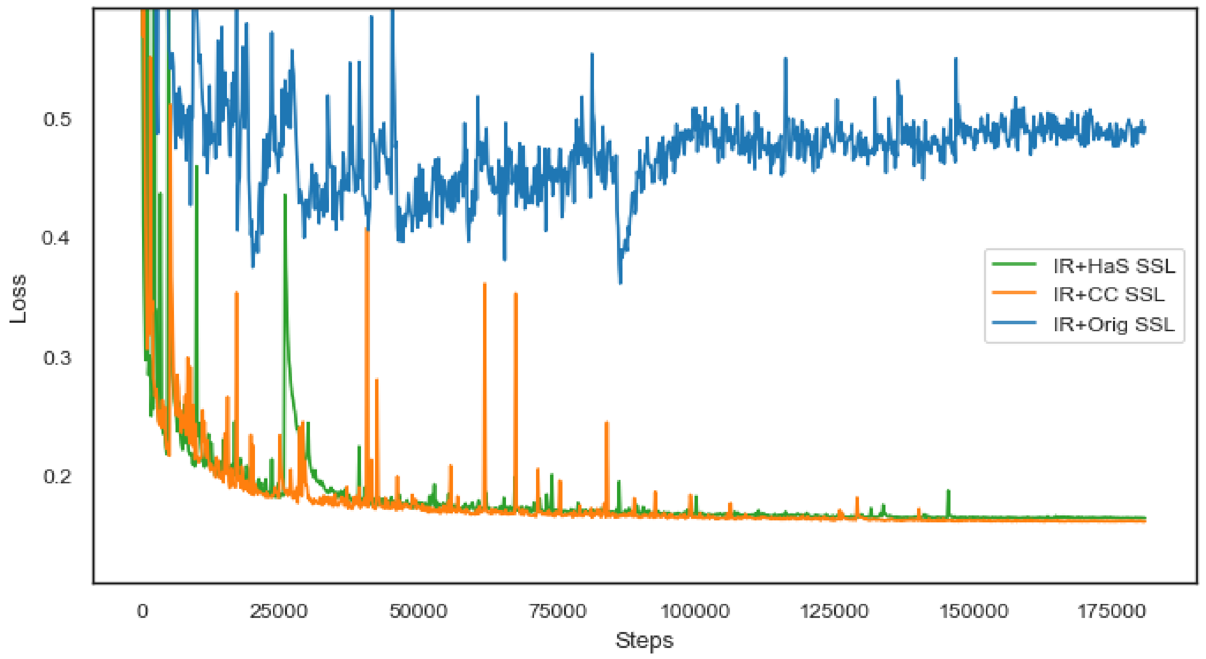


Figure A.3: Visualization of the validation loss of the Image Reconstruction SSL proxy tasks.

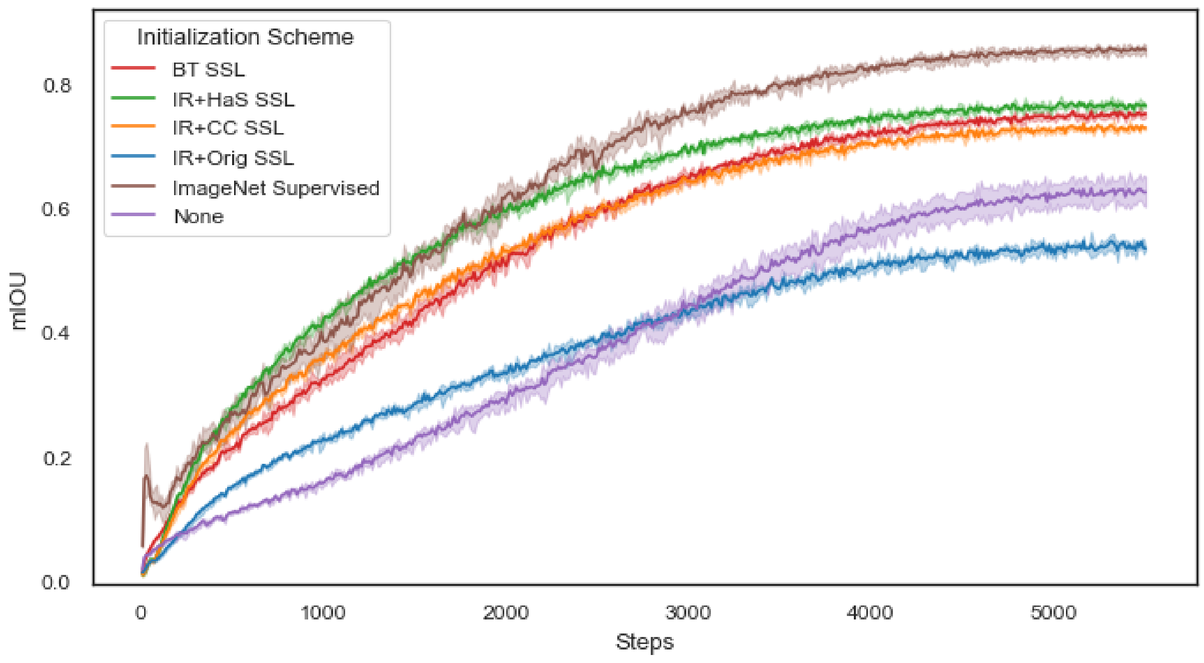


Figure A.4: Visualization of the training mIOU of the downstream segmentation tasks.

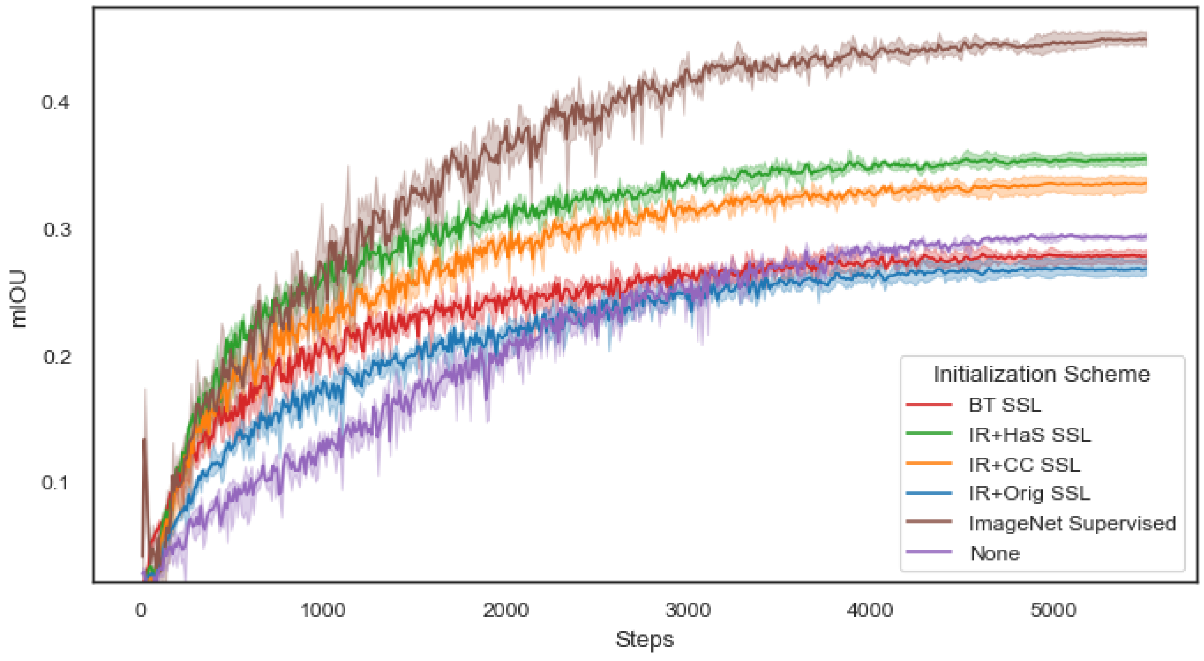


Figure A.5: Visualization of the validation mIOU of the downstream segmentation tasks.

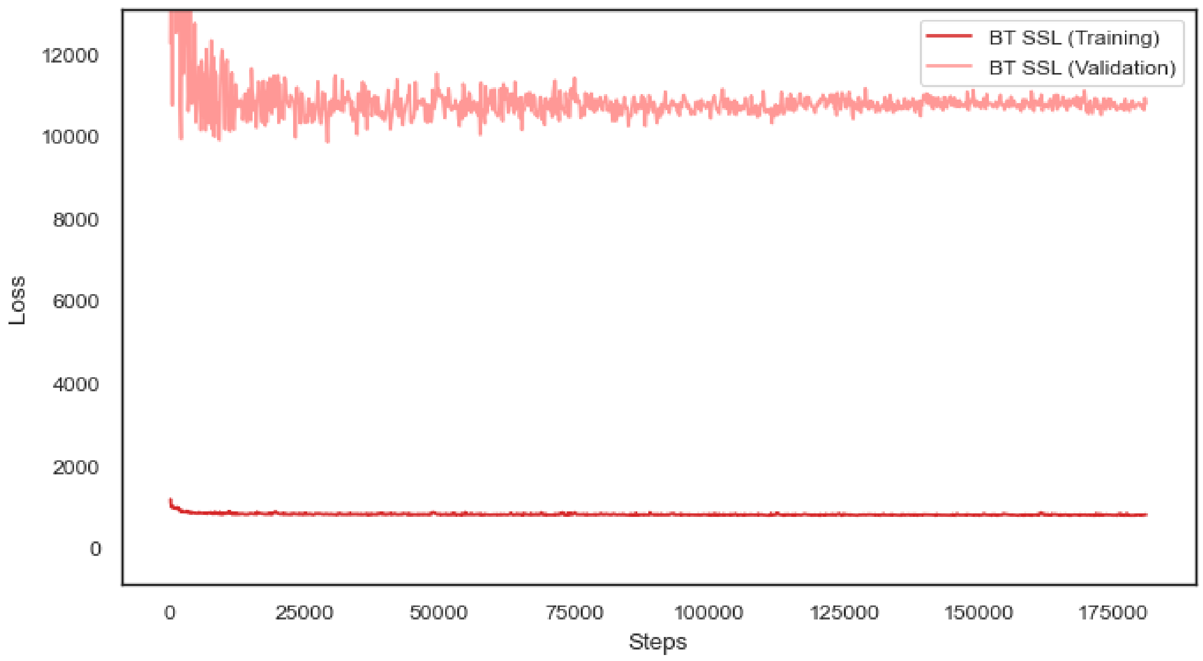
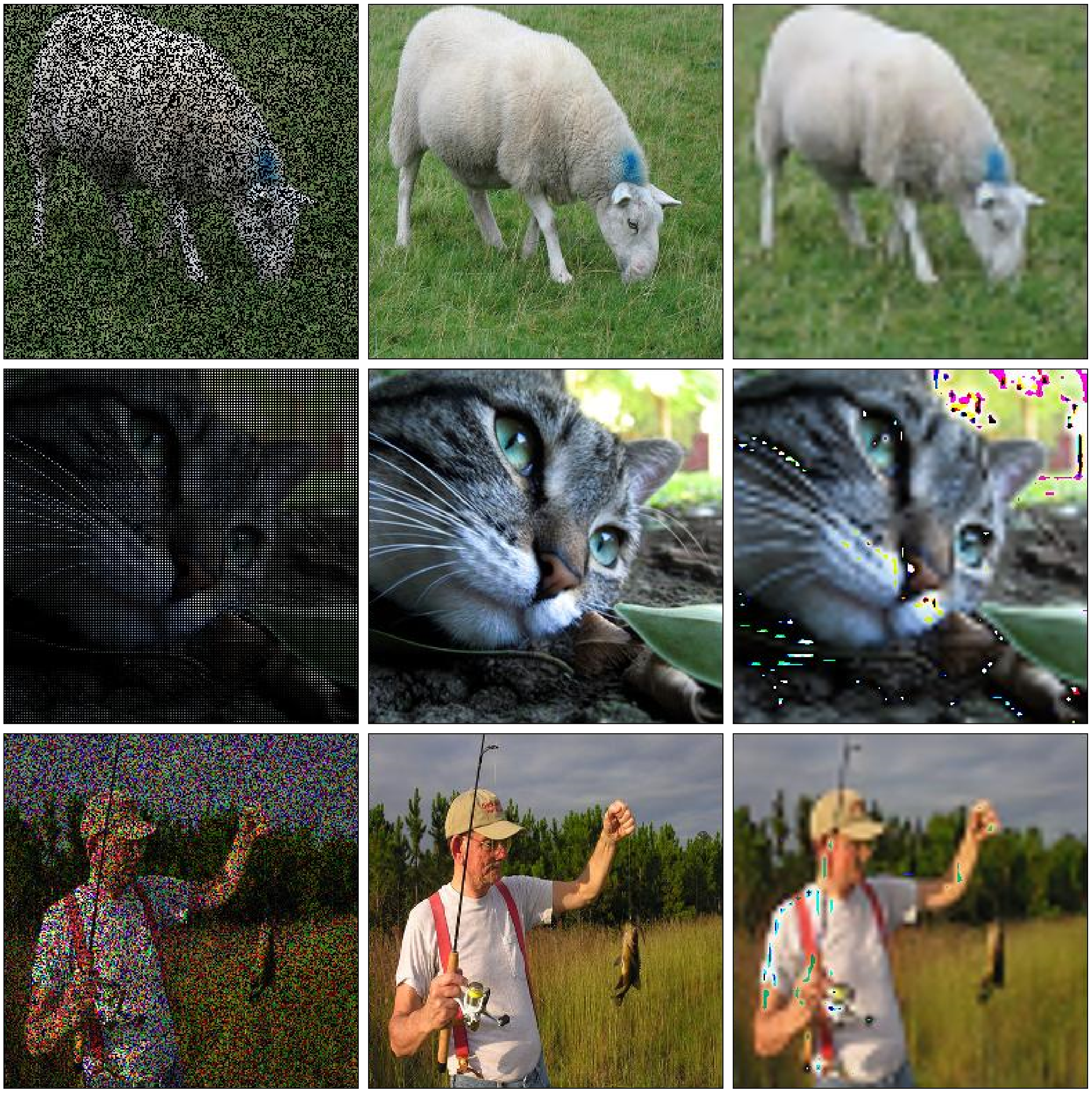
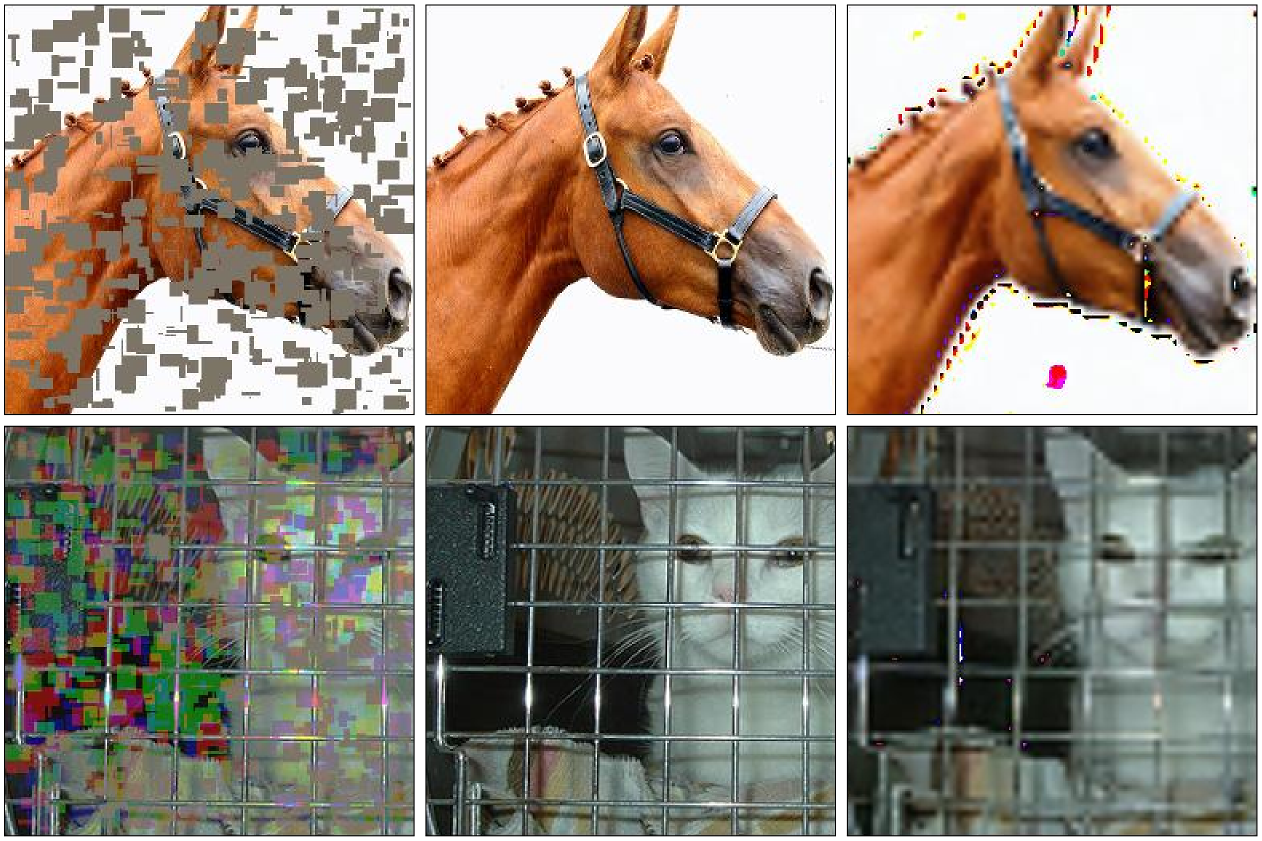


Figure A.6: Visualization of the training and validation loss of the Barlow Twins SSL pre-training task. Note the multiple orders of magnitude difference between the two curves.



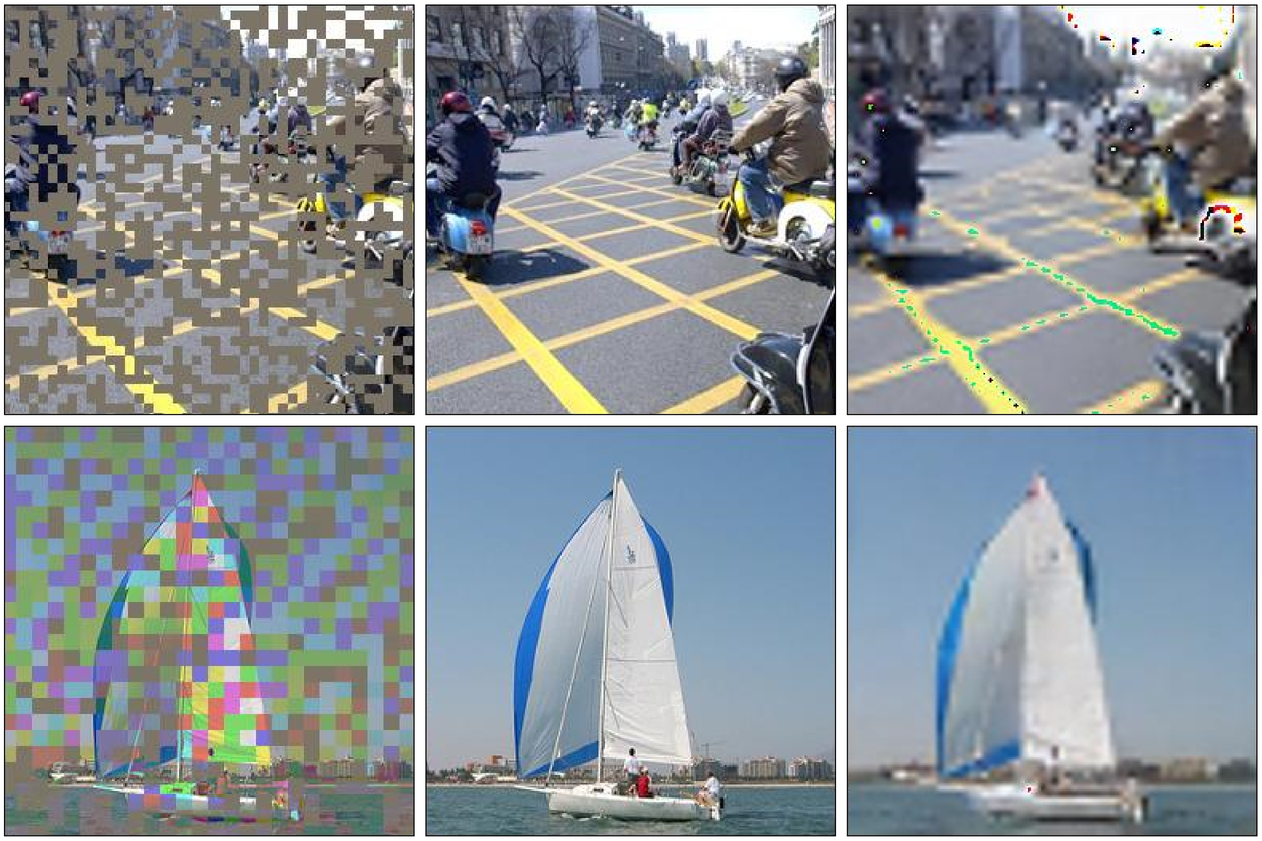
(a) Corrupted Input **(b)** Truth **(c)** Prediction

Figure A.7: Example images captured during IR pre-training with "Original" augmentations. Each potential corruption choice is shown. The artifacts present in the prediction are colors that exceed the normal 0.0 to 1.0 range. Note that the predicted image is half-resolution due to the architecture of the network.



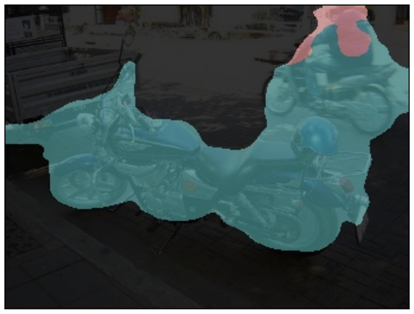
(a) Corrupted Input **(b)** Truth **(c)** Prediction

Figure A.8: Example images captured during IR pre-training with "Coarse Cutout" augmentations. Each potential corruption choice is shown. The artifacts present in the prediction are colors that exceed the normal 0.0 to 1.0 range. Note that the predicted image is half-resolution due to the architecture of the network.

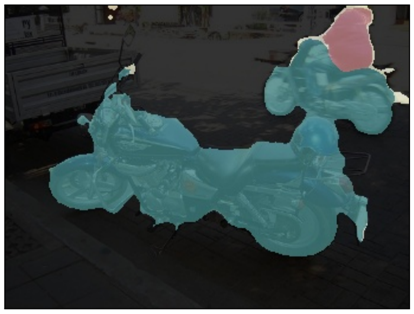


(a) Corrupted Input **(b)** Truth **(c)** Prediction

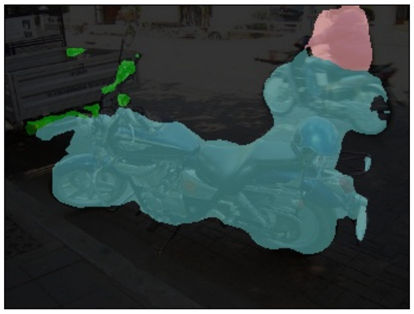
Figure A.9: Example images captured during IR pre-training with "Hide-and-Seek" augmentations. Each potential corruption choice is shown. The artifacts present in the prediction are colors that exceed the normal 0.0 to 1.0 range. Note that the predicted image is half-resolution due to the architecture of the network.



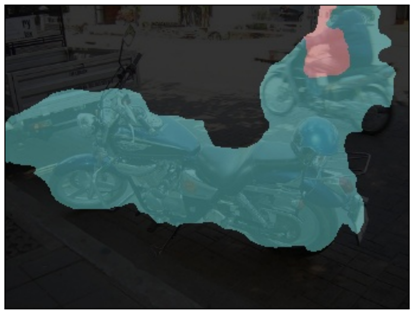
(a) No Pre-Training



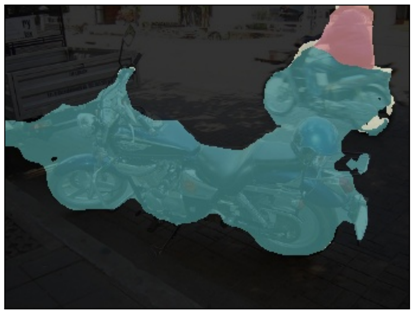
(b) ImageNet Supervised



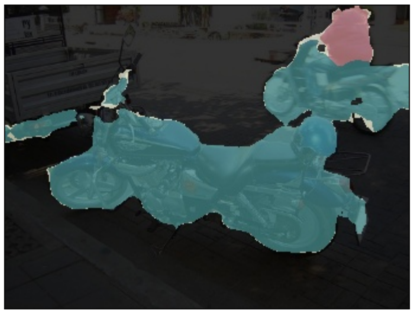
(c) BT SSL



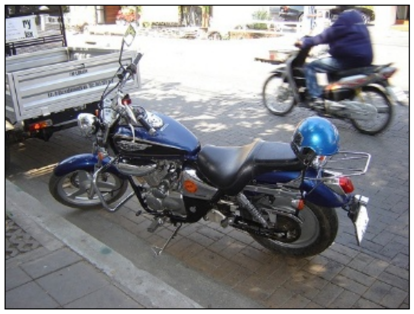
(d) IR+Orig SSL



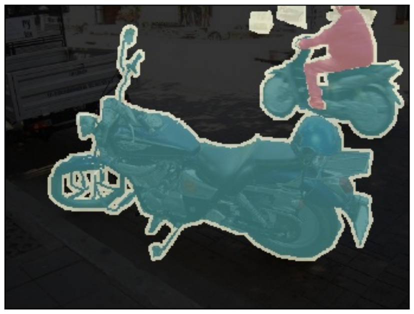
(e) IR+CC SSL



(f) IR+HaS SSL

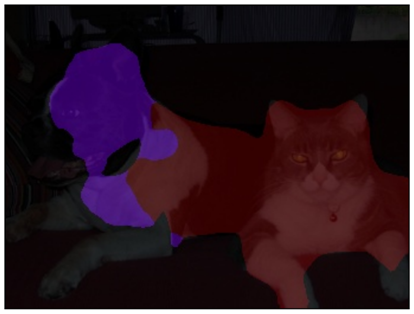


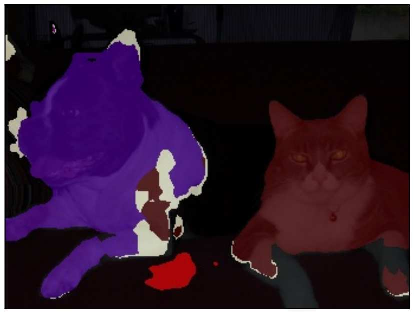
(g) Input Image

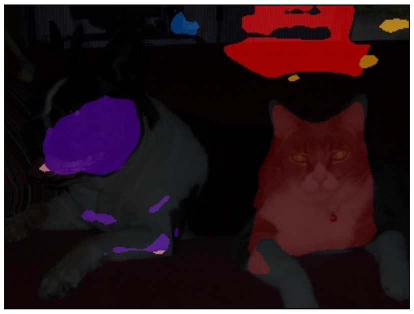


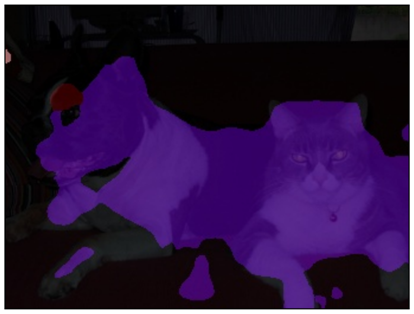
(h) Ground Truth

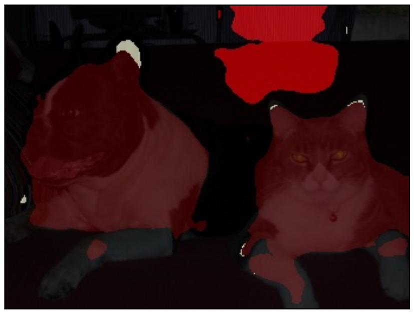
Figure A.10: An example of downstream segmentation output for each initialization scheme. Input image from [7].

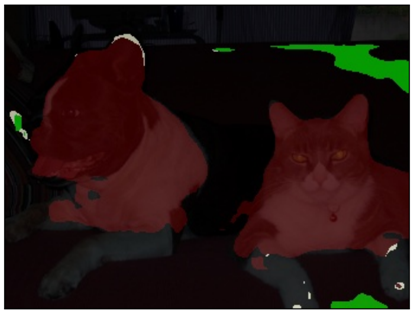

(a) No Pre-Training

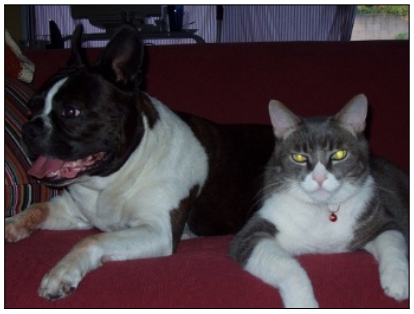

(b) ImageNet Supervised


(c) BT SSL


(d) IR+Orig SSL


(e) IR+CC SSL


(f) IR+HaS SSL


(g) Input Image

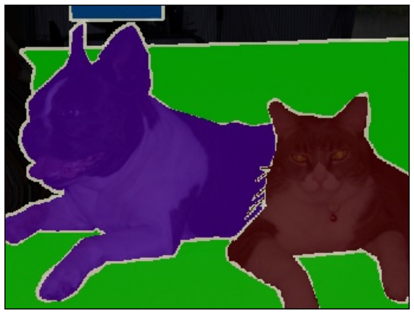

(h) Ground Truth

Figure A.11: An example of downstream segmentation output for each initialization schemes where all models struggled to produce a good output. Note that nearly all models failed to detect the sofa. Input image from [7].

Appendix B

Hardware, Software, and Model Configuration

B.1 Hardware

All models were trained on the University of Saskatchewan Copernicus Data Center. Our compute sessions requested the following resources, however the IR method likely can be trained with less compute resources.

- 4 NVIDIA V100 w/ 16 GB VRAM
- 200 gigs of RAM
- 32 CPU cores

B.2 Software

We used the following software platforms to develop the models in this thesis:

- PyTorch [21].
- PyTorch Lightning [8].
- PyTorch Toolbelt [13].
- Albumentations [1].
- TorchMetrics [5].
- We use the implementation of Adam found in [21].
- We use the implementation of Jaccard Loss found in [13].
- We used a SLURM environment for job scheduling.

B.3 Model Configuration

- The pre-trained ImageNet weights for ResNet34 are provided by [17]. It is pre-trained on ImageNet 1K, by resizing to 256x256 and cropping to 224x224. The documentation page claims results closely reproduce the results found in the original ResNet paper [10]. Visit the torchvision documentation for up-to-date details.
- We are using ResNet34 U-Net from [13] which makes use of the encoder described above. However, the decoder portion of the network is always initialized with default values.
- Metrics and Loss functions from [13] and [5]

B.4 Source Code and Data Availability

1. The PASCAL VOC dataset is available at: `http://host.robots.ox.ac.uk/pascal/VOC/`
2. Code is available at: `https://github.com/p2irc/ir-ssl-for-semantic-segmentation`