Scene Change Detection

Comparison of self-supervised and supervised semantic change detection using the contrastive learning approach

Master Thesis at Mathematical Sciences

Siddhant Som

Swaathy Sambath

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Scene Change Detection

Comparison of self-supervised and supervised semantic change detection using the contrastive learning approach

Siddhant Som
Swaathy Sambath

CHALMERS
UNIVERSITY OF TECHNOLOGY

Master Thesis at Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2022

Scene Change Detection

Comparison of self-supervised and supervised semantic change detection using the contrastive learning approach

Siddhant Som
Swaathy Sambath

Supervisor: Moritz Schauer, Department of Mathematical Sciences
Supervisor: Kristofer Norström, CEVT AB
Examiner: Daniel Persson, Department of Mathematical Sciences

Master's Thesis 2022
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Scene Change Detection

Comparison of self-supervised and supervised semantic change detection using the contrastive learning approach

Siddhant Som
Swaathy Sambath
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Scene Change Detection (SCD) identifies changes between the images taken at two different times, using pixel-level or point cloud approaches in most cases. Training a neural network for such a task requires a large number of images with annotated changes. Annotating changes is a slow, costly and time-consuming process. The state-of-the-art (SOTA) approach for SCD, like the DR-TANet paper, is based on transfer learning from large ImageNet datasets. This is a supervised technique and to overcome the challenges mentioned above, we introduce a self-supervised pretraining method with unlabeled datasets based on a existing D-SSCD approach that learns temporal-consistent representations of a pair of images. This project is an investigation of these approaches that can train and evaluate on available datasets through the use of a suitable loss function for the purpose of SCD. We compare results for different percentages of labeled data from different models and benchmark datasets such as Visual Localization CMU (VL_CMU_CD) and Panoramic change detection (PCD) datasets.

Keywords: Detection, pixel-level, slow, annotating

Acknowledgements

First and foremost, we want to express our gratitude to our CEVT AB supervisor, Kristofer Norstrom, for always taking the time to lead us through the project and for many interesting discussions.

We also want to thank everyone else at CEVT AB, notably the lost object team, for their assistance with our project.

Finally, a special mention about our supervisor at Chalmers University, Moritz Schauer for his guidance and support.

Siddhant Som, Swaathy Sambath Gothenburg, April 2022

List of Abbreviations

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

SCD	Scene Change Detection
SOTA	State-of-the-art
VL_CMU_CD	Visual Localization CMU
PCD	Panaromic Change Detection
SSL	Self-Supervised Learning
CNN	Convolutional Neural Networks
ReLU	Rectified Linear Unit
CosimNet	convolutional siamese metric network
DR-TANet	Dynamic Receptive Temporal Attention Network
SSCD	Self-supervised pretraining for Scene Change Detection
D-SSCD	Difference self-supervised pretraining for Scene Change Detection
DRTAM	Dynamic Receptive Temporal Attention Module
TAM	Temporal Attention Module
MoCO	Momentum contrastive
NLLLoss	Negative log likelihood loss

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

i,j	Indices for distribution network
t	Index for time step
x	Input for neurons

Parameters

h	Hidden layer
b	Bias
$E_{\sim}$	Expected value operator with respect to the distribution p for cross entropy loss
L_i	Log-likelihood
β	Moving average parameter
α	Learning rate
m	Aggregate of gradient
$e_{a,b}$	Relative positional encoding

Variables

W	Matrix of weights for hidden layer
Q, P	Cross entropy probability distribution
p_{y_i}	Represents the probability of the data point in NLLLoss
y_i	Represents particular class of data point in NLLLoss
f	Convolutional layer filters

w,h	Dimensions for Convolutional layer
Q	Queries
K	keys
V	Values
d_k	Dimension key

Contents

List of Figures

List of Tables

1

Introduction

1.1 Scene change detection

Scene Change Detection (SCD) compares images taken at two different times to identify changes between them. SCD has various practical applications such as self-driving cars, landscape change detection, medical diagnosis, urban landscape detection, and visual surveillance. SCD is considered as one of the most important problems in the field of computer vision. Convolutional Neural Networks (CNNs), which have been widely used in computer vision applications perform well to satisfy the objectives of SCD. ResNets and VGG are state-of-the-art (SOTA) CNNs that extract great feature maps to identify critical parts of an image. Such CNNs can also perform semantic segmentation to depict areas of change. For instance consider Figure 1.1: the task is then to construct a SCD model using these two images to show changed regions which is the top right image t_1, different from the regions in the top left image t_0. The bottom row of the image represents the labelled ground truth images and the result obtained. The challenges in SCD include coping with the various levels of noise in images as well as accounting for changing levels of illumination across images. Another major challenge is the labeling of data for training a model, which is an expensive and time consuming process. For example, on average to annotate each pair of image it takes around 20 minutes to 156 minutes [1]. To address these issues, we introduced self-supervised pretraining approaches that outperform supervised ImageNet pretraining under limited labeling.

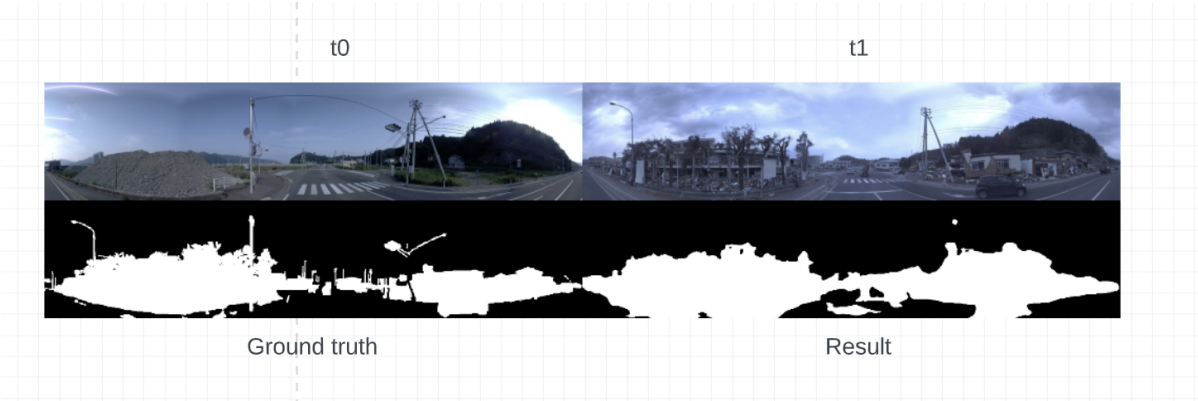


Figure 1.1: SCD illustration the images in the top row are from two different times. The bottom row compares the obtained result to the ground truth.

1.2 Self-supervised learning

Self-Supervised Learning (SSL) is a method used to solve challenges faced by models reliant on labeled data. SSL is a deep learning technique to train a model on data without any labels. In this method, a model trains itself to understand various patterns in the given dataset. For example, in a SCD dataset with no labels, the model would try to ensure that similar images would have similar representations. This self-supervised model can then be used for downstream tasks such as classification or segmentation. The primary objective of this project is to study SSL for semantic segmentation, which classifies each pixel in an image. The obvious advantage is the reduction in reliance on labelled data, which is what we attempt to exploit in this project. Annotation or labelling is a very expensive and time consuming process in general, and a reduction reliance on annotation is highly valuable.

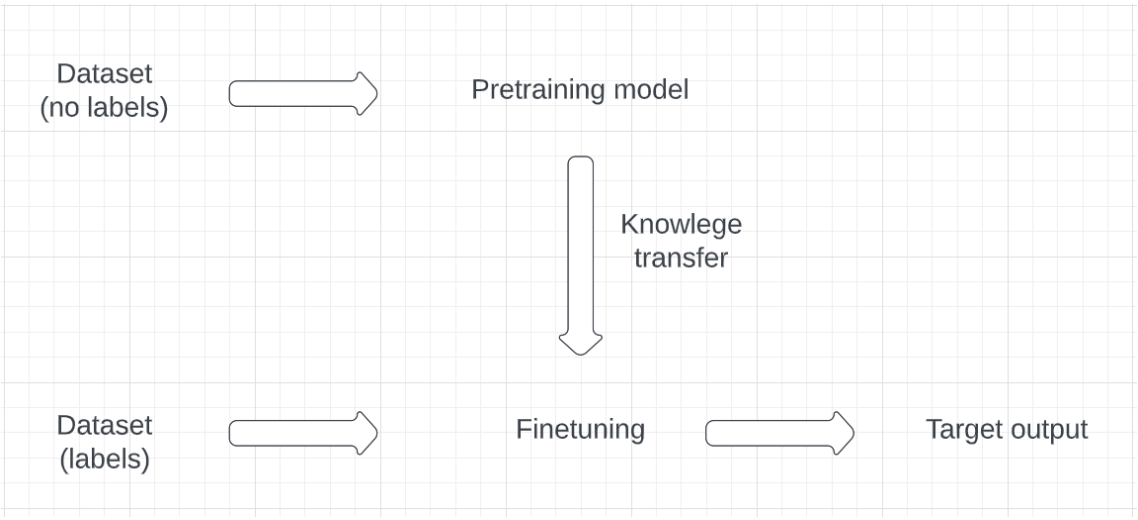


Figure 1.2: Self-supervised learning representation.

SSL pretrained models are not only useful for training on curated datasets like ImageNet, but they are also great few shot learners, attaining 75.1 percent with only 10% of ImageNet. On several downstream tasks, our model outperforms a supervised model trained on ImageNet, proving the benefits of SSL pretraining, even when applied on uncurated datasets [2].

1.3 Problem statement and purpose of the study

The application of this thesis is to develop a method to automate the process of identifying foreign objects in a given picture or scene. Furthermore, public SCD datasets are accessible for the experiments. The aim of the work is to answer or address the following questions or tasks:

- Implementation of self-supervised methods on a SOTA SCD model on publicly available datasets.

- Can the results of a SOTA SCD model on benchmarked datasets such as PCD and VL_CMU_CD be improved further, or is it possible to attain results of considerable accuracy even with a reduction in supervision?

- Identifying a pretraining method which utilizes a suitable loss function, to reduce the number of annotations necessary for obtaining the desired levels of accuracy or any other metrics used.

- Compare different pretraining methods that include pretrained ImageNet (DR-TANet), self-supervised MOCO, self-supervised SimCLR, self-supervised MoCO with pretrained ImageNet weights as starting point, self-supervised SimCLR with pretrained ImageNet weights as starting point.

- Is there an improvement in results of SCD when a model is pretrained on large datasets?

- Does the use of pretraining improve performance and reduce their cost of annotation?

The study's major goal is to reduce the cost of annotation involved in labeling pairs of images, used for SCD. This is beneficial for any semantic change detection and our thesis specifically supports CEVT AB where we do our thesis, who use SCD to automate the process of locating the missing object and identifying what the missing object it is. The most significant challenge in training a model for SCD at the moment is the cost of annotation needed in creating a desirable dataset. Finding ways to mitigate said cost, is highly valuable.

1.4 Contribution of the study

Neural networks require huge amounts of labeled data for training [24]. Finding huge amounts of labeled data is difficult, and labeling existing data is a time-consuming and expensive process, especially in a domain such as SCD. The pretraining and finetuning methods explored in this study perform well even with limited labeled data, which is the main contribution of this study.

1.5 Structure of the paper

This paper consists of five chapters. While Chapter 1 introduced the problem and had information on the purpose of the study, Chapter 2 gives an insight into the background and surveys the literature concerning SCD. Chapter 3 explains our approach. Chapter 4 shows the results obtained using our approach. Chapter 5 we discuss about the study and concludes the report.

2

Background and literature survey

2.1 Neural networks

Neural networks are models used in deep learning for tasks such as classification and regression. They map an input to a desired output and thus serve as a mapping function. A neural network generally consists of several layers, with each layer consisting of smaller units called neurons, which transform an input value. A single layer feed-forward network consists of an input layer and an output layer, while a multi-layer feed-forward network consists of at least one layer called the hidden layer between the input and output layer. Neurons in one layer are connected to neurons in the next layer and any input fed into the neural network passes through all the layers before an output is produced. Every neuron has an associated weight – an indication of the importance of the output of that neuron – which is adjusted as the network trains. Training a neural network is, in essence, finding the weights that the network will correctly map the input to the desired output. The output of a hidden layer is given as:

$$h = A(W^T x + b) \tag{2.1}$$

where W is a matrix of weights, x is the input vector, b is the bias vector and A is the activation function. Layers in which every neuron is connected to every neuron in the previous layer are called *fully-connected* layers. In Figure 2.1, the dimensions of W would therefore be 4x5.
The training process of a neural network involves the use of a loss function, back-propagation and an optimizer. These features are described in the following sections.

2.1.1 Loss functions

The output of a neural network is called a prediction. A loss function measures the quality of the predictions by comparing the predicted output with the desired output. A lower loss is an indication of a better prediction. As a model learns more during the process of training, its parameters get updated and loss reduces. Selecting a loss function depends on the application. A popular choice is the *cross entropy* loss, which indicates the difference between two probability distributions. It is given as:

$$H(P, Q) = -E_{x \sim P} \log(Q(x)) \tag{2.2}$$

Where Q and P are two different probability distributions. Another popular choice is the *Negative log likelihood loss (NLLLoss)* which is given as:

$$L_i = -\log(p_{y_i}). \tag{2.3}$$

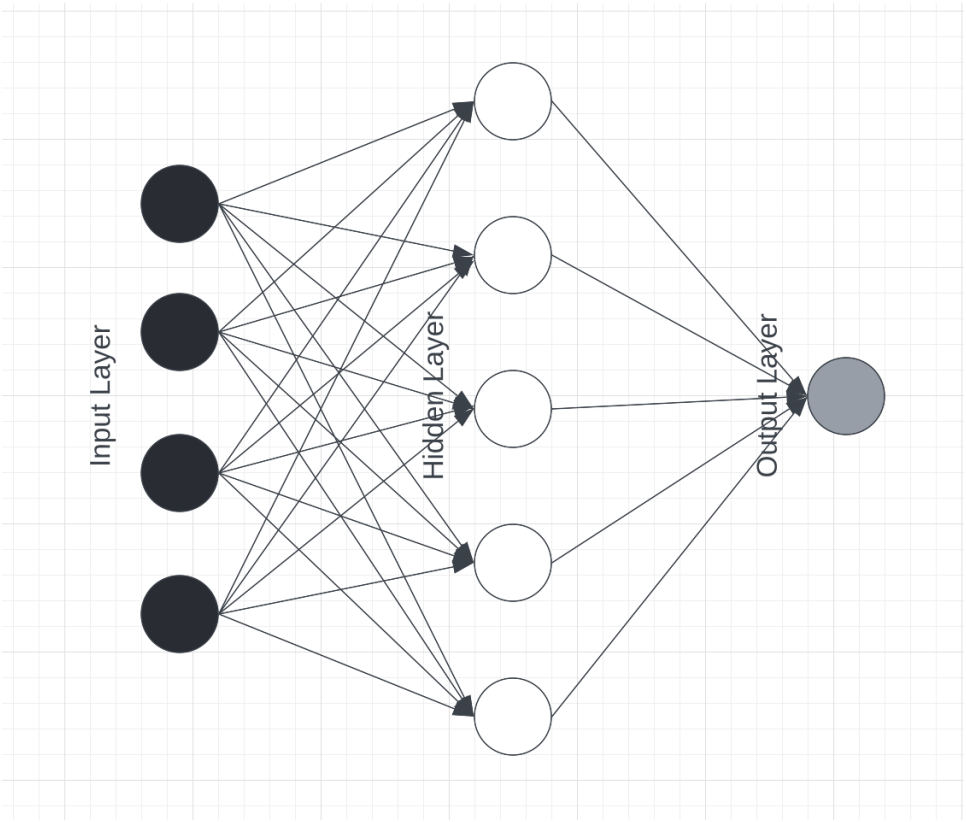


Figure 2.1: A neural network with single hidden layer and each neuron in input layer is connected to each of the five neurons in hidden layer. The outputs of neurons pass through an activation function are combined in the output layers.

where p_{y_i} represents the probability of the data point y_i belonging to a particular class and L_i is called the negative log-likelihood.

2.1.2 Backpropagation

Backpropagation involves the computation of the partial derivatives of the loss with respect to each of the parameters (weights and biases) in the neural network. It utilizes the chain rule to compute the derivatives backward. The computed derivatives are then used for optimization, by trying to minimize the difference between the obtained output and the correct expected output, to adjust the parameter values accordingly.

2.1.3 Optimization

An optimizer tries to find the parameters of the neural network which minimize the loss function. The most popular optimizer is the Adam optimizer, which is an extension to stochastic gradient descent and its equations are given as:

$$w_{t+1} = w_t - \alpha m_t. \tag{2.4}$$

where:

$$m_t = \beta m_{t-1} + (1 - \beta) \left[\frac{\partial L}{\partial w_t} \right]. \tag{2.5}$$

α is the learning rate, m_t is the aggregate of gradients at time t, m_{t-1} is the aggregate of gradients at time $t-1$, w_t represents the weights at time t, w_{t+1} represents the weights at time $t+1$, L is the loss function and β is the moving average parameter whose value is a constant and is usually kept at 0.9. Adam is different from classical stochastic gradient descent. In stochastic gradient descent, the learning rate remains the same for all the parameters. However, this is not the case with Adam, which combines the benefits of two other optimization algorithms called Momentum and RMSProp. It essentially adjusts the learning rate for each parameter separately.

2.1.4 Activation functions

An Activation Function influences whether or not a neuron is activated i.e whether or not the neuron's output contributes to the output of the network. It uses simpler mathematical operations to determine whether the neuron's input to the network is essential or not throughout the prediction step and whether it needs to be transformed. It is essentially a nonlinear mathematical function applied to the raw output value of a neuron to give the final output. The purpose of an activation function is to add non-linearity to the neural network. The simplest form of activation functions are ones that do not transform the data at all. However, when such simple functions are used, the model fails to learn complex mapping functions. This is why non-linear activation functions are preferred as they enable the model to learn more complex structures in the data. The most commonly used activation function is the rectified linear unit (ReLU) activation function, which is given as:

$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases}$$

A huge advantage of ReLU is its simplicity. This makes it easier to optimize the model. Another important advantage of ReLU is that models trained using it usually do not suffer from the problem of vanishing gradients, as the gradients remain proportional to the activations of the neurons. Another popular activation function is the sigmoid function given as:

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{2.6}$$

It transforms a value to the range [0, 1].

2.2 Convolutional neural networks

Convolutional neural networks (CNN) are neural networks that can deal with image data. A CNN can identify relevant features in an image and can classify it based on those features. In a feed-forward neural network, every neuron in one layer was connected to every neuron in the previous layer. However, this is not the case in a

CNN and weights are shared between neurons leading to a reduction in the number of parameters and complexity. Such layers where every neuron is not connected to every neuron in the previous layer are called *convolutional* layers. An image is made up of pixels and has several channels. Color images have three channels - RGB while grayscale images only have a single channel. Each channel has pixel values, which are used by a CNN to learn about the features in the image. Images are passed as inputs in the form of *tensors.*, which are essentially n-dimensional arrays. Images are passed as 4-dimensional tensors, in the form of (batch size, channels, height and width). A CNN reduces the images to a form that is easier to process without

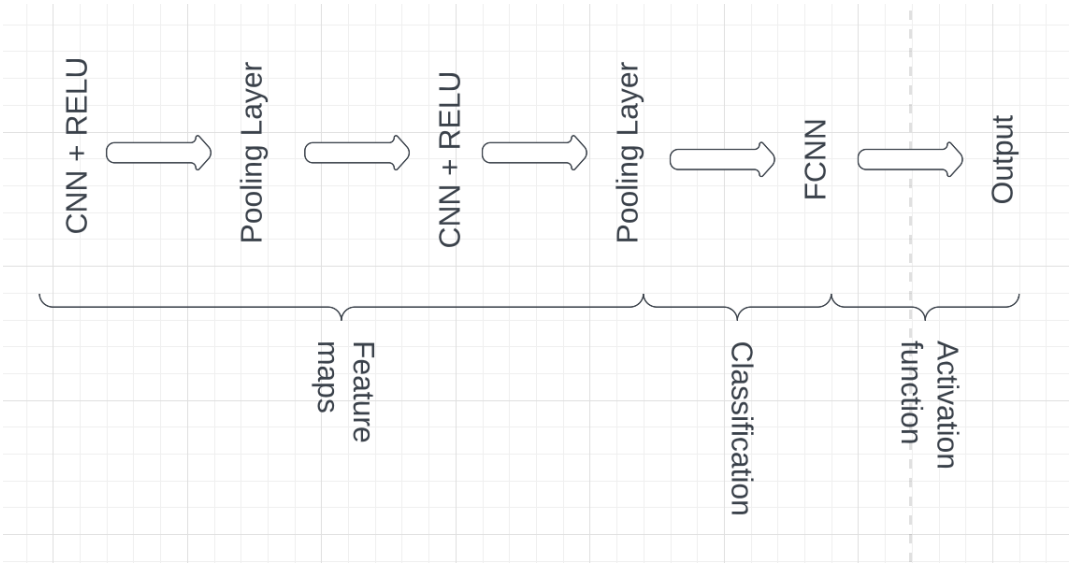


Figure 2.2: A simple CNN architecture

losing features that are critical for getting a good prediction. Every convolutional layer has many *kernels* or *filters*, which are essentially square matrices. Every kernel performs the process of convolution – which is an element-wise matrix multiplication and summation on the input channels. A single kernel convolves over each of the channels of the input image, and then the corresponding values of the results are added up. In other words, the pixel values in the image are transformed based on the values in the filter. Therefore, the number of output channels of a layer is equal to the number of kernels in the layer. Kernels are usually 3x3 or 5x5 and result in the down sampling of the input. The size of the output from a convolution operation depends on the number of positions on the input image, where the filter can be placed. If the input image is denoted by h, and a filter is denoted by f, then the process of convolution can be written mathematically as:

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k f[j,k]h[m-j,n-k]. \tag{2.7}$$

where G is the resulting matrix and m, n mark its rows and columns. At the boundary, the formula is slightly modified. Figure 2.1, as the input goes through various convolutional layers, and more and more convolutions are performed on it, the network learns the important features present in the image. The number of

parameters p (without bias) in a convolutional layer with f filters, of dimensions $w \cdot h$ receiving i channels as input, are computed as:

$$p = (w \cdot h \cdot i)f. \tag{2.8}$$

Similar to convolutional layers, *pooling* layers reduce the spatial size of the convolved feature to decrease the computational power required to process the data through dimensionality reduction. There are two types of pooling operations - max pooling and average pooling. Max Pooling returns the maximum value from the Kernel-covered region of the picture. Average Pooling, on the other hand, returns the average of all the values from the region of the picture covered by the Kernel.

2.3 Transformer

The transformer is a neural network architecture which was proposed in the paper "Attention is all you need" by Vaswani et al [12]. It is considered to be a state-of-the-art model in the field of natural language processing. However, the structure followed by a transformer can also be used for other applications, as was done in the case of scene change detection. A transformer consists of 2 parts – an encoder and a decoder.

2.3.1 Encoder

The encoder first converts the input into a form which can be processed by a computer. A computer can only work with vectors and matrices. This is referred to as the embedding of the input. Then the embedding is processed further before being passed into the decoder. In the case of scene change detection, the input images are converted into vectors and matrices, which are then processed by the neural network.

2.3.2 Decoder

The embedding may be further processed in the decoder before being converted into the output form. If the problem is one of natural language processing, then the output will be the output word or sentence. During scene change detection, the embedded vectors and matrices of the original images are converted into change detection masks indicating regions of change.

2.4 Related work

2.4.1 Semantic scene change detection

The SCD process involves a series of steps, such as the selection of training data, image pre-processing, image feature extraction, construction of the SCD algorithm, and evaluation of the model performance. Detecting regions of change has always been of interest to computer-vision researchers. Given two images, SCD identifies

semantic changes between the different times. Early SCD methods mostly utilize the difference between feature maps, with the aid of statistics and probability theory [3]. With the development and progress in the field of deep learning, the most popular choices generally involve the use of CNNs. CNNs are neural networks that can deal with image data, extract important information from images and perform various tasks. SOTA models generally follow an encoder-decoder architecture, with the encoder being a SOTA CNN such as VGG or ResNet, which are then used for tasks such as image classification or semantic segmentation [4] [5]. Throughout the encoder, paired images are downsampled into feature maps of different scales. The extracted features use information from the paired images. Siamese Networks are neural networks that compute changes between a pair of inputs. Guo et al [6], introduce the convolutional siamese metric network (CosimNet) which can deal with noisy changes across the images, as well as changes in illumination. They use the idea of contrastive loss. Sakurada et al [7], introduce the CDNet which is based on U-Net and accounts for differences in camera viewpoints during SCD [8] . Since supervision for SCD is expensive, Sakurada et al [7], introduce a novel semantic SCD method that involves only weak supervision. They propose an architecture mitigates the issue of difference in camera angles in SCD by using Siamese networks [9]. Chen et al [10], then introduce the dynamic receptive temporal attention network (DR-TANet) which is both lightweight and efficient and guarantees excellent performance with fewer parameters. DR-TANet will be explained in further detail in the following section. Since a model for SCD generally involves the use of a CNN as an encoder, it is also possible to use CNNs that have been pretrained on a large dataset such as ImageNet with millions of images. Raghavan et al [11], introduce a novel self-supervised pretraining method for SCD, which solves the issue of domain shift that occurs when SCD methods use transfer learning from ImageNet datasets. Their method is called Difference Street Scene Change Detection (D-SSCD) and it utilizes absolute feature differencing. D-SSCD learns important features in an image while separating them from the redundant ones.

2.4.2 Correlation of features

Correlation maps present the similarity between a region of the image at time t_0 and a fixed square search area around the same position at time t_1. The correlation mechanism can obtain a relatively good performance, but there is still room for improvement. In the field of natural language processing, the mechanism of self-attention – introduced by Vaswani et al [12], is used widely in transformer architecture. Self-attention helps to learn large-scale dependencies. Mathematically, self-attention is described as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.9}$$

where Q,K and V represent the query, key and value vectors.

$$\begin{aligned} Q &= W_Q X, \\ K &= W_K X, \\ V &= W_V X. \end{aligned} \tag{2.10}$$

X represents the input embedding, which is a vectorial representation of the input.

Huang et al.[13] introduce and apply criss-cross attention in a semantic segmentation task, which is an efficient method to harvest contextual information. Zhang et al [14] propose a new feature extractor that is built with self-attention to integrate the inferred compositional structure among visual elements in a local area. Ramachandran et al [15] show that self-attention can be a useful replacement for spatial convolutions due to its fewer parameters and floating-point operations.

2.5 Dynamic Receptive Temporal Attention Network

Chen et al [10] then introduce the concept of temporal attention in the DR-TANet paper, which is more computationally efficient than self-attention. This paper also explores the impact of the dependency scope size on the performance of the model, introduces the temporal attention module (TAM) to utilize the idea of temporal attention and then further improves upon TAM to introduce the dynamic receptive temporal attention module (DRTAM), which is more lightweight and efficient.

2.5.1 Architecture

DR-TANet is a highly efficient and lightweight SOTA model for SCD. It has an encoder-decoder architecture, including a dynamic receptive temporal attention module (DRTAM), which will be described in the following subsection. The encoder in the architecture is ResNet18, which has 18 layers. Since a scene change detection model has two images to compare, the encoder is split into two channels – one for time t_0 and the other for time t_1 – to perform the extraction of features. These extracted features are then passed into the DRTAM to find similarities between them at corresponding positions. Then the attention maps generated are passed into the decoder to perform upsampling and create a change mask that indicates the regions of change.

2.5.2 Temporal Attention Module

The idea of temporal attention is to calculate the dependency of a pixel in the t_1 channel, with pixels in the t_0 channel, in a given scope. One can use self-attention to compute the dependency of a pixel in the t_1 channel with every pixel in the t_0 channel. However this is computationally expensive and not representative of the relation between pixels and therefore it is more useful to compute dependencies in a given range or scope. This implies that an example pixel (i, j) of the feature maps

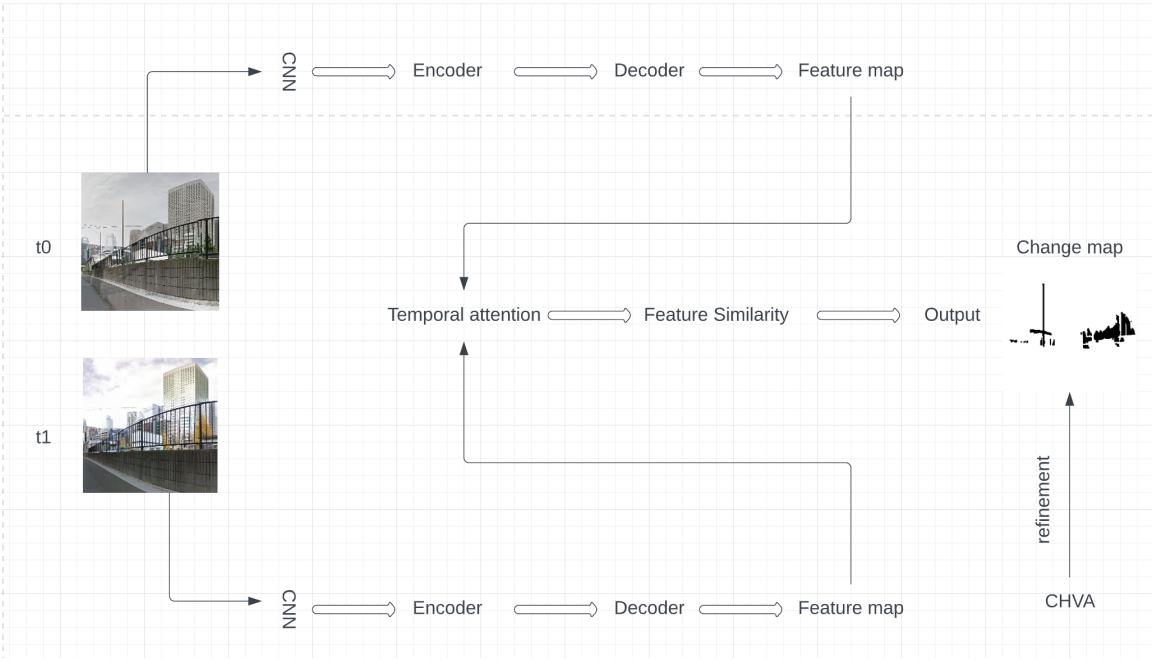


Figure 2.3: DR-TANet architecture

at channel t_1 will be dependent only on a fixed scope of files in channel t_0. The fixed scope size could be 1x1, 3x3, 5x5 or so on. Multi-head temporal attention is used where the feature maps are divided into groups and temporal attention will be performed on each group.

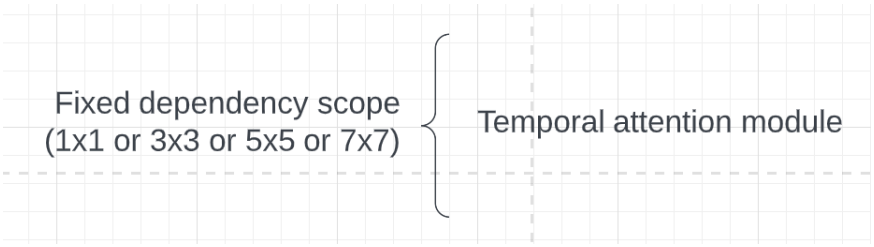


Figure 2.4: Fixed scope size

Formally temporal attention is defined as:

$$\begin{aligned} q_{ij} &= W_q {X_{ij}}^{t_0}, \\ k_{ij} &= W_k {X_{ij}}^{t_1}, \\ v_{ij} &= W_v {X_{ij}}^{t_1} \end{aligned} \tag{2.11}$$

$$A_{ij} = \sum_{a,b \in \mathbf{N}(i,j)} softmax_{ab}(q_{ij}^T(k_{ij} + e_{ab}))\nu_{ab} \tag{2.12}$$

where W_q, W_k, W_v are the weight matrices for generating the query, key and values at pixel position (i,j). N(i,j) represents dependency scope-size. $e_{a,b}$ is the relative positional encoding and the query calculates the inner product with the sum of key and positional encoding. Based on temporal attention, the temporal attention module (TAM) is constructed, which consists of four layers where each layer takes feature maps at two temporal channels as input and derives the attention map based on the temporal attention mechanism. The former calculated attention maps are downsampled and concatenated with the subsequent attention maps. every attention map will be inserted once more into the decoder through skip connections. This prevents information loss during the whole upsampling feature flow.

2.5.3 Dynamic Receptive Temporal Attention Module

TAM has a fixed dependency scope throughout the 4 layers. The only difference in DRTAM is that the dependency scopes are reduced from layer to layer, leading to a reduction in the number of parameters. In the vision models, during the downsampling path of the encoder, feature maps first identify the low-level properties (color, edges, etc.) and then gradually the high-level properties (texture, shapes, objects, etc.) will gain more focus. When feature maps gather the low-level features, at the initial temporal attention layer, the dependency-scope size is 7x7, then it reduces to 5x5, 3x3 and 1x1 in the following layers. Mathematically, it is given as:

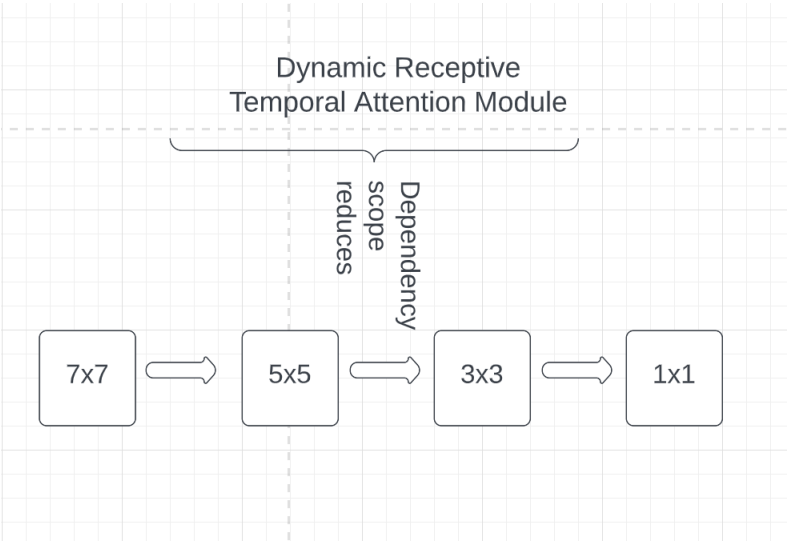


Figure 2.5: Dependency scope size varying from the first till the fourth layer.

$$TA_{ab} = \text{softmax}_{ab}(q_{ij}^T(k_{ij} + e_{ab}))\nu_{ab}$$

$$A_{ij}^1 = \sum_{\mathbf{N}_1(i,j)=7\times7}^{a,b\in\mathbf{N}_1(i,j)} TA_{a,b}, A_{ij}^2 = \sum_{a,b\in\mathbf{N}_2(i,j)}^{(\mathbf{N}_2(i,j)\text{ is }5\times5)} TA_{a,b}$$

$$A_{ij}^3 = \sum_{\mathbf{N}_3(i,j)=3\times3}^{a,b\in\mathbf{N}_3(i,j)} TA_{a,b}, A_{ij}^4 = \sum_{\mathbf{N}_4(i,j)=1\times1}^{a,b\in\mathbf{N}_4(i,j)} TA_{a,b} \tag{2.13}$$

2.6 Difference Street Scene Change Detection

This section describes an effective pretraining strategy - an alternative to the standard ImageNet pretraining technique - where a network is trained with a suitable loss function, and then parameters of the backbone of the said network are transferred to the backbone of DR-TANet. This is done to give the DR-TANet encoder a better starting point for the training process and to reduce its dependence on annotated data.

2.6.1 Architecture

The network used is the Simple Framework for Contrastive Learning of Visual Representations (SimCLR) architecture which was introduced by Chen et al [10]. The framework aims to learn effective representations for visual inputs without any supervision. It consists of an encoder and a projection head – which is essentially a feed-forward neural network. The input images are augmented before being passed into the framework. The encoder is chosen to be ResNet18 for simplicity. It extracts representations from the augmented images and then the projection head maps those representations to the desired latent space. When the two images are treated as augmentations of each other and passed into the framework, the technique is called *self-supervised pretraining for change detection*. The framework learns the changed region by maximizing cross-correlation between the two images. The other variation is the *difference self-supervised pretraining for change detection*, where absolute feature-differencing is used to learn the changed region directly. Figures 2.6 and 2.7 illustrate the two methods. The loss function used is called Barlow-twins, which is described in the following subsection. The images at t_0 and t_1 are con-

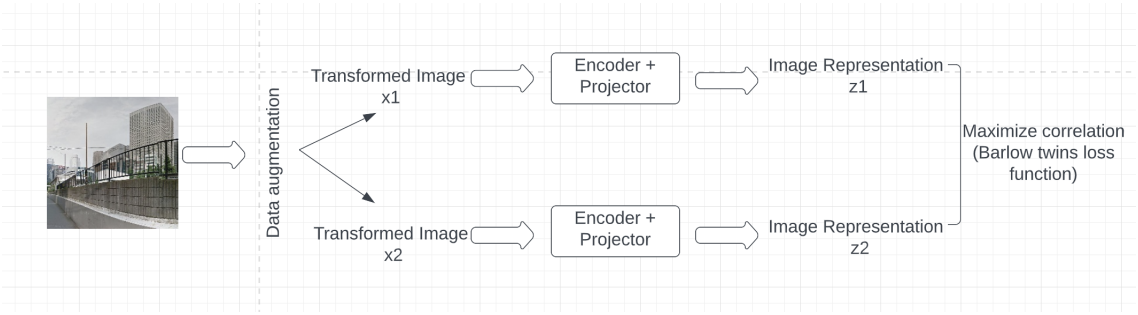


Figure 2.6: In Street Scene Change Detection (SSCD), the framework learns by maximising correlation between two images

sidered to be augmented versions of each other since they are images of the same scene at two different times. They are first passed through the encoder f_θ to get the feature vectors f'_0 and f'_1. These feature vectors are then passed through the projection head g_θ to be projected into the desired dimensions and obtain z'_0 and z'_1. Finally, they are passed into the loss function. D-SSCD involves the creation of two random augmentations for each of the input images. The augmentations may include Gaussian blur, color distortions, rotations and so on. Thus, the model gets a total of four images after augmentations – T'_0, T''_0, T'_1 and T''_1. All four images are

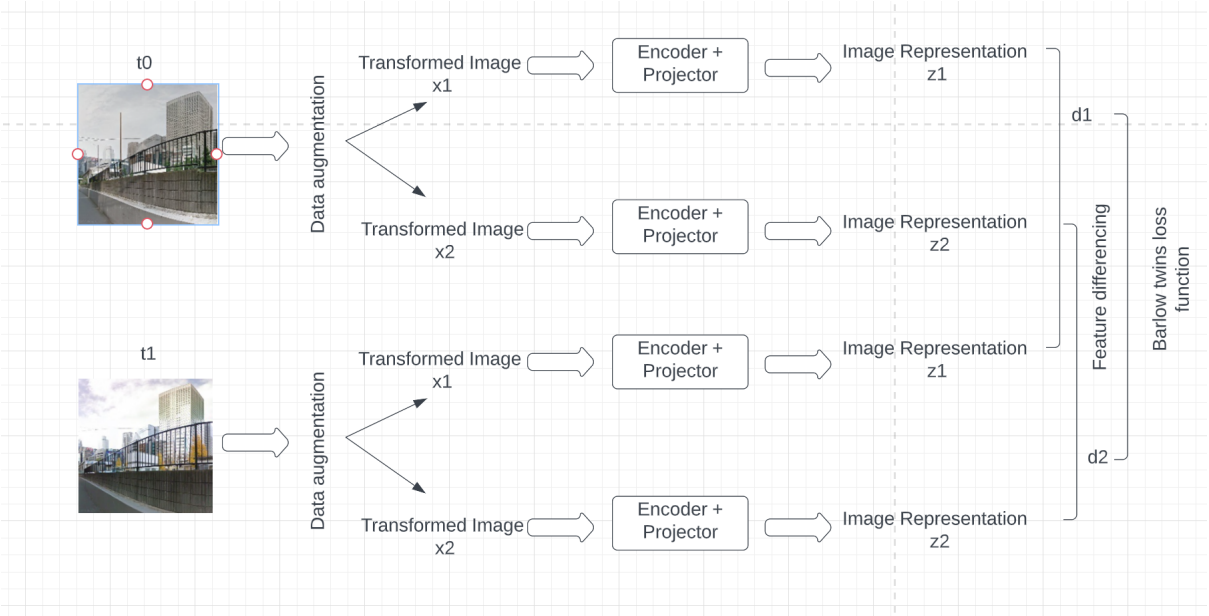


Figure 2.7: In D-SSCD model, the framework learns by absolute feature differencing.

then passed through the framework to finally obtain the latent representations z'_0, z''_0, z'_1 and z''_1. The difference representations are computed as:

$$d_1 = z'_0 - z'_1 \tag{2.14}$$

$$d_2 = z''_0 - z''_1 \tag{2.15}$$

The differences d_1 and d_2 are then passed into the loss function.

2.6.2 Barlow twins loss function

The central idea of this function is to separate the important features in an image from the redundant ones and also make the network invariant to distortions. It is based on the *redundancy-reduction principle*, first proposed by neuro-scientist Horace Barlow, and was introduced by Zbontar et al [17],
The input representations passed in are used to compute a correlation matrix, which gives an idea of the correlation of the various features. If two features are highly correlated, it means that one of them is redundant. Mathematically the loss is given as :

$$L_{BT} = \sum_i (1 - C_{ii})^2 + \lambda \sum_{i \neq j} C_{ij}^2 \tag{2.16}$$

where,

$$C_{ij} = \frac{\sum_b z^A_{b,i} z^B_{b,j}}{\sqrt{\sum_b (z^A_{b,i})^2} \sqrt{\sum_b (z^B_{b,j})^2}} \tag{2.17}$$

The first term in the objective function L_{BT} is called the *invariance* term and the second term is called *redundancy reduction* term. The objective function has to be ultimately minimized. The invariance term is minimized when C_{ii} is maximized,

which means that the same features across both the images are highly correlated, making the framework invariant to distortions. The redundancy reduction term is minimized when C_{ij} is minimized indicating minimal correlation between different features and consequently a redundancy reduction. Thus by learning to minimize this loss function, the framework learns to ignore distortions, while also differentiating between the important features and redundant ones.

3
Methods

The method in this project was inspired by a recent SCD study, where the pretraining method was based on the self-supervised D-SSCD approach[11], with a few modifications and finetuning was based on the idea of using different percentages of labeled data, in the approach using DR-TANet approach[10]. The pretraining method based on Barlow twins loss function uses the SimCLR architecture, which is a framework designed by Google for SSL with a possible modification to the MoCO framework. Evaluation was done on test datasets and the f1-scores were computed, indicating how well the changed regions were learned by the model. The results of the models with self-supervised pretraining and the models with self-supervised pretraining with pretrained ImageNet weights as a starting point were finally compared with DR-TANet with pretrained ImageNet weights, which was treated as the baseline. The performance of each of the models was evaluated on various percentages of labeled data.

3.1 Experimental setup

3.1.1 Image data representation

In computers, images are represented as arrays of numbers. One such representation uses a 3-dimensional array where the dimensions refer to the three color channels red, green, and blue (RGB)[16]. Each number in an array representing an image has a value ranging from 0 to 255. This experiment uses the publicly available change detection datasets VL_CMU_CD, PCD and ImageNet datasets.

3.1.1.1 VL_CMU_CD dataset

The VL_CMU_CD dataset contains complex and diverse scene pairs captured at various angles and light conditions as shown in Figure 3.1. It contains 11 different classes of images including construction-maintenance, traffic cones on the road, bins on the pavement, new sign boards and cars. There are 152 image sequences of distinct scene changes amounting to 1362 RGB image pairs. Each image sequence category has a minimum of 2 to a maximum of 41 image pairs. This dataset was taken from the city of Pittsburgh, Pennsylvania, USA, over a period of one year. Each image has dimensions of 1024 $\times$ 768 pixels. In this project, the dataset was modified and has 3732 image pairs for training the model. 429 pairs of images without any data preprocessing were kept aside for evaluating the model. Only images belonging to the training set without labels were used to pretrain the model.

During pretraining, these image pairs are resized to a resolution of 256×256. The Figure 3.1 below in which the first image is t_0 taken at first time frame and it is compared with the image t_1 taken at second time frame. Finally the last image is the image with ground-truth mask.

(a) Image t_0 **(b)** Image t_0 **(c)** Ground truth

Figure 3.1: An illustration of VL_CMU_CD dataset of image t_0, t_1 and labeled mask.

3.1.1.2 PCD dataset

The PCD dataset developed by Sakurata and Okatani [22], consists of two smaller datasets – 'TSUNAMI' and 'GSV', each of which has 100 pairs of images - each of dimensions 224 × 1024 pixels - with the manually labeled change masks. The images in the TSUNAMI dataset are from tsunami-affected areas in Japan and those from the GSV dataset are from Google street view. The images contain scene changes identified as 2D modifications of object surfaces (e.g., changes in the advertising board) and 3D structure changes (e.g., vanishing of buildings and vehicles). Changes caused by differences in lighting and photography conditions, as well as those caused by the weather are excluded, as are changes caused by reflection on building windows and changes in cloud and signs on the road surface. When comparing GSV dataset the TSUNAMI dataset is less affected by noise. During pretraining, these image pairs are also resized into 256×256 resolution.

(a) Image t_0 **(b)** Image t_0 **(c)** Ground truth

Figure 3.2: An illustration of Tsunami dataset of image t_0, t_1 and labeled mask.

3.1.1.3 ImageNet dataset

ImageNet is a large publicly available dataset that is frequently utilized in many fields of research. The total dataset contains millions of annotated images from a variety of categories, including animals, fruits, and flowers. Each of these categories

(a) Image t_0 **(b)** Image t_0 **(c)** Ground truth

Figure 3.3: An illustration of GSV dataset of image t_0, t_1 and labeled mask.

is further subdivided into sub-categories. The ImageNet dataset is utilized as the training set for the pretraining model in this master's thesis. This type of pretraining set, known as ImageNet-1K, has a total of 1,281,167 pictures for training and 1000 distinct classes. These ImageNet pretraining weights are also publicly available and it can be directly used for finetuning the supervised models.

3.1.2 Data preprocessing

The VL_CMU_CD original dataset has 1349 image pairs and during pretraining using 80:20 train_test_split we got 933 image pairs of VL_CMU_CD dataset that were resized to a resolution of 256×256 and 3 rotated augmentations of each image pairs were created. This resulted in 3732 image pairs to be used for pretraining (without masks) and different fraction of 3732 image pairs (with masks) were used for finetuning. The remaining 416 test pairs of images without preprocessing were the used for evaluation. The PCD dataset originally had 200 image pairs. Again the similar 80:20 train_test_split and we got 160 image pairs that were preprocessed into 9600 image pairs and its chosen for pretraining (without masks) along with the VL_CMU_CD dataset and different fraction of 9600 image pairs (with masks) were used for finetuning. In PCD dataset 160 of each image pair is extended into 60 patches with a 224x224 resolution by moving 56 pixels in width and data augmentation of plane rotation and that generates the 9600 image pairs. In 200 image pairs of original PCD dataset the remaining 40 test image pairs without preprocessing were used for evaluation.

3.1.3 Self-supervised pretraining on unlabeled images

Transfer learning outperforms building a new model from scratch in pretraining. As the backbone, the base encoder employed by the authors of the SimCLR paper is ResNet50 but we used the standard ResNet18 architecture due to limited GPU memory and longer training time. The loss function used is the Barlow Twins loss function, which was previously described. SimCLR, which is explained in the next section, was utilized as the framework. To generate two pairs of augmented images, random transformations such as color distortions and Gaussian blur are applied to the images. To generate feature representations, these augmented image pairs are sent into the encoder and projection head. The feature differencing is done across the projection outputs to learn the representation of the changing features between the two images (t_0, t_1). Once the pretraining is done the parameters of the encoder as shown is the Figure 3.4 are transferred to the downstream process of finetuning change detection.

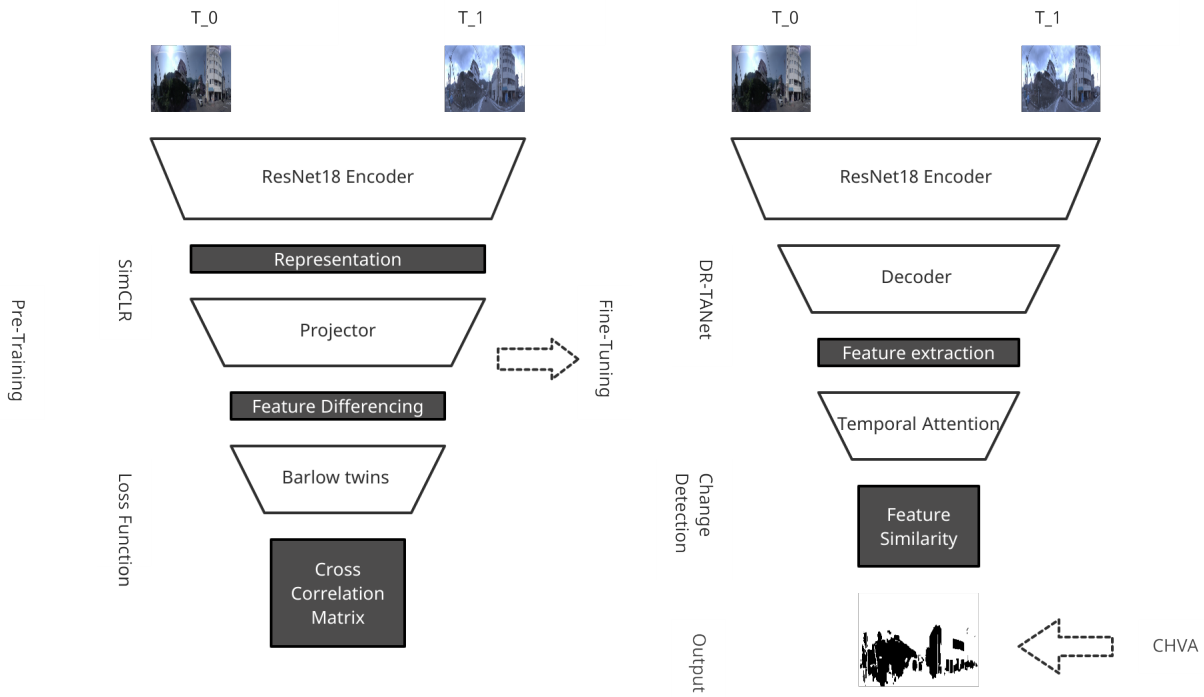


Figure 3.4: An experimental setup of pretraining (unlabeled images) followed by finetuning (labeled images).

3.1.3.1 SimCLR framework

The SimCLR is a framework proposed by Chen et al [19] to effectively learn visual representations using contrastive learning. It learns representations by maximising the agreement between augmented views of the same image (positive pairs), and minimising the agreement between augmentations of different images (negative pairs). It begins with a data augmentation module, which adds augmentations to the input images progressively. It creates two random augmentations per image. Once the augmentations are created, it applies three further augmentations on each of the augmented versions of the images, which are random cropping and resizing, random color distortions and random Gaussian blur. According to the authors, random crop and color distortions are important for boosting the performance. Now, in order to create representations for each of the two augmentations, SimCLR employs a base encoder f and a projection head g. The base encoder employed by the authors of the SimCLR paper is ResNet50. The projection head is a group of linear layers to map representations to a space where contrastive loss functions are applied. However, the loss function used by us is the Barlow Twins loss function, as employed in the D-SSCD paper [11].

3.1.4 Finetuning on limited labeled data

One of the major difficulties we face is the manual labeling of our unlabeled data. To train a good model, we often need to prepare a large amount of labeled data.

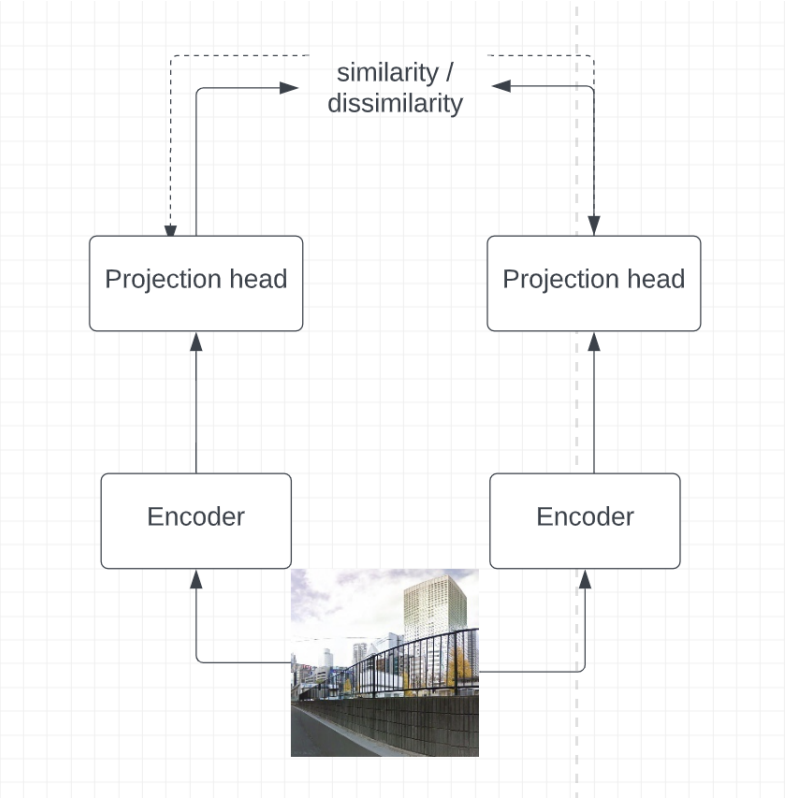


Figure 3.5: SimCLR architecture.

In this case, we may utilize the pretrained model from the labeled public dataset which is ImageNet or develop a pretrained model that learns temporal-consistent representations for the downstream task of supervised finetuning on limited labeled data.

The same ResNet18 which was used for pretraining was used as the backbone for finetuning so that pretraining parameters could be carried across. The paired input images (t_0, t_1) are passed to the encoder path, where they are divided into two channels for feature extraction as shown in Figure 3.5. They are then sent into the temporal attention to determine the similarity between the images in the given scope. The retrieved feature maps are then sent into the decoder, where the change mask is created. Different percentages of labeled data $(1\%, 10\%, 50\%, 100\%)$ are sampled for the training split of VL_CMU_CD, PCD datasets. The model training converges after 100 epochs.

3.1.5 Evaluation on different percentages of labeled data

Evaluation is performed based on finetuning the SOTA SCD model with five sets of pretraining strategies on PCD and VL_CMU_CD datasets. (1) Supervised ImageNet pretraining (DR-TANet mode), (2) self-supervised pretraining SimCLR, (3) Self-supervised pretraining MoCO, (4) Self-supervised SimCLR pretraining with pretrained ImageNet weights as starting point and (5) Self-supervised MoCO pretraining with pretrained ImageNet weights as starting point. These different models eval-

uated on the PCD dataset which has 40 test case image pairs and VL_CMU_CD which has 429 test case image pairs. The pretraining followed by finetuning helped to achieve better results.

$$\text{f1\_changed\_region} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.1}$$

$$\text{f1\_unchanged\_region} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.2}$$

$$\text{f1\_score} = \frac{\text{f1\_changed\_region} + \text{f1\_unchanged\_region}}{2} \tag{3.3}$$

The f1-score metric of equation (3.3) is used to evaluate the performance of change detection after finetuning. The f1-score is calculated separately for changed regions and unchanged changed regions. Finally, a mean for metrics of change and no change is calculated. The f1-score has a value between 0 and 1. The better the accuracy and recall, the higher the f1-score. Overall, the evaluation on VL_CMU_CD and PCD datasets shows that our suggested approaches for pretraining on unlabeled data (MoCO D-SSCD and SimCLR D-SSCD) can outperform the commonly used ImageNet pretraining, which uses over a million labeled images.

3.2 Baseline method

The methods described in the DR-TANet paper is chosen as the baseline methods. Authors of the DR-TANet and D-SSCD paper used ResNet50 as the backbone for both pretraining and finetuning. However, due to memory issues, we had to use ResNet18, which has fewer parameters and fits the available resources. The results achieved by our thesis outperforms the results that mentioned by our baseline methods. The architecture, hyper-parameters, augmentations, training and testing procedures were followed as mentioned in the papers. All models were trained for 100 epochs with batch size set to 16. The Tesla K80 GPU was used as default GPU for all the experiments.

3.3 Model extensions

In this section, additional methods that were implemented to improve on the baseline method, which is self-supervised SimCLR with pretrained ImageNet weights as starting point. The model was further extended using different pretraining strategies. MoCO was chosen due to its performance on many object detection tasks. MoCO uses memory queue and does not require large batch sizes, thereby solving memory issues.

3.3.1 MoCO framework

MoCO [20] manages to show an improvement in performance compared to SimCLR. Contrastive learning involves a comparison between positive and negative samples, so one optimization method is the use of more negative samples. For a particular

image, a negative counterpart would be any other image in the batch. If a batch contains N images, then after applying augmentations our model gets two representations per image, resulting in a total of 2N representations. For a particular representation x, there is only one other representation that forms a positive pair with it. This leaves us with (2N - 2) negative pairs involving x. In the case of SimCLR, a large number of negative samples can only be obtained with large batch size. This in turn leads to higher requirements in computational power. This is where MoCO comes in and provides an alternate approach to generating negative samples.

As opposed to SimCLR, MoCO works with two encoders and uses the concept of dynamic dictionaries. One of the encoders is a *query* encoder and the other is a *key* encoder. If a certain query matches a key, a positive pair is generated. This match occurs when the query and key come from the same image. For the negative pairs, it maintains a large dictionary that contains encoded keys from previous batches, which serve as negative samples to the query in question. This dictionary is in the form of a queue of batches. The latest batch is added to the queue, while the oldest batch is removed from it. The number of negative samples is dependent on the size of this queue. The key encoder is implemented as momentum-based moving average

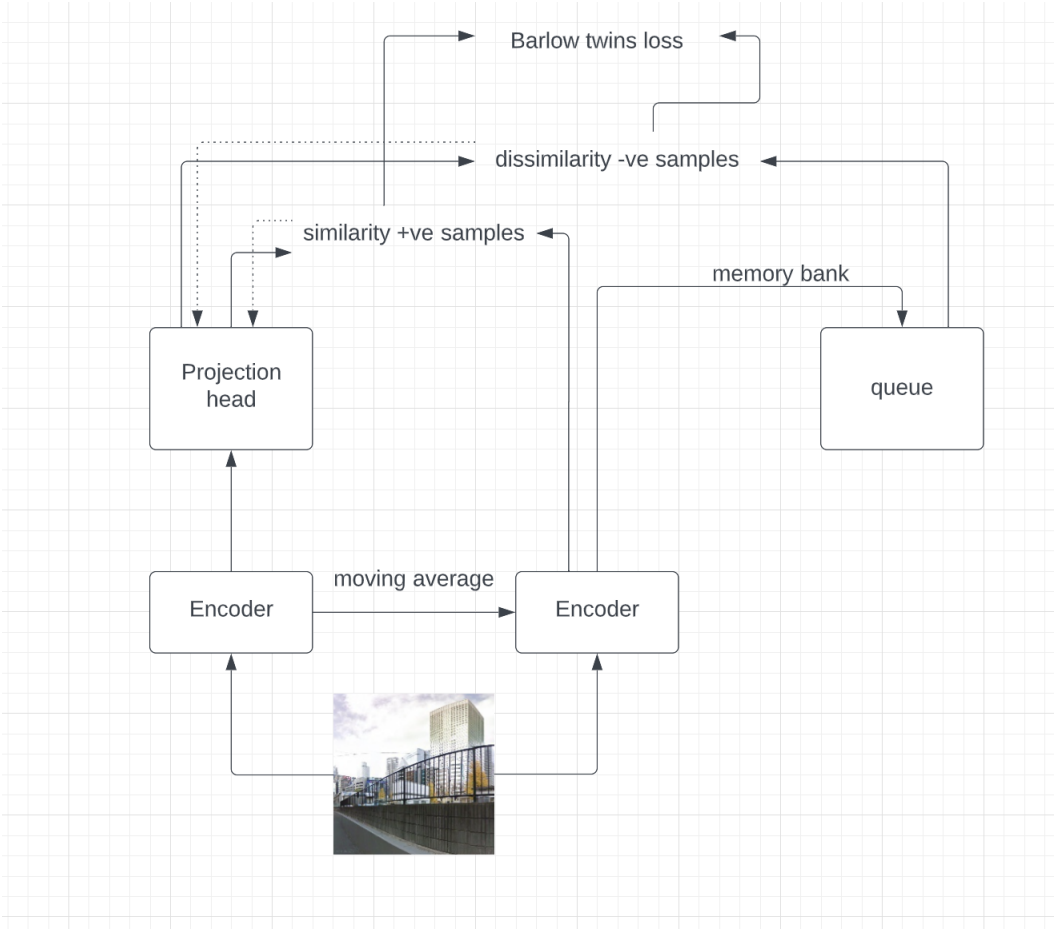


Figure 3.6: MoCO architecture.

of the query encoder, meaning that the parameters of the key encoder are updated as follow:

$$\theta_k \longleftarrow m\theta_k + (1-m)\theta_q \tag{3.4}$$

where m is generally close to 1.

3.3.2 Data augmentation

Performing data augmentation in SCD is important since it helps to reduce overfitting and improves performance of the model. It also helps in learning the model for good representations. The noise introduced allows the model to learn noise invariant representations. It has been shown that stronger augmentations lead to better representations [12] and these augmentations have also led to improvement in MoCO and SimCLR as well [13]. In our experiments, we used the same set of data augmentations as in MoCO and SimCLR. These include cropping, color jittering, color distortion, and Gaussian blur. The illustration of an original image and augmented image is shown below.

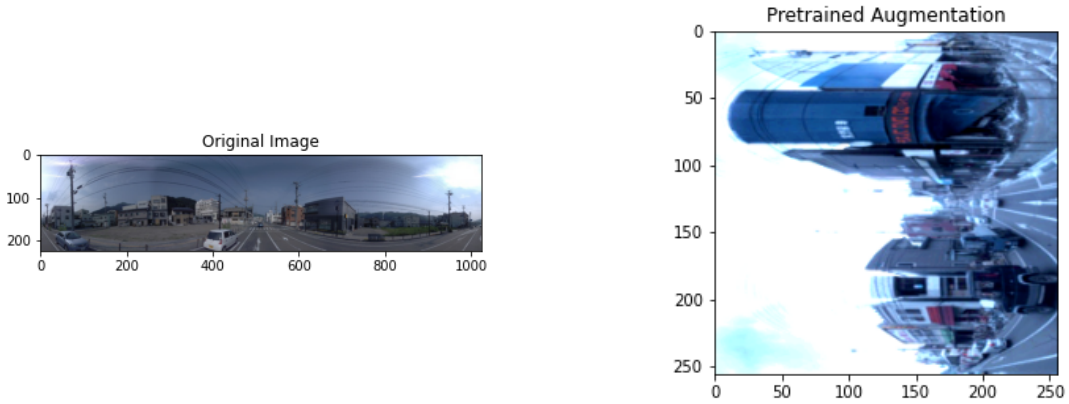


Figure 3.7: The instance of an original image and augmented image.

Random cropping and resizing: Random crop and resizing are used to crop and resize images into the desired sizes for the input images of self-supervised pretraining. Random crops are not considered for this preprocessing and only the input image are resized to 256 × 256 along with normalization.

Random color distortions: The random color distortions are applied for images to change the brightness, contrast, saturation and hue of an image. Color distortions for images are applied to resized images with a 50% probability of color jitter and a 20% probability of gray scale.

Random Gaussian blur: Blurs the image with Gaussian blur of 50% probability.

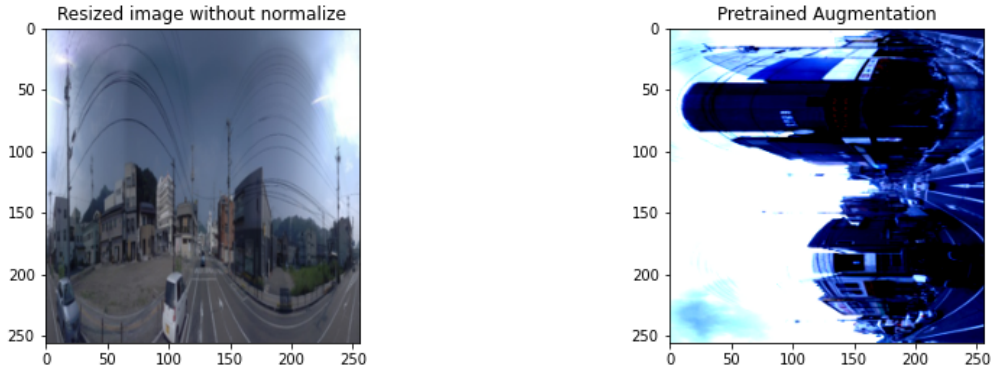


Figure 3.8: Random cropping and resizing.

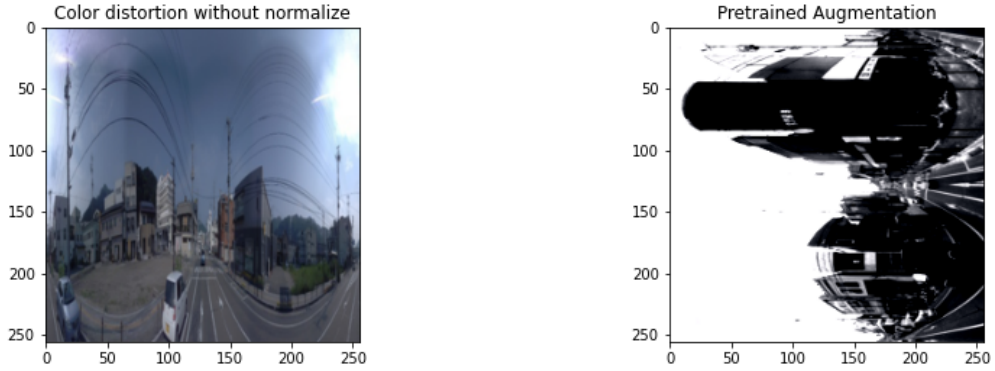


Figure 3.9: Random color distortion.

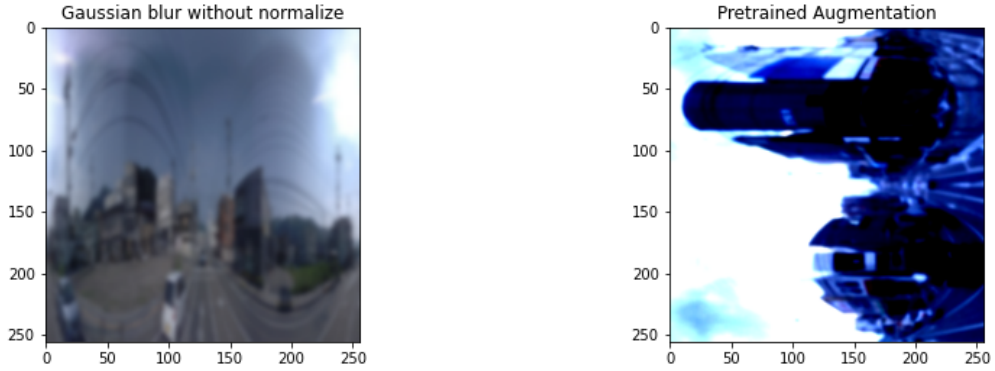


Figure 3.10: Random Gaussian blur.

4
Results

The first subsection describes the results of our baseline supervised ImageNet model (DR-TANet model). The second subsection describes the results of self-supervised MoCO and self-supervised SimCLR, and the third subsection compares the outcomes of self-supervised MoCO with pretrained ImageNet weights as starting point and self-supervised SimCLR with pretrained ImageNet weights as starting point. Finally, the models' performances are compared. In each case, the Barlow twins loss function was used for pretraining, and the f1-score was used as the metric for performance evaluation. ResNet18 served as the backbone encoder for all of our models. All models were evaluated for 100 epochs with a batch size of 16. On average, each pretraining process took about 72 hours to complete and was followed by a finetuning process with different percentages of labeled data. Based on the desired percentage of labeled data, a random selection was done on both VL_CMU_CD and PCD datasets. Five different fractions of supervised labeled SCD images using ImageNet 4.1 were created for each dataset and then evaluated upon as a baseline for all the experiments. The random selection of dataset for every fraction of performance in the models was fresh each time.

4.1 Performance analysis

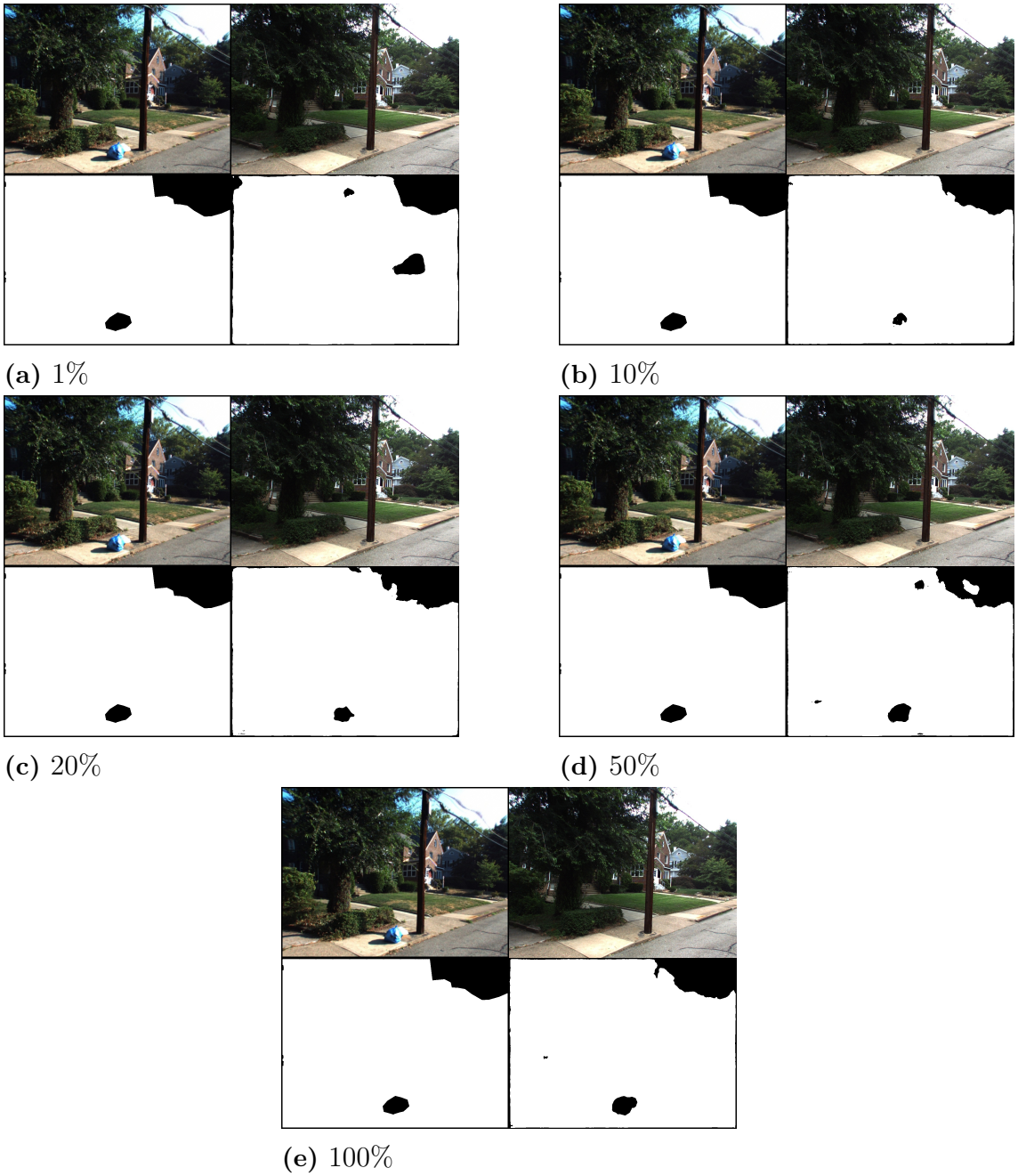
4.1.1 Supervised labeled SCD images using ImageNet

Table 4.1: Performance (f1-score) of DR-TANet model trained on different datasets.

Dataset	**Label fraction**				
	1%	**10%**	**20%**	**50%**	**100%**
VL_CMU_CD	0.630	0.834	0.801	0.825	0.813
PCD	0.653	0.754	0.778	0.770	0.770

The commonly used ImageNet dataset has millions of images. Models pretrained on ImageNet dataset weights and then using the resulting weights have often had

major gains in performance when used for other similar datasets. Models with pretrained ImageNet weights are widely and publicly available with no additional cost. Finetuning a model using the already available ImageNet pretrained weights makes the model parameters converge quicker than from scratch. The Table 4.1 above shows the performance of the DR-TANet model (SOTA) trained on different datasets using supervised ImageNet.



(a) 1%

(b) 10%

(c) 20%

(d) 50%

(e) 100%

Figure 4.1: Comparison of different fraction of labeled VL_CMU_CD dataset of **Supervised ImageNet**. The top row of each image illustrates scenes from two different times. The bottom row compares the obtained result to the ground truth.

As shown in Table 4.1 above, the model's performance fluctuated. When evaluated

on VL_CMU_CD dataset, the f1-score grew when 10% of the data was used for training and then fell for 20%, improved again for 50% and finally fell again for 100% of the data. Similarly, in case of the PCD dataset the performance saturated when 20% of the labeled data was used and did not improve beyond that. The performance of these models using pretrained ImageNet weights was considered as the baseline. It was then compared with two other categories of models. The first category had models which were pretrained from scratch using the Barlow Twins loss function. The second category also had models which were pretrained using the Barlow twins loss function, however, the pretraining did not start from scratch and instead pretrained ImageNet weights were used as the starting point.

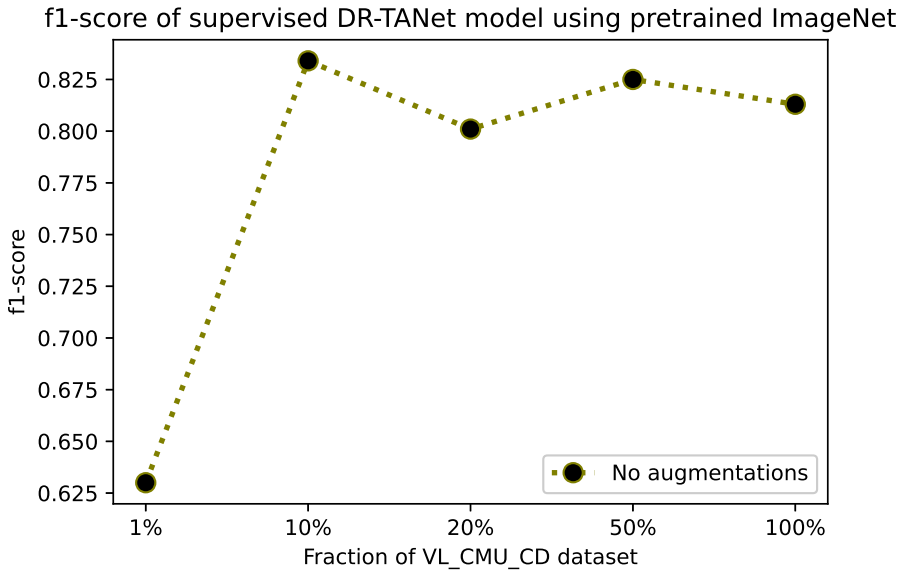
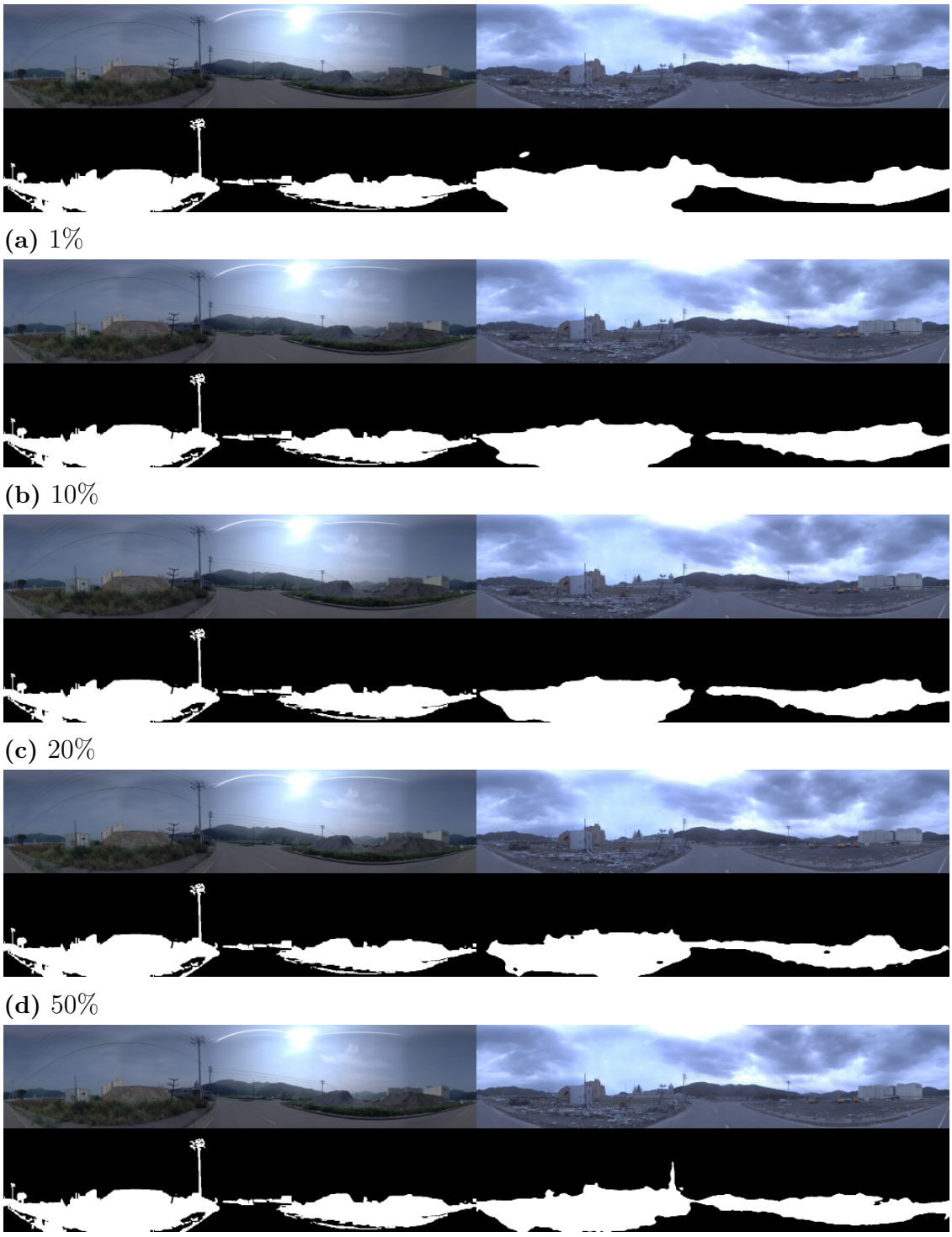


Figure 4.2: Performance (f1-score) of DR-TANet model (SOTA), trained on VL_CMU_CD dataset, using supervised ImageNet weights.

According to figure 4.2, accuracy and f1-score are higher in the case where 10% of the data is used, despite a significant difference in performance between the cases with 1% and 10% labeled data. Similarly, as shown in figure 4.5, the performance is higher when 20% of the labeled data is used, and there is a significant performance difference when compared with 1% labeled data.



(a) 1%

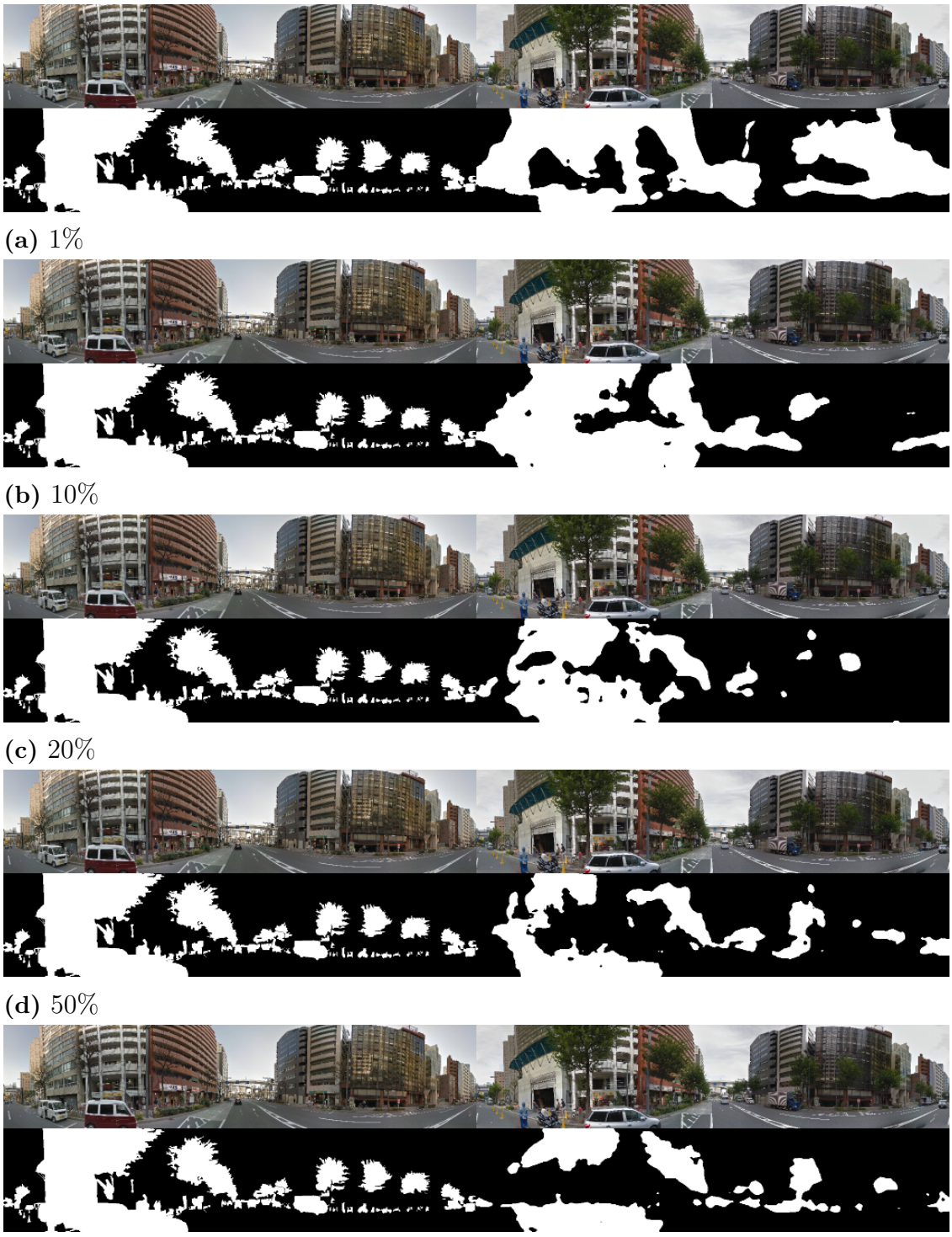
(b) 10%

(c) 20%

(d) 50%

(e) 100%

Figure 4.3: Comparison of different fraction of labeled TSUNAMI dataset of **Supervised ImageNet**. The top row of each image illustrates scenes from two different times. The bottom row compares the obtained result to the ground truth.



(a) 1%

(b) 10%

(c) 20%

(d) 50%

(e) 100%

Figure 4.4: Comparison of different fraction of labeled GSV dataset of **Supervised ImageNet**. The top row of each image illustrates scenes from two different times. The bottom row of each image compares the obtained result to the ground truth.

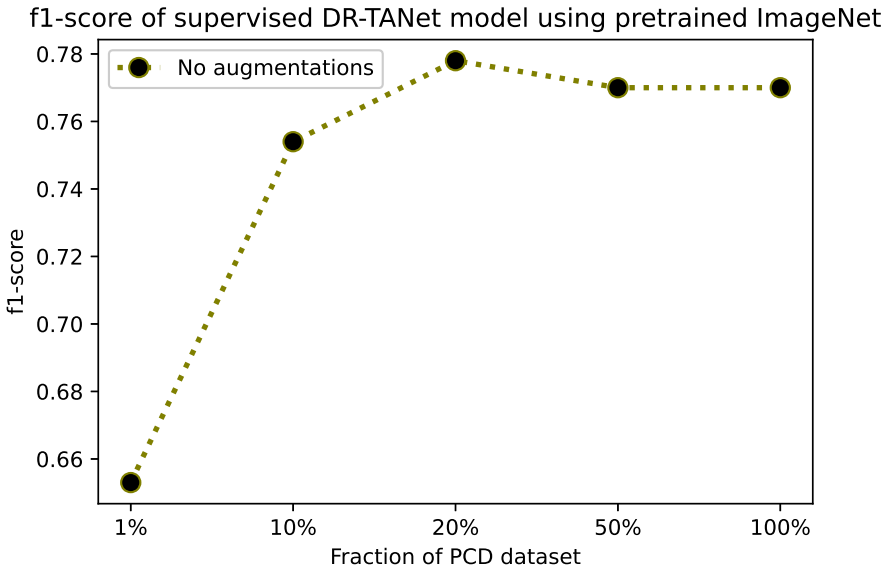


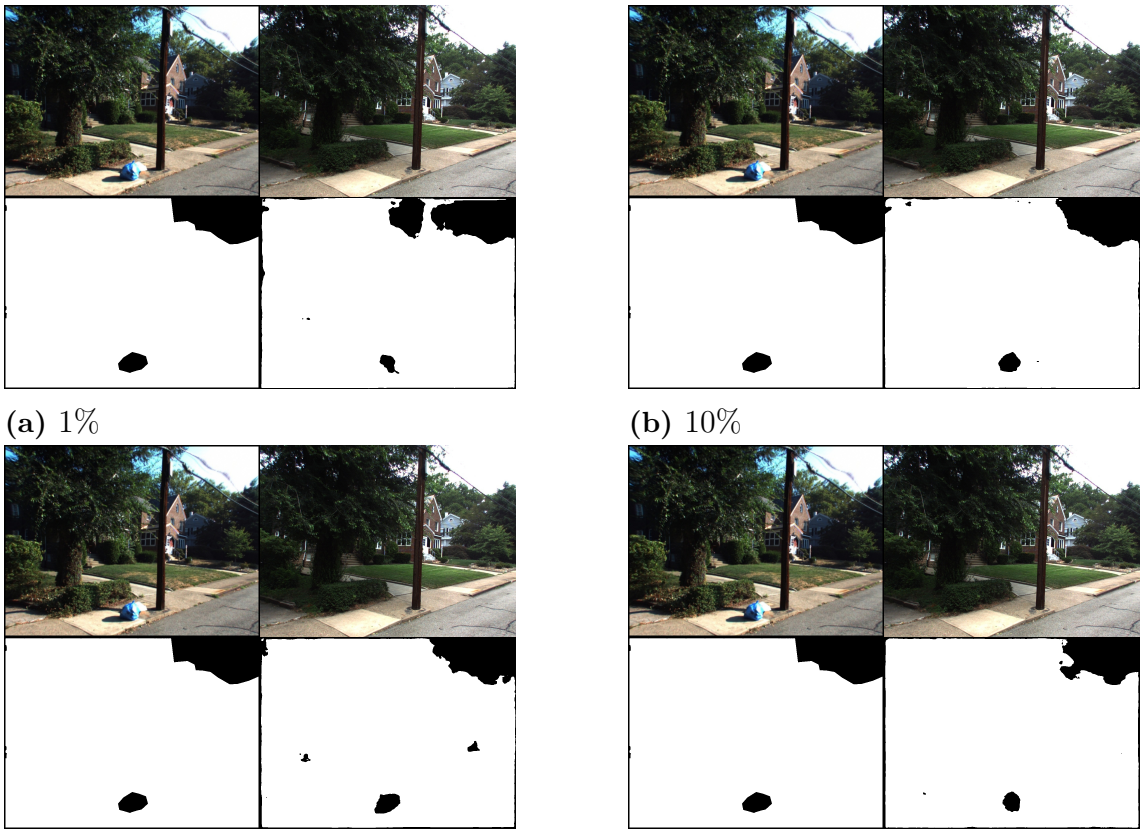
Figure 4.5: Performance (f1-score) of DR-TANet model (SOTA) trained on PCD dataset using supervised ImageNet weights.

4.1.2 Self-supervised unlabeled SCD images in pretraining

Table 4.2: Performance (f1-score) of self-supervised pretraining methods when model is evaluated on VL_CMU_CD dataset with augmentations.

Pretraining methods	**Label Fraction**				
	1%	**10%**	**20%**	**50%**	**100%**
SimCLR	0.780	0.823	0.806	0.833	0.814
MoCO	0.801	0.853	0.844	0.862	0.864

Due to the high cost involved in acquiring manual annotations, the availability of a large amount of labeled data is a significant challenge in SCD. A good SCD model performs well even when labeled data is limited. Tables 4.2 and 4.3 display the results of VL_CMU_CD and PCD datasets finetuned with five different fractions. From the VL_CMU_CD and PCD datasets, different percentages of labeled data - 1%, 10%, 50%, and 100% - were picked at random for finetuning. In all the limited labels cases, the proposed SOTA pretraining approaches involving SimCLR and MOCO outperform the commonly used pretrained ImageNet weights by a small margin.



(a) 1%

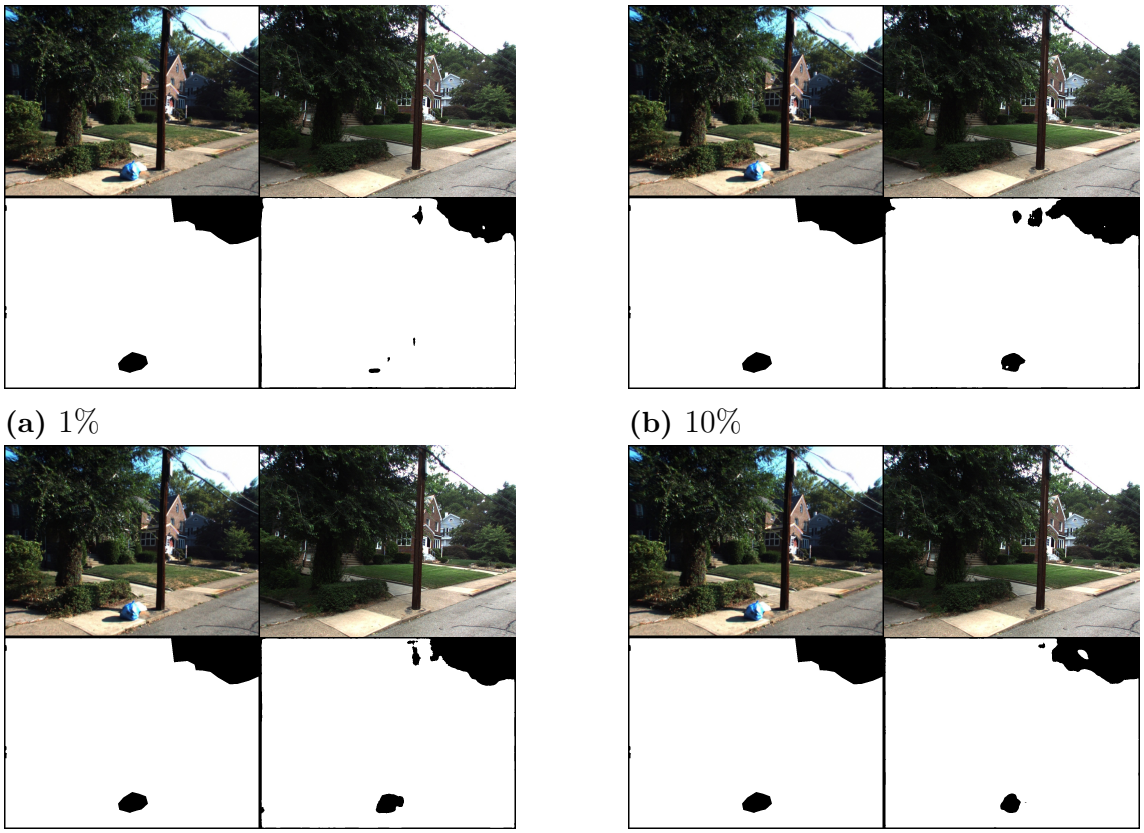
(b) 10%

(c) 20%

(d) 50%

(e) 100%

Figure 4.6: Comparison of different fraction of labeled VL_CMU_CD dataset of **self-supervised SimCLR model**. The top row of each image illustrates scenes from two different times. The bottom row compares the obtained result to the ground truth.



(a) 1%

(b) 10%

(c) 20%

(d) 50%

(e) 100%

Figure 4.7: Comparison of different fraction of labeled VL_CMU_CD dataset of **self-supervised MoCO model**. The top row of each image illustrates scenes from two different times. The bottom row compares the obtained result to the ground truth.

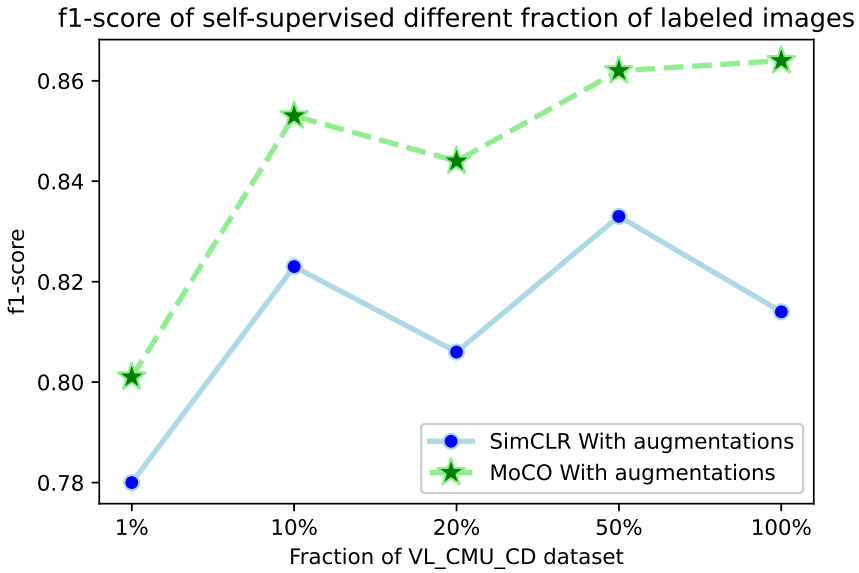


Figure 4.8: Performance (f1-score) of self-supervised model finetuned on fraction of labeled VL_CMU_CD dataset with augmentations.

As shown in tables 4.2 and 4.3 above, the self-supervised pretrained models outperform the supervised pretrained models by achieving higher gains on smaller datasets. As proven by this approach, self-supervised learning methods aid in learning from smaller datasets more efficiently. As shown in figure 4.8 and 4.13, where the performance of SimCLR and MoCO is compared, MoCO performs better for smaller datasets, although there appears to be no change between the models in the case where 20% of the labelled data is used. Therefore, inspite of some differences there is no statistically significant difference in performance (f1-score) between the models SimCLR and MoCO.

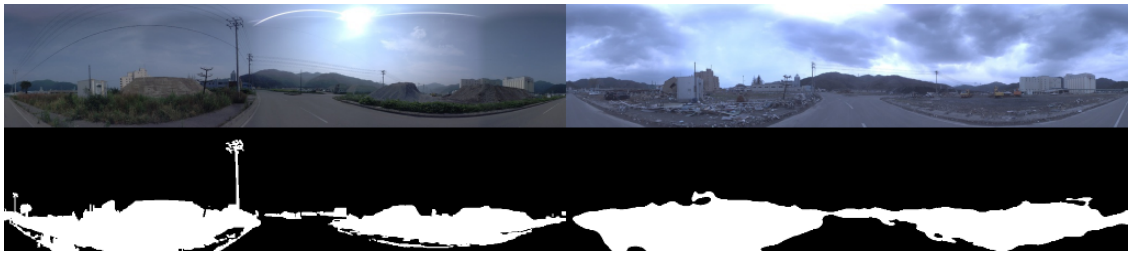
Table 4.3: Performance (f1-score) of self-supervised pretraining methods when model is evaluated on PCD dataset with augmentations.

Pretraining methods	**Label Fraction**				
	1%	**10%**	**20%**	**50%**	**100%**
SimCLR	0.694	0.749	0.781	0.788	0.722
MoCO	0.679	0.770	0.780	0.778	0.774

The baseline method chosen in this study is training with pretrained ImageNet weights as a starting point. This is further compared with self-supervised and the combination of self-supervised model with supervised ImageNet. As shown in Figure 4.22, an average performance of 20% outperforms other labeled fractions, and the same in Figure 4.23.

(a) 1%

(b) 10%

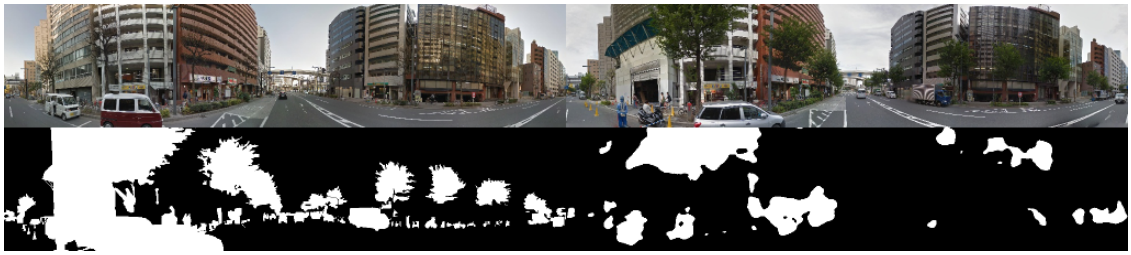


(c) 20%

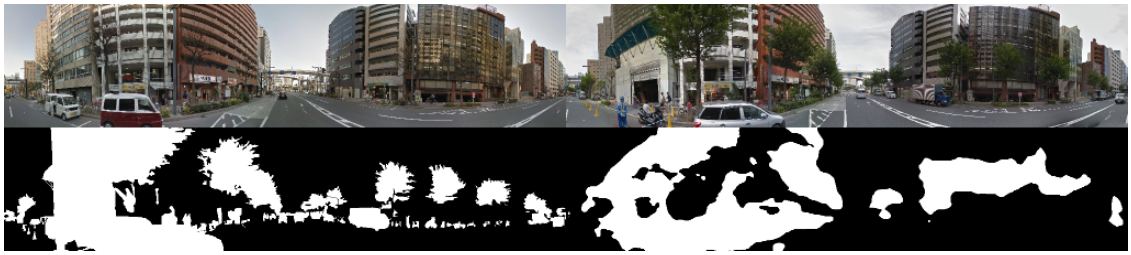
(d) 50%

(e) 100%

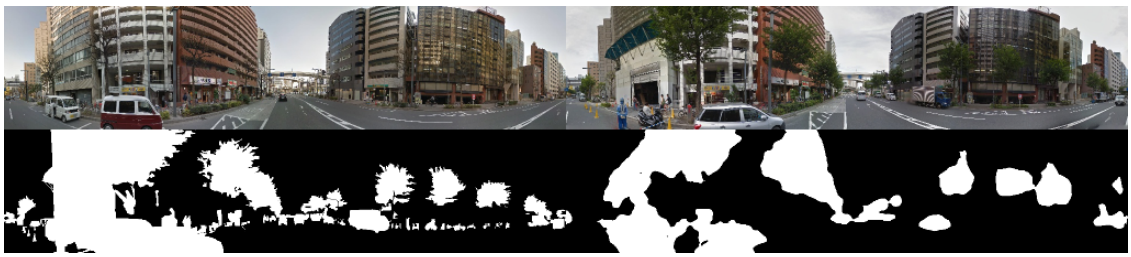
Figure 4.9: Comparison of different fractions of labeled TSUNAMI dataset of **self-supervised SimCLR model**. The top row of each image illustrates scenes from two different times. The bottom row compares the obtained result to the ground truth.



(a) 1%



(b) 10%



(c) 20%

(d) 50%

(e) 100%

Figure 4.10: Comparison of different fractions of labeled GSV dataset of **self-supervised SimCLR model**. The top row of each image illustrates scenes from two different times. The bottom row compares the obtained result to the ground truth.

(a) 1%

(b) 10%

(c) 20%

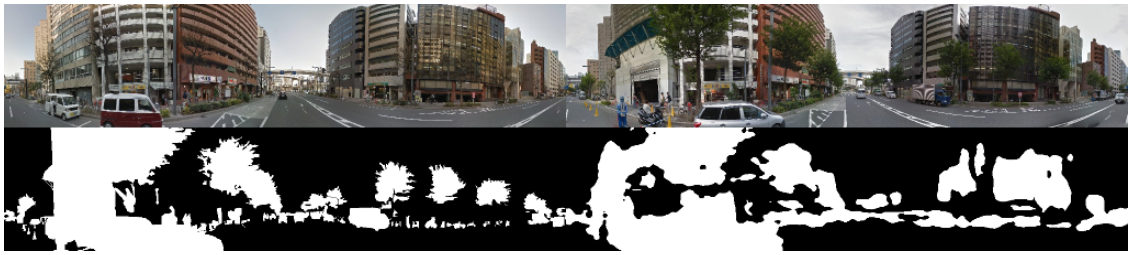
(d) 50%

(e) 100%

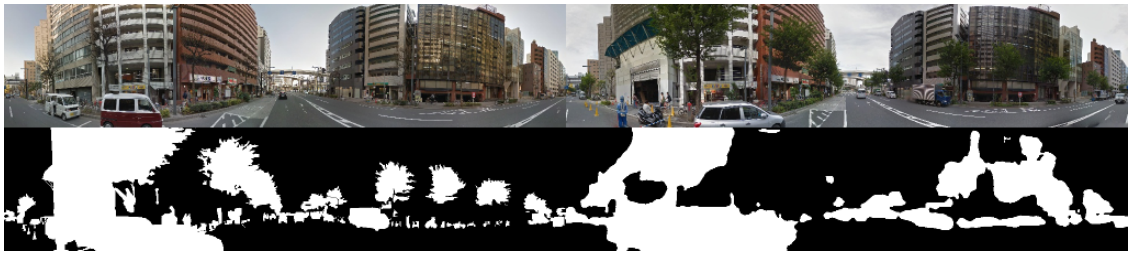
Figure 4.11: Comparison of different fractions of labeled TSUNAMI dataset of **self-supervised MoCO model**. The top row of each image illustrates scenes from two different times. The bottom row compares the obtained result to the ground truth.

(a) 1%

(b) 10%



(c) 20%



(d) 50%

(e) 100%

Figure 4.12: Comparison of different fractions of labeled GSV dataset of **self-supervised MoCO model**. The top row of each image illustrates scenes from two different times. The bottom row compares the obtained result to the ground truth.

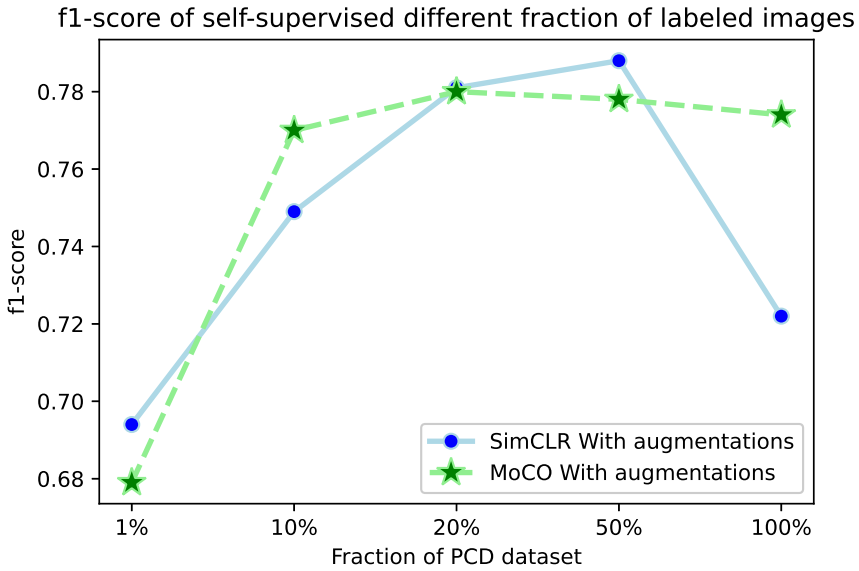


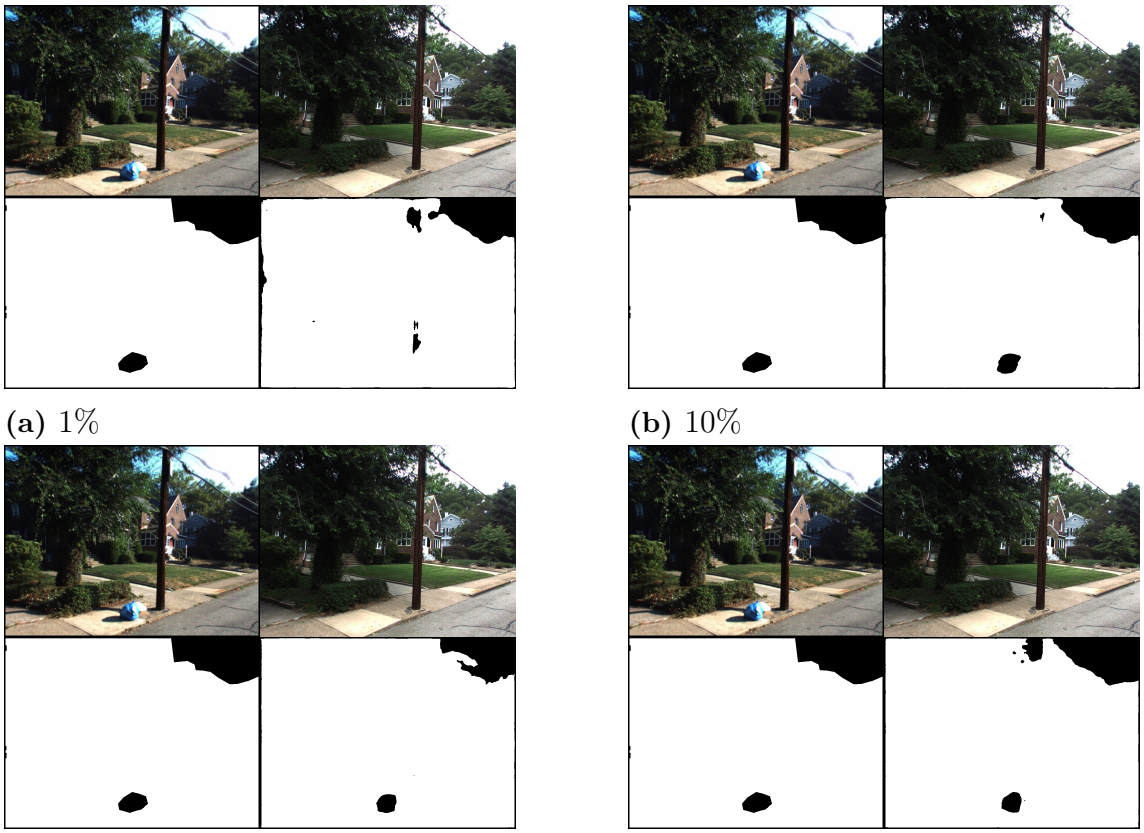
Figure 4.13: Performance (f1-score) of self-supervised model finetuned on fractions of labeled PCD dataset with augmentations.

4.1.3 Self-supervised pretraining using pretrained ImageNet weights as starting point

As stated previously, the availability of large amounts of labeled data is a significant challenge in SCD. This issue was addressed by conducting additional experiments with a two-stage pretraining approach. The proposed self-supervised pretraining models were initialized with supervised ImageNet weights as starting point (SimCLR + supervised ImageNet and MoCO + supervised ImageNet) and finetuned with different percentages of labeled data of 1%, 10%, 50%, and 100%.

Table 4.4: Performance (f1-score) of two stage pretraining methods when model is evaluated on VL_CMU_CD dataset with augmentations.

Pretraining methods	**Label Fraction**				
	1%	**10%**	**20%**	**50%**	**100%**
SimCLR (Sup-Im weights as starting point)	0.819	0.844	0.861	0.861	0.861
MOCO (Sup-Im weights as starting point)	0.808	0.840	0.855	0.845	0.863



(a) 1%

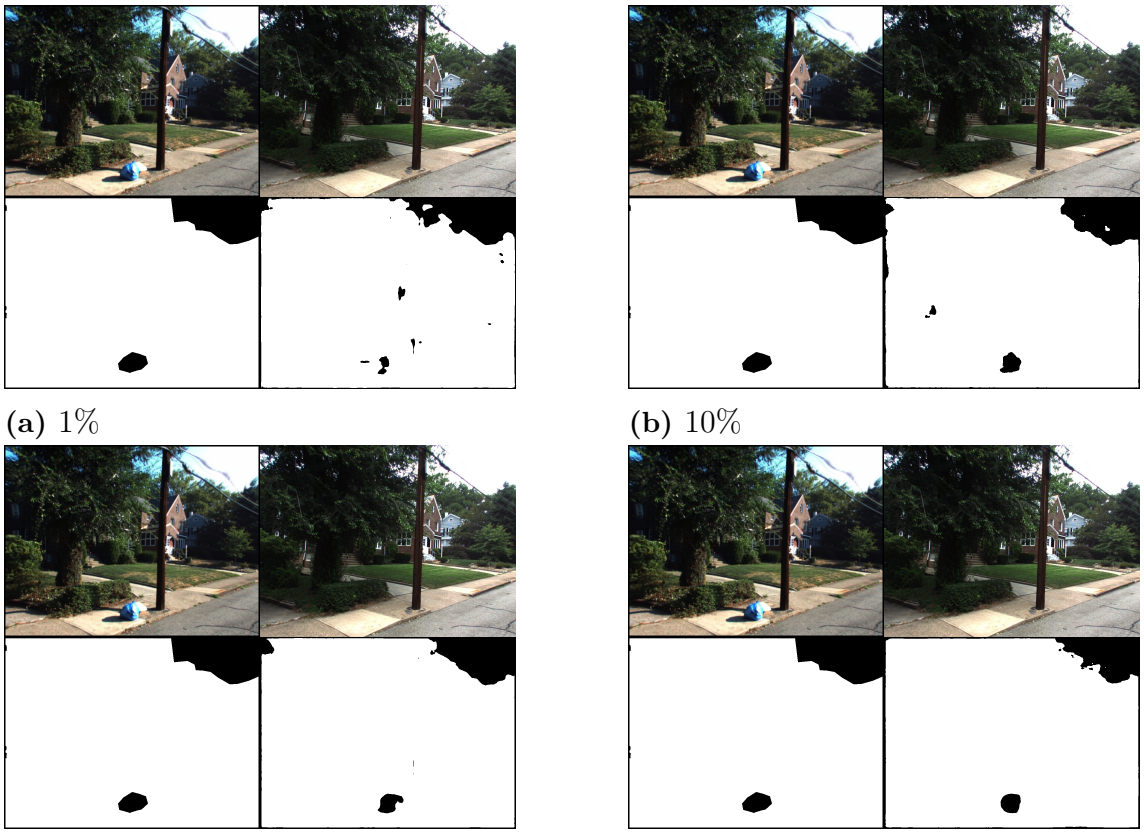
(b) 10%

(c) 20%

(d) 50%

(e) 100%

Figure 4.14: Comparison of different fraction of labeled VL_CMU_CD dataset combination of **SimCLR with pretrained ImageNet weights as starting point**. The top row of each image illustrates scenes from two different times. The bottom row compares the obtained result to the ground truth.



(a) 1%

(b) 10%

(c) 20%

(d) 50%

(e) 100%

Figure 4.15: Comparison of different fraction of labeled VL_CMU_CD dataset combination of **MoCO with pretrained ImageNet weights as starting point**. The top row illustrates images from two different times. The bottom row compares the obtained result to the ground truth.

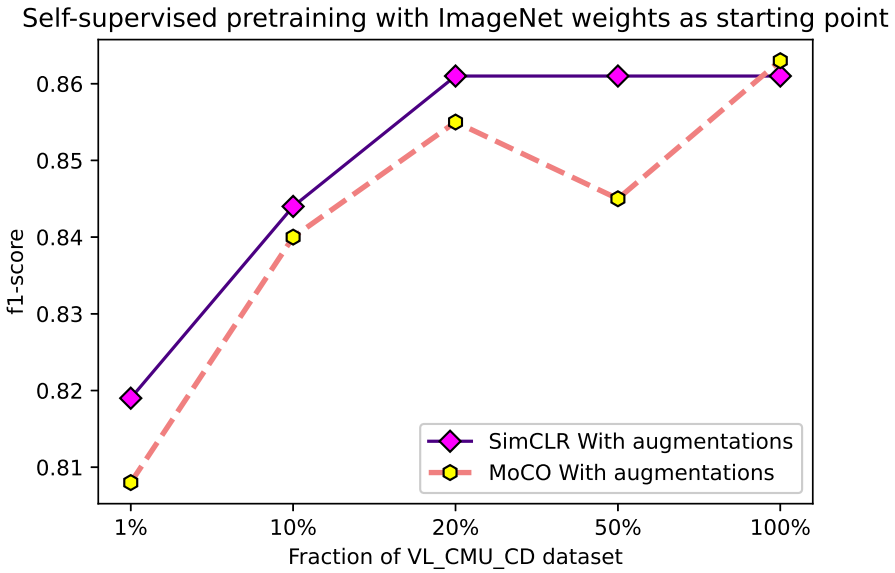
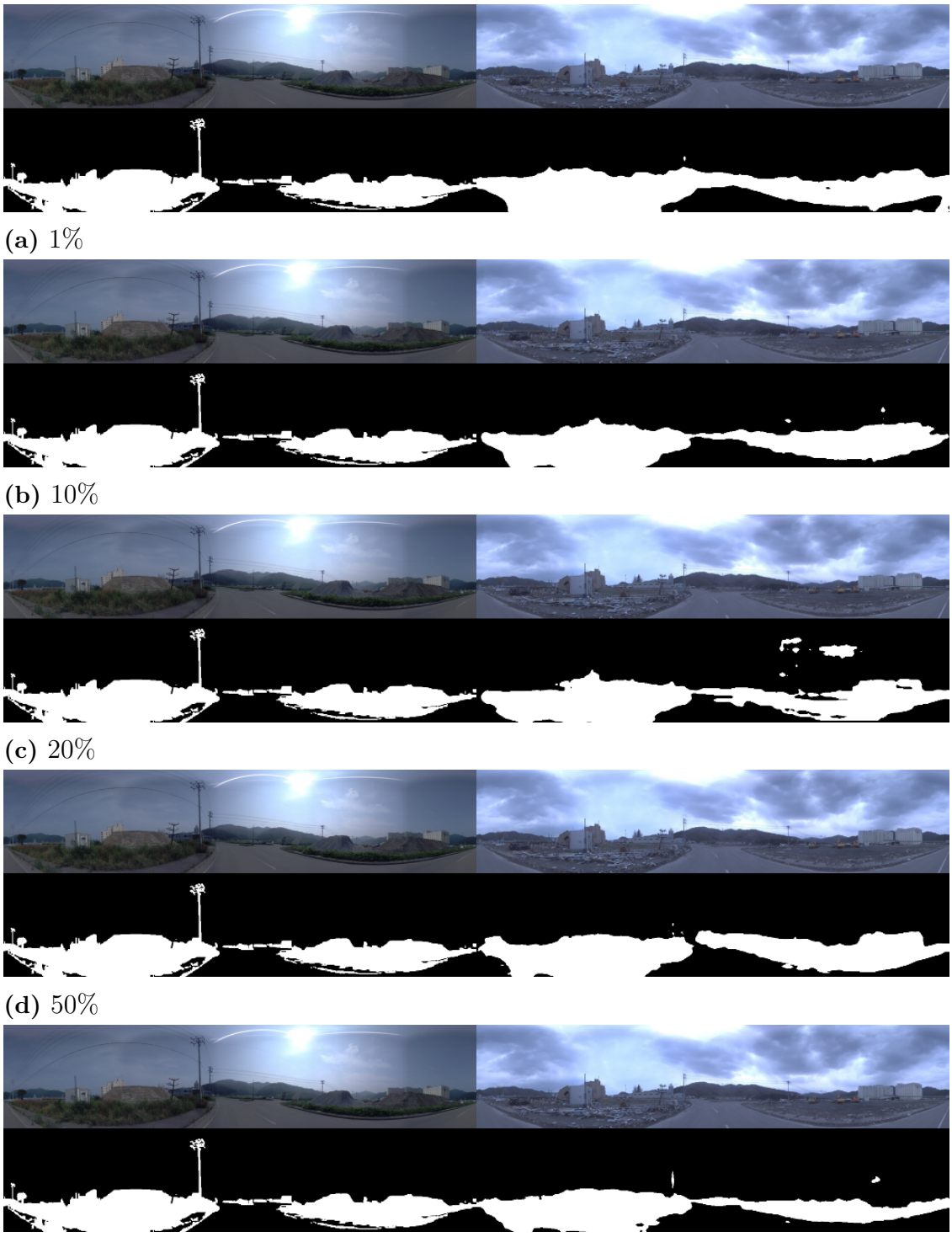


Figure 4.16: Performance (f1-score) of model pretrained on VL_CMU_CD dataset using the self-supervised method with pretrained ImageNet weights as a starting point.

As shown in the Table 4.4 and 4.5, self-supervised DSSCD with pretrained ImageNet weights as starting point outperforms both the baseline model and the self-supervised DSSCD model. The model with a better performance is given below.

Table 4.5: Performance (f1-score) of two stage pretraining methods when the model is evaluated on PCD dataset with augmentations.

Pretraining methods	**Label Fraction**				
	1%	**10%**	**20%**	**50%**	**100%**
SimCLR (Sup-Im weights as starting point)	0.693	0.768	0.770	0.787	0.785
MoCO (Sup-Im weights as starting point)	0.670	0.770	0.794	0.780	0.773



(a) 1%

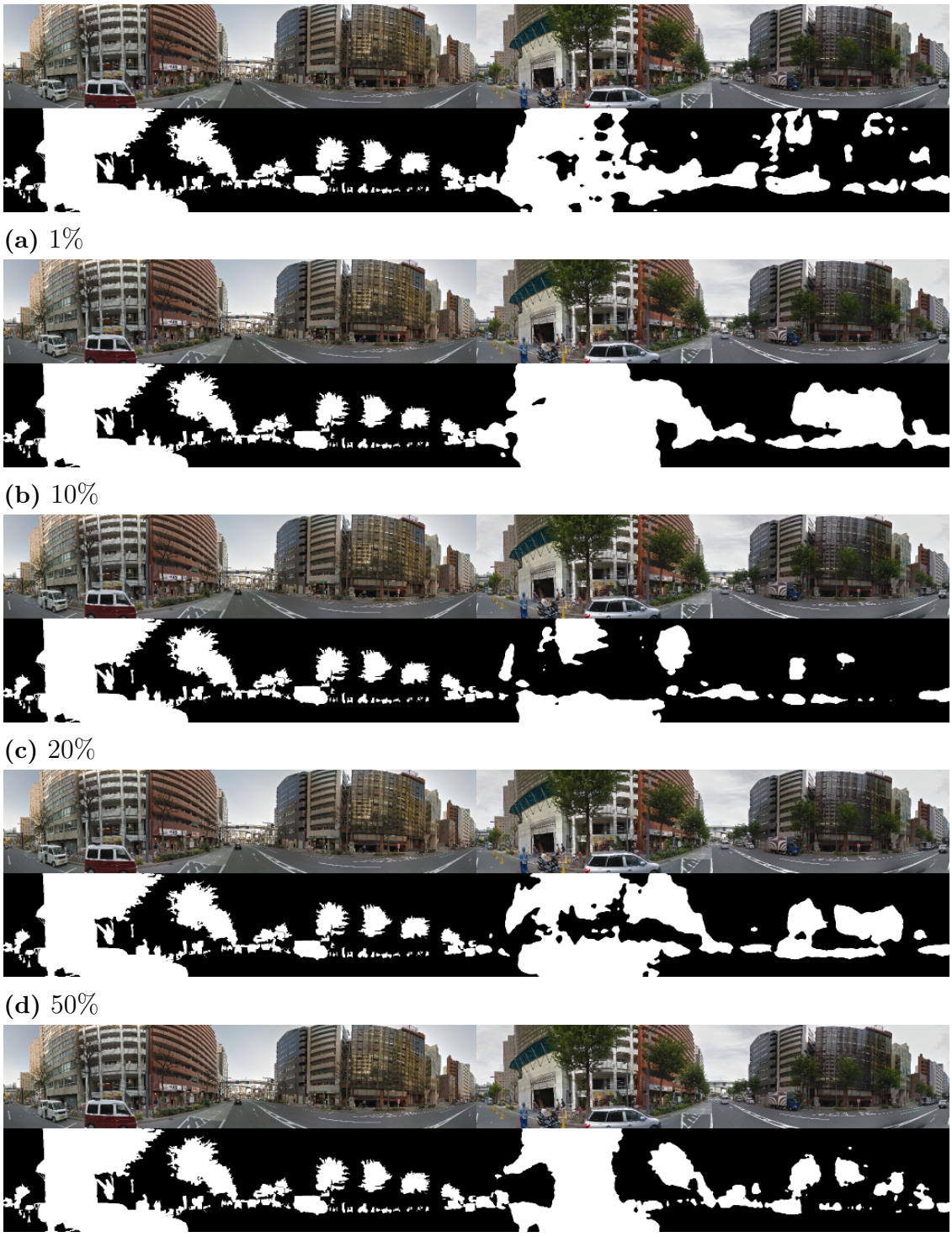
(b) 10%

(c) 20%

(d) 50%

(e) 100%

Figure 4.17: Comparison of different fraction of labeled TSUNAMI dataset using **SimCLR with pretrained ImageNet weights as starting point**. The top row of each image illustrates scenes from two different times. The bottom row compares the obtained result to the ground truth.



(a) 1%

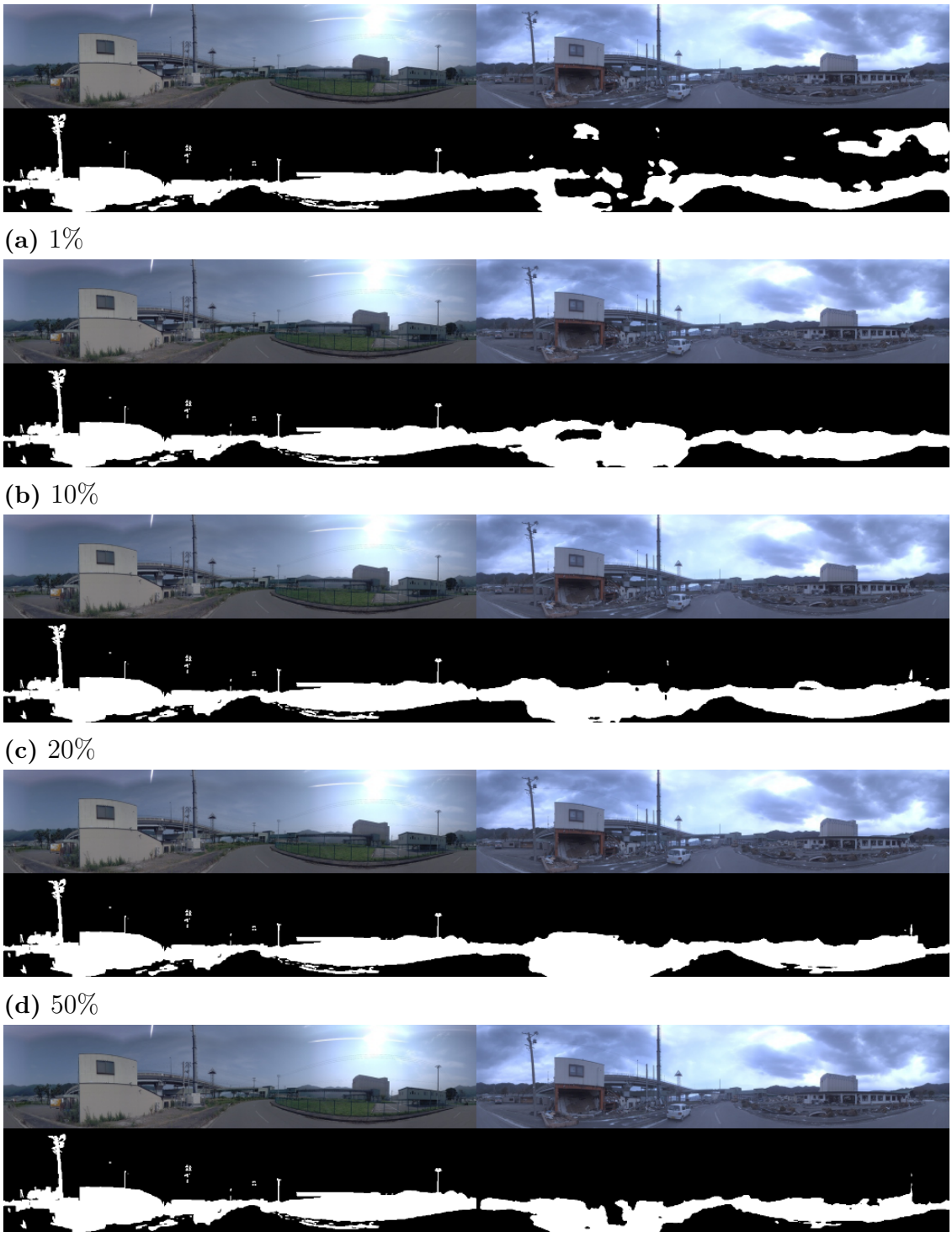
(b) 10%

(c) 20%

(d) 50%

(e) 100%

Figure 4.18: Comparison of different fraction of labeled GSV dataset using **SimCLR with pretrained ImageNet weights as starting point**. The top row of each image illustrates scenes from two different times. The bottom row compares the obtained result to the ground truth.



(a) 1%

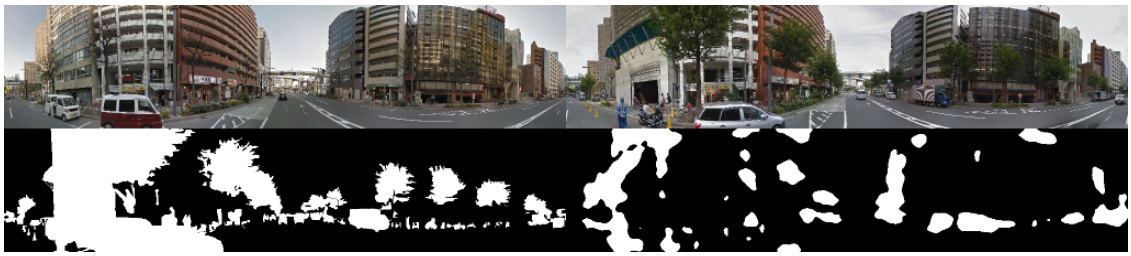
(b) 10%

(c) 20%

(d) 50%

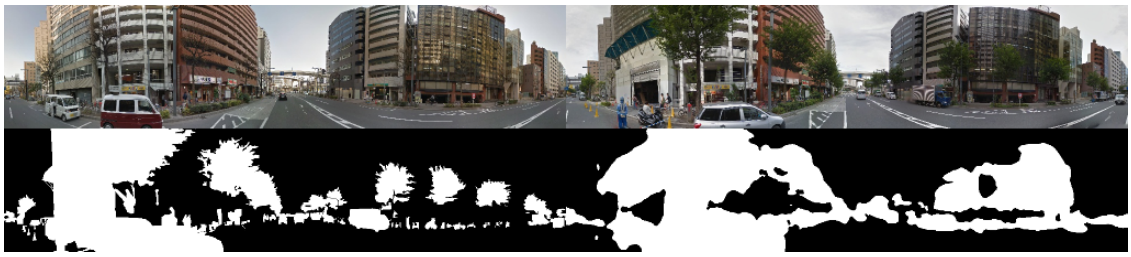
(e) 100%

Figure 4.19: Comparison of different fraction of labeled TSUNAMI dataset combination of **MoCO with pretrained ImageNet weights as starting point**. The top row of each image illustrates scenes from two different times. The bottom row of each image compares the obtained result to the ground truth.

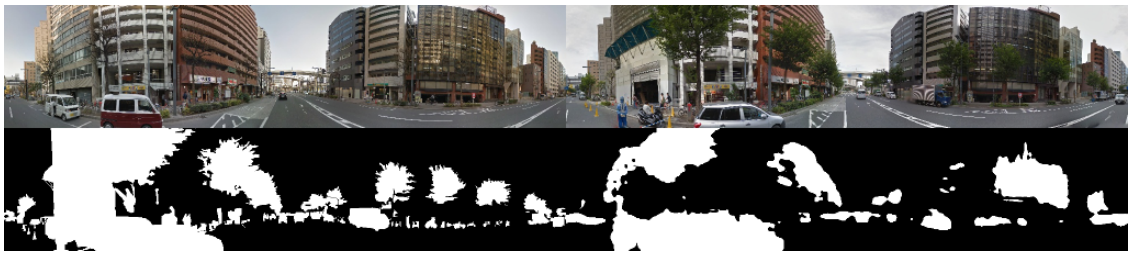


(a) 1%

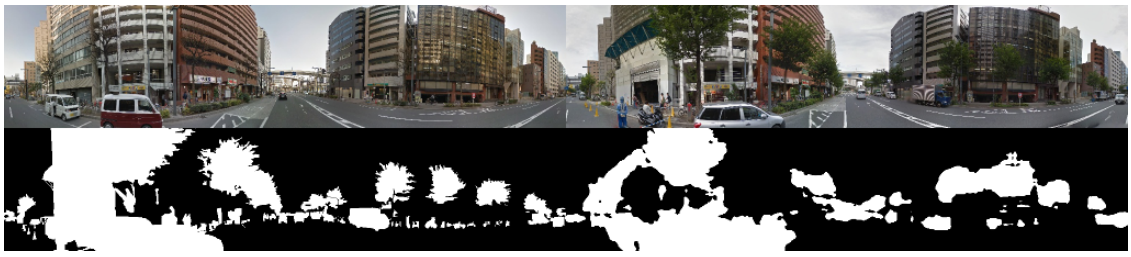
(b) z0%



(c) 20%



(d) 50%



(e) 100%

Figure 4.20: Comparison of different fraction of labeled GSV dataset combination of **MoCO with pretrained ImageNet weights as starting point**. The top row of each image illustrates scenes from two different times. The bottom row of each image compares the obtained result to the ground truth.

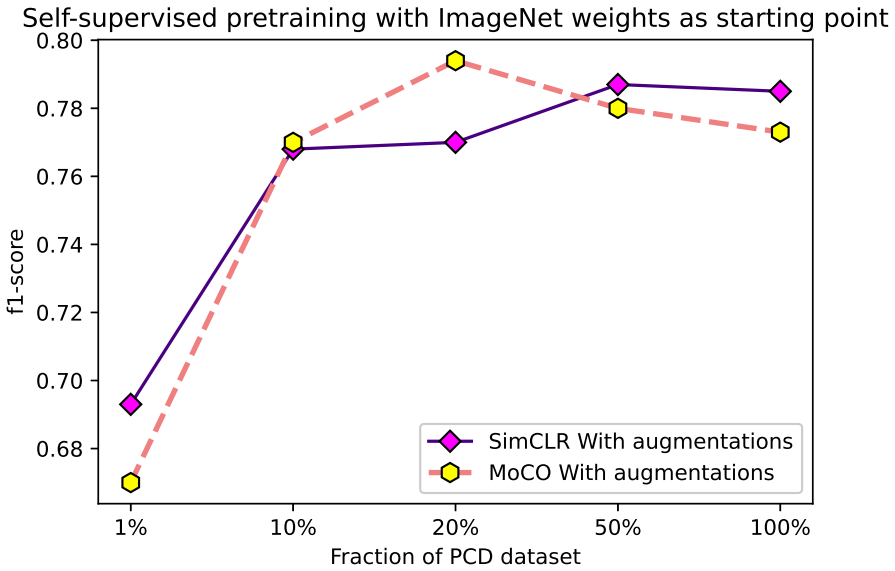


Figure 4.21: Performance (f1-score) of model on PCD dataset using self-supervised pretraining with pretrained ImageNet weights as starting point.

4.1.4 Comparing different pretraining models with baseline supervised ImageNet Model

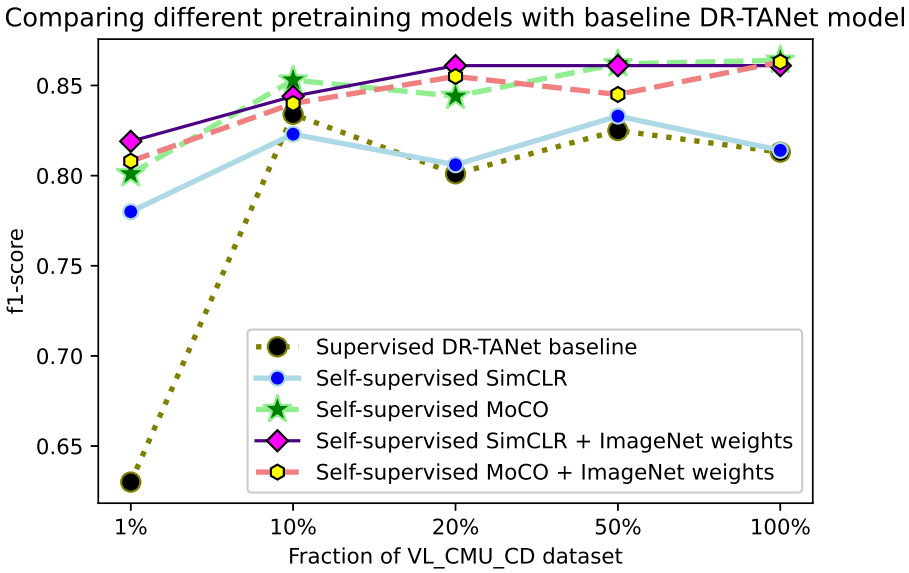


Figure 4.22: Performance (f1-score) comparing different pretraining models with baseline supervised ImageNet model on VL_CMU_CD dataset.

The baseline method chosen in this study is training a model with pretrained ImageNet weights as a starting point. This is further compared with self-supervised

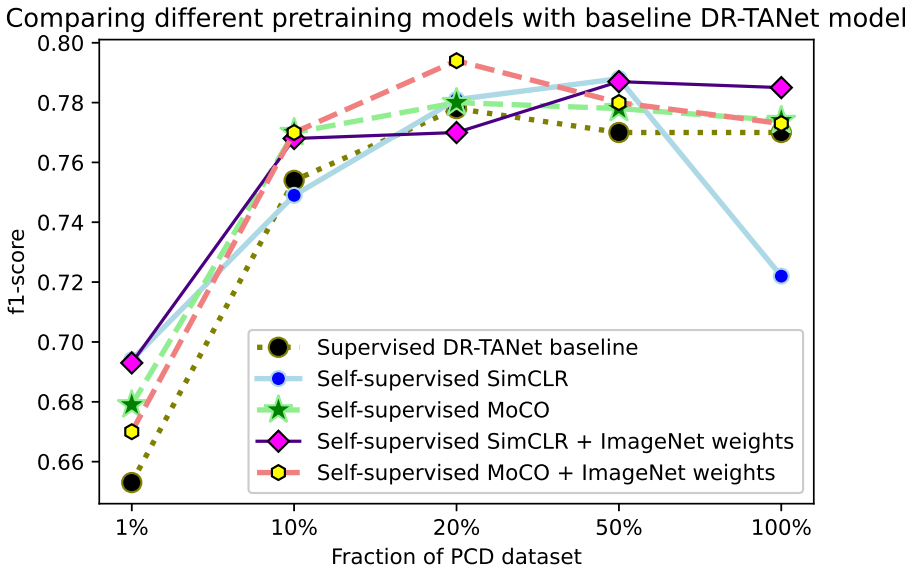


Figure 4.23: Performance (f1-score) comparing different pretraining models with baseline supervised ImageNet model PCD dataset.

and the combination of self-supervised model with supervised ImageNet. As shown in Figure 4.22, an average performance of 20% outperforms other labeled fractions, and the same in Figure 4.23.

5
Discussion & Conclusion

This section provides a conclusion to the project and attempts to answer the most important questions mentioned under the problem statements.

Which pretrained model achieved the best performance?

The results of models pretrained using different methods, evaluated on both the VL_CMU_CD and PCD datasets were presented in the previous section. The performance of the downstream task was tested with various percentages of labeled data. The model which achieved the highest f1-score was the one with self-supervised pretraining with ImageNet weights as starting point. In case of VL_CMU_CD dataset, the pretraining architecture with the best result was SimCLR, while in case of PCD dataset, the pretraining architecture with the best result was MoCO. It is important to note that the difference in performance between SimCLR and MoCO is not very significant. SimCLR had a higher f1-score by 0.005 on the VL_CMU_CD dataset, while MoCO had a higher f1-score by 0.024 on the PCD dataset. Strong arguments can be made in favour of both. While SimCLR creates stronger augmentations to make the model more robust, MoCO uses a memory bank, can store samples to create more negative pairs and as a result can work with smaller batch sizes, mitigating memory issues. The highest f1-score was achieved in the downstream task where 20% of the labeled data was used. The training was also faster than in the cases where 50% and 100% of the labeled data was used. In this study based on Figure 4.22 and 4.23, self-supervised pretraining gave the best results when a low percentage (around 20% or less) of labeled data was used for finetuning, but did not improve the results any further when a higher percentage of labeled data (above 20%) was used.

Limitations of self-supervised models and potential improvements

It was seen that an increase in the amount of labeled data for a downstream task did not improve the f1-score and accuracy of the models - with the various pretraining methods described - tested. This was mainly due to a random selection of labeled data rather than a class-balanced selection. A class balanced selection ensures that during the model would learn and gain information about all classes present during the training process. In the case of a SCD dataset, this would mean having equal amounts of images from all classes of images present in the dataset. However, in this study due to time and feasibility constraints, a random selection of VL_CMU_CD and PCD data does not ensure that images belonging to each class were selected.

This way the model does not learn about each category of images present in the dataset, resulting in lower f1-scores when the model has to predict changes in categories of images it has little to no knowledge. A limitation of this study is that while the proposed method increases performance for VL_CMU_CD and PCD datasets, it was only demonstrated for one backbone architecture (ResNet-18), owing to computing resource limitations. Using a backbone architecture with more layers, such as the ResNet-50 would have potentially led to a further improvement in f1-scores on account of its greater complexity, but would have taken longer to train and required more computational power [25].

The performance of the models (both with self-supervised pretraining and supervised pretraining) reached their maximum performance when 20% of the labeled data was used for finetuning. This is because the models learnt no new information with a further increase in the amount of labeled data. If the backbone architecture were to be changed to a network with more layers like the ResNet-50, it would be interesting to note where the performance of the models peaks. With an increase in the number of layers, the complexity of the network would increase and it may learn more information when the amount of labeled data passed increases. This has scope for further investigations.

Overall discussion

The f1-scores generated from the results and the accuracy of the image were highest when 20% of the labeled was used to finetune the models. Since annotation of images involves a lot of manual labour, small labeled data sets are efficient in many domains, where labeling is a costly and time-consuming procedure. In this SCD scenario, the performance gains obtained by the models for small labeled datasets became more significant. Self supervised learning is not limmited to applications involving curated training sets like ImageNet, but may be useful in applications involving uncurated data, as was seen during the evaluation of models on limited amounts of data from the VL_CMU_CD and PCD datasets. However, the comparison of several models led to the conclusion that a model with self-supervised pretraining starting with ImageNet weights performed better on account of its higher f1-score. It is expected that this research will help to promote the use of self-supervised methods and result in label-efficient and robust models that are suitable for real-world applications.

5.1 Conclusion

Finetuning the DR-TANet model using both self-supervised and supervised pretrained methods on publicly available scene change detection datasets (VL_CMU_CD and PCD) improved f1-scores on downstream tasks. All of the models pretrained using self-supervised learning methods outperformed the models trained using standard supervised learning methods as was shown in figures 4.22 and 4.23. Self-supervised pretraining resulted in good f1-scores even after finetuning with limited labeled data. The non-reliance of self-supervised pretraining on labels, as well as its

better performance made it superior to supervised pretraining.

Another important conclusion is that random selection of data is not the best way to proceed with a downstream task, and instead it is better to perform a class-balanced data selection. A minimum amount of labeled samples for each mini-batch should be set to reduce class imbalance in the dataset. According to the results shown in Figures 4.22 and 4.23, unless finetuning with limited input SCD datasets (VL_CMU_CD and PCD), self-supervised pretraining can help speed up convergence but not necessarily improve the f1-score. Finally, it is worth highlighting that the SCD semantic segmentation task is gaining as much popularity as object detection. This study hopes to motivate future research into self-supervised scene change detection for many applications advantageous to society.

Bibliography

[1] Sakurada, K., Okatani, T. and Deguchi, K. "Detecting changes in 3D structure of a scene from multi-view images captured by a vehicle-mounted camera. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 137-144)", 2013

[2] Priya Goyal, Mathilde Caron, Benjamin Lefaudeux, Min Xu, Pengchao Wang, Vivek Pai, Mannat Singh, Vitaliy Liptchinsky, Ishan Misra, Armand Joulin, Piotr Bojanowski, "Self-supervised Pretraining of Visual Features in the Wild", 2021

[3] R. J. Radke, S. Andra, O. Al-Kofahi, and B. Roysam, "Image change detection algorithms: a systematic survey," IEEE Transactions on Image Processing, vol. 14, no. 3, pp. 294–307, 2005

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in International Conference on Learning Representations (ICLR), 2015

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–77

[6] Enqiang Guo, Xinsha Fu, Jiawei Zhu, Min Deng, Yu Liu, Qing Zhu, and Haifeng L, "Learning to Measure Changes: Fully Convolutional Siamese Metric Networks for Scene Change Detection", arXiv, 2018

[7] K. Sakurada, W. Wang, N. Kawaguchi, and R. Nakamura, "Dense optical flow-based change detection network robust to a difference of camera viewpoints," arXiv preprint arXiv:1712.02941, 2017

[8] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in International Conference on Medical Image Computing and Computer-Assisted Intervention, 2015, pp. 234–241

[9] Ken Sakurada, Mikiya Shibuya, Weimin Wang, "Weakly Supervised Silhouette-based Semantic Scene Change Detection", arXiv, 2018

[10] Shuo Chen, Kailun Yang, Rainer Stiefelhagen, "DR-TANet: Dynamic Receptive Temporal Attention Network for Street Scene Change Detection", arXiv, 2021

[11] Vijaya Raghavan T. Ramkumar, Prashant Bhat, Elahe Arani, Bahram Zonooz, "Self-Supervised Pretraining for Scene Change Detection" NeurIPS, 2021

[12] A. Vaswani et al., "Attention are all you need," in Advances in Neural Information Processing Systems, 2017, pp. 5998–6008

[13] Z. Huang, X. Wang, L. Huang, C. Huang, Y. Wei, and W. Liu, "Ccnet: Criss-cross attention for semantic segmentation," in 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019

[14] H. Hu, Z. Zhang, Z. Xie, and S. Lin, “Local relation networks for image recognition,” in 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 3463–3472
[15] P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens, “Stand-alone self-attention in vision models,” in Advances in Neural Information Processing Systems, 2019, pp. 68–80
[16] Rukshan Pramoditha "How RGB and Grayscale Images Are Represented in NumPy Arrays", towardsdatascience, 2021
[17] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, Stephane Deny "Barlow Twins: Self-Supervised Learning via Redundancy Reduction", PMLR, 2021
[18] Alcantarilla, P.F., Stent, S., Ros, G., Arroyo, R. and Gherardi, R "Street-view change detection with deconvolutional networks. Autonomous Robots, 42(7), pp.1301-1322." 2018
[19] Ting Chen, Simon Kornblith, Mohammad Norouzi, Geoffrey Hinton "A Simple Framework for Contrastive Learning of Visual Representations" 2020
[20] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, Ross Girshick "Momentum Contrast for Unsupervised Visual Representation Learning", arXiv, 2019
[21] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations", arXiv, 2020
[22] Ken Sakurada and Takayuki Okatani, Change Detection from a Street Image Pair using CNN Features and Superpixel Segmentation, Proceedings of British Machine Vision Conference (BMVC), 2015
[23] X. Chen, H. Fan, R. Girshick, and K. He, "Improved baselines with momentum contrastive learning", arXiv, 2020
[24] Iqbal H.Sarker, "Deep Learning: A Comprehensive Overview on Techniques, Taxonomy", SN Computer Science, 2021
[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, "Deep Residual Learning for Image Recognition", arXiv, 2015

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY