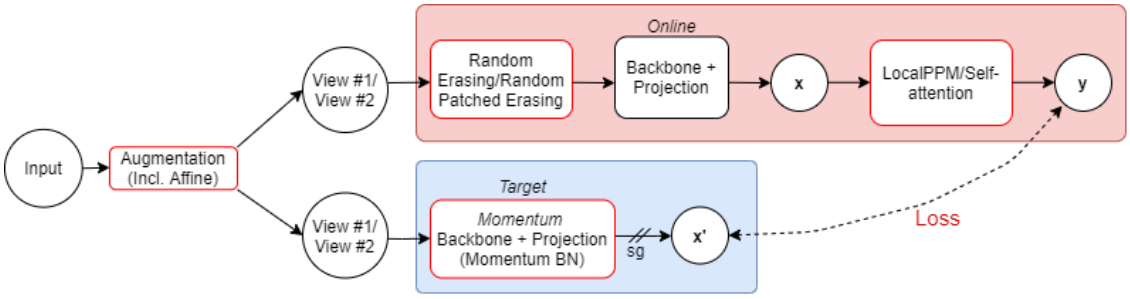
CHALMERS
UNIVERSITY OF TECHNOLOGY



Self-Supervised Representation Learning for Semantic Segmentation

Local Feature Propagation and Strong Data Augmentations for Learning Pixel-Level Feature Representations

Master's thesis in Systems, Control and Mechatronics

MATILDA RENMAN
NICOLE TRAN LUU

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021

Master's thesis 2021

Self-Supervised Representation Learning for Semantic Segmentation

Local Feature Propagation and Strong Data Augmentations for Learning Pixel-Level Feature Representations

MATILDA RENMAN
NICOLE TRAN LUU

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2021

Self-Supervised Representation Learning for Semantic Segmentation
Local Feature Propagation and Strong Data Augmentations for Learning Pixel-Level Feature Representations
MATILDA RENMAN
NICOLE TRAN LUU

Supervisor: Amer Mustajbasic, Volvo Cars
Supervisor: Viktor Olsson, Volvo Cars
Supervisor: Wilhem Tranheden, Volvo Cars
Examiner: Lennart Svensson, Department of Electrical Engineering

Master's Thesis 2021
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Architecture of the proposed method.

Typeset in LaTeX
Gothenburg, Sweden 2021

Self-Supervised Representation Learning for Semantic Segmentation
Local Feature Propagation and Strong Data Augmentations for Learning Pixel-Level Feature Representations
Matilda Renman
Nicole Tran Luu
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Self-supervised learning (SSL) is a machine learning method that recently has gained interest within the research community. The idea is to create algorithms that automatically generate labels, which can be used as supervisory signals when training on unlabelled datasets. Thus, the need for large and costly, manually labelled datasets is reduced. SSL is often used to pretrain a network by solving a pretext task, which is a task created to learn feature representations. Contrastive learning and consistency-based pretext tasks are popular methods for this. The feature representations are then fine-tuned to perform a downstream task.

In this thesis, we study self-supervised representation learning for semantic segmentation. Semantic segmentation is the task of classifying each pixel in an image, which means that the pretraining network needs to learn pixel-level feature representations. Consistency-based pretext tasks have shown success in achieving this, where one strategy is to find consistency in the output from two encoders when they are fed two augmented views of the same image. Here, data augmentations play an important role since it can be used to define pretext tasks for the network to solve and keeps it focused on relevant information.

We explore various data augmentations that have not been applied previously for consistency-based self-supervised semantic segmentation. In particular, we implement affine transformations and different erasing strategies. This creates a prediction task, which forces the network to utilise the full context of the image instead of only focusing on a few visual features. Additionally, we also propose a module that propagates pixel features locally to find the similarity between features. By doing this locally, larger feature maps can be used without increasing the complexity significantly. The thesis also explores the possibilities of applying self-attention to the network and give an evaluation of this.

The final result is a pretraining method that uses both the tested data augmentations and the local propagation module. This method is competitive with state-of-the-art when fine-tuned for semantic segmentation, even when trained with smaller batch size.

Keywords: Self-supervised learning, contrastive learning, computer vision, semantic segmentation, representation learning, consistency learning.

Acknowledgements

We are very thankful for the warm welcome we received from Volvo Cars and for showing interest in our work. We would like to especially thank our excellent supervisors at Volvo Cars, Amer Mustajbasic and Viktor Olsson, for their large amount of commitment and all the support and guidance they have given us throughout the thesis. We would also like to extend a special thank you to Wilhelm Tranheden, who also was our supervisor for the first half of the thesis, and who gave invaluable support when brainstorming ideas. In addition, a big thank you to our academic supervisor Lennart Svensson, at Chalmers, for his expertise and valuable discussions. These have been very supportive and enhanced the quality of our work.

Matilda Renman and Nicole Tran Luu, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1
Introduction

Machine learning is a technology that has gained great interest over the last couple of decades. It has been particularly notable in recent years as the field has evolved and the possible applications for it have increased. A major reason for this is the increase in computational power and accessible data. Both of these factors are important since they lay the foundation for the use of deep learning, which through statistical algorithms, uses the provided data to learn how to perform an assigned task. This way, computers can easily solve problems that are too complex or time-consuming for a human, and thereby save both time and money.

Deep learning can be applied to numerous applications such as self-driving cars, text translation and medical diagnosis. Computer vision is one field that utilises deep learning. The idea is to bridge the gap between human and computer vision and be able to build contextual perception from visual content such as images. A bottleneck for these applications is the need for large amounts of annotated data to train the deep learning network on.

Today, annotating data is done manually, which is both expensive and time-consuming. Because of this, there exist different approaches in research for handling this problem. One example is to use unlabelled data and automatically generate a supervised task for training. This could, for example, be to predict the colours of an image that is created by applying a greyscale filter to a coloured image. The network is fed the greyscale image and outputs a prediction of the colours in the image. Since the original image contains the correct colours, it can be used as a ground truth to compare with the predicted output. Thus, a supervised task is created. This is called self-supervised learning (SSL) and is the focus of this thesis.

SSL can, for example, be used to pretrain a network to learn useful structures and representations within the data without requiring manual labels. The learnt representations can then be transferred to another network and fine-tuned with a few labels for a supervised downstream task. In this way, the supervised network requires fewer labels for training while achieving better results in a shorter training time. A few examples of SSL pretraining methods that achieve state-of-the-art (SOTA) results and outdo the supervised pretraining counterpart can be found in [3], [5] and [6]. Nevertheless, it remains a relatively unexplored area, and there is room for improvement. Most of the current work on the subject focus on learning general representations that are suitable for downstream tasks such as classification. Many methods are also quite heavy to train, which means they might require special

computer resources for training, which are not easily accessible.

Semantic segmentation is the task of labelling and classifying each pixel in an image to a set of predetermined classes, for example, car or person. It is especially beneficial in self-driving cars where the vehicle needs information regarding the context of the driving environment. For example, it can help in lane detection or to identify other necessary information. However, obtaining labels for semantic segmentation is more challenging than for tasks, such as image classification, since each pixel needs to be labelled.

To reduce the need for large pixel-level labelled datasets, SSL pretraining can be used. However, pretraining methods for learning pixel-level feature representations that are more spatially sensitive and useful when discriminating pixels, are underrepresented in current work. The methods that do exist, [2, 7, 8] can be improved to achieve more accurate semantic segmentation results.

One of these, called PixPro [2], is a method that creates a consistency task with the objective to make the same prediction on two versions of the same input sample. In this thesis, this method is used as a basis when creating a new SSL pretraining network. This is done to learn pixel-level feature representations with the aim to improve the representation learning for semantic segmentation. The pretraining network is transferred and fine-tuned for the downstream task of semantic segmentation.

In particular, this thesis studies additional data augmentations than what has been previously tested in this context, and investigates two different similarity modules used to find the similarity between pixels. One of the modules uses local propagation of pixel features to find similarities, which means a larger feature map than previously tested can be used. A larger feature map means more information of the image is kept, which is important for semantic segmentation. The other module utilises self-attention to find the dependencies between pixel features. In addition, a smaller batch size for training is used to create a less computationally heavy model than related work. These areas were chosen for development since they have proven to be successful in different fields within deep learning.

1.1 Problem Statement

The aim of this thesis is to investigate how unlabelled data can be used to pretrain a network to learn pixel-level feature representations suitable for semantic segmentation. This is motivated by the fact that annotated images, especially on pixel-level, are expensive and time-consuming to acquire. To this end, self-supervised learning is used to pretrain a network on a large dataset containing various images, without using labels. One existing pretraining method which has room for improvement, PixPro [2], is used as a base and different mechanisms are added and modified with the aim to improve the representation learning.

The learned features are transferred and fine-tuned on a small, labelled dataset

to perform semantic segmentation. The end goal is to apply the semantic segmentation in a self-driving car to classify the environment while driving. Thus, a dataset containing annotated road images is used for semantic segmentation. To evaluate how good features the pretraining network has learned, the quality of the semantic segmentation is examined. To this end, the evaluation metric mean Intersection over Union is used to evaluate the semantic segmentation. The result is compared with the semantic segmentation from a supervised pretraining baseline and two SOTA pretraining methods. Since several mechanisms are explored, an acceptable outcome of the thesis is to give an evaluation of how well these perform, even if the result is not necessarily an improvement of the supervised baseline or SOTA methods.

1.2 Scope

The work done in this thesis, in large part, explores and builds upon related work within self-supervised pretraining for downstream tasks. The results obtained from our model when transferred to semantic segmentation, are compared with relevant existing works exploring self-supervised pretraining, specifically, MoCo [3, 4], which is a contrastive method, and PixPro [2], which is a consistency-based method. To this end, fine-tuning of parameter settings as well as basic building blocks for the models, such as the choice of network backbone and optimizer, is not the focus of development. The same is applied to the segmentation network, which is chosen to be the same as the one used by MoCo and PixPro to give a fair comparison. Only one dataset, ImageNet [9], is used for the pretraining and one dataset, Cityscapes [1], is used for semantic segmentation. These are chosen since they are the most common benchmarks in related work, and since ImageNet is sufficiently big for the pretraining and Cityscapes contains relevant images for the semantic segmentation.

1.3 Contributions

The main contribution of this thesis is an SSL pretraining method that utilises data augmentation and a local feature propagation module to learn feature representations suitable for semantic segmentation. The network builds upon PixPro [2] but differs in the data augmentations used and the local propagation module. It is also trained with a smaller batch size. The data augmentations used in the developed network include affine transformations and Random Erasing [10]. When developing the network, some general parameter settings for the augmentations as well as an additional erasing strategy were also studied and evaluated. The local propagation module which was developed for the network makes it possible to use a larger feature map, which contains more information, than in the PixPro network without a large increase of parameters. In addition to the local propagation module, a self-attention module was implemented and evaluated.

1.4 Related Work

There exists a range of networks that have been developed for semantic segmentation. The Fully Convolutional Network (FCN) [11] is one that replaces the fully connected layer in a convolutional neural network (CNN) with a 1×1 convolution. U-net [12] and DeepLab [13] are also commonly used networks where U-net builds upon FCN by adding shortcut connections, and DeepLab, amongst other things, adds atrous convolutions which helps with the large downsizing that is a problem with FCN. In this thesis, however, FCN is used for the segmentation evaluation since this is the network used by related methods and thereby will give the most fair comparison.

A few notable works that explore SSL pretraining for computer vision tasks include SimCLR [5, 14], MoCo [3, 4] and BYOL [6]. The first two methods use contrastive learning, which based on the distinctiveness of images seek to output embeddings that are similar for altered samples of the same image (positives) and dissimilar for samples of different images (negatives). The requirement for both positive and negative samples creates quite heavy models which are difficult to train without expensive computer resources. However, if the negative samples are removed, the network could collapse into outputting constant solutions.

A way to omit having negative samples without the network collapsing was found with the development of the third example mentioned above, BYOL [6]. The model consists of an architecture containing two different encoders where one is updated by gradient descent, while the other is a copy of the first encoder but is updated with an exponential moving average. The idea is that when fed different augmented views of the same image, both encoders should output the same feature maps. This can be seen as a teacher-student network where the student network is trained by comparing its output with the output from the teacher network.

BYOL has in turn inspired further development and improved methods. Even if BYOL manages to reduce the batch size without as much loss in performance as contrastive methods such as MoCo [3, 4], it still has a limit after a size of 256 where the performance seems to drop significantly. Momentum batch norm [15] was developed as a response to this. By implementing the Momentum batch norm, a momentum update is performed on the mean and standard deviation to normalize variables. This leads to the possibility of using smaller batch sizes than 256 while keeping the efficiency and stability intact.

Most of the previously mentioned SSL pretraining methods are aimed at learning general features useful for classification tasks. This means that the features they learn might not be optimal for a pixel-level task such as semantic segmentation, since this requires more detailed understanding. As a consequence of this, Xie et al. [2] introduced their method called PixPro which builds upon BYOL by adding a propagation module following one of the two encoders in the network. This module, called pixel-to-propagation module (PPM), computes a similarity transform of

the output from the encoder by, for each pixel feature, propagating all other pixel features within the same image. The model then seeks consistency between the two different branches that make up the architecture. In this way, they achieve SOTA performance on SSL pretraining for semantic segmentation.

One of the investigated methods in this thesis was inspired by inpainting where the network is given an image where a region of the image is missing, which forces the network to predict the content of the missing region. This way, the network has to utilise the full context of the image which can improve its representation learning. To create the missing region, there exist various data augmentation methods such as, Cutout [16], Random Erasing [10] and Hide-and-Seek [17]. Cutout and Random Erasing creates a rectangular region at a random point in the image and fills it with, for example, black or random values. Hide-and-seek splits the image into patches and erases a number of patches randomly.

In this thesis, we also investigate adding a self-attention module to our network. Self-attention in computer vision has been applied in a range of different ways. For example, SAGAN [18] implements self-attention in a Generative Adversarial Network (GAN) to generate images. They introduce a learnable parameter which is used to decide how much of the output from the self-attention layer should be added to the feature map. This is also applied in this thesis. Other networks such as DANet [19] and CMSA [20] use special self-attention structures to capture spatial long-range dependencies. However, neither of the above-mentioned networks are used in pretraining networks for representation learning, which is why we investigate it.

2
Theory

This chapter briefly covers the theory behind the methods and models relating to this thesis. Some prior knowledge in regards to basic concepts relating to machine learning is however assumed. Firstly, a short walkthrough of a few basic and essential deep learning mechanisms are given before semantic segmentation and the difference between supervised-, unsupervised- and self-supervised learning is explained. This is followed by digging deeper into representation learning and how it can be used. In the end, we describe some methods that have been particularly important for the development of our method.

2.1 Deep Learning

The idea behind machine learning (ML) is to teach computers to perform actions and tasks without being explicitly programmed to do so. Given a dataset, the ML model learns to understand patterns found in the data, for example, how the information of images or text sentences is formed. The ML model can then use this when making decisions or predictions. ML can be used in a variety of applications that are too complex to solve for traditional algorithms written by humans.

Deep learning is a subcategory of machine learning, inspired by how the human brain works and it builds upon artificial neural network architectures. The term deep refers to the fact that the neural network may contain numerous hidden layers. By making it possible to create deeper networks, the performance of the learning increases. Deep learning models also have the ability to automatically extract features when given raw data, which is beneficial for a range of applications. On the downside, deep learning requires large amounts of data to perform well. Besides, the models are computationally heavy to train, which makes them not easily accessible for everyone. A few key concepts regarding deep learning that is related to this thesis will be described below.

2.1.1 Loss Function

Training a neural network can be seen as an optimization process where the weights of the model are changed to achieve the desired output. Thus, a loss function is needed which should be minimized during training. Since the loss function essentially calculates the error of the model, it is important to choose a loss function that is suitable for the task at hand. For example, the loss function can be used

to minimize the error between the output from the network and the actual ground truth label in the dataset.

However, for training methods that do not rely on ground truth labels in the dataset, there exist other loss functions that might be more suitable. Contrastive loss is one of these which has been applied in a lot of SSL pretraining methods. The main idea is to find which features are similar and dissimilar. This is done by taking, for example, augmented views of the same image as a positive pair while defining all other images as negatives. The loss will be low if the positive pairs are encoded to be similar and the negatives are dissimilar. Grouping similar and dissimilar features in this way may help with, for example, linear classification since the distinction between features is made clear. There exist several formulations of the contrastive loss function [21, 22, 23] and related work to this thesis uses different variants [3, 5, 7, 8].

In this thesis, the use of negative samples is avoided since these create a heavier model for training. This means that a contrastive loss can not be used since it requires negative samples. Instead, the cosine similarity function is used as the loss function, as in [2],

$$\cos(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||||\mathbf{B}||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=0}^{n} B_i^2}}. \tag{2.1}$$

It calculates the angle, θ between two vectors $\mathbf{A}$ and $\mathbf{B}$ projected in a multi-dimensional space and is a common metric for finding the similarities between vectors. The output is a value between -1 and 1 where 1 corresponds to complete similarity and -1 to dissimilarity

2.1.2 Optimization

The neural network is a complex system and reaching the minimal loss function is not a simple task. Therefore, an optimizer is used to adjust the model parameters and the learning rate throughout the training, to faster reduce the loss towards the global minimum and reach the optimal results. One of the most common optimization methods is the Gradient Descend (GD).

The principal fundament of GD is that for each sample, during the back-propagation, the gradients for each of the parameters are calculated with regards to the loss function. These are used to update the model parameters, θ, towards the direction of the local minimum. The model parameters are also known as weights. However, this is computationally expensive since the gradients for all samples need to be calculated. Another problem is that the algorithm might get stuck in a local minimum instead of a global. Stochastic Gradient Descent (SGD) was developed to prevent these issues. For each iteration, SGD will calculate the gradient for one randomly chosen sample and use it to update θ,

$$\theta = \theta + \eta * \nabla_\theta L(\theta; x^i; y^i). \tag{2.2}$$

This randomness contributes to some interruptions in the updates, which will prevent them from getting held at a local minimum

Here, x^i is the chosen random sample, and y^i is the corresponding labels, and η, is the learning rate (LR) constant that determines the step size and the direction of the update. It is larger when the loss is far from the local minimum and small when getting closer to the minimal loss. The objective function L(θ) is the chosen loss function.

However, the computational complexity also depends on the batch size, and the presented SGD is not optimal when training with a large batch size. Therefore, a Layer-wise Adaptive Rate Scaling (LARS) [24] optimizer can be used instead for stable and accurate optimization when a large batch size is used. This is done by introducing a local LR, η^l, that adjusts between each layer l,

$$\eta^l = \lambda \times \frac{\|\theta^l\|}{\|\nabla_\theta L(\theta^l; x^i; y^i)\|}. \tag{2.3}$$

In the equation for the local LR, a coefficient $\lambda < 1$ is also introduced, which is used to define how much trust is put into the layer to change its weights during one update

2.1.3 Batch Normalization

One problem that further contributes to making the training of a neural network so difficult is that when the layers are updated through back-propagation, the distribution of previous layers changes. This is problematic since the update is based on an expectation that the weights of all prior layers are fixed. This means that the network update keeps chasing a moving target which slows down the training, and it creates a requirement for careful initialization of the parameters. One way to combat this is to use batch normalization (BN) [25]. In BN, a normalization step is used to standardize the activations of each variable from the hidden layers. First the mean μ of the activation x is calculated,

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i, \tag{2.4}$$

where m is the size of the mini-batch. From this, the variance σ^2 can then be calculated,

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2. \tag{2.5}$$

These values can then be used to normalize the activation,

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}, \tag{2.6}$$

where ϵ is a constant added for stability. Finally, the normalized activation is scaled and shifted,

$$x_i = \gamma\hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i). \tag{2.7}$$

The parameters γ and β are learnable and are thus updated during training. This is done for each mini-batch and it stabilizes the training as well as speeds it up.

When the standard BN is implemented for training on multiple GPUs it is called Synchronized BN. The difference is that the standard BN only normalize the data within every single GPU instead of across all GPUs. Synchronized BN is therefore more useful when training with a large batch size. It is for example used in related works such as PixPro [2] and BYOL [6]. A downside to the Synchronized BN is that it is inefficient and may require large batch sizes to achieve stable statistics. As a consequence of this, Li et al. introduced the Momentum BN [15]. This makes it possible to reduce the batch size. Instead of only using the current statistics, an exponential moving average of the historical BN statistics (mean, μ and variance σ^2) are taken to update and normalize

$$\begin{aligned} \mu_t &= (1-\alpha)\mu_t + \alpha\mu_{t-1} \\ \sigma_t^2 &= (1-\alpha)\sigma_t^2 + \alpha\sigma_{t-1}^2. \end{aligned} \tag{2.8}$$

Here, μ_t and σ_t are the running mean and variance at current batch and $\mu_t - 1$ and $\sigma_t - 1$ are from the previous batch. The parameter α is a momentum coefficient.

This adaptation of the BN is useful in teacher-student networks where two encoders are used, and one of the encoders, the teacher, are taken as a slow moving copy of the other, the student. This means the teacher is not updated by gradients, but as a momentum update of the student, see Section 2.4. In order to keep both accuracy and efficiency, the Momentum BN can be applied to the teacher network while keeping the Synchronized BN for the student network. This means that the parameters γ and β are not estimated in the Momentum BN, but are taken as momentum versions from the student.

2.1.4 Dilated Convolution

Convolutional layers are a common component in neural networks, especially in combination with image data. A convolutional layer can be seen as a filter of weights, a kernel, that is applied to the input in a sliding window fashion. The dot product is computed between the filter and each filter-sized patch of the input data it slides over. This operation leads to the output of a feature map, and makes it possible for the network to discover and learn spatial features in order to make sense of the content in an image. In Figure 2.1, a standard convolution is illustrated.

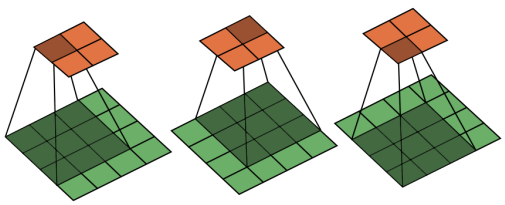


Figure 2.1: A standard convolution with a 3×3 kernel moving over a 4×4 input map (green). This results in an output (orange) of size 2×2.

Dilated, or atrous, convolution is a variant of convolution where a spacing in the form of zeros is introduced in between the values in a kernel. This increases the receptive field without increasing the computational cost since the number of parameters is kept the same. An illustration of a dilated convolution can be seen in Figure 2.2.

Since the downsampling in standard convolutions leads to a loss in spatial information, this technique has proven especially beneficial for networks that are used for dense prediction tasks [13, 26]. A higher resolution in the outputted feature map can be obtained by replacing the stride, which is the amount the kernel is shifted, with dilation. The dilation helps with keeping a larger receptive field without increasing the complexity.

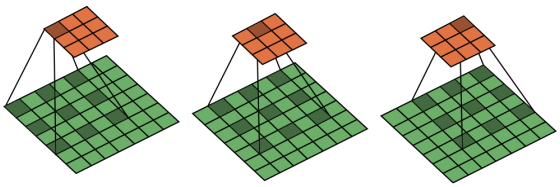


Figure 2.2: A dilated convolution. The dilation is set to 2 here which leads to a spacing of 1 between the values of the kernel which has size 3×3.

2.1.5 ResNet

A common approach for solving complex problems using neural networks is by adding more layers to the model, making it deeper. This way, increased accuracy can be reached as the layers progressively learn more features. However, this gives rise to another problem in the form of vanishing or exploding gradients, which in turn means that the error rate also increases. To solve this, a new architecture called Residual Network, or ResNet for short, was introduced [27]. The idea is that an initial mapping $\mathcal{H}$ can just as well be represented by a residual mapping $\mathcal{F} := \mathcal{H} - x$. This in turn gives $\mathcal{H} := \mathcal{F} + x$. The network then uses skip connections added to a feed-forward network, as can be seen in Figure 2.3. The skip connections perform identity mapping and connect to the outputs of the stacked layers. This gives the gradient an alternative shortcut to flow through which helps in solving the vanishing and exploding gradient problem.

ResNet can, for example, be used as a backbone structure of an encoder. This means that it is the network used to extract the features from the input data, and it is this way ResNet is applied in this thesis.

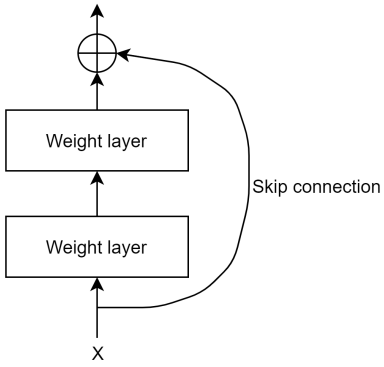


Figure 2.3: Building block of ResNet.

2.2 Semantic Segmentation

Semantic segmentation is a computer vision task where the goal is to classify each pixel in an image into a set of predefined classes. Compared to image classification, where the objective is to classify the image as a whole, this is considerably more difficult. One of the challenges with semantic segmentation is that it requires a detailed understanding of the semantics in the images. In Figure 2.4 an example of semantic segmentation of an image is shown. The segmented output can be seen in the image to the right and each coloured region corresponds to a class label.

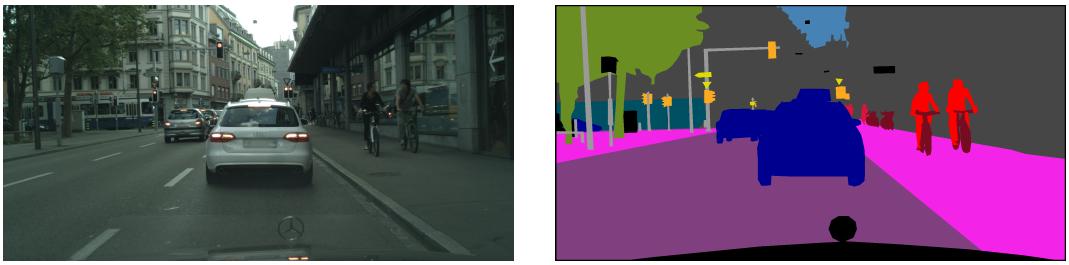


Figure 2.4: An illustration of a semantic segmentation of an image. The original image can be seen to the left and the corresponding segmentation to the right. The images are taken from the dataset Cityscapes [1].

2.2.1 Semantic Segmentation Network

In this thesis, the Fully Convolutional Network (FCN) [11] is used for the semantic segmentation task. FCN is one of the more commonly used semantic segmentation models, it is for example used by MoCo [3, 4], BYOL [6] and PixPro [2]. The evolution of this network was inspired by the classic convolution neural network (CNN), in which the encoder consists of multiple convolutional-, pooling- and activation layers and the decoder consists of upsampling layers used to perform pixel-wise prediction, see Figure 2.5.

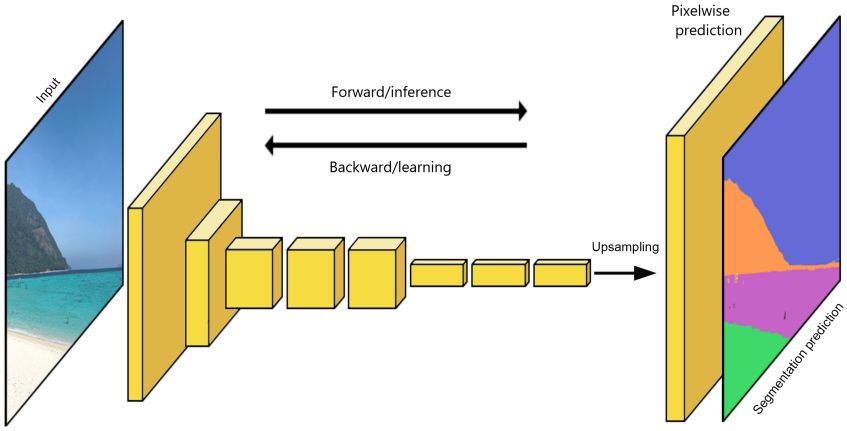


Figure 2.5: Architecture of a FCN model.

There are three types of FCN: FCN-32, FCN-16, and FCN-8. The main difference between them is how fine details they capture. FCN-32 has the most upsampling, 32, and thereby gives the roughest output. This is because deep features can be obtained when going deeper into the network, but positional information is also lost. FCN-8 in turn gives the finest boundaries in the output. In this thesis, FCN-16 is used for semantic segmentation since this is used by previous work.

2.3 Supervised-, Unsupervised- and Self-Supervised Learning

The more established methods for semantic segmentation are trained using supervised learning. However, due to the fact that supervised learning requires a manually labelled dataset, which is expensive and time-consuming to acquire, the interest in unsupervised learning has grown and been the focus in numerous recent research.

In supervised learning, the network learns a pattern to map the input features to the corresponding output labels. This is done by including ground truth labels, which are used as targets for the network to train against. The objective is that the predicted output should be as close to the target as possible. Supervised learning is generally defined into two categories, classification and regression. Classification maps the outputs into different categories, while regression maps the output to real values. A standard classification example is the cat-dog problem, where the network is trained on annotated images of dogs and cats to learn to classify if an image contains a dog or a cat. A regression problem could be, for example, predicting the price of a house given the size of the house.

The opposite of supervised learning is unsupervised learning, which learns from unlabelled data. The network tries to learn a trend of the input data and output

an underlying structure or distribution of the data. A common example of unsupervised learning is clustering, where the data is divided into groups with certain similarities. It is more difficult for a network to learn how to perform a given task without having any type of ground truth to compare with. This means that an unsupervised network may require more data and training time, and there is a higher risk for inaccurate results.

Self-supervised learning (SSL) is a learning method where a supervisory signal is created from unlabelled data. For example, as illustrated in Figure 2.6, a supervised task can be created by simply rotating the images. This way, the semantic information of the image stays intact and can be used to predict the angle of rotation. In a way, SSL can be thought of as fusing supervised learning and unsupervised learning, getting the main advantages of both techniques. SSL is the focus of this thesis.

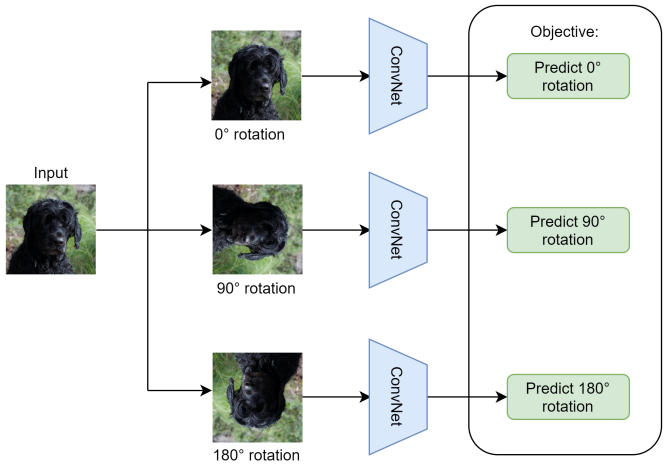


Figure 2.6: Illustration of SSL where the input image is rotated and the network is used to predict the amount of rotation.

2.4 Representation Learning

SSL is often used to pretrain a network to learn good visual representations of an image. The idea is that the learnt parameters can be transferred and act as an initialization of the weights for a supervised downstream network. This is useful when there are insufficient annotated data samples to train the downstream network directly. The downstream network can then be trained on the actual downstream task, see Figure 2.7. The downstream task can, for example, be classification, object detection, or semantic segmentation. The quality of the downstream tasks can be used to evaluate how good feature representations the pretraining network has learnt.

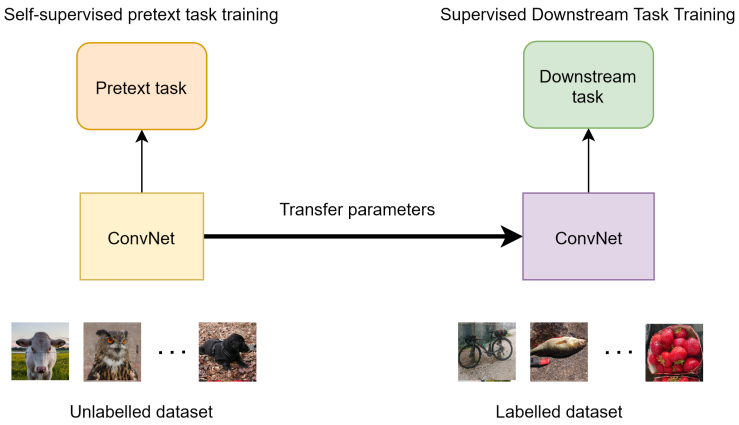


Figure 2.7: Illustration of the general pipeline for self-supervised learning. The representation features are learned by training a network to solve the pretext task. When this training is finished, the learnt parameters can be transferred and fine-tuned to perform a downstream task such as classification or semantic segmentation.

There exist various approaches for learning feature representations, where most are based on visual pretext tasks. A pretext task is what the network aims to solve in order to learn representations, and there exist several different techniques that can be used. Most of the time, the pretext task is made up and how well the network performs this task in itself is unimportant. What is important are the representations, which may include good semantic or structural meanings, that the network learns while solving the pretext task. For example, a pretext task could be something as simple as predicting the rotation in an image [28], as illustrated in Figure 2.6, or something more challenging such as inpainting, which means that the network should predict the missing pixels in an image with an erased section [29].

As shown by several SOTA models, contrastive learning is one way to form a pretext task that provides reliable training results [3, 5, 8, 21]. This means that a contrastive loss is used to learn good representations where, in the image representation space, similar features are pulled together and dissimilar features are pushed apart. The similar features are defined by positive pairs, which can, for example, be two cropped views of the same image, and the dissimilar features are defined by negative samples, which would then be all other images. In other words, contrastive learning can be thought of as comparing the inputs to find which samples are similar and which are not.

Several of the above-mentioned methods are focused on instance discrimination suitable for classification tasks, which are more simple relative to semantic segmentation tasks. For tasks such as semantic segmentation, features need to be learned on pixel-level to include more details. There is also an issue with the need to define both positive and negative samples to keep a stable network. By requiring both, large batch sizes are needed, which are not computationally efficient. As an example, SimCLR [5] shows that the contrastive learning benefits from larger batch sizes ranging

from 256 to 8192. This requires special cloud TPUs (Tensor Processing Units) which are not commonly accessible. The risk, however, with omitting the negative samples when training is that the network possibly could collapse into outputting the same vector despite what image is used as input.

In order to prevent this, pretraining methods such as BYOL [6] were developed. The inventors of BYOL found a way to train a network to find representations without requiring negative samples. The network is asymmetrical with two parallel encoders where one, the online, is updated as regular by gradients while the other, the target, is a momentum encoder taking an exponential moving average (EMA) of the online encoder. This means that if denoting the parameters of the online encoder by θ, the parameters of the target encoder by ξ, and the momentum coefficient by $m \in [0, 1]$, we have:

$$\xi \longleftarrow m\xi + (1 - m)\theta. \tag{2.9}$$

The collapsing of the network is avoided in BYOL, in large part because of the stop-gradient that is applied to the target encoder [30]. The stop-gradient prevents the parameters in the target encoder from being updated by gradients in the direction of minimal loss, and this prevents the network from outputting a constant representation.

The network can be seen as a teacher-student network where the goal is to train the online encoder, the student, by comparing its output with the output from the target encoder, the teacher, and seek consistency between them when they are each fed with two differently augmented views of the same image. When the training is completed, only the parameters from the online encoder are kept. An overview of the network can be seen in Figure 2.8.

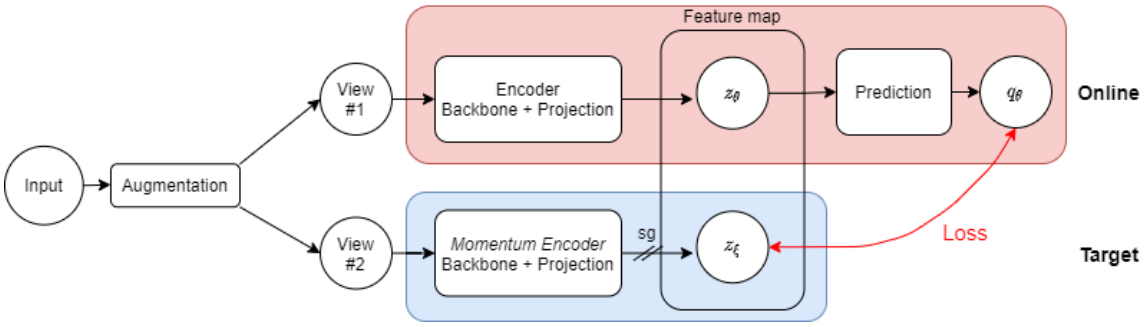


Figure 2.8: The architecture of BYOL. Two differently augmented views of the input image are fed to the two branches. The encoders in both branches consist of a backbone network and a projection head. The target encoder is updated by taking an EMA of the parameters from the online encoder. This means that a stop-gradient (sg) is applied after the target encoder in order to prevent gradient update. This also prevents the network from collapsing into trivial solutions. The encoders output feature maps z_θ and z_ξ respectively. The online branch also includes a prediction head which consists of linear layers. This creates an asymmetrical network. The loss is minimized between the prediction q_θ and z_ξ.

2.4.1 PixPro

PixPro is a pretraining method introduced by Xie et al. [2] which has the same base asymmetrical architecture as BYOL [6]. The model focuses on learning pixel-level feature representations and achieves SOTA results when transferred for object detection and semantic segmentation, while it remains relatively lightweight since it does not require negative samples. Having a more lightweight model means it does not require as much computational resources, which means it is easier to train. For these reasons, it was used as a base when developing our model. An illustration of the network architecture can be seen in Figure 2.9.

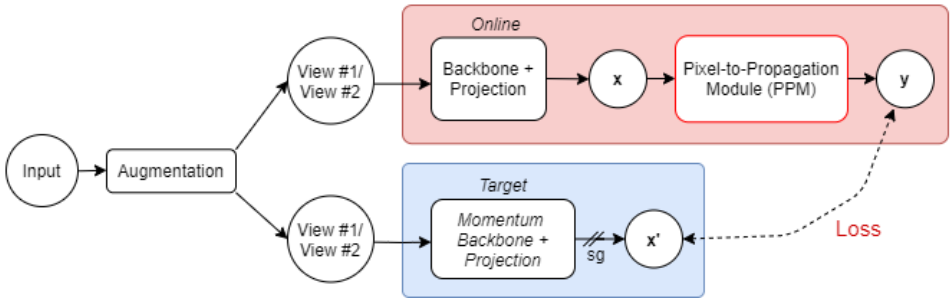


Figure 2.9: The architecture of PixPro. Similarly to BYOL, two augmented views of the same image are taken and sent through an online and a target encoder. The output is then projected into feature maps $\mathbf{x}$ and $\mathbf{x}'$. In contrast to BYOL, the prediction head here is the PPM that outputs a transform $\mathbf{y}$. The loss is taken between $\mathbf{y}$ and $\mathbf{x}'$ where consistency in the output is sought. At the end of the training, as in BYOL, only the parameters of the online encoder are kept and used for transferring.

The main difference between BYOL and PixPro is the additional module in the prediction head, pixel-to-propagation module (PPM). The idea is that this module should encourage spatially close pixels to be similar, which may be beneficial in areas where the pixels belong to the same label. In this module, a transform y_i is computed for each pixel feature x_i by propagating all other pixel features x_j within the same image, Ω, to it

$$y_i = \sum_{j \in \Omega} s(x_i, x_j) \cdot g(x_j). \tag{2.10}$$

Here, g$(\cdot)$ is defined as a linear transformation function. It is instantiated by a linear layer followed by BN and a ReLU activation function. The similarity function, s$(\cdot,\cdot)$, is defined by the cosine similarity function, as in eq. (2.11), with γ being an exponential used to decide the sharpness of the function with default value 2

$$s(x_i, x_j) = (\max(\cos(x_i, x_j), 0))^{\gamma}. \tag{2.11}$$

The feature maps, $\mathbf{y}$ and $\mathbf{x}'$, outputted from the two branches in the network are encouraged to be consistent. Only the positive pixel pairs are considered when seeking consistency and they are assigned according to an assignment rule,

$$A(i,j) = \begin{cases} 1, & \text{if dist(i,j)} \leq \tau, \\ 0, & \text{if dist(i,j)} > \tau, \end{cases} \tag{2.12}$$

where i and j are pixels from two different, augmented views of the same image. The threshold τ is set by default to 0.7 and dist(i,j) is the normalized distance between pixels i and j in the representation space. The positive pixel pairs i and j are then used to formulate the loss function

$$\mathcal{L}_{PixPro} = -\cos(y_i, x'_j) - \cos(y_j, x'_i). \tag{2.13}$$

The final loss is calculated as an average over all positive pixels in the same image, and then as an average over the images in the mini-batch. This loss is used to back-propagate and update the weights in the online encoder.

2.5 Attention

Attention in deep learning can be interpreted as adding weights of importance to an input. In particular, it has been important for natural language processing, such as translation applications, by utilising attention as a mechanism to find the relationship between words in a sequence. For example, in the sentence *The cat is eating the mouse*, attention will help in finding the relation between *eating* and *mouse*. Attention can also be used in computer vision and intuitively this can be related to how humans focus on certain parts of a visual input in order to understand the context. In simple terms, attention can be thought of as memory through time.

2.5.1 Self-Attention

Self-attention is a mechanism used to find and embed how related each hidden state is to all other hidden states. This is done using queries, keys, and values, which are vectors comprised of the input vectors to the self-attention layer. By adding weights to these vectors, the matrices Q (query vectors), K (key vectors), and V (value vectors) are formed, which are the learnable parameters that are updated during training. The output from the self-attention layer is calculated by Scaled Dot-Product Attention [31]

$$\text{Attention}(Q,K,V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{2.14}$$

Here d_k is the dimension of the queries and keys. The scaling by d_k comes from the fact that the Softmax function can be sensitive to very large input values.

If looking at the PPM, as was described in the previous section, it can intuitively be thought of as a modified self-attention module without learnable Q and K, see Figure 2.10. The similarity function seen in eq. (2.11) can be considered as the dot product between Q and K, without learnable weights, and the linear transformation

function g(·) can be related to the values matrix V in self-attention. The PPM and self-attention also have similar purposes, which are to find the similarities in the input embedding. For this reason, self-attention is further investigated in the development of our method.

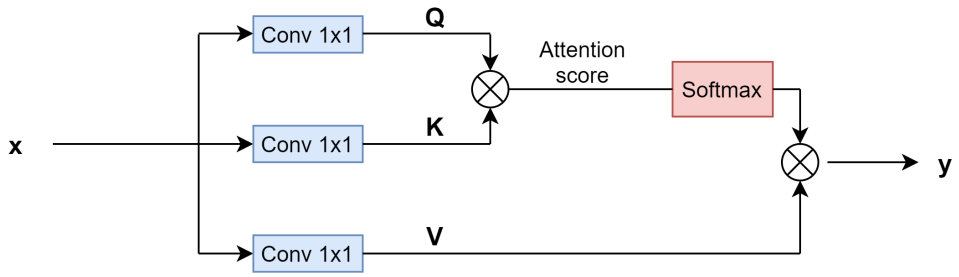


Figure 2.10: Illustration of a self-attention module used for computer vision.

The transformer [31] is a popular model which relies completely on self-attention. The model also includes positional encoding (PE) to keep track of the relative position of the input. The PE can, for example, consist of sinusoidal waves as in [31]. These positions are added to the embedded representations that are fed to the self-attention layer. When evaluating self-attention in our model, positional encoding is added to keep the spatial information intact.

3
Methods

This chapter presents the overall network architecture for the method developed in this thesis. The base architecture of the network is the same as the one used in PixPro [2], which in turn is based upon BYOL [6], with a few modifications. These modifications and how they are implemented are described in detail following the description of the network architecture. Two areas have mainly been investigated, data augmentations and the prediction head of the network. These areas have proven essential for the success rate of related work in terms of creating pretext tasks and keeping the learned information relevant. Subsequently, the datasets used and the data augmentations applied for creating the views going into the network, are presented.

3.1 Network Architecture

The network we have used for pretraining is built upon the network used for PixPro [2], as is described in Section 2.4.1, with the added mechanisms that are investigated in this thesis. These are described in detail in the following sections. The objective of the network is to seek consistency in the outputted feature vectors. In Figure 3.1, the suggested architecture can be seen, including the studied mechanisms, which are affine transformation, erasing, LocalPPM, Momentum BN, and self-attention.

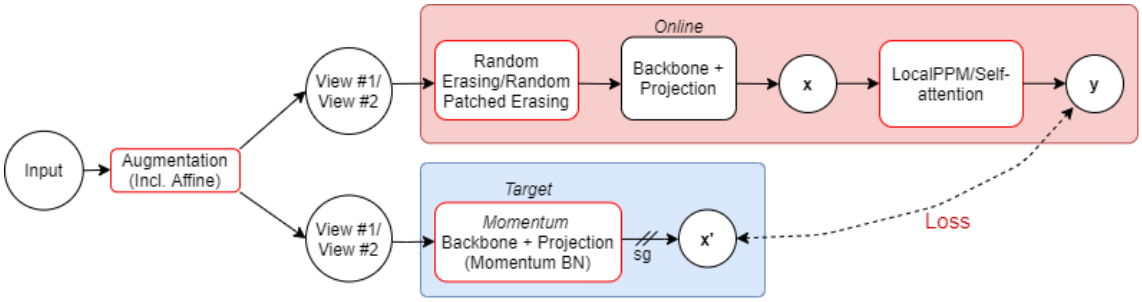


Figure 3.1: Illustration of the network architecture used for our method. In this image, all tested mechanisms are included to give an overview. In the final method, only affine transformation, the Random Erasing, the Momentum BN, and the LocalPPM are included. The base of the network is the same as in PixPro [2], and the objective of the network is to seek consistency between the feature vectors $\mathbf{y}$ and $\mathbf{x}'$. At the end of the training, only the parameters in the online encoder are kept for transferring.

A batch of images are taken as input to the network and data augmentations are applied to each image. This includes taking two differently cropped views, v_1 and v_2, of the same image, and letting both views pass through both the online- and the target encoder. One of the two views, v_1, is slightly more strongly augmented than v_2 since it also includes the affine transformation. The affine transformation is added to modify the geometrical structure of the images to see if this could improve the learning.

Before passing through the online encoder of the network, part of the view is erased. This is another modification made to the network to create a prediction task where the network needs to predict the missing part. The idea is that this will force the network to leverage the full context of the image and not only focus on a few visual features.

The online encoder consists of a ResNet-50 [27] backbone using synchronized batch normalization and a projection head, and it outputs a feature map $\mathbf{x}$. The feature map $\mathbf{x}$ is either propagated through the LocalPPM or the self-attention module, depending on which mechanism is used, and this gives the transform $\mathbf{y}$. The LocalPPM is a modified version of the PPM in PixPro [2], and it makes it possible to use larger feature maps without a big increase in complexity by propagating pixel features locally. The self-attention module is a simple self-attention structure used to investigate if the network could benefit from finding similarities through self-attention.

The target encoder also consists of a ResNet-50 backbone and a projection head. In addition, Momentum BN [15] is applied to the target encoder. This is added to the network to make it possible to train it on a smaller batch size without a loss in performance. A stop-gradient operation is applied to the output of the target encoder, which is updated as a momentum encoder by taking an EMA of the online encoder's parameters. The target encoder outputs a feature map $\mathbf{x}'$.

As mentioned, the objective of the network is to output consistent feature representations from each branch regardless of what view of an image is processed. This is the same objective as in PixPro [2] and the same cosine similarity loss function is used, seen in eq. (2.13), with the same average calculations and assignment rule to find the positive pixel pairs, see eq. (2.12). If there is no overlap between the cropped pair of views, then no loss is computed. However, the majority of all cropped pair of views do contain at least some overlap. At the end of the training, only the parameters learnt in the online encoder are transferred for semantic segmentation in a FCN network.

3.1.1 LocalPPM

The size of the feature map has a big impact on the segmentation performance since it is related to the receptive field. A larger feature map includes more information of the image, which is important for semantic segmentation where a detailed un-

derstanding of the image content is necessary. By replacing the stride with dilation of 2 in the fourth block of the ResNet-50 [27] backbone used in both encoders, the feature map dimension is doubled to a size of 14×14 compared to the previous size of 7×7. However, a larger feature map also occupies more memory, which creates a heavier model requiring more computational resources when looking for similarities between pixels. To reduce the increased complexity this entails, the LocalPPM was developed, which is inspired by the PPM in PixPro [2] and the concept of local self-attention [32].

The basic idea of the LocalPPM is the same as the PPM in PixPro where pixel features are propagated to encourage spatially close pixels to be similar, see Section 2.4.1 for a detailed explanation. The difference is that instead of propagating all of the pixel features within the same image, only the pixels within a local section are propagated. That is, the feature map is divided into local sections that fit into a frame of size n×n and only the pixel features that fit into the same frame are propagated to each other, see Figure 3.2. This reduces the amount of computations needed since fewer parameters are used to compute the transform of each pixel feature. Combining this with a larger feature map should give the best of two worlds, more spatial information is kept with the higher resolution, while the necessary computations this imposes can be reduced.

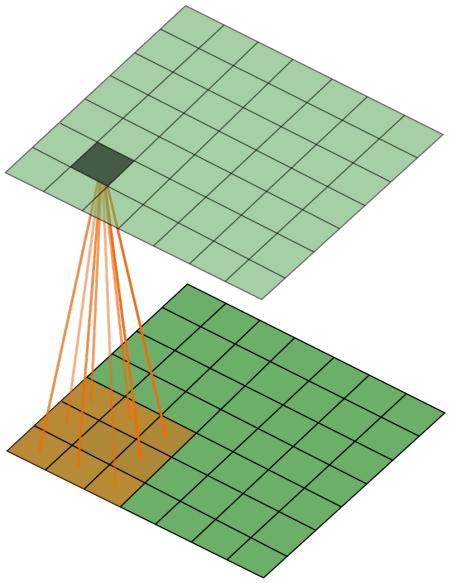


Figure 3.2: Illustration of the idea behind the local propagation in the LocalPPM. The orange shows a local area that, in this case, fits within a frame of size 3×3. Each pixel feature within this local area is propagated with all other pixel features within the same local area. The whole feature map is divided into local sections like this, where pixel features are propagated locally. Zero padding is added to the feature map in case this is needed to fit the frames.

3.1.2 Self-Attention Module

Another concept studied in this thesis is if further modifications to the prediction head could lead to any improvement. The PPM used in PixPro [2] has a similar purpose as self-attention, where both mechanisms seek to find the similarities in the input. In addition, the PPM has a comparable structure to self-attention. If the input to the similarity function, see eq. (2.11), in the PPM can be thought of as queries and keys, one difference from self-attention is that these do not have learnable weights. Another difference is how the linear transformation function g(·) in eq. (2.10), which can be considered as the value matrix in self-attention, is defined. The intuition behind why these design choices of the prediction head were made in the original article [2] is not clear. Thus, we decided to investigate this further by replacing the PPM with a simple self-attention module.

The first step of the self-attention module is to add PE to the input, which provides the positional information of each pixel in the image [33, 34]. This has shown to be essential for transformers where self-attention is used, especially for larger data such as images. As described in [34], since the self-attention does not contain the absolute position of objects in the images, it is difficult for the network to learn and identify the relationship between objects or pixels. The PE is implemented as an extended Sinusodal Positional Encoding [31], see eq. (3.1). The PE has the same dimension, d_x, as the feature map used as input to the self-attention module, and it is added to the input before passing through the self-attention module. Since the PE in [31] was developed for text sentences, which are one-dimensional vectors, the mechanism is adapted for image inputs by repeating the same equation in both x- and y-direction, following [34]

$$PE_{(pos,2i)} = sin\left(\frac{pos}{10000^{\frac{2i}{d_x}}}\right), \tag{3.1a}$$

$$PE_{(pos,2i+1)} = cos\left(\frac{pos}{10000^{\frac{2i}{d_x}}}\right). \tag{3.1b}$$

Here *pos* is the position index and i is the dimension. The dimension, i, plays an important role to produce unique PE for each of the pixels, which is described in detail in [31, 34].

Motivated by the articles [18, 19], the weights for Q, K and V are implemented as 1×1 convolutional layers, which are updated by the optimizer. The feature map $\mathbf{x}$ is fed through these layers to generate Q, K and V. The Q matrix is multiplied with K to obtain an attention score, to which a softmax function is applied. The output from the softmax function is multiplied with V to achieve the self-attention output, o. This is as summarized in eq. (2.14).

The output, o, from the self-attention calculation is defined as a correlation to the input, $\mathbf{x}$:

$$\mathbf{y} = \mathbf{x} + o * \gamma. \tag{3.2}$$

Here, γ is a learnable scalar initialized as 0. The motivation behind this is to let the network first focus on the local area and gradually put more weight on long-range dependencies [18, 20, 35] . An illustration of the full self-attention module can be seen in Figure 3.3.

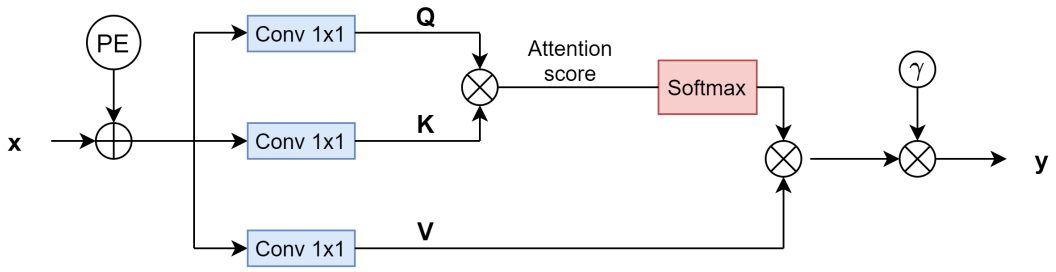


Figure 3.3: Illustration of the proposed self-attention module.

3.1.3 Erasing Augmentation

Data augmentations play an essential role in SSL representation learning since they can be used to create pretext tasks for the network to solve to learn features. The pretext task often relies on some sort of prediction, for example, and as mentioned previously, to predict the rotation of images or the colour map of a greyscale image.

Inspired by augmentation methods such as Cutout [16] and Random Erasing [10] as well as inpainting [29] and various other predictive methods [36, 37], we apply a prediction pretext task to our model by erasing part of the view going into the online encoder. By only applying the erasing before the online encoder and keeping the view going into the target encoder unchanged, the output from this can be used as a supervisory signal to predict the missing region in the online encoders output. The reasoning behind why this should improve the learning is that the network is forced to utilise the full context of the image, instead of focusing on a few visual features.

The implementation of this is done in two different ways, which is illustrated in Figure 3.4. In Figure 3.4b, Random Erasing [10] is shown which erases a randomly selected, rectangular region I_e within an image I with size $S = H \times W$. First, a point is randomly initialized, $\mathcal{P} = (x_e, y_e) \in I$. This point determines the position of the erasing region. Thereafter, the area to erase, $S_e \in [s_l, s_h]$, is also randomly initialized where the ratio range is defined by $\frac{S_e}{S}$. The aspect ratio r_e of the rectangle is in turn randomly initialized between $[r_1, r_2]$. The size of I_e is selected as

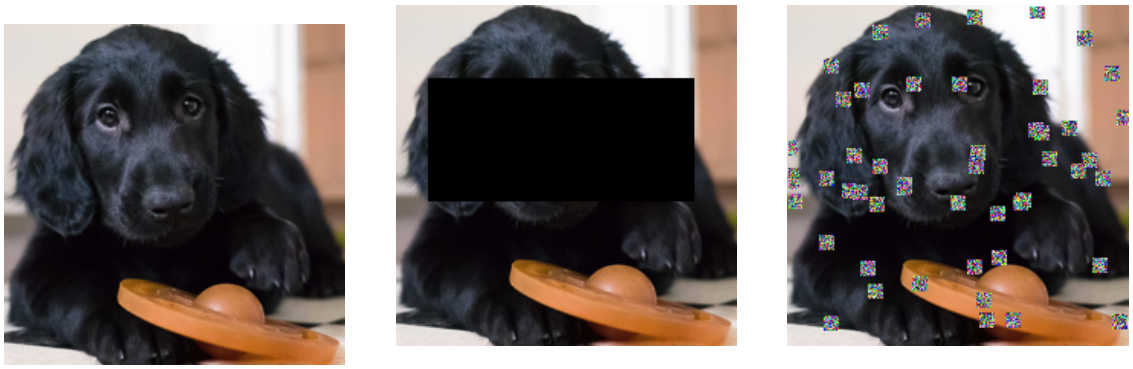
$$H_e = \sqrt{S_e \times r_e}, \quad W_e = \sqrt{\frac{S_e}{r_e}}, \tag{3.3}$$

and if the condition that $x_e + W_e \leq W$ and $y_e + H_e \leq H$ holds, then the region to erase is selected as

$$I_e = (x_e, y_e, x_e + W_e, y_e + H_e). \tag{3.4}$$

Each pixel within this region is set to zero, i.e., the region is filled with black, or to Gaussian noise across all three channels, depending on settings. The probability for Random Erasing to be applied to the image is p, which means that the probability of it being kept unchanged is $1 - p$.

The second way, named Random Patched Erasing, is illustrated in Figure 3.4c, and is inspired by [17, 38]. In this augmentation, the erased regions are randomly spread out in small patches. This is in contrast to Random Erasing which is only applied in one larger rectangular region. By instead using smaller and spread out regions, different visual features in the image may be occluded instead of blocking out one large part. The implementation is similar to Random Erasing. The positions of the patches are randomly selected to create vectors $\mathbf{P_x}$ and $\mathbf{P_y}$ containing x and y positions of length n, where n is the total number of patches. The size of the patches is a square with side length s. A mask is then created as $\mathbf{M} = (\mathbf{P_x} - s, \mathbf{P_x} + s, \mathbf{P_y} - s, \mathbf{P_y} + s)$ and applied to the image. The masked region is then either filled with black or with Gaussian noise across all channels depending on what setting is used. A probability for applying the augmentation is set in the same way as for Random Erasing.



(a) Original image.

(b) Image with Random Erasing.

(c) Image with Random Patched Erasing.

Figure 3.4: Two different implementations of erasing were tested. For the Random Erasing in (b), $s_l = 0.02$, $s_h = 0.4$, $r_1 = 0.3$ and $r_2 = 1/0.3$. For the Random Patched Erasing in (c), 50 patches of size 10×10 was used. As seen, the patches may overlap each other. The area erased can either be filled with black as seen in (b) or with Gaussian noise across all three channels as seen in (c).

3.2 Data

Two public datasets are considered in this thesis. ImageNet is used for the pre-training network since it is a large and accessible dataset, and as it is the dataset used in related work. For training the semantic segmentation network, Cityscapes [1] is used, which contains pixel-level annotations on road images and is a standard benchmark for semantic segmentation.

3.2.1 ImageNet

ImageNet is a large public dataset widely used within research. The full dataset contains millions of annotated images from a wide selection of categories [9]. Some of the high-level categories include animals, fruit, and flowers. These are, in turn, divided into several sub-categories.

In this master's thesis, the dataset from ImageNet Large Scale Visual Recognition Challenge 2017 (ILSVRC2017) is used as the training set for the pretraining [39]. This training set is often referred to as ImageNet-1K and has a total of 1,281,167 images for training and contains 1000 different classes.

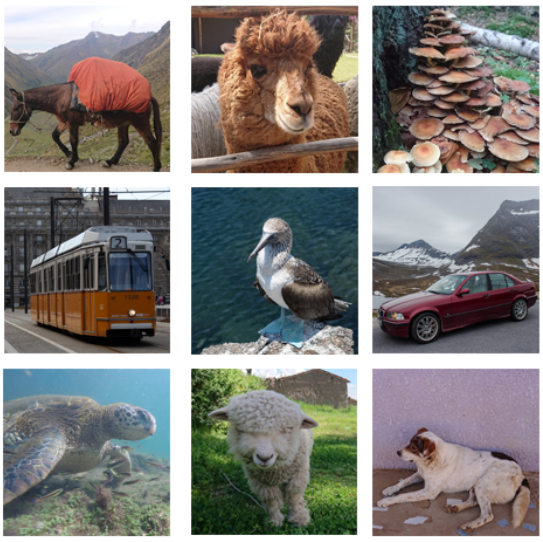


Figure 3.5: Examples of what images in the ImageNet dataset might look like.

3.2.2 Cityscapes

The public dataset, Cityscapes, contains a mixture of images taken in an urban environment by a camera placed in the front of a car, and it includes pixel-level annotations [1]. The dataset is freely available to use for research or personal experimentation. It is comprised of images of street scenes from 50 different cities during different seasons of the year and with varying weather conditions, all with a resolution of 2048 $\times$ 1024. In addition, the dataset contains 5000 finely pixel-level annotated frames as well as 20 000 coarsely annotated frames. In this thesis, only the finely annotated frames are used. These are split into 2975 images for training, 500 for validation, and 1525 for testing. The dataset has in total 30 different classes, but since some are rare, they are excluded from the benchmark leaving 19 remaining classes. The class distribution of the labels can be seen in Figure 3.6. An example image with a corresponding segmentation map can be seen in Figure 2.4.

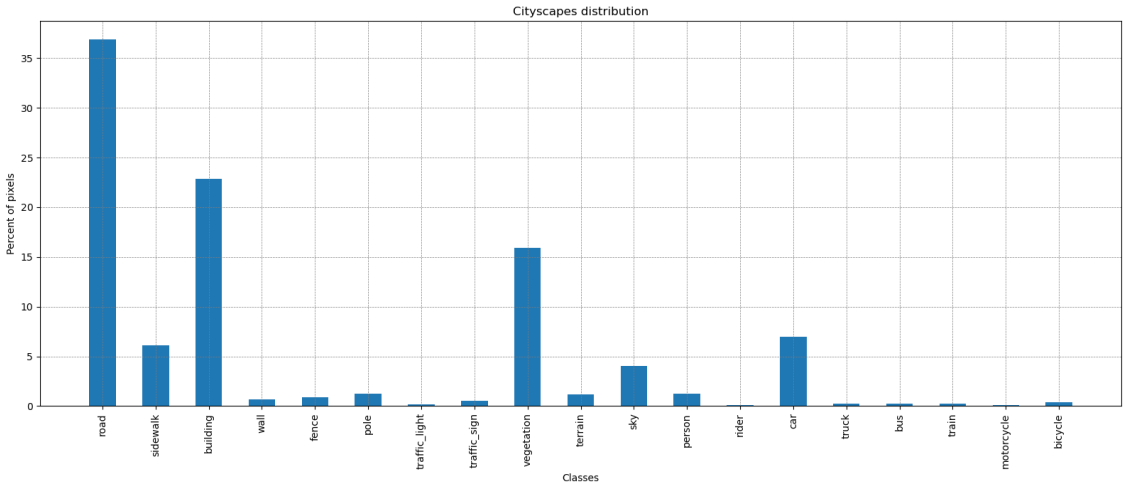


Figure 3.6: Distribution of labelled classes in the Cityscapes dataset.

3.3 Data Augmentations

In computer vision, data augmentations play an important role in the success of training a network. Data augmentations may help with keeping the data relevant by introducing noise. For example, by adding colour transformations, it can be avoided that the network only learns the specific colour histogram of the images. If the network only learns this information, it makes it vulnerable to sudden changes in the data, and it might not be relevant information to learn. As mentioned, in SSL, data augmentation may also be used to create a pretext task, as is done in this thesis with the erasing.

Many related works for SSL pretraining only include a few data augmentations such as crop, flip and colour transformations. Thus, it is interesting to investigate the effect of adding additional augmentations and to see if it could provide a more robust network. In this thesis, we use the following data augmentations for the pretraining:

Crop	Two random crops are taken from the input image and resized to 224 × 224.
Horizontal flip	The images are randomly flipped in horizontal direction.
Colour distortion	Changes the colours in the images by adjusting brightness, contrast, hue and saturation. Applied with a probability of 80%.
Gaussian blur	Blurs the image with Gaussian blur.

Affine transformations	Applies random affine transformations, including rotation, translation and shear. Affine transformation is a geometric transformation that preserves parallelism. It could be advantageous in order for the network to become consistent for geometrical transformations.

Following PixPro [2] and BYOL [6], the first four data augmentations listed are applied to both of the two views passing through the network to obtain two randomly transformed views of the same image. In addition to this, the affine transformations are added to one of the two views to get even stronger data augmentations than what previous methods have tested. The purpose of this is to teach the network to output consistent feature maps despite geometrical transformations. By applying affine transformations in one of the two views, the network can learn the relationship between an affine transformed view and a view without affine transformation. To this end, after passing through the network, the outputted feature map for the affine transformed view is inverse transformed. This way, consistency can be sought between the feature maps outputted from the two branches in the network.

4
Results

This chapter presents the results of our proposed model and the experiments that were conducted to evaluate each studied mechanism. First, all details regarding the setup for the implementation of the networks and the training algorithms are given, followed by a description of the evaluation metric mIoU that was used. To motivate the choice of mechanisms for our proposed model, the segmentation results from different experiments, where each mechanism was individually applied to the pretraining network, are presented. This is followed by a presentation of the final results, which were obtained by training our model with the mechanisms that showed the best individual results. All results are from transferring the pretrained parameters to perform semantic segmentation.

4.1 Implementation Setup

4.1.1 Practical Setup

The code for this thesis was written in Python together with Pytorch [40] as the machine learning framework used for building and training the networks. To our disposal, we had two clusters of three NVIDIA GeForce GTX 1080 Ti GPUs and four NVIDIA GeForce RTX 2080 Ti GPUs respectively. Each GPU has 11 GB of memory, and all pretraining and some segmentation training have been performed on these. Occasionally, we also had access to an additional cluster of four NVIDIA GeForce RTX 2080 Ti GPUs. These were only used for training the FCN-16 for semantic segmentation.

4.1.2 Setup for Pretraining

Since the search space is vast and tuning every parameter would have taken too much time, most settings for the pretraining were kept the same as the setup for PixPro [2]. ResNet-50 [27] was used as the backbone of the architecture, which is also most consistently used in previous research. The LARS optimizer was used together with a cosine learning rate scheduler. A base learning rate of 1.0 was set and the LR was linearly scaled with the batch size as $\text{LR} = lr_{base} \times \frac{\#bs}{256}$. The momentum was set to 0.9 and the weight decay to 0.0004. All experimental runs were performed over 30 epochs and with a batch size of 64, except for two experiments where a batch size of 48 was used. The results are all based on one pretraining run since these were time-consuming to perform. For the training of our final proposed method, four

GPUs were used and a batch size of 64. The choice of batch size was limited by the provided resources. ImageNet-1K [39] was used for all pretrainings.

4.1.3 Setup for Semantic Segmentation

The settings for the semantic segmentation network followed in large part the settings of MoCo [3, 4], since these were also used in the article of PixPro [2] to evaluate transferability to semantic segmentation. A FCN-16 architecture was used which had a ResNet-50 backbone architecture, the same was used for evaluating segmentation in [2, 3, 4, 6]. The 3×3 convolution in the fifth convolutional block in the ResNet backbone was modified to have dilation of 2 and stride of 1, which was followed by two additional 3×3 convolutions of 256 channels and by BN and ReLU. To get the pixel classifications, a 1×1 convolution was then used. For the two extra 3×3 convolutions, we set a dilation of 6, in line with the field-of-view given in [13]. Random scaling [0.5, 2.0], crop of size 768 and random horizontal flipping were used for data augmentations. As for the optimizer, we used a standard SGD with an initial LR of 0.01. The momentum was set to 0.9 and the weight decay to 0.0001. The Cityscapes dataset [1] was used for segmentation, see Section 3.2.2. The training ran on one GPU for 100 epochs with a batch size of 7. It took approximately 20 hours for one training to finish. All of the segmentation results are taken as an average of three independent runs, except the class-wise results which are taken from individual runs.

4.2 Performance Evaluation Metric - Mean Intersection Over Union

Intersection over Union (IoU) is an evaluation metric often used to measure the quality of semantic segmentation models. It calculates the percentage of overlap between the predicted output and the ground truth. The IoU is given by

$$IoU = \frac{target \cap prediction}{target \cup prediction}. \tag{4.1}$$

The numerator is comprised of the pixels present both in the prediction and the ground truth, thereby making them correctly classified pixels (true positives). The denominator is compromised of all pixels present in either the predicted set or the ground truth. This includes the true positives, the pixels that have been wrongly classified (false positives), and the pixels that have been wrongly not classified (false negatives). This calculation yields a value between zero and one and can be thought of as a percentage of how well the model is performing the segmentation task.

To get a global score, this is calculated for each class separately and then averaged, which returns the mean IoU (mIoU). If the network fails at classifying a class completely, this will affect the mIoU noticeably since it is an average over all classes. Thus, the network needs to correctly classify small and large objects alike to achieve a good mIoU score.

4.3 Results of the Experiments

The experiments were performed by making changes to the PixPro [2] architecture. That is, each mechanism was added and trained individually to study the impact each mechanism would have on the network. The results from these experiments are divided into two sections based upon the two areas that have been the focus for development. The first section presents the effects of applying various data augmentations and the second section presents the effects of modifying the prediction head, which is the PPM in PixPro. On average, each pretraining for the experiments took 75 h to complete. The experiments followed the implementation setup as previously detailed and ImageNet-1K was used for the pretraining while the Cityscapes dataset, trained on FCN-16, was used to perform the segmentation task. As a baseline for these experiments, the PixPro network is used.

4.3.1 Results of the Experiments: Data Augmentations

The baseline PixPro [2] applies crop, horizontal flips, colour distortion, and Gaussian blur as data augmentations for both views. Thus, these data augmentations were also included in all of our experiments and were applied to both cropped views of the network. The augmentations that were added to PixPro as experiments include affine transformations as well as two erasing strategies. Affine transformations were applied to one of the two cropped views that are sent to both the online and target encoder, while the erasing augmentations were only applied to the views before the online encoder to create the prediction task. The results of the data augmentation experiments are presented in Table 4.1.

Table 4.1: Segmentation results for a pretrained PixPro [2] where various data augmentations have been applied. The pretrained model has run for 30 epochs on 3 GPUs. The results are taken as an average of three separate segmentation runs and the standard deviation for these are included. Affine (fixed) means that fixed hyperparameters were used during the training, and Affine (rand. init) means randomly initialized hyperparameters. The notation Black indicates that the erased pixels were assigned as zero values while Gaussian means they were assigned as Gaussian noise across all channels. The notation ($p = 0.5$) indicates that the probability of the images undergoing erasing was 50%.

Settings	**mIoU**
Baseline (PixPro)	70.90 ±0.79
Rotation	71.71 ±0.58
Affine (fixed)	71.60 ±0.47
Affine (rand. init.)	71.80 ±1.19
Random Erasing Black	**72.56** ±0.61
Random Erasing Black ($p = 0.5$)	71.87 ±1.07
Random Erasing Gaussian	71.48 ±1.02
Random Patched Erasing Black	71.07 ± 1.00
Random Patched Erasing Gaussian	72.35 ±0.28

The first experiment was an implementation where only a rotation transformation, with a fixed angle of 30°, was added to the network. This was implemented to investigate if the network could handle additional augmentations than previously implemented. Rotation achieved an average of 71.71 mIoU when transferred for semantic segmentation. In the second experiment, translation and shear transformations were added to the rotation, creating a set of affine transformations. For this experiment, fixed hyperparameters were used. The rotation angle was set to 30°. The translation factor, which defines the number of pixels to translate in the respective direction, was set to 10 in x- and y-direction. The shear factor, which represents the shear angle for the respective axis, was set to 10° in both x- and y-direction. This training showed some odd behaviour of the images when shear was applied in both directions combined with rotation. Thus, only shear in x-direction was applied in the experiment with random initialization of the hyperparameters. Random initialization indicates that, for each batch, the transformation hyperparameters were randomly chosen within a boundary. The rotation angle had a boundary between ±45°, the translation factor between ±100 in both directions and the shear factor between ±10° in the x-direction. As shown in Table 4.1, all of these experiments achieved an improvement compared to the baseline.

The Random Erasing augmentation was applied with the hyperparameters for the area ratio range of erasing as $s_l = 0.02$ and $s_h = 0.4$, and for the aspect ratio range of erasing as $r_1 = 0.3$ and $r_2 = \frac{1}{0.3}$. These parameters were chosen as they were shown to provide the most reliable results in the original article that introduced Random Erasing [10]. The probability p for erasing was set to $p = 1$ to have all images going into the online encoder undergo erasing, except in one experiment where $p = 0.5$. The erased pixels were either assigned as zeros, denoted as Random Erasing Black in the Table 4.1, or with Gaussian noise across all channels, denoted as Random Erasing Gaussian. The average result of pretraining PixPro with added Random Erasing Black ($p = 1$) achieved a mIoU of 72.56, which is the best result for all experiments and is an improvement of 1.66 mIoU compared to baseline.

For Random Patched Erasing, the hyperparameters were set as a number of $n = 50$ patches with side length $s = 10$. This meant that about 10% of the image was erased. Two experiments were performed in the same way as for the Random Erasing, where the erased pixels were either assigned as zeros or as Gaussian noise across all channels. Random Patched Erasing Black only achieved slightly better results than the baseline, while Random Patched Erasing Gaussian improved the results more noticeably by achieving an increase of 1.45 mIoU compared to baseline.

The idea behind testing Random Patched Erasing was to investigate if smaller and spread out erased regions could help the network to learn the features of tiny or thin objects more accurately. Compared to using Random Erasing, there is less risk of the smaller objects being erased altogether. To verify if this idea held, the class-wise mIoU for the 19 classes present in Cityscapes can be inspected. Figure 4.1 shows the class-wise mIoU for Random Erasing Gaussian and Random Patched Erasing Gaussian for a segmentation run where the total mIoU for each is very close to the

other. It shows that for smaller and thinner objects such as fence, pole and traffic light, Random Patched Erasing Gaussian achieves slightly better results.

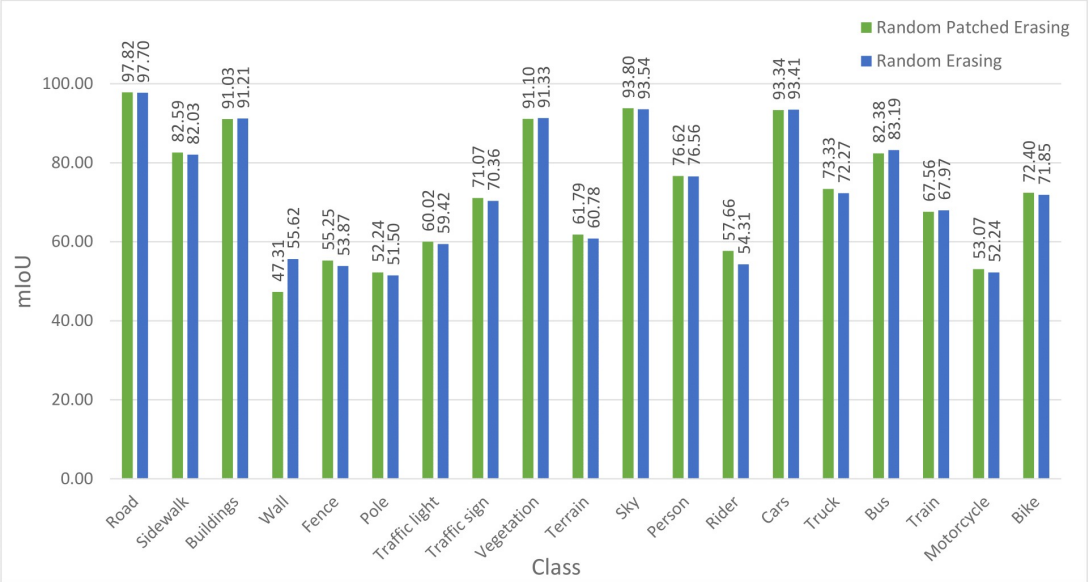


Figure 4.1: The class-wise mIoU for Random Erasing with Gaussian filling (blue) and Random Patched Erasing with Gaussian filling (green). These numbers were extracted from individual segmentations with a total mIoU of 72.56 and 72.65 respectively.

4.3.2 Results of the Experiments: Prediction Head

Two implementations with LocalPPM were made, one with a local frame of 7×7, and one with a local frame of 5×5. As mentioned in Section 3.1.1, the implementation of LocalPPM included adding dilation to the ResNet-50 backbone. This modification produced a 14×14 feature map.

When experimenting with a local frame of 7×7, the batch size was decreased to 48, compared to 64 as was used in the other experiments, due to memory limitations. Because of this, Momentum BN [15] was implemented in the target encoder in order to maintain the performance when a smaller batch size was used. Momentum BN was then kept for the two experiments with a frame size of 5×5, which used a batch size of 64 and 48, respectively. To evaluate the effects of applying the Momentum BN, a separate experiment was made where only the Momentum BN was implemented. Both in this implementation and that of LocalPPM, Momentum BN was only added to the target encoder. The semantic segmentation results for the experiments are presented in Table 4.2 together with the result for the self-attention implementation.

Table 4.2: Segmentation results for a pretrained PixPro [2] where the prediction head has been changed. The pretrained model has run for 30 epochs on 3 GPUs. The results are an average of three separate segmentation runs, and includes the standard deviation. The notation LocalPPM n×n + Momentum BN means that a local frame size of n×n was used together with Momentum BN. The notation (48) means that the batch size for that experiment was 48 instead of 64. The fifth row presents the results of implementing self-attention to the network. The last row shows the results with only Momentum BN implemented.

Settings	**mIoU**
Baseline (PixPro)	70.90 ±0.79
LocalPPM 7×7 + Momentum BN (48)	70.93 ±1.94
LocalPPM 5×5 + Momentum BN	**71.46** ±0.91
LocalPPM 5×5 + Momentum BN (48)	69.60 ±1.34
Self-attention	69.69 ±1.54
Momentum BN	71.33 ±1.27

The first experiments for the prediction head were with LocalPPM. Initially, the frame size of the LocalPPM was set to 7×7 since this was the size of the original feature map outputted from the encoders without added dilation. The semantic segmentation performance of this implementation reached a mIoU of 70.93. It can be noted that this was achieved with a smaller batch size of 48 for the pretraining compared to a batch size of 64 which was used for training the baseline.

To evaluate LocalPPM further and with the same batch size as the baseline and the other experiments, we decreased the frame size from 7×7 to 5×5. This showed an improvement of almost 0.6 mIoU compared to the baseline. Further on, LocalPPM with a frame size of 5×5 was also trained with a batch size of 48 to evaluate how the frame size affects the learning.

A feature map of 14×14, as was obtained by adding dilation to the backbone network, is still rather small compared to other articles that also implement local attention for similar purposes. An example of this is [41], which uses a feature map of 128×128. An experiment with more dilation added to the backbone was made, following the implementation in [26], which gave a feature map of 28×28. However, even with LocalPPM, this was too computational heavy for our computer resources to handle without running out of memory. The largest feature map used for this thesis was therefore of size 14×14.

A larger feature map means that less information is lost through downsampling. Intuitively, this should thereby make it easier for the network to recognise the features of small or thin objects. To investigate if this is the case, the class-wise mIoU for one of the LocalPPM experiments compared to the baseline PixPro can be observed. In Figure 4.2, LocalPPM with a frame size of 7×7 is shown together with PixPro. For these two implementations, the feature maps outputted from the encoders were 14×14 and 7×7 respectively. Interestingly, no clear advantage for the

LocalPPM in terms of recognising thin and small objects, such as traffic light and pole, can be seen.

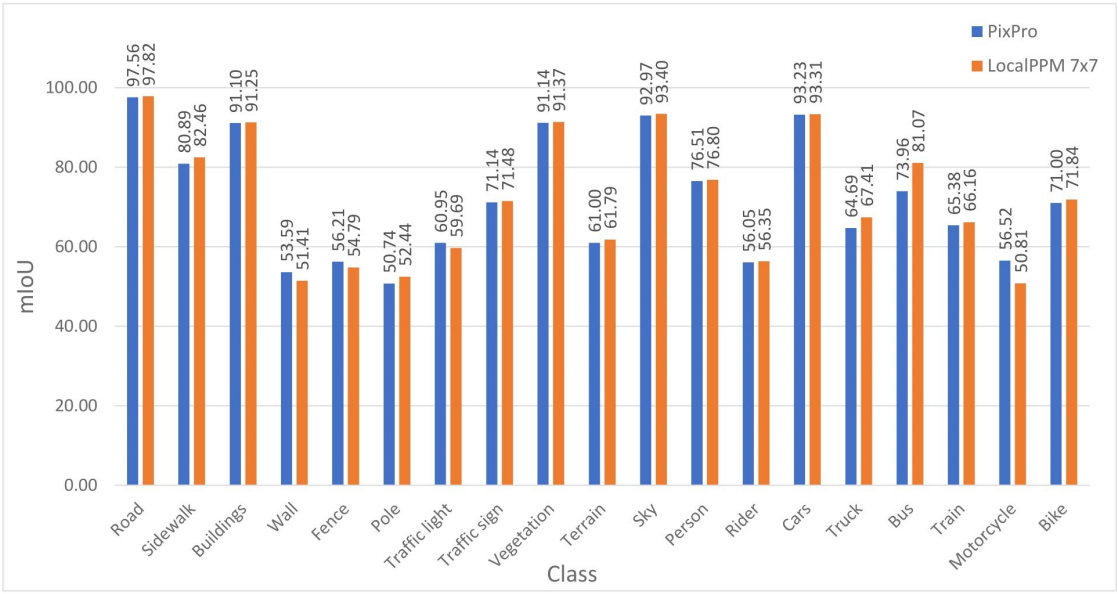


Figure 4.2: Class-wise mIoU for the baseline PixPro (blue) compared to LocalPPM (red) with 7×7 frame size and Momentum BN applied. These numbers were extracted from individual segmentations with a total mIoU of 71.85 and 72.19 respectively.

The final implementation tested was to replace the LocalPPM module with a simple self-attention module as described in Section 2.5.1. However, as shown in Table 4.2, applying a self-attention module decreased the mIoU by about 1%. The class-wise mIoU for self-attention compared to the baseline is shown in Figure 4.3, where the self-attention implementation achieved comparable performances as the baseline for some of the classes, for example, road, vegetation, and cars.

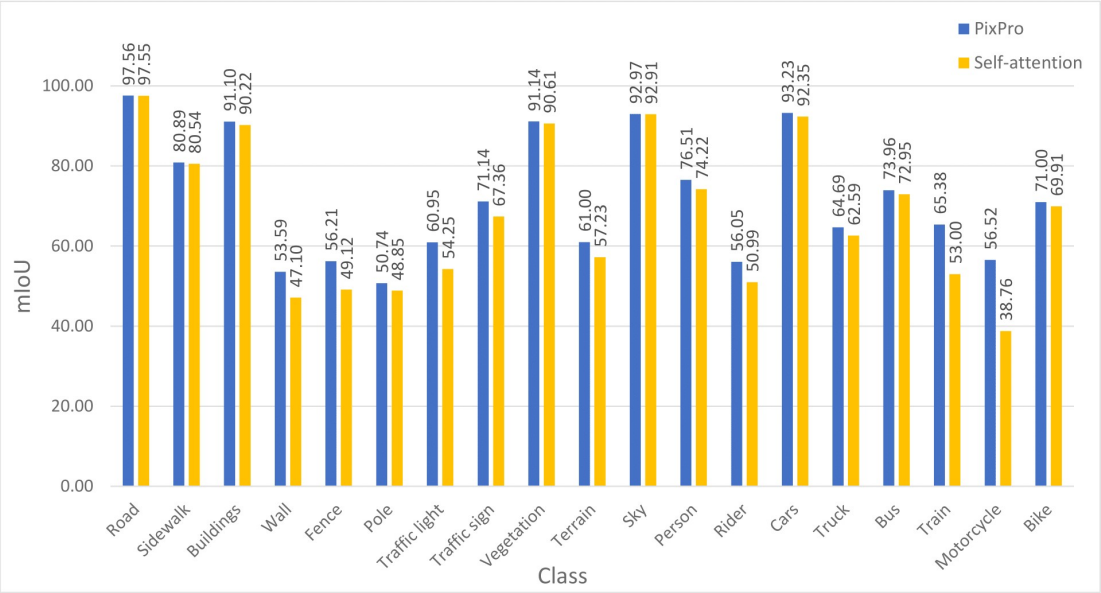


Figure 4.3: Class-wise mIoU for the baseline PixPro (blue) compared to self-attention (yellow). These numbers were extracted from individual segmentations with a total mIoU of 71.85 and 67.92 respectively.

4.4 Final Result

Table 4.3 presents the results of our final method, which was pretrained on ImageNet-1K and then transferred to a FCN-16 network that was trained for semantic segmentation on the dataset Cityscapes. The final method includes affine transformation with randomly initialized hyperparameters, Random Erasing Black, Momentum BN in the target encoder, and a LocalPPM with a frame size of 5×5. It was pretrained for 100 epochs on 4 GPUs and with a batch size of 64. The pretraining took about 12 days to complete.

The table also shows the segmentation results from a completely supervised pretraining, using pretrained weights from Pytorch [40], as well as two other related SOTA pretraining methods, namely MoCo [3] and PixPro [2]. These give a baseline to compare our results against. As a sanity check, a segmentation training with randomly initiated weights, in other words no pretraining, are also included in the table.

The baseline models have been pretrained on ImageNet-1K for 100 epochs, except for MoCo, which was trained for 200 epochs. The pretrained weights for MoCo and PixPro were retrieved online and loaded into the same segmentation network as used for all results. This was done to save time since both methods are rather computationally heavy to train. MoCo used a batch size of 256 trained over 8 GPUs, as described in [3], while PixPro was trained using a batch size of 1024 over 8 V100 GPUs, as described in [2]. For MoCo, only pretrained weights trained for 200 epochs were available.

An example of the visual semantic segmentation results can be found in Figure 4.4. In this figure, the segmentation map for our method is shown together with the corresponding ground truth label and the original image.

Table 4.3: Semantic segmentation on different pretraining models with FCN-16 as the semantic segmentation network. The results are taken as an average of three segmentation runs and the standard deviation for this is included. The first row presents the result for random initiated weights. The second row shows the results for a supervised pretraining followed by results for MoCo [3, 4] and PixPro [2]. Finally, our proposed method is shown in the last row.

Methods	Epochs	Cityscapes mIoU
Random init.	-	55.78 ±2.41
Supervised	100	75.08 ±0.94
MoCo (retrieved online)	200	74.47 ±0.24
PixPro (retrieved online)	100	75.91 ±0.18
Our method	100	74.15 ±0.45

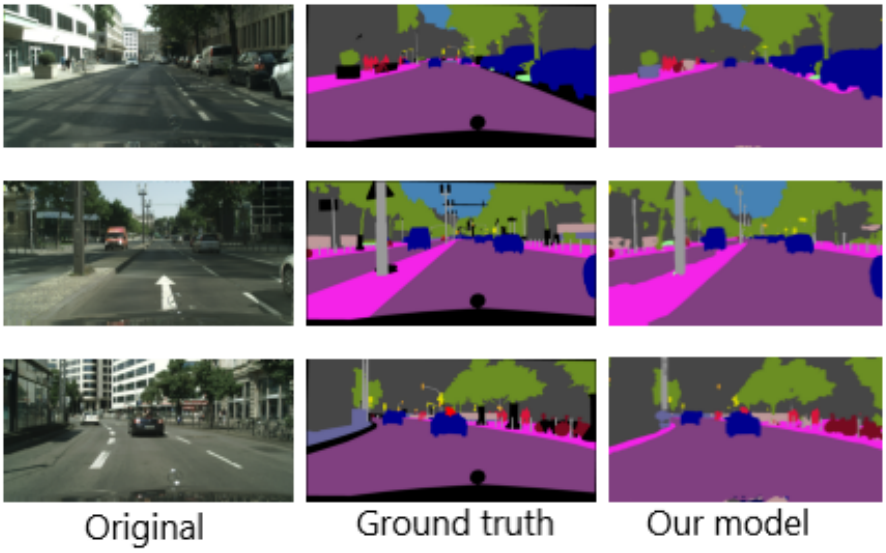


Figure 4.4: An illustration of the semantic segmentation map for our proposed pretraining method. The original image can be found in the first column, followed by the ground truth labelled image in the second column, and our proposed method (74.15 mIoU) in the last column. It can be noted that the ground truth includes some of the classes that are usually omitted when training a network for segmentation on the Cityscapes dataset since these do not contain any interesting information. An example of this is the ego vehicle visible in black at the bottom of all the ground truth images. The original images, as well as their ground truth labels, are taken from [1].

5
Discussion

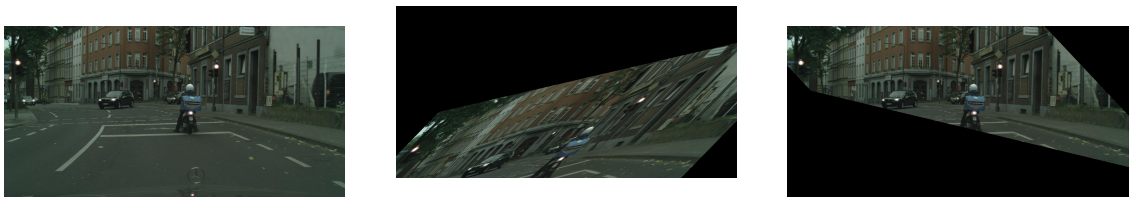
In this chapter, we will discuss the implemented methods and the results presented in the previous chapter. First, we will discuss the results relating to the data augmentations, and secondly, we will discuss the results from experimenting with the prediction head. After, the results from our final model, where the mechanisms that achieved the best individual results are combined, are discussed. At the end of the chapter, we give suggestions for possible future developments relating to our work.

5.1 Discussion of Results

5.1.1 Discussion of Data Augmentations

All tested data augmentations achieved an improved result compared to the baseline network PixPro [2]. These results are not completely unexpected since there already exists several examples that show the importance of data augmentations for good representation learning [5, 29, 42, 43]. It would have been interesting to experiment with more and even stronger data augmentations to investigate if the results could be improved even further. However, due to time limitations, we will leave this for future work.

As shown in Table 4.1, applying affine transformations with fixed parameters to the pretraining network improves the segmentation by approximately +1 mIoU. However, compared to only rotating the images, the affine transformations with fixed parameters has a slightly lower mIoU. A possible reason for this is the choice of hyperparameters, specifically in relation to shear combined with rotation. When applying rotation combined with shear in both the x- and y-axis, the transformation gives an unpredicted behaviour. Figure 5.1 clearly shows that when using the same parameter setting as in the affine transformation experiment with fixed hyperparameters, a shear angle of 10° for the respective axis in combination with rotation, roughly half of the image is discarded. In addition, in our network, the image is converted and downsampled to a 7×7 feature map, which in this case, could lead to a loss of important feature information and less optimal pretrained weights. Thus, experiments with shear angles between [1°, 50°] were performed to analyse the view of the transformed image. The image became a thin line and sometimes disappeared from the image frame completely, which means that Figure 5.1b became all black. Thus, for these reasons, we only applied shear in one direction when training with affine transformations with random parameters.



(a) Original image. Before applying the affine transformation.

(b) Image after applying affine transformations.

(c) Inverse affine transformation of (b)

Figure 5.1: An example of applying affine transformation with fixed hyperparameters, rotation angle: 20°, translation: (10,10), shear: (10°,10°). The image is taken from Cityscapes [1].

The results from affine transformation with random hyperparameters showed a slight improvement compared to having fixed parameters, achieving 71.80 mIoU. One reason for this could be that applying the shear transformation in both directions leads to damaged images. This was avoided in the experiment with affine transformation with random parameters since it was implemented with shear in only one of the two directions. Another possible reason is that randomness provides a more robust network and forces the network to become more invariant to different geometric relations instead of just adapting to the relationship of a fixed transformation.

Still, affine with random parameters only achieved a slightly higher mIoU compared to the rotation transformation. This might indicate that combining the affine transformations are not the optimal choice and leads to too strong data augmentation. It is also possible that the chosen boundary of the random parameters might have been too large, which leads to a loss in spatial information since pixel features might be lost after the inverse transform, as in Figure 5.1b. Because of this, the choice of hyperparameters, or what transformations are combined, could be crucial in achieving optimal performance with affine transformation. However, since the results between the three affine transformations that were used in experiments are close, it is hard to draw any definite conclusion.

The various Random Erasing and Random Patched Erasing experiments showed noteworthy results, see Table 4.1. This can be attributed to the fact that the network is forced to attend to the full context of the image and learn to generalize better when parts of the views going into the online encoder are missing. By still having views without missing parts going into the target encoder, the network can use this information to predict the missing parts in the output from the online encoder.

As can be seen, for the Random Erasing, the version with black filling gave the best results, while for the Random Patched Erasing, Gaussian filling worked the best. The fact that the Random Erasing with black filling gave the best overall result was a bit unexpected since in the original article introducing Random Erasing [10], it was shown that they achieved the best results by assigning each erased pixel with a random value between [0, 255], or with the mean ImageNet pixel value. Intuitively, a reason for this could be that, with the Gaussian filling, too much random noise was added to the network. However, since the Random Patched Erasing with Gaussian noise showed better results than the Random Patched Erasing with black, the problem rather seems to be that the network is sensitive to having too much Gaussian noise in one continuous region. Looking at the example images in Figure 5.2, where colour distortion and Gaussian blur are also added, as they were in all experiments, it can be seen that the Random Erasing Gaussian adds quite a bit of noise to the images. This may disturb the feature recognition.

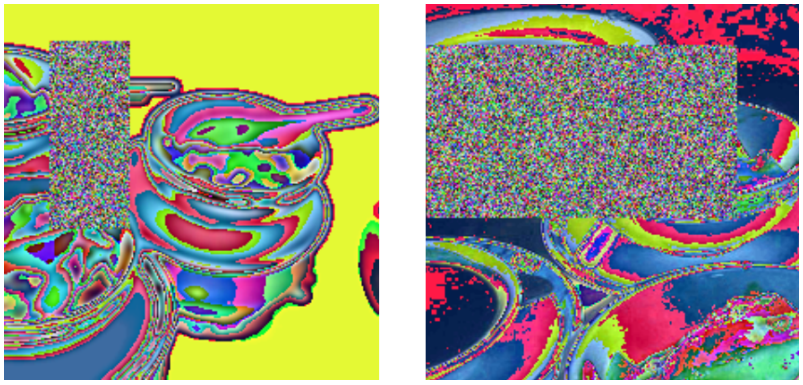


Figure 5.2: Two images where Random Erasing Gaussian has been applied together with colour distortion and Gaussian blur.

In the original article for Random Erasing [10], experiments had shown that the best results were achieved by having a probability $p = 0.5$ for erasing, which is why this was also tested by us in one experiment. Interestingly, this achieved worse segmentation results compared to $p = 1$. This could further indicate the value of erasing part of the view for this type of network.

5.1.2 Discussion of Prediction Head

Replacing the original PPM in PixPro [2] with LocalPPM showed no apparent improvement when a frame size of 7×7 was used. The result for this experiment was however achieved even though a smaller batch size had been used for the pretraining, with a batch size of 48 compared to a batch size of 64. This is interesting since the batch size seems to play a big part in the performance of these types of pretraining networks [5, 6].

Although, this could be attributed to the Momentum BN that was also added together with the LocalPPM. The Momentum BN is used to obtain stable statistics

without large batch sizes, so it is not unexpected if it contributes to the performance. Looking at the results for when the network was pretrained with only Momentum BN, a small change from the baseline can be observed. This strengthens the reasoning that the Momentum BN might have contributed to the LocalPPM 7×7 reaching the same results as the baseline. To truly be able to verify if this is the case, experiments with even larger feature maps should be conducted, but as stated, this was not possible for this thesis due to limited computer resources.

The results for a frame size of 5×5 with a batch size of 64 achieved a small improvement from the baseline, but again, this could be attributed to the Momentum BN. Looking at the results between only using the Momentum BN and using the LocalPPM with a frame size of 5×5 together with Momentum BN, the difference is small.

Neither the class-wise results for LocalPPM compared to PixPro, as seen in Figure 4.2, showed any significant differences. As mentioned, this was not expected since the expectation was that the LocalPPM might help with recognising small and thin objects since a larger feature map could be used. Again, it is possible that the feature map still remains too small for better capturing the feature information relating to these objects, or it could be that the network does not benefit from propagating locally. When propagating locally, not as many pixel features are used in the similarity calculation as in the original, full-size PPM. This means that some dependencies between features that do not lie within the same local region might get lost. If looking at the results for a frame size of 5×5, when a batch size of 48 was used, the result is lower than for a frame size of 7×7 with the same batch size. This indicates that propagating locally might not be optimal, or at least that a larger frame size should be chosen if possible. As in our case, with limited computer resources, it seems more important to prioritise having a larger batch size than having a larger frame size.

As can be seen in Table 4.2, replacing the PPM in PixPro [2] with a self-attention module did not improve the results compared to the baseline. Several implementations were tested when investigating the self-attention module, ranging from making small modifications of the PPM to straight out replacing it. The modifications are listed below in the order they were tested.

- Learnable weights were implemented in the similarity function in the PPM, see eq. (2.11). This was done by applying two 1×1 convolutional layers to the pixel feature vector to form Q and K respectively. This implementation made the network collapse. This was not entirely unexpected since no softmax function was added to map the similarity function to values between [0,1], which essentially means the dot product between Q and K could be a value between negative and positive infinity.

- Building upon the previous implementation, the transformation function g(.) was also replaced with a 1×1 convolution and a softmax function was added

together with a scaling factor. This gave a standard self-attention mechanism, as described in Section 2.5.1. Again, the network collapsed with this implementation.

- Motivated by [18], self-attention performs better with larger feature maps. Thus, we expanded the feature map by adding dilation to the backbone as described in Section 3.1.1. Neither this implementation worked and also collapsed.

- Instead of the dilation, the self-attention was applied as a correlation factor to the input feature map, $y = x + \gamma o$, described in Section 3.1.2. This time the network did not collapse, however, the network did not learn well with this.

- The self-attention mechanism was elaborated with a positional-encoder, as presented in [31], due to the spatial information being necessary, especially when working with images. This was initially trained without the correlation implementation, but then the network collapsed. Thus, the correlation was added and the network became stable. However, the segmentation performance did not transcend the baseline, the results shown in Table 4.2.

As can be seen, the network is rather sensitive and collapsed in many of the listed implementations, leading to the loss either not improving or diverging to infinity. These types of networks that operate with only positive image pairs seem to, in general, be rather sensitive to changes as shown in the ablation study for PixPro, where removal of the PPM altogether also resulted in a collapse of the network [2]. Often, the reason behind the collapse is some sort of information leakage between the online encoder and the target encoder.

In terms of the self-attention layer, it is possible that when back-propagating through the online branch of the network, the weights in the self-attention layer are updated to become constant or that they are updated very slowly, leading to bad representations. The correlation term might help with this since it decides how much of the output from the self-attention layer should be added to the feature map. By learning how much attention should be added to the feature map, it is possible that collapsed solutions are avoided. It is also possible that the self-attention module could benefit from training for more epochs since this would give more time to update the weights and the learnable parameter, γ, to find better similarities. Related to this, the LR might be important as shown in [30, 44] where similar networks are used and trained with a smaller learning rate, which can be important for the robustness and stability of the network.

The PE implementation seems to be important for the network to stay stable. This follows the argumentation in [34], which says PE is essential for self-attention since the self-attention itself does not provide the absolute position of each pixel, making it difficult to sequentially track the pixel positions. However, in related articles [19, 34, 45], no PE was implemented and yet, the network did not collapse. A possible reason for this could be that our network seeks consistency between the

representations for positive pixels. The positive pixels are defined by an assignment rule, see eq. (2.12), which is a distance relationship between the pixels in two views. To keep track of the distance between the pixels, PE might therefore be needed.

5.1.3 Discussion of Final Result

The final model, which includes affine transformation, Random Erasing Black, Momentum BN, and LocalPPM, achieved a mIoU of 74.15 when pretrained for 100 epochs and with a batch size of 64. The mechanisms were chosen for the final model since they achieved the best individual mIoU in the experiments, and the improvements seem to transfer when they are combined in the final model. It could still be that some combinations of the mechanisms used in the final model are not optimal. In particular, the affine transformation in combination with the Random Erasing could potentially be problematic since they both distort the feature maps.

To investigate if this could be the case, an additional experiment was performed after the training of the final model, where only affine transformation with fixed hyperparameters and Random Erasing Black was included. As with all other experiments, this was trained for 30 epochs on 3 GPUs and a batch size of 64. The average result from this experiment was 71.46 mIoU, which is lower than both individual results for fixed affine transformation and Random Erasing Black. This gives reason to consider that affine transformation in combination with erasing might not be optimal. Based upon the results from the previous experiments, it seems most advantageous to keep the Random Erasing. Due to time limitations, we did not have the opportunity to explore this further.

According to previous work [6], the batch size has a big impact on the performance of the network and a small batch size could affect the learning negatively. Due to limitations of computer resources, our network was trained with a batch size of 64, which was the largest batch size that could fit within the available memory. However, our model still achieved results that are only slightly below the baseline, even if it was trained with a batch size of 64 compared to the baseline which was trained with a batch size of 1024. This is a big difference and it shows the potential of our model since a smaller batch size is advantageous due to it being easier to train. In addition, better results could possibly be obtained for our model by fine-tuning it further.

5.2 Future Work

While working on this thesis and exploring relevant literature, countless different directions would have been interesting to investigate further. However, due to limitations of time and resources, we did not have the opportunity to explore all of these. Here we will list a few ideas for possible future work.

5.2.1 Experiment With More Data Augmentations

In this thesis, two of the more commonly used data augmentations, affine transformations and erasing, were analysed. Both of these showed promising results for representation learning. Therefore, it would be interesting to further investigate additional data augmentations. One possible direction is to dig deeper with the erasing transformation since this has shown to give the greatest improvement in the semantic segmentation results. For example, it would be interesting to further analyse the parameters for the Random Patched Erasing since this showed significant results, outdoing the Random Erasing in terms of recognising small and thin objects. Investigating if progressively adding more erased patches, [38], can increase the representation learning would also be interesting since this has been beneficial for the related technique dropout [46].

Another example would be to use CutMix [43] instead of erasing. CutMix is a similar augmentation to Random Erasing, but instead of erasing a region, part of another image is pasted into the region. This creates a mix between the two images. This could lead to superior results compared to erasing since no pixel information is lost but only replaced.

In addition, since fine-tuning of parameter settings were not the focus of development due to time limitations, optimal parameter settings for the data augmentations may achieve even better results. To investigate this, an ablation study might be in order where different parameter settings are tested.

5.2.2 Larger Feature Maps

As discussed, one of the reasons for applying LocalPPM is to achieve a larger feature map that contains more information. It would be interesting to investigate if a larger feature map than used in this thesis could improve the results further. This could, for example, be achieved by adding more dilation to the backbone network or possibly by adding a decoder to upsample the feature maps.

5.2.3 Developing the Self-Attention Module

One of the areas investigated in this thesis was to replace the prediction head with a self-attention module. Due to the timeframe, the implemented self-attention module is relatively simple. It did also not achieve better results than the baseline. However, it is noteworthy that even a simple self-attention module still performed almost as well as the baseline.

In Figure 4.2, the class-wise mIoU for the self-attention module and the baseline can be seen. This shows that self-attention predicts the dominant classes, for example, wall, sky, and cars equally as well as the baseline. It only performs a little weaker when segmenting the smaller classes or the objects with more complex geometries such as motorcycle and traffic light. Thus, further investigations could be made into local attention, which focuses on local dependencies. Examples of this

are LANnet [47] and the blockwise self-attention mechanism presented in [41]. In addition, the multi-head attention in [31] is also supposed to provide information dependency in subspaces. Self-attention could also be expanded by stacking several attention mechanisms or by implementing self-attention in different stages of the network.

6
Conclusion

The aim of this thesis was to investigate how unlabelled data can be used to pretrain a network to learn pixel-level feature representations suitable for semantic segmentation. This aim was met by the development of a SSL pretraining method that is trained on unlabelled images and reaches close to SOTA results when fine-tuned for semantic segmentation. When pretraining for 100 epochs, it achieves an average of 74.15 mIoU, which is a difference of -0.32 mIoU and -1.76 mIoU respectively, compared to two related baseline methods. These results were reached even if a smaller batch size of 64 was used for the pretraining, compared to the baseline methods which were trained on batch sizes of 256 and 1024 respectively.

In particular, a module called LocalPPM was introduced which finds similarities between features by propagation locally. This module makes it possible to use a larger feature map than what is used in related work, since it reduces the complexity that follows with a larger feature map. In addition, stronger data augmentations are also used in the developed method by implementing affine transformation and Random Erasing, which we show improves the representation learning of the network.

Bibliography

[1] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016 December, pp. 3213–3223, 2016, doi: 10.1109/CVPR.2016.350.

[2] Z. Xie, Y. Lin, Z. Zhang, Y. Cao, S. Lin, and H. Hu, "Propagate Yourself: Exploring Pixel-Level Consistency for Unsupervised Visual Representation Learning," *arXiv preprint arXiv:2011.10043*, 2020.

[3] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum Contrast for Unsupervised Visual Representation Learning," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 9726–9735, 2020, doi: 10.1109/CVPR42600.2020.00975.

[4] X. Chen, H. Fan, R. Girshick, and K. He, "Improved Baselines with Momentum Contrastive Learning," *arXiv preprint arXiv:2003.04297*, 2020.

[5] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 1597–1607. [Online]. Available: http://proceedings.mlr.press/v119/chen20j.html

[6] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, B. Piot, k. kavukcuoglu, R. Munos, and M. Valko, "Bootstrap your own latent - a new approach to self-supervised learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 21 271–21 284. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/f3ada80d5c4ee70142b17b8192b2958e-Paper.pdf

[7] E. Xie, J. Ding, W. Wang, X. Zhan, H. Xu, Z. Li, and P. Luo, "DetCo: Unsupervised Contrastive Learning for Object Detection," *arXiv preprint arXiv:2102.04803*, 2021.

[8] K. Chaitanya, E. Erdil, N. Karani, and E. Konukoglu, "Contrastive learning of

global and local features for medical image segmentation with limited annotations," *arXiv preprint arXiv:2006.10511*, 2020.

[9] Jia Deng, Wei Dong, R. 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*, pp. 248–255, 2009, doi: 10.1109/cvprw.2009.5206848.

[10] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random Erasing Data Augmentation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34(07), pp. 13 001–13 008, 2017, doi: 10.1609/aaai.v34i07.7000.

[11] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440, doi: 10.1109/CVPR.2015.7298965.

[12] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," vol. 9351, pp. 234–241, 2015, doi: 10.1007/978-3-319-24574-4_28.

[13] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2018, doi: 10.1109/TPAMI.2017.2699184.

[14] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. E. Hinton, "Big self-supervised models are strong semi-supervised learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 22 243–22 255. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/fcbc95ccdd551da181207c0c1400c655-Paper.pdf

[15] Z. Li, S. Liu, and J. Sun, "Momentumˆ2 Teacher: Momentum Teacher with Momentum Statistics for Self-Supervised Learning," *arXiv preprint arXiv:2101.07525*, 2021.

[16] T. Devries and G. W. Taylor, "Improved Regularization of Convolutional Neural Networks with Cutout," *arXiv preprint arXiv:1708.04552v2*, 2017.

[17] K. K. Singh and Y. J. Lee, "Hide-and-seek: Forcing a network to be meticulous for weakly-supervised object and action localization," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 3544–3553, doi: 10.1109/ICCV.2017.381.

[18] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, June 2019, pp. 7354–7363. [Online]. Available: http://proceedings.mlr.press/v97/zhang19d.html

[19] J. Fu, J. Liu, H. Tian, Y. Li, Y. Bao, Z. Fang, and H. Lu, "Dual attention network for scene segmentation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 3141–3149, doi: 10.1109/CVPR.2019.00326.

[20] L. Ye, M. Rochan, Z. Liu, and Y. Wang, "Cross-modal self-attention network for referring image segmentation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 10 494–10 503, doi: 10.1109/CVPR.2019.01075.

[21] A. Van Den Oord, Y. Li, and O. Vinyals, "Representation Learning with Contrastive Predictive Coding," *arXiv preprint arXiv:1807.03748*, 2019.

[22] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised Feature Learning via Non-parametric Instance Discrimination," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3733–3742, 2018, doi: 10.1109/CVPR.2018.00393.

[23] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio, "Learning deep representations by mutual information estimation and maximization," *arXiv preprint arXiv:1808.06670*, 2019.

[24] Y. You, I. Gitman, and B. Ginsburg, "Large Batch Training of Convolutional Networks," *arXiv preprint arXiv:1708.03888*, 2017.

[25] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 July 2015, pp. 448–456. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.html

[26] F. Yu, V. Koltun, and T. Funkhouser, "Dilated residual networks," *In Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition*, pp. 636–644, 2017, doi: 10.1109/CVPR.2017.75.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2015, doi: 10.1109/CVPR.2016.90.

[28] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised Representation Learning by Predicting Image Rotations," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://hal-enpc.archives-ouvertes.fr/hal-01832768

[29] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2536–2544, doi: 10.1109/CVPR.2016.278.

[30] X. Chen and K. He, "Exploring Simple Siamese Representation Learning," *arXiv preprint arXiv:2011.10566*, 2020.

[31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[32] S. Hofstätter, H. Zamani, B. Mitra, N. Craswell, and A. Hanbury, "Local self-attention over long text for efficient document retrieval," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 2021–2024, doi: 10.1145/3397271.3401224.

[33] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran, "Image transformer," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 July 2018, pp. 4055–4064. [Online]. Available: http://proceedings.mlr.press/v80/parmar18a.html

[34] J. Lee, S. Park, J. Baek, S. J. Oh, S. Kim, and H. Lee, "On recognizing texts of arbitrary shapes with 2d self-attention," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 2326–2335, doi: 10.1109/CVPRW50498.2020.00281.

[35] A. Sinha and J. Dolz, "Multi-scale self-guided attention for medical image segmentation," *IEEE Journal of Biomedical and Health Informatics*, vol. 25, no. 1, pp. 121–130, 2021, doi: 10.1109/JBHI.2020.2986926.

[36] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, 2019, pp. 4171–4186, doi: 10.18653/v1/N19-1423.

[37] T. H. Trinh, M.-t. Luong, Q. V. Le, and G. Brain, "Selfie: Self-supervised Pretraining for Image Embedding," *arXiv preprint arXiv:1906.02940v3*, 2019.

[38] W. Less, "Progressive Sprinkles: A new data augmentation for CNN's," 2019, Accessed: 2021-05-03. [Online]. Available: https://lessw.medium.com/progressive-sprinkles-a-new-data-augmentation-for-cnns-and-helps-achieve-new-98-nih-malaria-6056965f671a

[39] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015, doi: 10.1007/s11263-015-0816-y.

[40] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[41] J. Jue, S. Elguindi, U. Hyemin, S. Berry, and V. Harini, "Local block-wise self attention for normal organ segmentation," *arXiv preprint arXiv:1909.05054v1*, 2019.

[42] A. Zhao, G. Balakrishnan, F. Durand, J. V. Guttag, and A. V. Dalca, "Data augmentation using learned transformations for one-shot medical image segmentation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 8535–8545, doi: 10.1109/CVPR.2019.00874.

[43] S. Yun, D. Han, S. Chun, S. J. Oh, Y. Yoo, and J. Choe, "Cutmix: Regularization strategy to train strong classifiers with localizable features," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 6022–6031, doi: 10.1109/ICCV.2019.00612.

[44] M. Caron, H. Touvron, I. Misra, H. Jegou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging Properties in Self-Supervised Vision Transformers," *arXiv preprint arXiv:2104.14294v2*, 2021.

[45] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *arXiv preprint arXiv:2010.11929v1*, 2020.

[46] P. Morerio, J. Cavazza, R. Volpi, R. Vidal, and V. Murino, "Curriculum Dropout," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3564–3572, 2017, doi: 10.1109/ICCV.2017.383.

[47] L. Ding, H. Tang, and L. Bruzzone, "Improving Semantic Segmentation of Aerial Images Using Patch-based Attention," *arXiv preprint arXiv:1911.08877v1*, 2019.