6-8-2020

Self-Supervision Initialization for Semantic Segmentation Networks

Kenneth Alexopoulos
ksa6262@rit.edu

Self-Supervision Initialization for Semantic Segmentation Networks

Kenneth Alexopoulos

Self-Supervision Initialization for Semantic Segmentation Networks

Kenneth Alexopoulos
June 8th, 2020

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

RIT | Kate Gleason College of Engineering

Department of Computer Engineering

Self-Supervision Initialization for Semantic Segmentation Networks

Kenneth Alexopoulos

Committee Approval:

Dr. Raymond Ptucha *Advisor* Date
Associate Professor

Dr. Andreas Savakis Date
Professor

Dr. Guoyu Lu Date
Assistant Professor

Acknowledgments

I would like to give a big thank you to my advisor Dr. Raymond Ptucha for his guidance in academics and life. I am thankful for the support of my thesis committee members Dr. Andreas Savakis and Dr. Guoyu Lu. I am grateful for the members of the Machine Intelligence Lab who were all so welcoming and helpful. I would also like to thank my friends and family for their love and support during this challenging endeavor.

Abstract

Convolutional neural networks excel at extracting features from signals. These features are able to be utilized for many downstream tasks. These tasks include object recognition, object detection, depth estimation, pixel level semantic segmentation, and more. These tasks can be used for applications such as autonomous driving where images captured by a camera can be used to give a detailed understanding of the scene. While these models are impressive, they can fail to generalize to new environments. This forces the cumbersome process of collecting images from multifarious environments and annotating them by hand. Annotating thousands or millions of images is both expensive and time consuming. One can use transfer learning to transfer knowledge from a different dataset as an initial starting place for the weights of the same model training on the target dataset. This method requires that another dataset has already been annotated and salient information can be learned from the dataset that can aid the model on the second dataset. Another method that does not rely on human generated annotations is self-supervised learning in which annotations can be computer generated using tasks that force learning image representations such as predicting the rotation of an image. In this thesis, self-supervised methods are evaluated specifically to improve semantic segmentation as the primary downstream task. Data augmentation's affect during pre-training is observed in context of its effect on downstream performance. Knowledge from multiple self-supervised tasks are combined to create a starting point for training on a target dataset that outperforms either method individually.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Breakthroughs with deep learning have enabled practical uses in many fields such as recommendation engines, medical image analysis, language understanding, autonomous vehicles, and much more. In recent times people are with increasing frequency coming into contact with systems that use deep learning to solve difficult tasks. In the past decade the field of computer vision has been aided by convolutional neural networks (CNNs) that perform remarkably on difficult vision tasks such as object classification, object detection, and semantic segmentation. With the help of ever growing amounts of labeled and unlabeled data these networks can better generalize and model their input space. Work done by Long et al. [8] show CNNs can be adapted into fully convolutional networks (FCNs) by adding upsampling layers to perform pixel-wise predictions. Paszke et al. [9] showed how an efficient architecture that minimizes parameters and floating point operations can make inferences at real-time speeds while maintaining performance similar to models with many more parameters. Efficient architectures allow for semantic segmentation to be performed by resource constrained platforms or in real-time applications such as autonomous driving where the network can help give a more detailed understanding of the surrounding environment.

With access to large amounts of unlabeled data many have developed methods to train models to learn useful features without the need to annotate samples by hand. Works by Le et al. [10] used a nine layer multilayer perceptron (MLP) autoencoder trained on 10 million unlabeled images to recognize faces and objects without supervised data. Turchenko et al. [11] trained a FCN autoencoder to extract latent feature representations that were used to classify handwritten digits. [12] and [13] trained a FCN to fill in images with select regions removed. They then used the feature extractor to perform object recognition, detection, and semantic segmentation. [14] and [15] used pretext tasks that created their own labels to train a CNN. Dosovitskiy et al. [16] showed that data augmentation can be used to create pseudo-labels to train a CNN using contrastive learning to extract features. Doersch et al. [17] showed how multi-task learning can be used to train a network on multiple self-supervised tasks to perform better on downstream tasks than an individual self-supervised task. Others used multi-task learning to train using unsupervised, self-supervised, and supervised loss together [18]. Goyal et al. [19] looked at the amount of data used, model capacity and problem complexity when pre-training a network using different self-supervised tasks. They observed downstream performance improved when increasing any of these three areas and that performance further improved when increasing all three together. By using these self-supervised pretext tasks a model is able to receive supervisory signals from unlabeled data. The weights from these self-supervised tasks are then able to provide a better initialization point for a downstream task than random initialization.

1.2 Motivation

Supervised learning has made great achievements in recent time. Powerful models that are extremely deep and wide are able to perform targeted tasks very well, and sometimes even at super human performance levels. As model capacity grows larger

the need for labeled data to train these models grows as well. As datasets grow larger the cost and time to have humans label these datasets grows too. With the size of these large datasets, the noise of the ground truth annotations can make learning more difficult and allow ambiguity in the task. By leveraging unsupervised and self-supervised learning we can help mitigate and overcome the burden of creating labeled data. These methods can leverage almost unlimited data and possibly allow models to learn features from environments and objects that are not captured in supervised datasets. This could help them generalize to new information. Platforms such as autonomous driving would also be able to update their models with new scenarios without the need to create labels. This can help quicken the response time to problems and possibly prevent them from occurring.

While larger models that require large amounts of computational power, energy, and time generally outperform optimized efficient architectures, this limits their use in real world applications. Large models are great for tasks that do not need to be performed in real-time such as medical image analysis. Efficient architectures are needed for tasks that take place on consumer devices or embedded platforms. They are needed for real-time applications such as autonomous driving were reaction time is critical. Many methods and works are developed and tested with large models in mind and experiments are run using only these architectures. This is because these architectures perform better on benchmarks, but lessons learned do not always directly transfer to other architectures. That is why for this work we will focus on Efficient Residual Factorized Network (ERFNet) which is a state of the art efficient real-time network for semantic segmentation.

Semantic segmentation is the target downstream task for this thesis work. It offers a more detailed understanding of the scene than object recognition or object detection. Inferences can be made just as quickly as object detection, and do not require region proposals. Using camera intrinsics, infrared light, deep learning algorithms, or other

techniques depth can be predicted on a per pixel basis. Depth can then be combined with pixel-wise semantic labels to help autonomous agents in path planning and object avoidance. By increasing semantic segmentation performance models will be better at generalizing to new data, better at classifying objects, and better at predicting object boundaries.

1.3 Contributions

The principal contributions of this thesis research are outlined as:

- Examine a robust offering of self-supervised learning methods and benchmark their effects on downstream semantic segmentation.
- Provide a novel way of combining pre-training methods in encoder/decoder architectures.
- Compare the affect of data augmentation during self-supervised learning and its effect on downstream semantic segmentation.
- Benchmark the affect of using in-domain unlabeled data with out-of-domain labeled data.
- Find sparse sub-networks using self-supervised pre-trained weights as a basis for pruning weights before training the downstream task.

1.4 Document Structure

Chapter 2 discusses the background on supervised learning, unsupervised learning, self-supervised learning, fully convolutional neural networks, semantic segmentation, and sparse networks. Chapter 3 will discuss the deep learning architecture used for experimentation and the loss function used for training. It will also discuss the

different self-supervised methods that are explored. Chapter 4 will outline the training and benchmark datasets used, along with the implementation details. Chapter 5 will analyse the results of pre-training using different self-supervised methods and the ways used to initialize ERFNet. Chapter 6 will summarize the contributions made and outline directions for future work.

Chapter 2

Background

2.1 Supervised Learning

Supervised learning in the context of deep learning is the task of mapping an input to an output label using pairwise sets of training examples. This process takes place with a human in the loop when generating labels for a specific dataset. This type of learning is very effective in training convolutional neural networks. While supervised learning is a very accurate method for training models it is both costly and time consuming to have humans label each individual sample in a dataset which can range in size from hundreds to millions of samples. In machine learning supervised learning has advanced considerably. Recent successes on impressive tasks such as The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [20] have demonstrated the abilities of deep convolutional neural networks. These networks are able to learn what features to extract, and using those features classify images at super human performance levels.

2.2 Unsupervised Learning

Unsupervised learning is an algorithm used to train neural networks without labels. Unlike supervised learning each training sample does not have an associated label. Instead networks use loss functions that do not rely on labels such as clustering

[21], mean squared error [22], or KL-divergence [23]. Unsupervised learning looks for patterns in the data, grouping samples with similar features, or computes probability densities based on the input space.

2.3 Self-Supervised Learning

Between unsupervised learning and supervised learning exists self-supervised learning. Self-supervised learning does use training sample pairs, but unlike supervised learning the label is generated by the computer. By using these automatically generated labels deep learning models can learn supervisory signals that can be more beneficial than classical unsupervised methods. Shown in Figure 2.1 is RotNet [1] a network that learns to predict how much an image has been rotated by. The computer randomly picks a multiple of 90 degrees to rotate the image by and uses that amount as the label to be passed to the loss function along with the network's prediction.

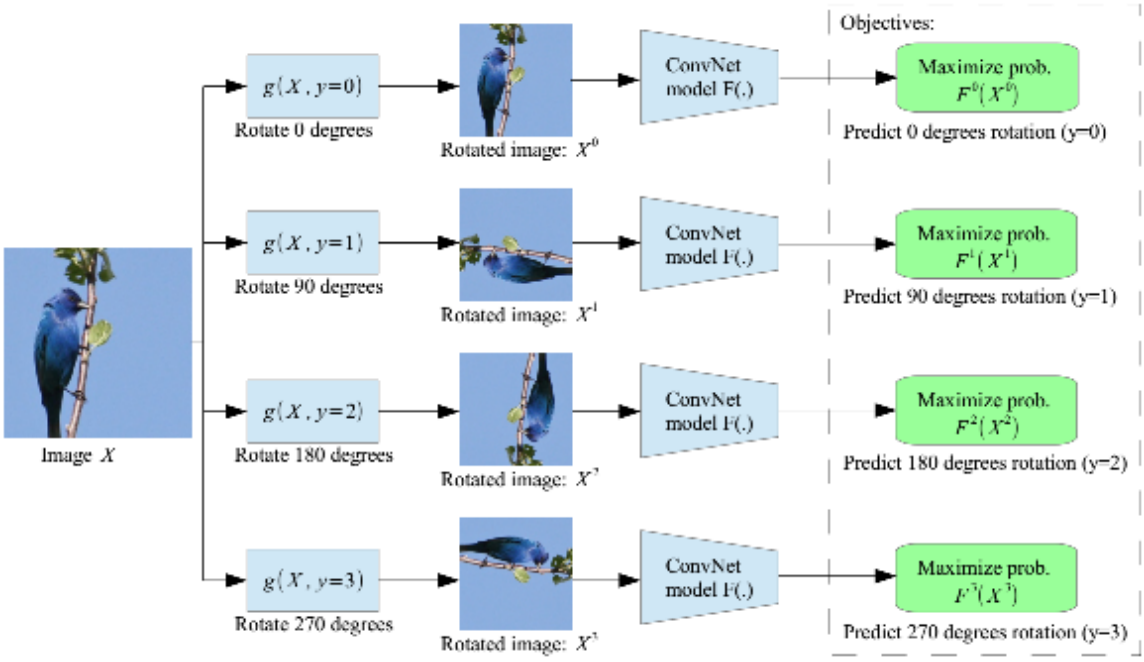

Figure 2.1: RotNet Architecture for predicting an image's rotation. The model uses a shared CNN feature extractor [1].

After training, the weights used in the fully convolutional feature extractor can then be used in other tasks such as classification, detection, or segmentation. These weights typically perform better than random initialization. Other algorithms for generating labels include generative methods such as trying to colorize a RGB image converted to grayscale [24] or restoring a region of an image that has been removed [12]. Other pretext tasks include trying to find the relative location of an image patch [15], reordering frames from a sequential video [25, 26], clustering similar objects or images together [18, 21, 27], detecting blurred regions [28], or using synthetic images [29]. Kolesnikov et al. [30] has shown that specific pretext tasks perform well with different model architectures, and that overall like the rest of the field, deeper and wider models outperform shallower models.

2.4 Fully Convolution Neural Networks

FCNs are a type of neural network that are able to produce predictions that match the spatial resolution of the input image. FCNs are comprised of convolutional layers, transposed convolutional layers, pooling layers, normalization layers, and activation functions. Unlike traditional neural networks that consist of multiple layers of perceptron nodes that are fully connected to the previous and following layer, or CNNs that start with convolutional layers that increasingly reduces the spatial resolution of the features extracted as the input passes through the network before passing the feature vector to any number of fully connected perceptron layers, a FCN typically does not contain any fully connected layers of perceptrons. A FCN acts like a typical CNN with sequential convolutional and pooling layers that reduce the spatial resolution and increase the depth of the representation, but before the feature vector is completely flattened and passed to a fully connected layers like in the previously described CNN the feature vector will be passed to any number of upsampling layers that reduce depth and increase the spatial resolution before the final layer will make

a prediction at the same spatial resolution as the input sample with a depth equal to the number of classes. Each depth map will contain a probability at each pixel location representing the likelihood the model thinks the pixel belongs to that class.

2.5 Pixel Level Semantic Segmentation

Pixel wise semantic segmentation is the task of giving each pixel location in an image a semantic label. Long et al. [8] showed that a FCN could be pre-trained on image classification, then fine-tuned end to end on semantic segmentation, resulting in state of the art performance on predicting pixel level class labels at the time. Current state of the art models such as deeplabv3+ [31] and ERFnet [3] utilize decoders with transpose convolutions to upscale the predicted labels with finer detail. More computationally intense models such as deeplabv3+ utilize a Spatial Pyramid Pooling layer between the encoder/decoder which combines features from the encoder after passing through different pooling and convolutional layers. Shown in Figure 2.2 is a three channel RGB image passing through a fully convolutional neural network with encoder and decoder. The network outputs a predicted pixel map of the same spatial resolution as the input image.

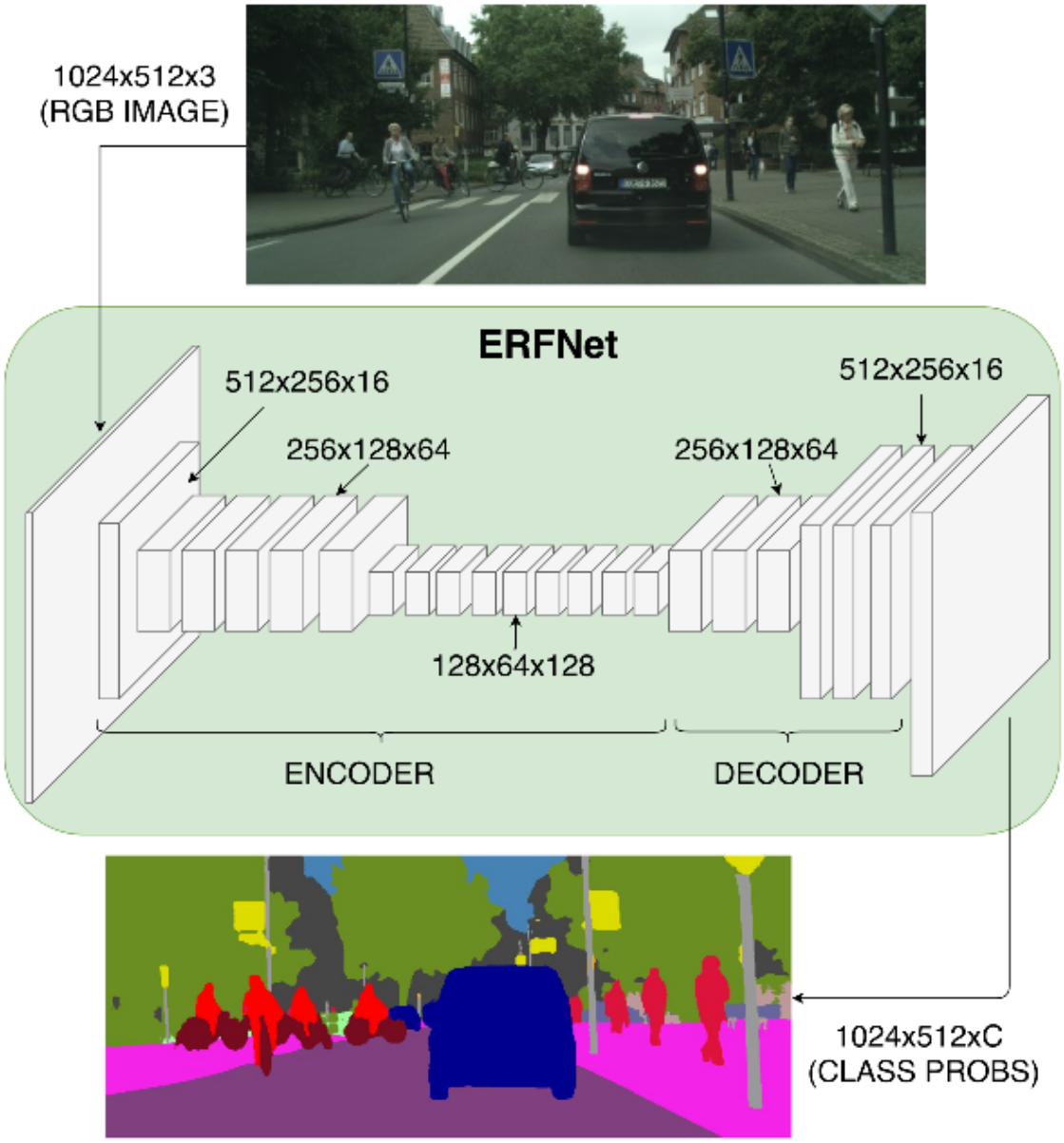

Figure 2.2: Forward pass through ERFNet.

2.6 Lottery Ticket Hypothesis

The lottery ticket hypothesis [32] is the idea that that inside dense feed-forward networks contain sub-networks (winning tickets) that when trained can perform as well as the full network. These sub-networks are found by first training a dense

network on a target task then after training some percent of the weights are pruned by setting them to zero, and a mask is created from each weight that was pruned. The network is then reverted back to the original weight values that existed before training. Using the mask the weights in the network are pruned again, and frozen at zero. Frankle et al. [32] found that pruned networks with their initial weight values could perform up to eight times better than the dense network without any training. They also found that some sub-networks with around 80 percent of their weights pruned could perform better than their dense parent network after training. Zhou et al. [2] hypothesises that using masks to prune weights acts like a type of training setting weights to zero that were initially moving there during the initial training phase. Shown in Figure 2.3 is a common masking criteria used to find winning ticket sub-networks. After the initial training phase weights are sorted from highest to lowest magnitude, the weights with the lowest magnitude are pruned. Other masking criteria may look at weights before and after training and only prune weights that started with a large magnitude and moved to having a small magnitude after training. Others might prune large magnitude weights but exclude weights that flipped sign during training.

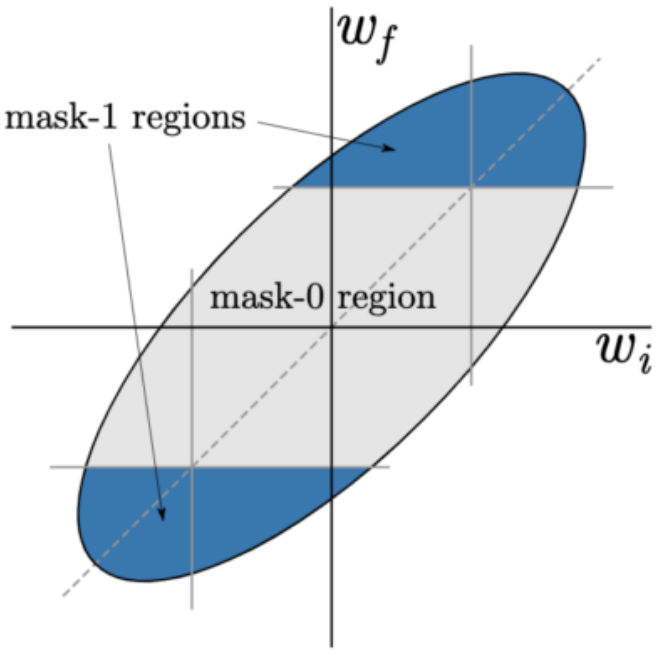

Figure 2.3: Mask criteria for finding winning ticket sub-networks, w_i is the initial weights values and w_f are the final weight values [2].

Chapter 3

Methodology

In this section, we introduce the ERFNet architecture used to explore how semantic segmentation is influenced by pre-trained weights. The goal is to increase performance on semantic segmentation datasets by utilizing traditional transfer learning from large datasets such as ImageNet along with self-supervision performed on in-domain datasets associated with the downstream semantic segmentation datasets such as Cityscapes [33]. Further we use weights from one or more self-supervised tasks to initialize either the encoder or both the encoder and decoder of ERFNet.

3.1 Efficient Residual Factorize Network

ERFNet [3] is a FCN that offers a balance between speed and accuracy. It uses asymmetric convolution blocks with residual connections that allow for real-time performance while attempting to maximize accuracy. To maintain real-time speeds the network forgoes techniques that other higher performing but non-real time networks adopt. Methods such as conditional random fields [34], pyramid pooling [35], multi-resolution blocks [36], or extremely deep and wide backbone feature extracting networks. ERFNet attempts to get the biggest bang for its buck using an unbalanced encoder decoder architecture with larger 16 layer encoder and smaller 7 layer decoder. The network employs heavy downsampling of spatial resolution at the start of the encoder to reduce the number of computations performed. It does this using two

Table 3.1: ERFNet architecture with input image of 512×1024

	Layer	Type	out-F	out-Res
ENC	1	Downsampling block	16	256x512
ENC	2	Downsampling block	64	128x256
ENC	3-7	non-bt-1D block	64	128x256
ENC	8	Downsampling block	128	64x128
ENC	9	non-bt-1D(dilated 2)	128	64x128
ENC	10	non-bt-1D(dilated 4)	128	64x128
ENC	11	non-bt-1D(dilated 8)	128	64x128
ENC	12	non-bt-1D(dilated 16)	128	64x128
ENC	13	non-bt-1D(dilated 2)	128	64x128
ENC	14	non-bt-1D(dilated 4)	128	64x128
ENC	15	non-bt-1D(dilated 8)	128	64x128
ENC	16	non-bt-1D(dilated 16)	128	64x128
DEC	17	Transposed Conv	64	126x256
DEC	18-19	non-bt-1D	64	126x256
DEC	20	Transposed Conv	16	256x512
DEC	21-22	non-bt-1D	16	256x512
DEC	23	Transposed Conv	C	512x1024

sequential downsampling blocks that reduce spatial resolution by a factor of four. It is then followed by five non-downsampling blocks, proceed by a final downsampling block and 8 more non-downsampling blocks. The final eight non-downsampling blocks use a pattern of dilated convolutions to increase the receptive field, the dilated value follows the order of 2,4,8,16 then repeats. Next the decoder using transposed convolutions with a stride of two upsamples the encoder features. The decoder uses two layers of non-downsampling layers after each transposed convolutional layer and ends with a final transposed convolution. ERFNet is able to make a prediction on an RGB image of size 512×1024 in 16.9 ms on a NVIDIA 1080 Ti. Therefore it can make predictions at up to 59 FPS on half resolution Cityscapes images and even faster at lower resolutions enabling the network to run in real-time pipelines.

3.1.1 ERFNet Blocks

ERFNet uses two different building blocks. Shown in Figure 3.1 are the two blocks used. (a) is the non-bottleneck-1D block that maintains spacial resolution, and (b) is the downsampler block that reduces spatial resolution by half. The non-bottleneck-1D block uses a skip connection to learn a residual from the previous layer. The skip connections helps combat the vanishing gradient problem, and it is hypothesized that a residual function is easier to optimize for than an unreferenced mapping [37]. The block uses a cascading of four asymmetric 1×3 and 3×1 convolutional layers separated by activation maps using rectified linear units (ReLU) [38]. A 1×3 convolutional layer and 3×1 convolutional layer are used to approximate a 3×3 convolutional layer using six parameters, as opposed to nine parameters as a 3×3 convolutional layer. The downsampler block concatenates a 3×3 convolutional layer with stride of two with a max pooling layer also with a stride of two. These layers reduce the spatial resolution by a factor of two. The max pool layer acts as a skip connection that can reduce the spatial resolution. The model includes batch norm [39] at the output of each set of asymmetric layers, the output of the downsampler block, and the output of the transposed convolutional layer in the decoder. The model also uses dropout [40] at the output of the asymmetric convolutional branch to prevent overfitting.

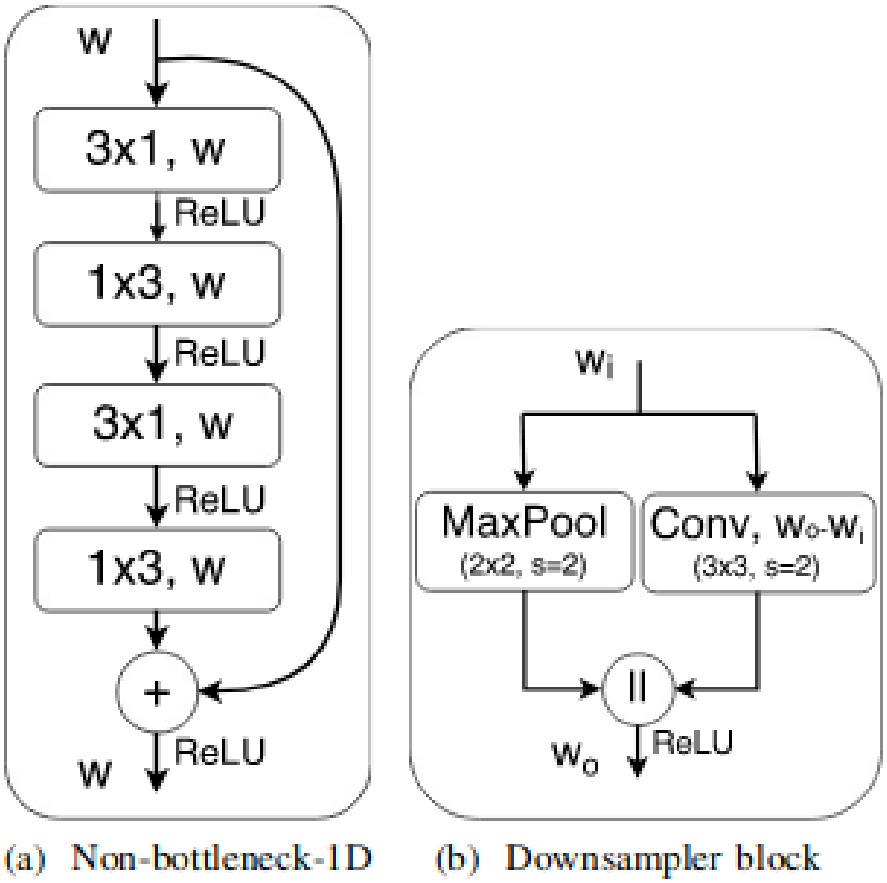

Figure 3.1: Building blocks of ERFNet [3]. (a) is a non-bottleneck block used to maintain spacial resolution. (b) is the downsampler block used to reduce spacial resolution by a factor of two.

3.1.2 Classification Head

For tasks that do not require matching the spatial resolution of the input a classification head replaces the decoder. The classification head takes the features from the encoder at $128 \times H/8 \times W/8$. The spatial dimensions are reduced using a 2D max pooling layer to remove the spatial dimensions. The feature vector is then passed to a 128 neuron linear layer followed by a ReLU activation, and finally to a prediction linear layer with neurons equal to the number of classes, or for SimCLR the size of the final feature representation.

3.1.3 Cross Entropy Loss

Equation (3.1) is the categorical cross entropy loss used to train ERFNet for semantic segmentation. Loss is computed across all pixels, where x is the unnormalized $C \times H \times W$ predictions from ERFNet and class is the ground truth label. The unnormalized score is normalized to a probability from 0 to 1 using a multi-class softmax operation. The negative log is taken of this class prediction. No loss is accrued if a probability of 1 is predicted for the correct class and the loss exponentially increases as the correct classes probability goes to 0. The loss function can be simplified to equation (3.2) and a per class weighting can be added as in equation (3.3).

$$loss(x, class) = -log(\frac{e^{x[class]}}{\sum_j e^{x[j]}}) \tag{3.1}$$

$$loss(x, class) = -x[class] + log(\sum_j e^{x[j]}) \tag{3.2}$$

$$loss(x, class) = weight[class](-x[class] + log(\sum_j e^{x[j]}) \tag{3.3}$$

3.2 Pre-training Tasks

3.2.1 Random Initialization

When initializing a neural network from scratch millions of parameters need to be set. In FCNs each convolutional layer needs to have its weights initialized. Even efficient networks like ERFNet contain millions of parameters. If we randomly initialize the weights with no heuristic layer outputs can explode leading to instability [41]. One common heuristic is to initialize weights by sampling from a [-1,1] uniform distribution that is scaled by a factor $1/\sqrt{n}$ where n is the number of weights in the layer [42].

This type of initialization can have its own problem. Glorot et al. [42] showed how using this initialization in deeper networks can lead to a vanishing gradient problem where deeper layers have activation values that converge towards zero. This causes gradients in deeper networks to vanish as loss is backpropagated to earlier layers. Glorot et al. proposed normalized initialization now termed Xavier initialization that has weights randomly sampled from a uniform distribution described by (3.4). n_i is the number of incoming connections and n_{i+1} is the number of outgoing connections of the layer being initialized. This initialization helped the network maintain variance of activation values, and maintain the variance of weight gradients across all layers. This helps the model converge quicker and reach a higher accuracy.

$$w \sim U[-\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}] \tag{3.4}$$

Recent advances in activation functions use ReLU activation [38] over previously used activation functions such as tanh for deep CNNs. This activation undermines Xavier initialization where outputs will no longer have a mean close to zero and a standard deviation near one. He et al. [43] showed that CNNs using ReLU activations that have more than 30 layers can fail to converge with Xavier initialization. They noticed how the vanishing gradient problem reappears in very deep CNNs that use ReLU activation and Xavier initialization. To train very deep networks He et al. proposed a new initialization method with ReLU activation in mind. They proposed sampling from a gaussian distribution with a standard deviation of $\sqrt{2/n}$ where n is the number of incoming connections. Using this initialization strategy the networks outputs are much closer to having a mean of zero and standard deviation of one. They used this method termed Kaiming initialization to train a 22 layer CNN faster than Xavior initialization and to train a 30 layer CNN that Xavior initialization failed to converge on.

When training a FCN (which uses ReLUs) from scratch Kaiming initialization

is the most popular method to initialize the network for the first time. However, this is not the best method to train a FCN for a targeted task. Transfer learning from another out-of-domain dataset will allow the network to learn features before training on the target dataset. Even better self-supervised learning can allow the model to learn features from an in-domain dataset that is typically much larger than the target dataset. By pre-training the network it can achieve higher performance on the downstream task than using Kaiming or Xavior initialization.

3.2.2 Rotational Task

The self-supervised rotational task [1] is a pretext task in which an input image undergoes a geometric rotational transformation. The amount the image is rotated is used as ground truth label to train a CNN to predict the transformation performed on the input image. Using lessons learned from Gidaris et al. [1] we turn this task into a discrete classification problem with four classes to choose from. Those classes correspond to a rotation of 0, 90, 180, or 270 degree rotation. They showed that increasing the number of classes to eight using multiples of 45 degrees or less classes using multiples of 180 degrees resulted in degraded downstream performance. Since the task simply involves rotating a square RGB image, the task can be extended to almost any image. Shown in Figure 3.2 is a cropped sample from the Cityscapes [33] dataset. The image on the left undergoes a rotation of 0 degrees while the image on the right was rotated by 90 degrees. The task supplies strong supervisory signals that force the model to learn semantic features that transfer well to other visual tasks. Data augmentation used for the rotation task includes random cropping, image translation, color jitter, and normalization.

Figure 3.2: Cityscapes sample showing the cropped input for the rotational task. Input is rotated counterclockwise by a multiple of 90 degrees.

3.2.3 Jigsaw Task

The jigsaw task [44] is a pretext task that has a CNN solve nine piece jigsaw puzzles to learn semantic visual features. A nine piece jigsaw puzzle has 362,880 permutations of pieces. Instead of predicting the correct location of each patch from one of the nine possible locations, the task is framed to predict the permutation that was used to shuffle the patches. Further the number of permutations is restricted to a subset of all permutations. By controlling the size of the subset of permutations used the difficulty of the task can be changed. When size of the subset is increased the difficulty increases and downstream performance increases. Noroozi et al. [44] also saw increased performance when the average hamming distance between permutations in the subset was maximized. This means that the number of different tile locations between two permutations is maximized. This helps mitigate ambiguous guesses between permutations by the model. For example when all the tile locations are the same except for two and those tiles could be extremely similar such as patches of pavement with no distinguishing features. Shown in Figure 3.3 is the process of extracting the patches from the sample image. First a random region from the image

is cropped, this region is then resized to 225×225 then from each 75×75 pixel region in a 3×3 grid has a 64×64 random crop taken from it and becomes one of the final nine patches. These patches shown in green do not touch each other making the task harder to solve. Shown in Figure 3.4 is how the patches might be shuffled after being permuted. The patches are sequentially fed to the ERFNet encoder for feature extraction then each feature vector is concatenated and passed to a 4096 neuron fully connected layer with ReLU activation. The output is then passed to a final fully connected layer with neurons equal to the number of permutations being predicted. The resolution of each patch is smaller, and has a different aspect ratio than downstream tasks. Generally it is best practice to keep input sizes and aspect ratios the same between tasks, but we do not do this due to the computational overhead of using 512×1024 patches. FCNs can handle different size images, but it is unclear how it effects the features learned. Shown in Figure 3.5 is how a solved jigsaw puzzle will look when constructed from randomly cropped patches. Data augmentation used in this task involve random horizontal flipping and translation before cropping, along with color jitter and normalization applied to each patch separately.

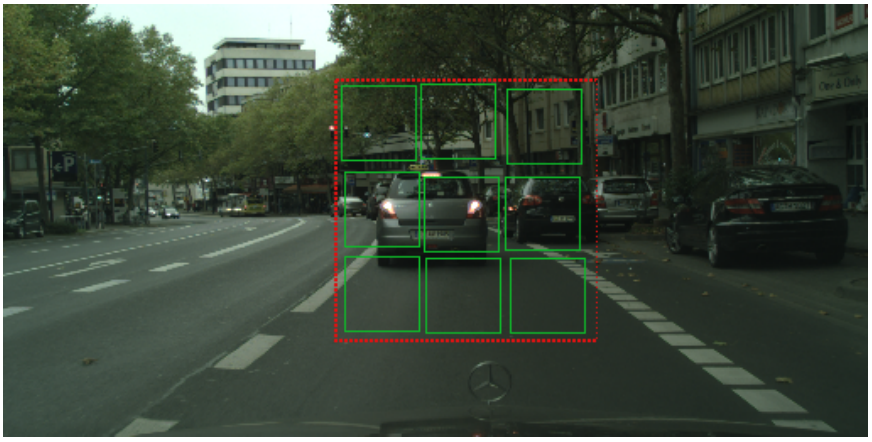

Figure 3.3: Cityscapes crop and patch extraction.

Figure 3.4: Cityscapes patch reordering.

Figure 3.5: Cityscapes puzzle solution.

3.2.4 Relative Patch Task

The relative patch task [15] is similar to the jigsaw puzzle task. In this task, patches are extracted in the same way as the jigsaw task. Instead of seeing each patch in a shuffled order and asked to predict the permutation the model will first see the center patch then at random one of the eight adjacent patches will be fed to the model. The location of the randomly chosen patch is treated as the ground truth and model must try to predict where the second patch came from in relation to the center patch. This task is considered more difficult to solve than the jigsaw task because it only allows the model to see two of nine patches before prediction. This task also uses the same data augmentation as the jigsaw task. That being random horizontal flipping and translation before random cropping, and random color jitter and normalization

to each extracted patch.

3.2.5 Colorize Task

The colorize task [24] is a task that has a FCN learn how to predict the color of a gray scale image. By starting with a colored image we first convert to grayscale and then are able to use the color of the image as ground truth to train the model. Following work done by Zhang et al. [24] we first convert the image to the CIEL*A*B* color space [4]. In this color space the L* channel is lightness from black to white in the range [0:100]. The A* and B* channels represent the chromaticity from green to red and from blue to yellow in the range[-128,128]. The color space is useful for this task because the changes in the A* and B* channels have roughly the same perceived change in color throughout the color space. To allow for easier transfer learning to semantic segmentation as a downstream task we concatenate the L* channel three times so it has the same number of dimensions as an RGB image. Then the grayscale image passes through both the encoder and decoder of ERFNet to produce a predicted pixel map of size $2 \times H \times W$ where the first dimension is the predicted A* channel values for each pixel location and the second is the predicted B* channel values. Loss is computed by taking the mean squared error (MSE) between the predicted values of each color channel and the ground truth values. MSE is the squared euclidean distance between two points or the squared L2 norm. Shown in Figure 3.6 is a visualization of a forward pass through the network during training. This task has the advantage of generating trained weights for both the encoder and decoder. Data augmentation for this task includes random horizontal flipping, random translation, and random cropping. We also perform random color jittering before converting to the CIEL*A*B* color space.

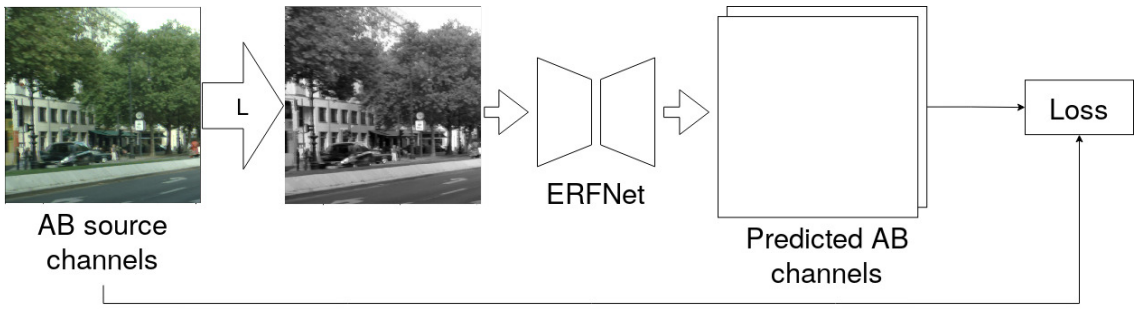

Figure 3.6: Colorize task pipeline. RGB source images are converted to CIEL*A*B* color space [4]. The lightness channel is passed the the ERFNet model and the model predicts the A*B* color channels.

3.2.6 Otsu's Method Clustering Task

The Otsu's method clustering task has a CNN predict the classes derived by Otsu's method [45]. The simplest form of Otsu's method is to pick a single threshold based on pixel intensity of a grayscale image and separate the pixels into two classes. Class one is every pixel at or below the threshold and class two is every pixel above the threshold. The threshold is calculated by iterating through each number in the range of all possible grayscale values and calculating the between-class variance when the two classes are determined using that threshold. The optimal threshold is that which maximizes the between-class variance and equivalently that minimizes the within-class variance. This method is then extended to multiple classes by performing a multilevel Otsu's method thresholding algorithm to determine multiple thresholds to calculate classes that maximizes the between-class variance. Due to an exploding time complexity that increases with each class the number of classes is restricted to a maximum of four. Shown in Figure 3.7 is the algorithm applied to a Cityscapes training image with four clusters being computed. The clusters generated by the algorithm is then used as ground truth to train ERFNet to perform semantic segmentation. This method is in some sense the closest to the downstream task except the classes are not based on real world objects but cluster labels assigned by the algorithm. To make

the task harder we use some of the same data augmentation as the other tasks, that being random horizontal flipping, random translation, and random cropping.

Figure 3.7: Cityscapes RGB input image and associated otsu class labels.

3.2.7 SimCLR

A simple framework for contrastive learning of visual representations named SimCLR [5] is a recent framework for contrastive learning. Contrastive learning is a unsupervised learning method that learns representations by forcing similar samples to be in agreement and dissimilar samples to be in disagreement. SimCLR is a visual framework and learns visual representations by extracting feature representations computed by CNN's from RGB images. Similar samples or positive samples are generated from the same RGB image and negative samples are ones generated from different images.

Samples are generated in pairs by applying random stochastic data augmentations to a base image. This means that in one minibatch each sample will have one positive pair and the rest will be negative pairs. Each generated sample will then be passed through the ERFNet's encoder for feature extraction. The feature representation then goes through a nonlinear projection head described in Section 3.1.2. The final representation is then passed to the normalized temperature-scaled cross entropy loss (NT-Xent) along with every other representation in the minibatch. Shown in Figure 3.8 is an example of how a positive sample is generated and used during training. The data augmentation pipeline used to generate positive pairs is the sequential application of random cropping with resizing to original size, random horizontal flipping, random color jitter, random grayscale, and gaussian blurring. Due to the random nature of all these transformations it is extremely unlike two samples will come out exactly the same, but since they come from the same base image they will share visual features.

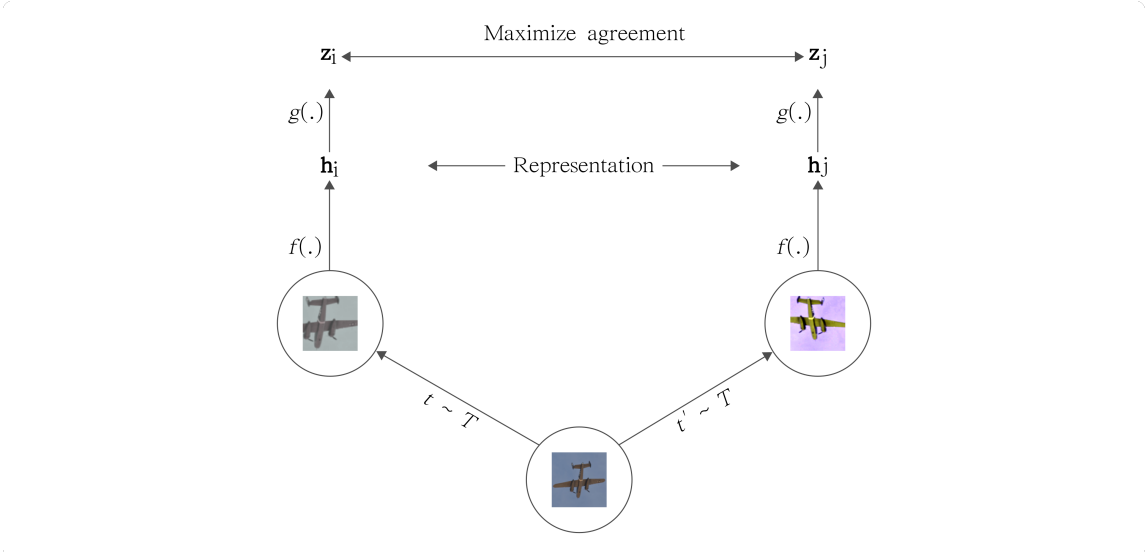

Figure 3.8: The source image is randomly augmented twice. The modified images are passed into a feature extractor f(.). The feature representation h_j is then passed to the projection head g(.). The final representation z_j is then used for comparison with other feature representations in the minibatch [5].

3.2.7.1 Normalized temperature-scaled cross entropy loss

For a minibatch starting with N images $2N$ augmented samples will be generated. Each sample will have one positive sample and $2(N-1)$ negative samples. Shown in Equation (3.5) is the NT-Xent loss used to compute loss for sample i with j being its positive example. sim is defined as the cosine similarity between two vectors. z_i is the final feature representation for sample i and z_j is the final feature representation for sample j. τ is the temperature parameter used to scale the similarity measure. τ in this thesis is set to 0.5. z_k when $k \neq i$ and $k \neq j$ represents the feature representation of the negative samples. $\mathbb{1}$ is an indicator function that is equal to one when $k \neq i$.

$$loss(i,j) = -\log \frac{\exp(sim(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}[k \neq i] \exp(sim(z_i, z_k)/\tau)} \tag{3.5}$$

Chapter 4

Implementation

4.1 Datasets

4.1.1 Cityscapes

The Cityscapes dataset [33] is a European city centric dataset with images taken from the hood of a sedan. The dataset is commonly used to report semantic segmentation performance and contains 2975 training samples and 500 validation samples. The dataset also contains coarse labels. Shown in Figure 4.1 is a sample image from the dataset.

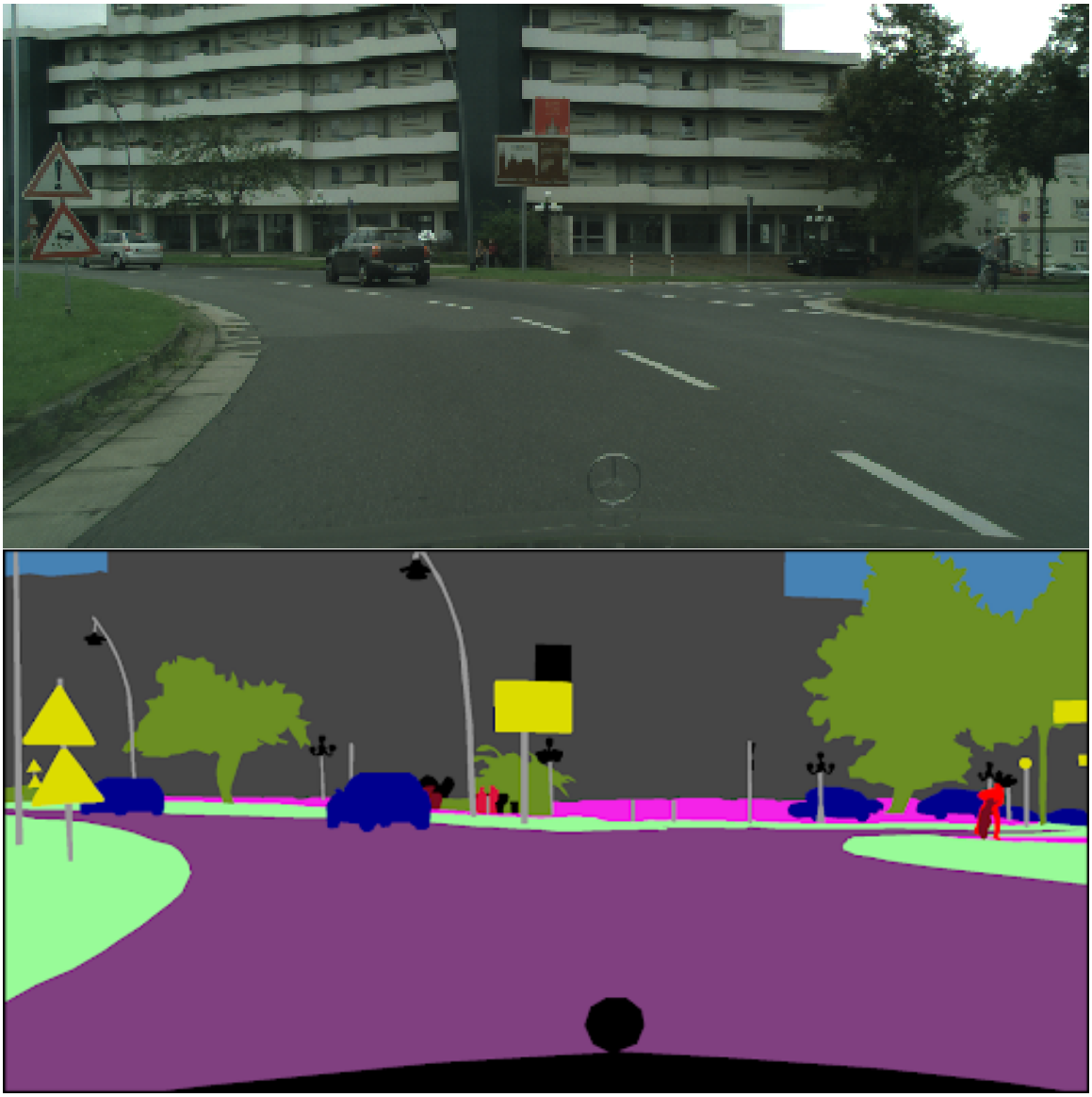

Figure 4.1: Sample image and pixel level label from Cityscapes. Each image is taken from the hood of the test car.

4.1.2 Camvid

The Cambridge-driving Labeled Video Database (Camvid) [46] is another driving centric semantic segmentation dataset, captured in London. The images are extracted from four different video segments taken at different times of day. The videos are captured from a car's perspective with a fixed camera. The version of the dataset

used contains 12 classes including an unlabeled class. The datasets contains many road variations like pavement and sidewalk, and temporal objects such as cars and pedestrians. Figure 4.2 shows images sampled from the Camvid dataset showing the urban environment. Below each image is its pixel level semantic segmentation mask.

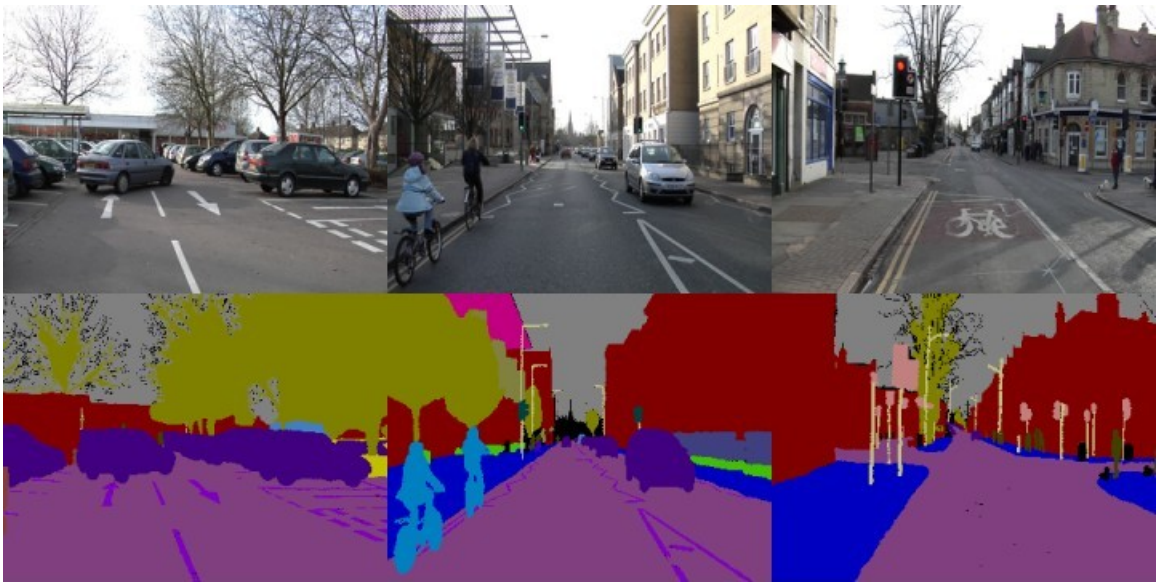

Figure 4.2: Sample images and pixel level labels from Camvid. Each image is taken from a car's perspective.

4.1.3 Imagenet

ImageNet [6] is a large-scale object classification dataset with 1000 classes. Each class has between 800-1300 examples resulting in the train set having over one million training samples. The dataset also provides 50,000 samples for validation. This dataset is commonly used to pre-train encoders for feature extraction. Shown in Figure 4.3 is a subset of the images contained in ImageNet's training set.

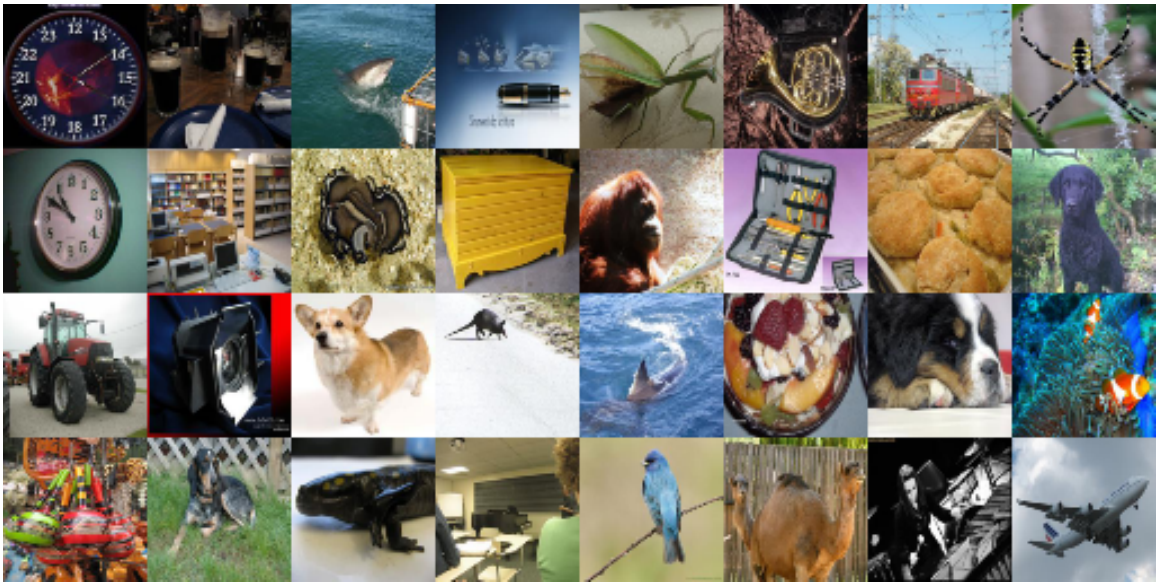

Figure 4.3: Sample images from the ImageNet Dataset [6].

4.2 Implementation

The ERFNet architecture is trained on the Cityscapes and Camvid dataset to perform semantic segmentation. The model is either initialized with pre-trained weights from another task or is randomly initialized. The pre-trained weights can include the encoder and decoder of ERFNet when trained on tasks that utilize both. If the pre-training task does not use the ERFNet decoder the model is initialized with another pre-trained task's decoder weights, or is randomly initialized. When transferring from an encoder only task the projection head used to perform classification or produce embeddings is discarded. When transfer learning from a task that utilizes the decoder the last convolution layer is discarded and replaced with a new convolutional layer used to predict the class probabilities of the down stream task. The PyTorch framework is used to perform all experiments using Nvidia GPUs. A batch size of 20 is used for semantic segmentation tasks with a Learning rate of 5e-4. A polynomial learning rate scheduler is used to reduce the learning rate on a per epoch basis. The popular Adam optimizer is used to perform backpropagation and update the weights

of the network after each iteration. The loss backpropagated is calculated using cross entropy loss with class weighting. The class weights are computed using the weighting scheme outlined in the ENet paper [9], calculated by equation (4.1). Self-supervised tasks uses a batch size of 128 for tasks that only use the encoder and 20 for tasks the used both encoder and decoder. All tasks use a learning rate of 5e-4, the Adam optimizer and a polynomial learning rate scheduler. Loss criterion is specific to the task, but all classification tasks use cross entropy loss.

4.2.1 Class Weighting

The datasets used in this thesis work contain unbalance classes. Classes such as road, building, and sidewalk dominate the city centric datasets. To remedy this we introduce class weighting to help the model perform better on classes with low probability of showing up. Common class weighting strategies use inverse class probability weighting to inflate the loss of low probability classes that helps the model learn to predict these classes. The ENet weighting strategy [9] uses an additional hyper parameter c to add bounds to the weights calculated by equation (4.1). c is set to 1.02 which restricts the interval from [1.42,50.49].

$$w_{class} = \frac{1}{\ln(c + p_{class})} \tag{4.1}$$

4.2.2 Adam Optimizer

Adaptive moment estimation (named Adam) [47] is a optimization method to back-propagate loss through a neural network to update the weights to obtain a more optimal solution. Adam combines successful methods from AdaGrad [48] and RMSProp [49]. Adam adaptively estimates the first and second moments of the gradient, and computes scaled learning rate for each parameter by multiplying the base learning rate by the first order moment divided by the square root of the second order

moment. This optimizer is used in many computer vision tasks and was used by both ENet and ERFNet. The optimizer helps speed up convergence, and performs similarly if not better than stochastic gradient descent.

4.2.3 Polynomial Learning Rate Scheduler

Using a learning rate scheduler helps the model settle deeply into a local minima by reducing the step size as the model plateaus. Some methods reduce the learning rate by half or a factor of ten after a set number of epochs while others reduce the learning rate after a set number of epochs without decrease in loss for the validation set. The polynomial learning rate steadily reduces the learning rate each epoch according to equation (4.2). When using a base learning rate of 5e-4 the learning rate will decrease as seen in Figure 4.4. Polynomial learning rate has shown to converge faster than step policy and with higher accuracy. It is used by the PyTorch ERFNet implementation and others such as [50].

$$lr = baselr * (1 - \frac{epoch}{maxepochs})^{0.9} \tag{4.2}$$

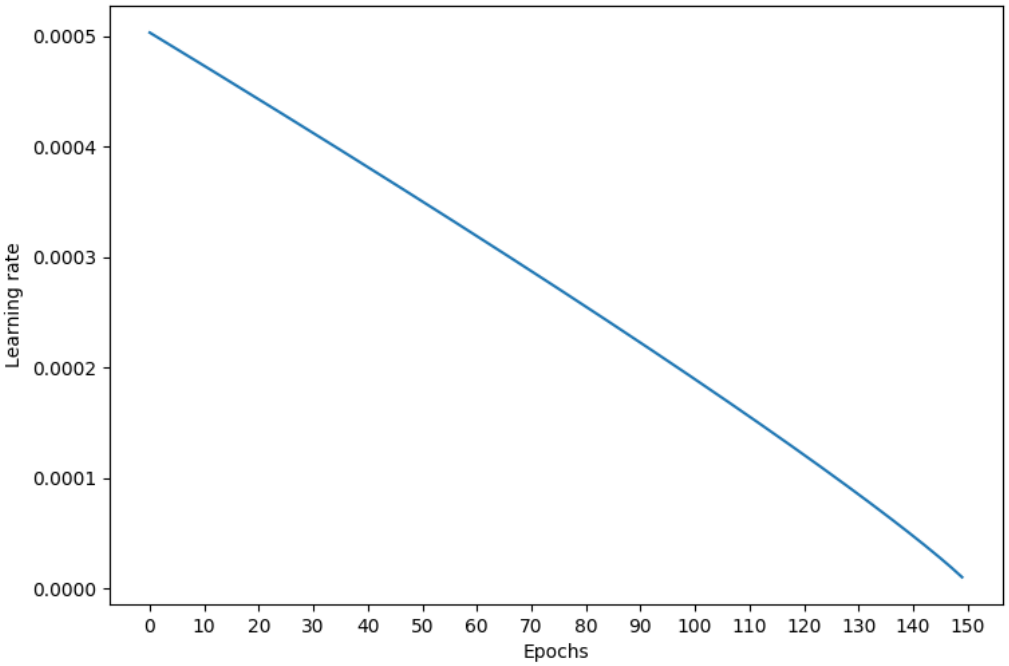

Figure 4.4: Learning rate decay per epoch when using the polynomial learning rate scheduler.

4.2.4 Metric

4.2.4.1 Intersection Over Union

To evaluate the performance of ERFNet on semantic segmentation the commonly used intersection over union (IoU) is used. IoU is also used for object detection where predicted bounding boxes are compared to their ground truth label. The calculation for IoU for detection can be visualized in Figure 4.5. Here the intersection of the predicted and ground truth bounding boxes are divided by the total union of the two bounding boxes. This metric encourages models that predict bounding boxes that cover the entire ground truth bounding box while minimizing the area that extends outside the ground truth box. For semantic segmentation the metric is extended from rectangles to pixel overlap. The IoU for a pixel prediction would take the

total number of the correctly predicted pixels over the total number of the correctly predicted pixels, incorrectly predicted pixels, and unclassified pixels.

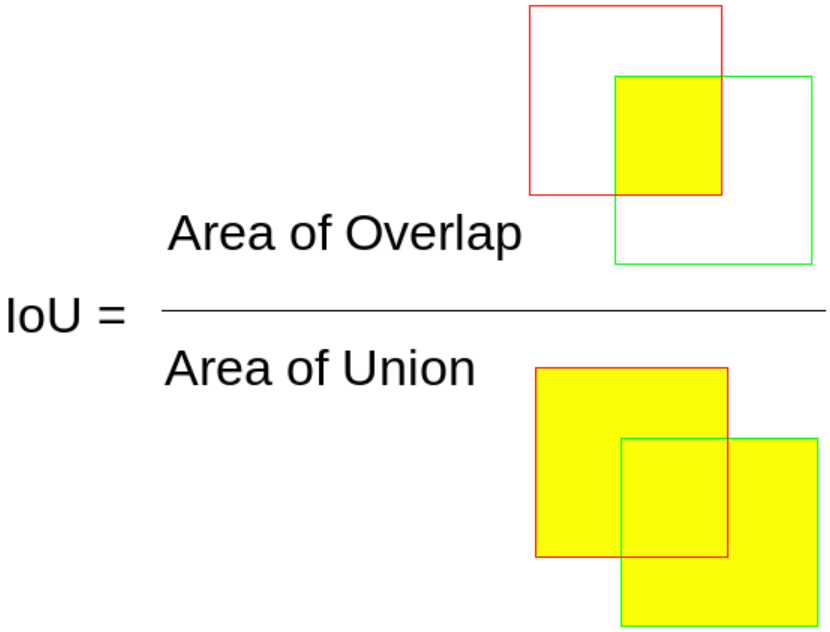

Figure 4.5: Visualization of the IoU equation.

4.2.4.2 Classification Accuracy

Classification Accuracy is used as an evaluation metric for the ImageNet dataset. It is also used for the self-supervised learning tasks rotation, jigsaw, and relative patch. Other metrics for classification include top-5 predictions used by the ILSVRC [20] that look at the five highest confident predictions in descending order. Accuracy for this thesis can be seen as a top-1 prediction only taking into account the highest confident prediction. The metric is calculated by simply taking the number of correct predictions over the total number of predictions. A more complicated metric is not used because the goal is to simply gauge how well the pre-training task is performing so its knowledge can be transferred to the downstream task.

Chapter 5

Results and Analysis

5.1 Results

The ERFNet model is benchmarked on the Cityscapes validation set using class mean intersection over union. Mean intersection over union (mIoU) is the average of the class-wise IoUs computed for each sample in the validation set. The model is fine-tuned on the 2975 fine-grain training labels. The number of training epochs, learning rate, optimizer, data augmentation, loss function, and class weights are all fixed for comparison. The weights used to initialize the network are varied using the learned weights provided by self-supervised methods described in Section 3.2. Shown in Table 5.1 is the downstream performance of ERFNet on Cityscapes validation set using different pre-training methods. All of the self-supervised methods are first initialized with an encoder pretrained on ImageNet classification. The model is then trained on the self-supervised task on either the ImageNet dataset or the Cityscapes extended dataset. The Cityscapes extended dataset includes an extra ∼19k images with no semantic labels. From Table 5.1 it can be seen that some self-supervised methods hurt the downstream performance of the model when compared to ImageNet classification pre-training alone, while the Rotation task increases performance by 0.9%. Kolesnikov et al. [30] shows how wider and deeper models are able to learn more and perform better at evaluation tasks when pre-trained via self-supervision. ERFNet compared

to non-real time models such as ResNet50 or even VGG19 has much more limited learning capacity. This limited capacity to learn from self-supervised methods could be why most self-supervised tasks even when initialized by ImageNet classification weights perform worst. The model must adapt features it learned previously to solve the new self-supervised task. These new features must be less helpful as a starting point to fine-tune for semantic segmentation than the previous supervised task even when performed on in-domain data. As for why the Rotational task performs better than all the other tasks it could be that the features learned are more helpful than ImageNet classification alone. It is hypothesised that the rotation task is able to best adapt the ImageNet classification weights to the rotation task and tune them to the Cityscapes extended training set. This is thought because the rotation task is the quickest task to train to maximum validation accuracy, taking less than 10 epochs even with full data augmentation. By comparison when we train the same task with no ImageNet pre-training, the rotation task takes six times as long to converge. Based on Table 5.3 the rotational task with no ImageNet classification initialization performs worse than random initialization. It could be that the rotational task does not need to learn much when initialized by ImageNet classification. It could also be that when the task is trained from scratch the model finds a simpler way to achieve similar validation accuracy on the rotational task. The task could be increasing performance because it updates the batchnorm layers with in-domain data, but when we reset the batchnorm parameters after pre-training the performance does not decrease. Inspired by other self-supervised experiments that use earlier convolutional layers to perform linear classification and archive slightly higher accuracy, we reset some of the layers later in the encoder to simulate transferring from an earlier layer. When resetting layer 8-16 the model performed worse, but when resetting only layer 13-16 a slight performance increase is observed.

Shown in Table 5.2 is downstream performance on Cityscapes when also includ-

Table 5.1: Comparison of single pre-training methods when ERFNet is fine-tuned on Cityscapes. Results shown on the unseen validation set. mIoU is mean intersection over union. mIoU averages the IoU of each sample in the validation set. Check marks indicate if the dataset is used for pre-training. All pre-training methods can be used for encoder initialization. Only Otsu's method and colorize can be used for decoder weights. Rotation-waug-layer12 randomly initializes all the layers of the encoder after layer 12.

Pre-train	ImageNet	Cityscapes	mIoU
Kaiming initialization	✗	✗	68.32
Encoder: ImageNet Classification Decoder: Kaiming initialization	✓	✗	70.5
Encoder: Cityscapes Otsu-noaug Decoder: Cityscapes Otsu-noaug	✓	✓	69.51
Encoder: Cityscapes Colorize-noaug Decoder: Cityscapes Colorize-noaug	✓	✓	69.74
Encoder: ImageNet SIMCLR Decoder: Kaiming initialization	✓	✗	69.74
Encoder: Cityscapes Relpatch-waug Decoder: Kaiming initialization	✓	✓	69.91
Encoder: Cityscapes Jigsaw-waug Decoder: Kaiming initialization	✓	✓	70.29
Encoder: Cityscapes Rotation-waug Decoder: Kaiming initialization	✓	✓	71.4
Encoder: Cityscapes Rotation-waug-layer12 Decoder: Kaiming initialization	✓	✓	71.56

Table 5.2: Results for when ERFNet is fine-tuned on Cityscapes when combining a strong performing encoder pre-training method with a decoder trained on another task. Results shown on the unseen validation set. Check marks indicates if the dataset is used for pre-training.

Pre-train	ImageNet	Cityscapes	mIoU
Encoder: ImageNet Classification Decoder: ImageNet Otsu-noaug	✓	✗	70.99
Encoder: Cityscapes Rotation-waug Decoder: ImageNet Otsu-noaug	✓	✓	71.6

ing decoder weights to the previous well performing self-supervised methods. Decoder initialization methods include Otsu's method clustering and colorization. When using self-supervised decoder weights in conjunction with ImageNet classification or rotation encoder weights the downstream performance is increased by 0.49% and 0.2% respectively. Compared to random initialization the use of self-supervised pretext tasks should help both the encoder and decoder perform better. Random initialization helps prevent the network from experiencing an exploding or vanishing gradient problem but does not provide any knowledge to the network. Pretext tasks are able to teach the network what features to extract, and does so using in-domain data making these features less difficult to transfer to the downstream task. Comparing these pretext tasks to transfer learning from other semantic datasets, the latter could perform better especially if the datasets are very similar, such as transfer learning from Cityscapes to Camvid. Both of these datasets are in urban environments taken from a car's perspective and many of the classes are the same between the two datasets. However a self-supervised task might perform better than transfer learning from Camvid to Cityscapes due to the reduced size and difficulty of the Camvid dataset. The self-supervised pretext tasks will be able to utilize over 50 times more samples using unlabeled data from Cityscapes extended training set than using the supervised Camvid training set.

Table 5.3: Comparison of how the method for initializing pre-training tasks effect downstream performance. Results shown on the Cityscapes Validation set. Both rotation tasks are trained using the Cityscapes extended dataset.

Self-Supervised Task	Rotation Task Init	ImageNet	Cityscapes	mIoU
Encoder: Rotation-waug Decoder: Kaiming initialization	Kaiming init	✗	✓	67.52
Encoder: Rotation-waug Decoder: Kaiming initialization	ImageNet Class	✓	✓	71.4

Table 5.4: Comparison of training procedures for transfer learning to semantic segmentation. Fine-tuned on Cityscapes, results on Cityscapes Validation set. First the model is loaded with encoder weights from the rotation task. Then the intermediary stage freezes all the weights of the encoder and trains the decoder for 40 epochs. Finally the encoder is unfrozen and the entire model is trained for 150 epochs.

Self-Supervised Task	Intermediary Stage	mIoU
Encoder: Cityscapes Rotation-waug Decoder: Kaiming initialization	✓	70.66
Encoder: Cityscapes Rotation-waug Decoder: Kaiming initialization	✗	71.40

Table 5.5: Cityscapes results with training data reduced by half. Check mark indicates the dataset used for pre-training.

Pre-train	Imagenet	Cityscapes	mIoU
Kaiming initialization	✗	✗	60.0
Encoder: ImageNet Classification Decoder: Kaiming initialization	✓	✗	64.5
Encoder: ImageNet Otsu-noaug Decoder: ImageNet Otsu-noaug	✓	✗	64.0
Encoder: Cityscapes Rotation-waug Decoder: Kaiming initialization	✓	✓	65.93

Shown in Table 5.5 are different methods effects on the Cityscapes downstream performance when the fine Cityscapes labels are reduced by 50%. Here we can see how the rotation task pre-training gives almost a 6% increase over random initialization. This gives a 2× greater difference when compared to using all the training labels. Pre-training methods help the model perform even better when less downstream supervised data is present. The margin between the rotation task and ImageNet classification also increases showing that the inclusion of the self-supervised task trained on in-domain unlabeled data is increasingly helpful when labeled data is scarce.

Data augmentation's effect on downstream performance has been observed by many who have contributed to self-supervision [5, 15, 30]. The same is true for self-supervised learning for semantic segmentation, Table 5.6 shows the rotation task train on both the Cityscapes and ImageNet dataset. The table contains downstream performance on the Cityscape validation set for the rotation task trained with and without data augmentation. In both cases the rotation task performs better with the inclusion of data augmentation. The features learned with heavy data augmentation are able to better generalize to new tasks. The heavy data augmentation allows the model to receive supervisory signals from a larger variety of samples. Data augmentation in the rotation task includes random translation, cropping, and color jittering, along with normalization. This forces the model to look at different parts of the im-

Table 5.6: Effects of data augmentation on the downstream task when added during the pre-training stage. Results shown on the Cityscapes validation set. Without data augmentation the image is just resized and randomly rotated. When data augmentation is introduced the pipeline includes random translation, random cropping, random color jittering, and normalization.

Self-Supervised Task	Data Augmentation	mIoU
Encoder: ImageNet Rotation Decoder: Kaiming initialization	✗	62.11
Encoder: ImageNet Rotation Decoder: Kaiming initialization	✓	64.11
Encoder: Cityscapes Rotation Decoder: Kaiming initialization	✗	70.26
Encoder: Cityscapes Rotation Decoder: Kaiming initialization	✓	71.40

age for features to determine rotation, along with different objects that move slightly due to translation and objects that look differently due to color jittering effecting the inputs brightness, contrast, saturation, and hue.

Shown in Table 5.7 is numerous experiments that attempt to modify a pre-trained ERFNet encoder at the individual layer level. 0.75-Cityscapes-Rotation-0.25-Kaiming starts with an encoder pre-trained on the rotation task using the Cityscapes extended dataset. Then we calculate the energy of each layer's feature maps by summing the magnitude of each weight in the feature map. We sort the feature maps by their energy levels and replace 25 percent of the lowest energy feature maps with new ones using Kaiming initialization. We do this using the hypothesis that the self-supervised pretraining tasks are not saturating the capacity of ERFNet, resulting in redundant or unused feature maps. By reinitializing the low energy maps they can be used to learn new features for the downstream task. This was not beneficial, so we increased the percent of feature maps retained and replaced them instead with high energy feature maps from other pre-training tasks. This also did not help, so using the finding that only using layers 1-12 from the rotation task performed better than using all the layers

in the encoder, we initialize layers 13-16 from the Otsu's method task to see if it will increase downstream performance. We use the Otsu's method task because it can be viewed as a pseudo-semantic segmentation task and features learned in the later layers of the encoder could be more beneficial to the downstream task than Kaiming initialization. This also hurts the downstream learning, so we try one final weight manipulation strategy. Using lessons from Frankle et al. [32] we prune individual weights of each layer, instead of entire feature maps. We create a sparse network by setting 50 percent of the lowest magnitude weights obtained by pre-training ERFNet on the rotation task to zero. We then create a mask of these pruned weights and keep them set to zero while fine-tuning on the Cityscapes semantic segmentation task. While this also fails to increase performance over using all the weights from the rotation task, we do achieve a mIoU of 70 percent on the Cityscapes validation set using only 50 percent of the weights in the encoder. A loss of one or two percent mIoU for a sparse network with significantly less parameters could be worth while for faster inference times, and less memory usage.

Shown in Table 5.8 is a comparison between ERFNet and other State-of-the-art (SOTA) models. Most models in the line up contain over 20 times the number of parameters than ERFNet's full model. BiSeNetV2 [51] contains 1.8 times the number of parameters than ERFNet but is able to achieve faster inference times than ERFNet. BiSeNetV2 is a very recent architecture that achieves higher mIoU on Cityscapes and has faster inference times than ERFNet due to its dual branch architecture and booster training strategy. This architecture is much different from ERFNet using a bilateral segmentation backbone instead of a encoder-decoder design. ERFNet does have the advantage of using less memory than BiSeNetV2 and the memory efficiency increases further with ERFNet's pruned model. Compared to the top performing model Hierarchical multi-scale Attention (HMSA) [7], HMSA has a mIoU that is 14.7 percentage points higher than ERFNet. HMSA uses a high resolution large

Table 5.7: Results of modifying weights of the pre-trained encoder at the individual layer level. 0.75-Cityscapes-Rotation-0.25-Kaiming means that at each layer we keep 75 percent of the highest energy feature maps, and reinitialize the other 25 percent using Kaiming initialization. 0.9-Cityscapes-Rotation-0.1-ImageNet-class keeps 90 percent of the highest energy feature maps and replaces 10 percent with the highest energy feature maps from the encoder pre-trained on ImageNet classification. Cityscapes-Rotation-layer1-12-Cityscapes-otsu-layer13-16 uses weights from the rotation task for layers 1-12, and uses weights from the Otsu's method task for layers 13-16. Cityscapes-Rotation-lotto-0.5 sets 50 percent of the smallest magnitude weights to zero, and keeps them at zero while fine-tuning on Cityscapes. Cityscapes-Rotation-lotto-0.75 prunes only 25 percent of the weights. All entries are trained on Cityscapes with results on the Cityscapes validation set.

Self-Supervised Task	mIoU
Encoder: 0.75-Cityscapes-Rotation-0.25-Kaiming Decoder: Kaiming initialization	70.43
Encoder: 0.9-Cityscapes-Rotation-0.1-ImageNet-class Decoder: Kaiming initialization	69.23
Encoder: 0.9-Cityscapes-Rotation-0.1-Cityscapes-Otsu Decoder: Kaiming initialization	68.56
Encoder: 0.9-Cityscapes-Rotation-0.1-Cityscapes-jigsaw Decoder: Kaiming initialization	66.79
Encoder: Cityscapes-Rotation-layer1-12-Cityscapes-otsu-layer13-16 Decoder: Kaiming initialization	69.92
Encoder: Cityscapes-Rotation-lotto-0.5 Decoder: Kaiming initialization	70.00
Encoder: Cityscapes-Rotation-lotto-0.75 Decoder: Kaiming initialization	70.67
Encoder: Cityscapes-Rotation-lotto-0.2 Decoder: ImageNet-Otsu-lotto-0.2	66.58

Table 5.8: Comparison with state-of-the-art on Cityscapes validation set. Both ERFNet models are fine-tuned on Cityscapes after being initialized with the rotation task (encoder) and the Otsu method task (decoder). The pruned version has 80 percent of the weights set to zero before and during training. Hierarchical multi-scale Attention [7] does not report the number of parameters but their backbone network contains 65.8M parameters so their full network must contain more than that.

Model	Number of Parameters	Inference Speed (ms)	mIoU
ERFNet-pruned	0.42M	16.9	66.58
ERFNet-full-best	2.06M	16.9	71.6
BiSeNetV2 [51]	3.65M	6.7	73.4
WASPnet [52]	47.482M	-	74.0
DeepLabV3+ [31]	43.48M	200	79.55
Auto-DeepLab-L [53]	44.42M	-	80.33
Hierarchical-MSA [7]	65.8M+	1170	86.3

capacity backbone network that extracts features at multiple resolutions and uses multiple segmentation heads that make predictions at multiple scales then combines those predictions in a hierarchical fashion. These heads also use attention layers and training uses auto-labelling to boost the size of the training set. Due to the multiple scales and size of the backbone network this network takes over a second to make an inference preventing it from being used in real time applications. Other methods performs better than ERFNet but contain many more parameters and have much slower inference times.

Chapter 6

Conclusions

6.1 Conclusions

In this thesis we have examined a number of self-supervised methods that have made a wide impact on the field of self-supervised learning. We used these self-supervised methods as a way to initialize the weights of a real-time fully convolutional neural network to perform pixel level semantic segmentation on various city centric datasets. We compared the affects of data augmentation during pre-training with its influenced on the downstream semantic segmentation performance. We demonstrated how in-domain unlabeled data can be utilized along side out-domain label data to help models on their target datasets. Finally we explored ways of combining self-supervised tasks, and different ways to transfer learn to maximize the performance of ERFNet.

6.2 Summary of Work

To summarize the results and lessons of this thesis,

- All self-supervised methods used in this thesis can increase performance on semantic segmentation over random initialization, when the self-supervised task is initialized with ImageNet classification weights.

- Only the rotation task trained on in-domain data offers benefit over ImageNet

classification alone when used to initialize the ERFNet encoder before fine-tuning on the target dataset.

- The Otsu's method clustering task can be used to initialize the weights of the ERFNet decoder to increase performance over random initialized weights. This can be used in conjunction with a ERFNet encoder pre-trained on the rotation task to achieve the best results.

- We found that allowing ERFNet to update all weights after transfer learning resulted in better performance, compared to the common best practice of freezing pre-trained layers and only updating new layers added for the target task for a number of epochs, before allowing the optimizer to update all weights.

- To further increase downstream performance by pre-training with the rotation task using in-domain data we only transfer the weights before layer 13 of the ERFNet encoder and randomly initialized the other layers before fine-tuning.

- Heavy data augmentation is useful for self-supervised learning tasks to generate better weights for transfer learning to downstream tasks.

- Sparse networks were found using weights from a self-supervised task. The resulting network was trained on the target downstream task using less than half the weights of the full network with minimal drop in performance.

6.3 Future Work

This thesis research was focused on only a small subset of all self-supervised learning methods. It also only used a few of the available semantic segmentation dataset those being ones that having a wide use in the literature. They are gathered from a vehicles perspective, and can be used to simulate how a vehicle could use semantic

segmentation to understand its surroundings. Other larger datasets such as Mapillary contain a magnitude more samples and could result in different findings. Some possible directions for future work based on this thesis are:

- Analyze more self-supervised methods. Some methods could include generative tasks by a generative adversarial network (GAN). A GAN could also be used in the colorize task as an alternative to MSE loss.

- Increase or decrease in-domain unlabeled data to see how it effects downstream performance, and see if using magnitudes more unlabeled data helps other tasks perform better than ImageNet classification, or remove the need to initialize self-supervised methods with ImageNet classification weights to perform well on the downstream task.

- Explore other semantic segmentation datasets such as PASCAL VOC2012 [54] or MSCOCO [55]. These datasets can be used for unlabeled data for self-supervised learning, or for transfer learning from their supervised labels.

- Additional tasks can also be looked into such as segmentation and boundary detection with datasets such as BSDS500 [56] as a supervised task to transfer learn from. This task is easier to create labels for than pixel level semantic segmentation and could help the network predict better object boundaries. Other methods for segmetation could also be used for self-supervised learning such as handcrafted feature descriptors [57] or texture based methods [58].

- Test how individual data augmentation transforms during self-supervised pre-training influences the downstream task performance.

- Test how knowledge transfer [59] could help pass better visual representations onto ERFNet by training a much larger and wider model on the self-supervised learning tasks before transferring the knowledge onto the smaller network.

ERFNet would then learn these visual representations by model distillation [60] using the probability distributions predicted by the larger network as ground truth labels. ERFNet can also be trained using the cluster centers extracted from the larger network to assign pseudo-labels to each unlabeled image in the dataset as in [59].

- Explore new masking criteria described by Zhou et al. [2] to set individual weights in each layer of a network to zero. They set the percent of pruned weights on a per layer basis, and train the model by updating the mask of each layer instead of the layers weights.

Bibliography

[1] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," *arXiv preprint arXiv:1803.07728*, 2018.

[2] H. Zhou, J. Lan, R. Liu, and J. Yosinski, "Deconstructing lottery tickets: Zeros, signs, and the supermask," *CoRR*, vol. abs/1905.01067, 2019. [Online]. Available: http://arxiv.org/abs/1905.01067

[3] E. Romera, J. M. lvarez, L. M. Bergasa, and R. Arroyo, "Erfnet: Efficient residual factorized convnet for real-time semantic segmentation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 263–272, 2018.

[4] *Image technology colour management Extensions to architecture, profile format and data structure*, International color consortium, 2019, profile version 5.0.0.

[5] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," *arXiv preprint arXiv:2002.05709*, 2020.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[7] A. Tao, K. Sapra, and B. Catanzaro, "Hierarchical multi-scale attention for semantic segmentation," 2020.

[8] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014. [Online]. Available: http://arxiv.org/abs/1411.4038

[9] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," *CoRR*, vol. abs/1606.02147, 2016. [Online]. Available: http://arxiv.org/abs/1606.02147

[10] Q. V. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean, and A. Y. Ng, "Building high-level features using large scale unsupervised learning," 2011.

[11] V. Turchenko, E. Chalmers, and A. Luczak, "A deep convolutional auto-encoder with pooling - unpooling layers in caffe," 2017.

[12] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," 2016.

[13] I. Katircioglu, H. Rhodin, V. Constantin, J. Spörri, M. Salzmann, and P. Fua, "Self-supervised training of proposal-based segmentation via background prediction," *CoRR*, vol. abs/1907.08051, 2019. [Online]. Available: http://arxiv.org/abs/1907.08051

[14] X. Wang and A. Gupta, "Unsupervised learning of visual representations using videos," 2015.

[15] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1422–1430.

[16] A. Dosovitskiy, P. Fischer, J. T. Springenberg, M. Riedmiller, and T. Brox, "Discriminative unsupervised feature learning with exemplar convolutional neural networks," 2014.

[17] C. Doersch and A. Zisserman, "Multi-task self-supervised visual learning," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2051–2060.

[18] T. W. Tsai, C. Li, and J. Zhu, "Ds^{3}l: Deep self-semi-supervised learning for image recognition," *CoRR*, vol. abs/1905.13305, 2019. [Online]. Available: http://arxiv.org/abs/1905.13305

[19] P. Goyal, D. Mahajan, A. Gupta, and I. Misra, "Scaling and benchmarking self-supervised visual representation learning," *CoRR*, vol. abs/1905.01235, 2019. [Online]. Available: http://arxiv.org/abs/1905.01235

[20] 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 (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[21] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," *CoRR*, vol. abs/1807.05520, 2018. [Online]. Available: http://arxiv.org/abs/1807.05520

[22] S. Laine and T. Aila, "Temporal ensembling for semi-supervised learning," *arXiv preprint arXiv:1610.02242*, 2016.

[23] T. Miyato, S.-i. Maeda, M. Koyama, and S. Ishii, "Virtual adversarial training: a regularization method for supervised and semi-supervised learning," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 8, pp. 1979–1993, 2018.

[24] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," *CoRR*, vol. abs/1603.08511, 2016. [Online]. Available: http://arxiv.org/abs/1603.08511

[25] D. Xu, J. Xiao, Z. Zhao, J. Shao, D. Xie, and Y. Zhuang, "Self-supervised spatiotemporal learning via video clip order prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 334–10 343.

[26] I. Misra, C. L. Zitnick, and M. Hebert, "Unsupervised learning using sequential verification for action recognition," *CoRR*, vol. abs/1603.08561, 2016. [Online]. Available: http://arxiv.org/abs/1603.08561

[27] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox, "Discriminative unsupervised feature learning with convolutional neural networks," in *Advances in neural information processing systems*, 2014, pp. 766–774.

[28] A. Alvarez-Gila, A. Galdran, E. Garrote, and J. van de Weijer, "Self-supervised blur detection from synthetically blurred scenes," *Image and Vision Computing*, vol. 92, p. 103804, Dec 2019. [Online]. Available: http://dx.doi.org/10.1016/j.imavis.2019.08.008

[29] Z. Ren and Y. Jae Lee, "Cross-domain self-supervised multi-task feature learning using synthetic imagery," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 762–771.

[30] A. Kolesnikov, X. Zhai, and L. Beyer, "Revisiting self-supervised visual representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2019, pp. 1920–1929.

[31] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," *CoRR*, vol. abs/1802.02611, 2018. [Online]. Available: http://arxiv.org/abs/1802.02611

[32] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Training pruned neural networks," *CoRR*, vol. abs/1803.03635, 2018. [Online]. Available: http://arxiv.org/abs/1803.03635

[33] 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," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[34] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected crfs with gaussian edge potentials," *CoRR*, vol. abs/1210.5644, 2012. [Online]. Available: http://arxiv.org/abs/1210.5644

[35] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," *CoRR*, vol. abs/1612.01105, 2016. [Online]. Available: http://arxiv.org/abs/1612.01105

[36] K. Sun, Y. Zhao, B. Jiang, T. Cheng, B. Xiao, D. Liu, Y. Mu, X. Wang, W. Liu, and J. Wang, "High-resolution representations for labeling pixels and regions," *CoRR*, vol. abs/1904.04514, 2019. [Online]. Available: http://arxiv.org/abs/1904.04514

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[38] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[39] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[40] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[41] J. Dellinger, "Weight initialization in neural networks: A journey from the basics to kaiming," Apr 2019. [Online]. Available: https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-from-the-basics-to-kaiming-954fb9b47c79

[42] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015. [Online]. Available: http://arxiv.org/abs/1502.01852

[44] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," *arXiv preprint arXiv:1603.09246*, 2016.

[45] P.-S. Liao, T.-S. Chen, P.-C. Chung *et al.*, "A fast algorithm for multilevel thresholding," *J. Inf. Sci. Eng.*, vol. 17, no. 5, pp. 713–727, 2001.

[46] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognition Letters*, vol. xx, no. x, pp. xx–xx, 2008.

[47] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[48] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of machine learning research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[49] Y. Bengio, "Rmsprop and equilibrated adaptive learning rates for nonconvex optimization," *corr abs/1502.04390*, 2015.

[50] W. Liu, A. Rabinovich, and A. C. Berg, "Parsenet: Looking wider to see better," *CoRR*, vol. abs/1506.04579, 2015. [Online]. Available: http://arxiv.org/abs/1506.04579

[51] C. Yu, C. Gao, J. Wang, G. Yu, C. Shen, and N. Sang, "Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation," 2020.

[52] B. Artacho and A. Savakis, "Waterfall atrous spatial pooling architecture for efficient semantic segmentation," *Sensors*, vol. 24, p. 5361, 12 2019.

[53] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

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

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

[56] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proc. 8th Int'l Conf. Computer Vision*, vol. 2, July 2001, pp. 416–423.

[57] A. Suga, K. Fukuda, T. Takiguchi, and Y. Ariki, "Object recognition and segmentation using sift and graph cuts," in *ICPR 2008 19th International Conference on Pattern Recognition*. Los Alamitos, CA, USA: IEEE Computer Society, dec 2008. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICPR.2008.4761400

[58] R. Zwiggelaar and E. R. E. Denton, "Texture based segmentation," in *Digital Mammography*, S. M. Astley, M. Brady, C. Rose, and R. Zwiggelaar, Eds. Springer Berlin Heidelberg, 2006, pp. 433–440.

[59] M. Noroozi, A. Vinjimoor, P. Favaro, and H. Pirsiavash, "Boosting self-supervised learning via knowledge transfer," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9359–9367.

[60] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.