I would like to thank Prof. Siniša Šegvić and Petra Bevandić for guidance during the making of this thesis. I would also like to thank my family and friends for being supportive throughout my studies.

CONTENTS

1. Introduction **1**

2. Deep Learning **2**

2.1. Building Blocks 2

2.1.1. Multilayer Perceptrons (MLP) 2

2.1.2. Activation Functions 3

2.1.3. Convolutional Neural Networks 6

2.1.4. Batch Normalization (BN) 8

2.1.5. Residual Connections 9

2.2. Optimization 11

2.2.1. Loss Functions 11

2.2.2. Stochastic Gradient Descent 12

2.2.3. Adam 12

2.2.4. Backpropagation 14

2.2.5. Regularization 15

2.3. Semantic Segmentation 16

2.4. Model Architecture 17

2.4.1. SwiftNet 17

3. Datasets **21**

3.1. Cityscapes 21

3.2. PASCAL VOC 22

3.3. Augmentations 23

4. Semantic Segmentation On Multiple Datasets **25**

4.1. Previous Work 25

4.2. The Concat Model 25

4.3. Universal Taxonomy 26

4.3.1. Cross-Dataset Confusion Matrices 26
4.3.2. Finding Overlapping Labels 28
4.3.3. Constructing The Taxonomy 29
4.3.4. Comparison with the Concat-only approach 30

5. Implementation **31**
5.1. Supporting Libraries 31
5.2. Models 31
5.3. Taxonomy Algorithm 32

6. Experimental Results **33**
6.1. Baseline Models 33
6.2. The Constructed Taxonomy 33
6.3. Applying The Universal Taxonomy 35

7. Conclusion **36**

A. Confusion Matrices **38**

Bibliography **39**

1. Introduction

Semantic segmentation is a branch of computer vision that concerns itself with attributing class labels to individual pixels. This thesis tries to solve the problem of training models on multiple datasets with overlapping labels by creating a *universal taxonomy*. A universal taxonomy is a set of disjunct classes that is analogous to establishing a hierarchical relationship between elements of two (or more) sets of class labels. For the purposes of this thesis, the developed methods will be demonstrated on two image segmentation datasets, namely Cityscapes [1] and PASCAL VOC [2]. Our experiments use the SwiftNet-RN18 model, however the presented techniques are model agnostic and should work with any semantic segmentation architecture.

Semantic segmentation models are typically trained on a dataset consisting of images that are all from the same domain. The Cityscapes dataset is one such example. It contains images of urban environments captured from a moving vehicle. However, in many practical applications, it is desirable to be able to train a semantic segmentation model on images from multiple domains. For example, a self-driving car company may want to train a semantic segmentation model on images from both the Cityscapes and PASCAL datasets, in order to better handle the variety of environments that the car may encounter.

However, training a semantic segmentation model on images from multiple domains is a challenging task, due to the fact that the different domains may have overlapping labels that go beyond textual similarities. For example, the Cityscapes and PASCAL datasets both have a label for "person", which is straightforward to map to a common label by simply using the symbolic meaning of the labels. However, in the Cityscapes dataset, "person" only includes pedestrians, and "rider" is a separate class. On the other hand, in the PASCAL dataset, "person" also includes riders on bicycles and motorcycles. As a result, a naive concat semantic segmentation model trained on images from both the Cityscapes and PASCAL datasets would have difficulty correctly labeling pixels that correspond to riders, since they belong in both "city-rider" and "PASCAL-person".

2. Deep Learning

Deep learning is a form of machine learning that uses many non-linear layers to process data. A deep model learns increasingly complex data representations, leading to high-capacity modelling of the training data. Learning is done by backpropagation (2.2.4), which adjusts the weights of the model to minimize the loss (2.2.1). Deep models were computationally unfeasible until the early 2010s, when a series of architectural breakthroughs and utilisation of GPU hardware lead to a resurgence of interest in deep learning. From that point on, deep learning has been the dominant method in many subfields of artificial intelligence, including the field of computer vision.

2.1. Building Blocks

Deep learning models can be seen as a composition of basic 'building blocks'. Some blocks (e.g. skip connections) are ubiquitous in modern deep learning architectures. These building blocks often solve fundamental underlying issues and are seldom longer than a couple dozen lines of code. However, their apparent simplicity cannot be taken for granted, as they often have non-obvious interactions with the rest of the components in the training loop (Batch Normalization is especially guilty of this [3]). Below we will go through the building blocks needed to understand the SwiftNet-RN18 architecture. This is by no means an exhaustive list of all possible blocks, as even just a single block can have dozens of variations in its own right. The block conceptualisation is also apparent in architecture diagrams, where each component is represented by a block (Figure 2.1).

2.1.1. Multilayer Perceptrons (MLP)

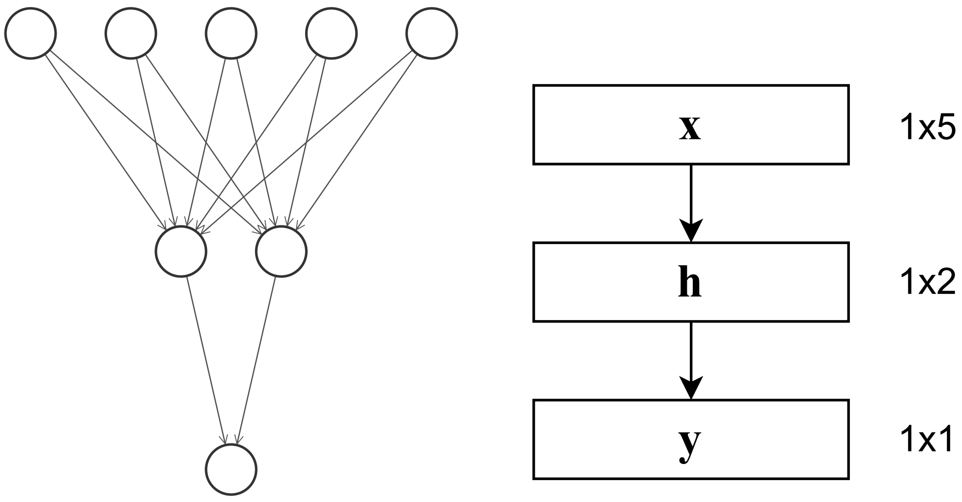
A Multilayer Perceptron (MLP) is a class of fully connected *feed forward* artificial neural networks. An MLP consists of multiple sequential layers of computation. Each layer is comprised of individual neurons, which are then connected to every other neuron in the subsequent layer. The computation of a single neuron can be represented with a *dot product* of the neuron's weight vector **w** (ingoing connections from the previous layer) with the input **x**, offset by a scalar bias b. The weight vector **w** and the bias make up the *parameters* of that neuron. The result is then run through an activation function σ:

$$h = \sigma\left(\mathbf{w} \cdot \mathbf{x} + b\right).$$

In turn, the computation done by a single layer is a linear transformation followed by a (potentially) non-linear activation function:

$$\mathbf{h} = \sigma\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)$$

where **W** is the *weight matrix* and **b** is the *bias vector*.



(a) Explicit representation **(b)** Block representation

Figure 2.1: Equivalent representations of an MLP. The block representation is preferred as it conveys the same amount of information with less visual noise. The block representation is also very convenient for representing abstractions of larger modules. One such example can be found in Figure 2.13, where the encoder and decoder are abstracted into larger blocks.

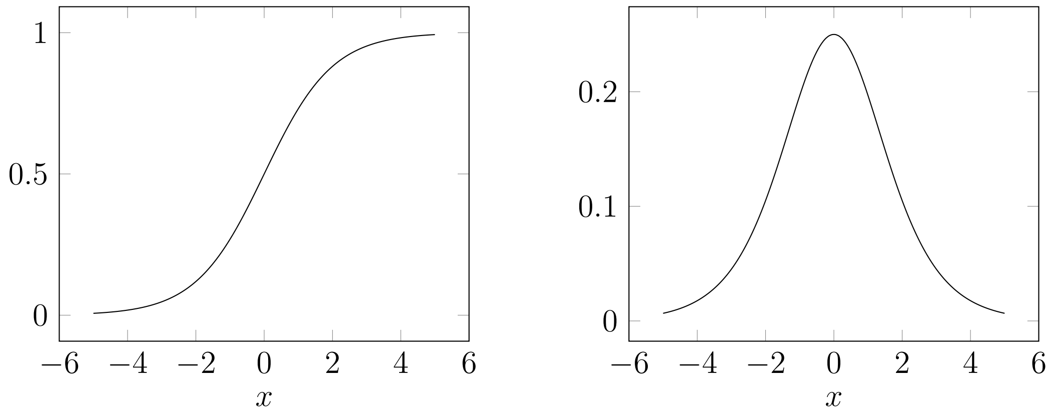
2.1.2. Activation Functions

Due to composability of linear transformations, a completely linear model of arbitrary depth is equivalent to a single layer. Activation functions play a crucial role in deep learning models and are responsible for introducing non-linearity in the computation. Non-linearity is essential in deep learning since it allows us to express models as a composition of many layers of computation, which can provide enough capacity to learn complex relationships in the data.

Sigmoid activation

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

The sigmoid (logistic) activation function maps the set of real numbers onto the unit interval: $x \in \mathbb{R} \mapsto \langle 0, 1 \rangle$. The smooth, monotonically increasing output in the $\langle 0, 1 \rangle$ range makes it especially suitable for modelling probability. The use of this activation function fell out of favour in deep neural networks, since its derivative has a range of $\langle 0, 0.25 \rangle$ (Figure 2.3). During backpropagation (2.2.4) in an N-layer deep network, the derivative of the activation function is multiplied N times. This can lead to vanishing gradients. Now it can typically only be seen in the output layer, while ReLU (or variations) are used elsewhere.



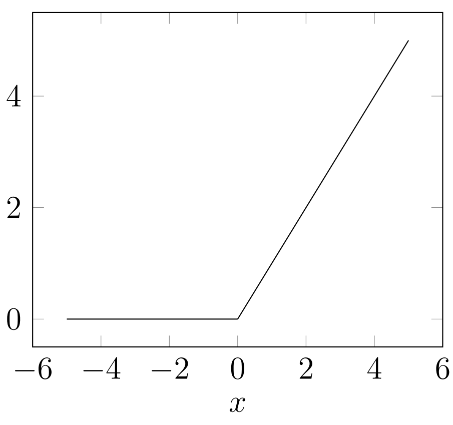
(a) f(x) **(b)** f'(x)

Figure 2.2: Plot of the sigmoid activation function. The graphs expose an important problem with the sigmoid activation. If the magnitude of the input is very large, the gradient through the neuron is very close to 0. We say that the neuron is *saturated*. This can significantly slow down training.

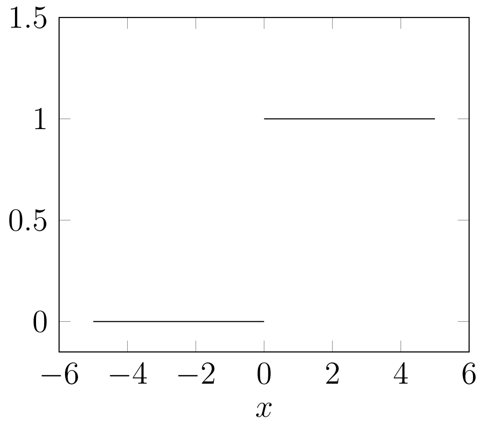
ReLU activation

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

The Rectified Linear Unit (ReLU) activation function maps: $x \in \mathbb{R} \mapsto [0, \infty\rangle$. Given that the derivative is equal to 1 for $\forall x > 0$, the activation function avoids the previously mentioned problem of vanishing gradients. The function is also much faster to compute than the sigmoid function. Due to these properties, ReLU (and its variants, e.g. Leaky ReLU) are the most widely used activation functions in modern architectures. However, the ReLU function can suffer from *dead units* (neurons that produce 0 for all input values, and stay that way due to gradients being 0). This is known as the dying ReLU problem, which is exacerbated by increasing depth [4]. Luckily, there are many variations of the ReLU function that avoid this issue, such as the Leaky ReLU, as well as smart initialization strategies and the use of Batch Normalization (2.1.4).



(a) f(x)



(b) f'(x)

Figure 2.3: Plot of the ReLU activation function and its derivative. The derivative is 0 for all x below 0. This means that during backpropagation (2.2.4), gradients for that specific neuron and example are 0. If this happens for all training examples, the neuron in question will never have a chance to change, i.e. it can be considered dead.

Softmax activation

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

The softmax activation function is a generalization of the sigmoid to multiple dimensions. As the name might suggest, the function serves as a smooth approximation of the argmax function. The output of the softmax is a probability distribution. This is especially useful in classification scenarios. The softmax is often combined with the cross-entropy loss (2.2.1) and one-hot targets to train a classification model.

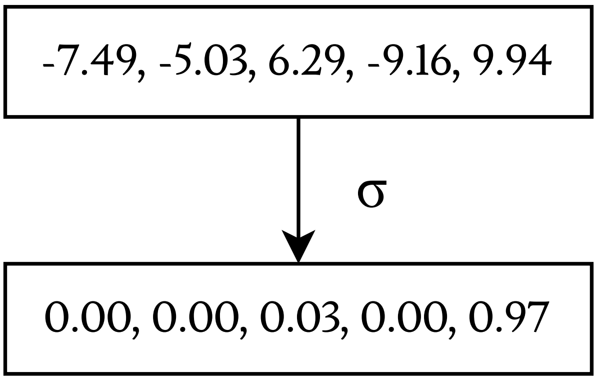


Figure 2.4: Softmax on a 5-element vector. The output sums to 1 and can be looked at as a probability distribution. Due to its exponential nature, the function is quite extreme, as evidenced by the fact that in this example only two values have an output larger than 1%.

2.1.3. Convolutional Neural Networks

Convolutional Neural Networks (CNNs), much like MLPs (2.1.1), contain neurons with weights and biases. The computation of those neurons is again modeled by the dot product. However, there are four major ideas that differentiate CNNs from MLPs.

Local receptive fields

Each neuron is only connected to a small local area of the previous input (Figure 2.5). This is known as the local receptive field of the neuron. The size of the local receptive field is determined by the size of the kernel (filter). A filter in this context is an $N \times N$ matrix of weights. Sizes for filters typically range from 7x7 to 3x3.

Spatial awareness

In traditional MLPs a multidimensional input is "flattened" into a 1d vector. This ignores the inherent topology of multidimensional data. In the context of images, this causes the model to treat spatially distant pixels the same as those that are close together. On the other hand, due to the arrangement of the local receptive fields — usually square in shape — CNNs keep the spatial structure of the data.

Weight sharing

CNNs reduce complexity further by sharing a set of weights across neurons in a layer. Due to weights being shared, the resulting operation looks like a sliding dot product of the learned filter across the input image. Applying a filter across an input **x** creates a *feature map* (also called a *channel*). CNNs often have multiple filters making up a single layer (and by extension, multiple feature maps). Feature maps make up the output of the CNN layer.

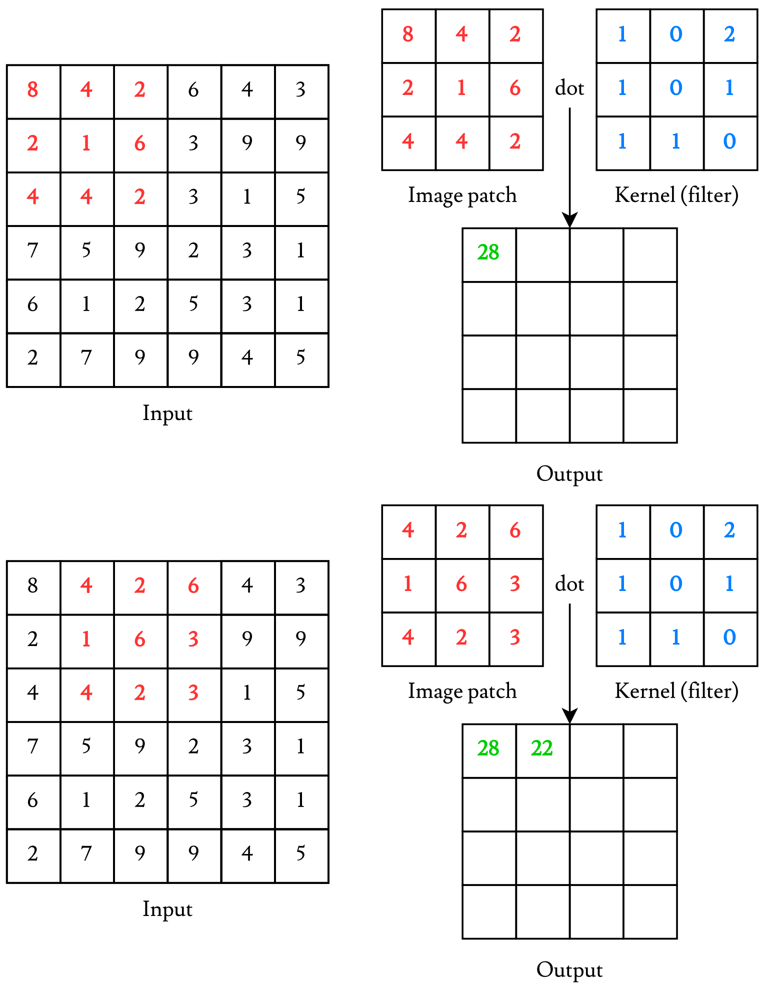


Figure 2.5: Local receptive fields of two neurons with a 3x3 filter. The first neuron is responsible for the top-left 3x3 input patch, while the second is shifted by one pixel to the right. Notice that the filter is the same in both situations, this is due to weight sharing (2.1.3). Also notice that the output is *smaller* than the input. Due to the size of the filter the pixels at the edges cannot occur in the center of a local receptive field. Thus, a convolution with a $k \times k$ kernel shrinks the representation by $\lfloor k/2 \rfloor$ at each edge. This can be avoided by padding convolution inputs, for example using 0-padding.

Pooling layers

Pooling layers are subsampling layers that typically reduce the resolution of a convolutional representation by pooling local regions. That is usually done to reduce the computational footprint of the model. The most common pooling operations are max-pooling and average pooling. A pooling layer applies a pooling function (max, avg) to an $N \times N$ pixel region of individual channels. Thus, it effectively reduces the size of the feature map by a factor of s on each spatial dimension, where s is the stride (distance between the centres of two regions).

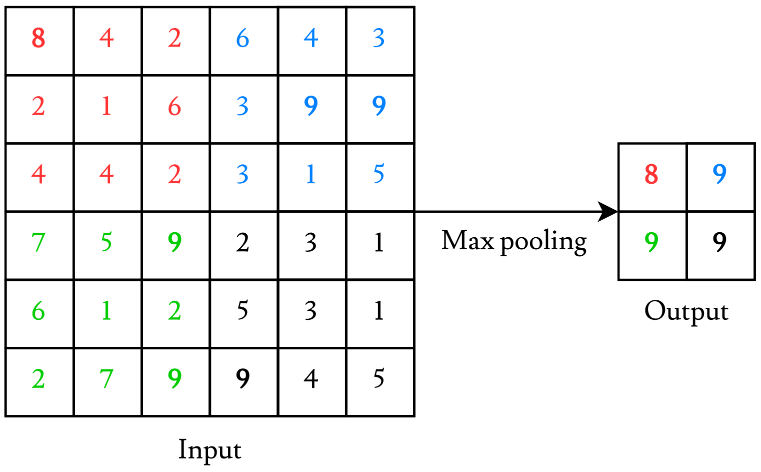


Figure 2.6: 3×3 Max-pooling operation on a single channel input with a stride $s = 3$. In effect, this reduces the spatial resolution by 3x along each dimension. It is worth considering how backprop (2.2.4) works through such an operation. In this case the gradients are passed only to the neurons that are responsible for the maximum value(s). In the rare case where there are multiple equal maximum values, the behaviour depends on the implementation.

2.1.4. Batch Normalization (BN)

Batch Normalization (BN) [5] is a block for normalizing the activations of a given layer, so that they have zero mean and unit variance. When dealing with the outputs of a CNN, this is done per-channel. The pixels of an individual channel are normalized by the same mean and variance[1]. This can be seen as a form of whitening. The initial motivation behind BN was to reduce internal covariate shift. Internal covariate shift denotes that the distribution of layer inputs varies ("shifts") during training. However, more recent research has shown that the benefits of BN aren't due to reducing internal covariate shift, but rather due to the loss landscape becoming smoother [6]. BN also has a useful effect when combined with ReLU activations, since it ensures (on average) that half of the signals will be positive and pass through the activation.

[1]This means that for an input of shape $(\mathrm{M}, \mathrm{W}, \mathrm{H}, \mathrm{C})$, BN will calculate C means and standard deviations (one for each channel), as opposed to calculating $\mathrm{W} \times \mathrm{H} \times \mathrm{C}$ (one for each scalar element). The latter is done when there is no weight sharing across the spatial dimensions.

For simplicity, the algorithm will be shown on scalar data. During training, Batch Normalization on a minibatch (a group of training examples) $y_i = \mathrm{BN}_{\gamma,\beta}(x_i)$[2] is done as follows:

1. Calculate the mean and variance from the minibatch:

$$\mu_B = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \sigma_B^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_B)^2$$

2. Normalize using the computed values:

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

where ϵ is a small number to avoid numerical instability.

3. Scale and shift the normalized activations by learned parameters γ and β:

$$y_i = \gamma\hat{x_i} + \beta.$$

During inference, instead of computing the μ and σ^2 from the batch, an exponential running average from training time is used. This is done to keep results deterministic and to allow use in low batch-size scenarios (such as real time inference). The running averages are updated during training as follows:

$$\mu = \alpha\mu + (1-\alpha)\frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\sigma = \alpha\sigma + (1-\alpha)\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

where α is a hyperparameter between 0 and 1.

2.1.5. Residual Connections

Residual connections were introduced in the ResNet paper [7] in 2015 and are still widely used in modern architectures. A typical residual unit for small models (also called a ResNet block) consists of two ReLU-activated layers of computation (typically convolutional layers + BN), with an additive residual connection.

[2]Here i denotes the index of the element within a minibatch.

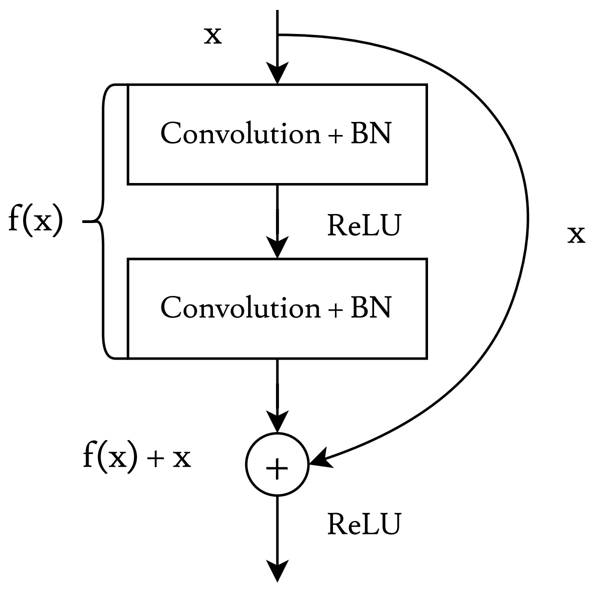


Figure 2.7: A typical residual unit for small models. If the input and output have differing dimensions, the identity residual connection can be replaced with a linear projection matrix W_s and suitable subsampling. In that case, the equation of the computed function becomes: $\hat{\mathbf{y}} = f(\mathbf{x}) + W_s\mathbf{x}$

The purpose of the residual connection is to reframe the problem as learning the residual function of f rather than f itself. Due to the additive connection, the computed function becomes:

$$\hat{\mathbf{y}} = f(\mathbf{x}) + \mathbf{x}.$$

Thus, the *target function* for $f(\mathbf{x})$ is actually the difference between the target $\mathbf{y}$ and the input $\mathbf{x}$. This makes the model learn the function that describes the change with respect to the input, rather than a function that directly maps the input to the desired output. The motivation for this formulation stems from the desire to make it easy for the model to learn the identity function. In the residual case, the identity function is achieved by simply using filters filled with zeros. Various factors (including L2 regularization, which pushes the weights down) make it extremely easy to learn weights with values 0. However, in the case without residual connections, the equivalent filters would be those with a 1 in the middle and 0 everywhere else. These filters are significantly harder — they're as hard to learn as any other filter. Due to this change, deeper models can easily (at the very least) emulate the behaviour of smaller models, by making the additional blocks model the identity function. In the original ResNet paper [7], He et al. have measured no training loss degradation even with a 1202 layer model (validation loss was however higher due to overfitting). On the other hand, without residual connections degradation could be seen with models as shallow as 34 layers.

2.2. Optimization

2.2.1. Loss Functions

Loss functions are functions that take in a prediction y_p and a ground truth y_t and output a single scalar value that reflects the mistake of the prediction. The goal of the training process is to minimize the loss. This is typically done through gradient descent (2.2.2). The loss function is often augmented with auxiliary objectives, such as regularization (2.2.5). There are many different loss functions that can be used for different tasks, however the most common are Mean Squared Error (MSE) and Cross-Entropy.

Mean Squared Error (MSE)

Mean Squared Error is a loss function that measures the average squared error between the prediction and the target. It is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where y_i is the label of the i-th example in the minibatch and $\hat{y}_i$ is the model's prediction.

Cross-Entropy (CE)

Cross-entropy loss is a loss function that is widely used in classification settings. Minimizing CE will in turn minimize the Kullback–Leibler (KL) divergence between the output distribution and the target distribution. In other words, it will minimize the 'distance' between the two distributions. The cross-entropy loss is defined as:

$$\text{CE} = \sum_{i=1}^{n}\sum_{j=1}^{|S|} y_{ij}\log(\hat{y}_{ij})$$

where y_{ij} is the j-th value of the one-hot encoded label of the i-th example in the minibatch and $\hat{y}_{ij}$ is the model's prediction for that label. In a semantic segmentation context this is calculated on a per-pixel level, and the per-pixel losses are then (usually) averaged to give the final loss.

2.2.2. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is the most common optimization algorithm for training deep neural networks. SGD is an iterative algorithm that performs the following update to the parameters in each training iteration:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t, x).$$

θ are the model parameters, η is the learning rate and $L(\theta, x)$ is the loss function. The magnitude of the learning rate coefficient effectively controls the size of the steps the weights make in the parameter space. The above updates are typically made after processing a single minibatch. The main advantage of SGD compared to other optimization algorithms is that it can be implemented with little memory, since only the gradient of the current minibatch needs to be stored.

Momentum

Momentum is a common modification of SGD that is analogous to momentum in physics, where the weight vector is thought of as a particle travelling through parameter space with a certain speed **v**. Momentum takes into account the direction and magnitude of prior gradients in order to reduce oscillations and speed-up learning. In theory, momentum can also help avoid getting stuck in local minima, but these occur very rarely in deep models due to their large number of parameters. The updates to the parameters become:

$$v_{t+1} = \gamma v_t - \eta \nabla_\theta L(\theta_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

where γ is the momentum coefficient (typically around 0.9). In effect, momentum adds an exponential moving average of the previous updates to the current weight update.

2.2.3. Adam

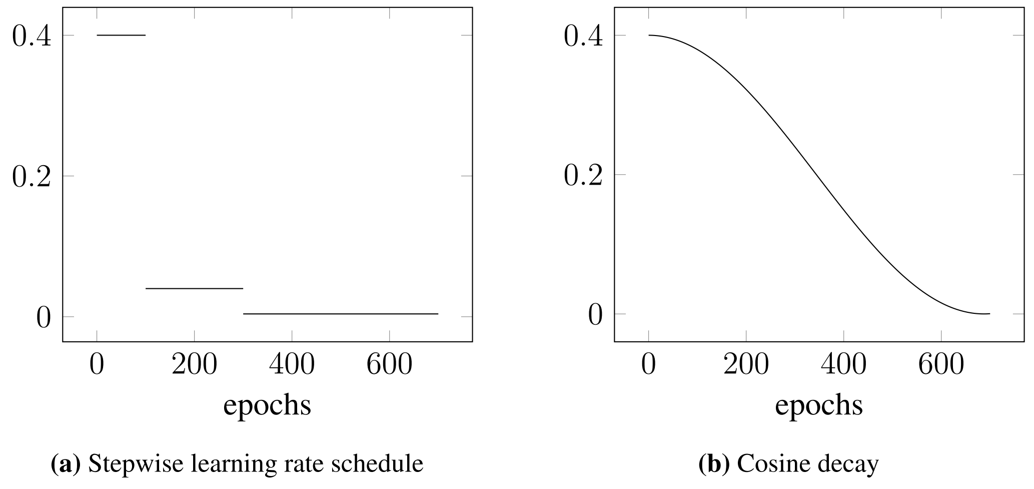
Adam [8] is an optimization algorithm that combines the benefits of momentum and dynamic scaling of the learning rate. It is one of the most popular optimization algorithms for training deep neural networks. Adam maintains two moving averages of the gradient, one for the first moment (mean) and one for the second moment (uncentered variance). They are used to dynamically scale the learning rate. The idea is that if the mean is large compared to the variance, we can be confident that the gradient is pointing in the right direction. The updates to the parameters become:

$$
\begin{aligned}
m_{t+1} &= \beta_1 m_t + (1-\beta_1)\nabla_\theta L(\theta_t) \\
v_{t+1} &= \beta_2 v_t + (1-\beta_2)\nabla_\theta L(\theta_t)^2 \\
\theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{v_{t+1}}+\epsilon} m_{t+1}
\end{aligned}
$$

where m_t and v_t are exponential moving averages of the first and second moments of the gradient. The first moment is simply the gradient, and the second moment is the gradient squared. The coefficients β_1 and β_2 are typically set to 0.9 and 0.999 respectively. The term ϵ is used in the denominator to avoid numerical instability when the denominator is close to zero.

Learning Rate Scheduling

Learning rate scheduling is a strategy for gradually reducing the learning rate during training. The idea behind this is that, as training progresses and the loss plateaus, the model will need to make smaller steps in order to reach a better point in the parameter space. The simplest approach is the *step learning rate schedule*, where the learning rate is decreased by a constant factor (commonly 10x) at regular intervals. Another popular strategy is the *cosine learning rate schedule*, where the learning rate is decreased according to a cosine function [9].



(a) Stepwise learning rate schedule **(b)** Cosine decay

Figure 2.8: Learning rate schedules. Since the parameters of the cosine schedule have to be determined ahead of time, the stepwise learning rate schedule is preferred in instances where the loss is unpredictable or irregular between training runs. Reinforcement learning is one such context.

2.2.4. Backpropagation

Backpropagation is an algorithm for computing the gradient of the loss function with respect to the parameters. In effect, it is an algorithm for backtracking the gradient through a computational graph. The basic idea is that the gradient is initially only known with respect to the loss function, and thus needs to be backtracked through the computational graph to the parameters of interest. Given that deep models are a composition of functions, backpropagation can be seen as an application of the chain rule of differentiation.

Chain rule

If we have a function $f(g(x))$ and we want to know the derivative of f with respect to x, we can use the chain rule to get the following result:

$$\frac{df(g(x))}{dx} = \frac{df(g(x))}{dg(x)}\frac{dg(x)}{dx}$$

where df/dg is the derivative of f with respect to g and dg/dx is the derivative of g with respect to x.

Computing the gradient

The computational graph is a directed graph where nodes are operations and edges are tensors. The computational graph allows for automatic computation of the gradient of the loss with respect to the parameters of the model [10]. This is done by tracing the gradient back through the edges of the graph towards the leaf nodes. The gradient of the loss with respect to a node can be computed by simply applying the chain rule of differentiation to the outgoing edges of the node (Figure 2.9).

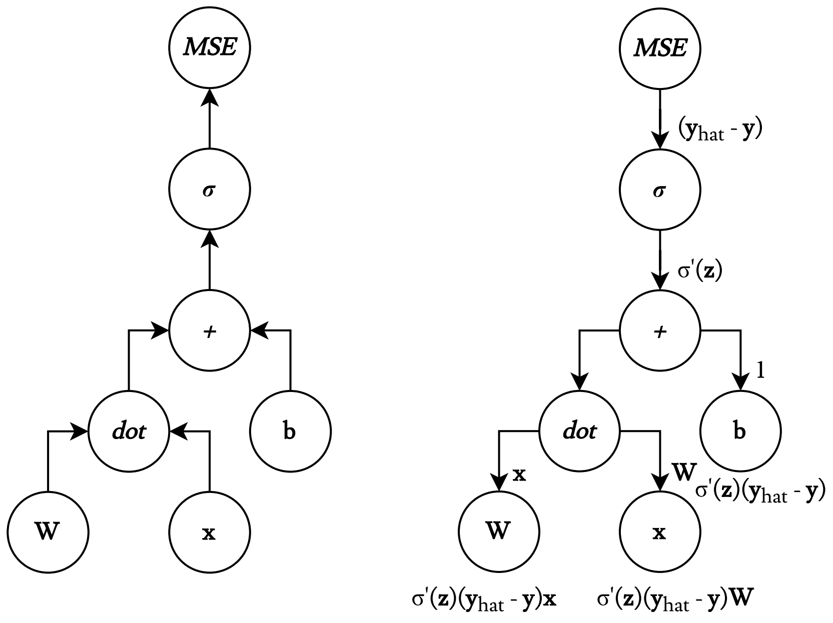


Figure 2.9: Computational graphs of the forward and backward pass of an MLP layer. $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$. Backward operations are used to compute the gradient of the loss with respect to the parameters. The gradients of any part of the process (even the input) can be determined by walking through the backward pass graph and applying the chain rule.

2.2.5. Regularization

Regularization is a set of techniques that may alleviate overfitting, i.e. training a model too closely to the training dataset. These techniques strive to minimize the generalization gap of the model. The generalization gap is the difference between the training error and the validation error. Overfitting occurs when the generalization gap is large. An overfitted model memorizes the training data and is not able to generalize to new data. This is a common problem with deep neural networks, since they tend to overfit very easily [11].

Weight decay

Weight decay is a regularization technique whereby the loss function is augmented with the sum of the squares of the weights:

$$L' = L + \lambda \sum_{i=1}^{n} w_i^2.$$

Here, L' denotes the new loss function, L is the original loss function and λ is the weight decay coefficient. The effect of this addition is to penalize models with large weights. Hence, the goal is to find a hypothesis with smaller weights (and therefore a smaller complexity), while still minimizing the loss.

Dropout

Dropout [12] is a regularization technique that randomly drops out (i.e. sets to 0) neurons during training. The amount of regularization is controlled by the dropout probability p. This technique forces the model to learn to classify a given example through multiple mechanisms, since at any given moment in training some neurons are not available. In other words, the model needs to learn to classify without the use of some of its features. This reduces the interdependence of individual neurons. Due to the fact that on average only $1 - p$ of the neurons are non-zero during training, dot products (i.e. outputs of the layers) are going to be smaller by a factor of $1 - p$ than during inference. To combat this, the outputs are scaled by $1/(1 - p)$ during training.

Data augmentation

Data augmentation is a technique that takes a single image and produces new augmented versions from the original. These images are modified in such a way that they are still recognisable as the original image, but they are not exact copies. Typical augmentations are cropping, rotation, translation, colour shifting, etc. This reduces overfitting by both artificially increasing the size of our dataset and by making the individual examples harder to learn. Data augmentation is especially prevalent in low-data scenarios.

2.3. Semantic Segmentation

Semantic segmentation is a task that aims to predict the label of every pixel in an image. The model needs to be able to categorize every pixel in an image into one of the classes in the training dataset (Figure 2.10). The most common metric used to evaluate the performance of semantic segmentation models is Intersection over Union (IoU). IoU is defined as:

$$\text{IoU} = \frac{\text{intersection}}{\text{union}} = \frac{|Y \cap \hat{Y}|}{|Y \cup \hat{Y}|}.$$

Y is the set of pixels contained in the ground truth segmentation and $\hat{Y}$ is the set of pixels contained in the predicted segmentation. Since IoU is calculated on a per-class basis, a mean aggregate score is commonly used – mIoU (mean IoU).

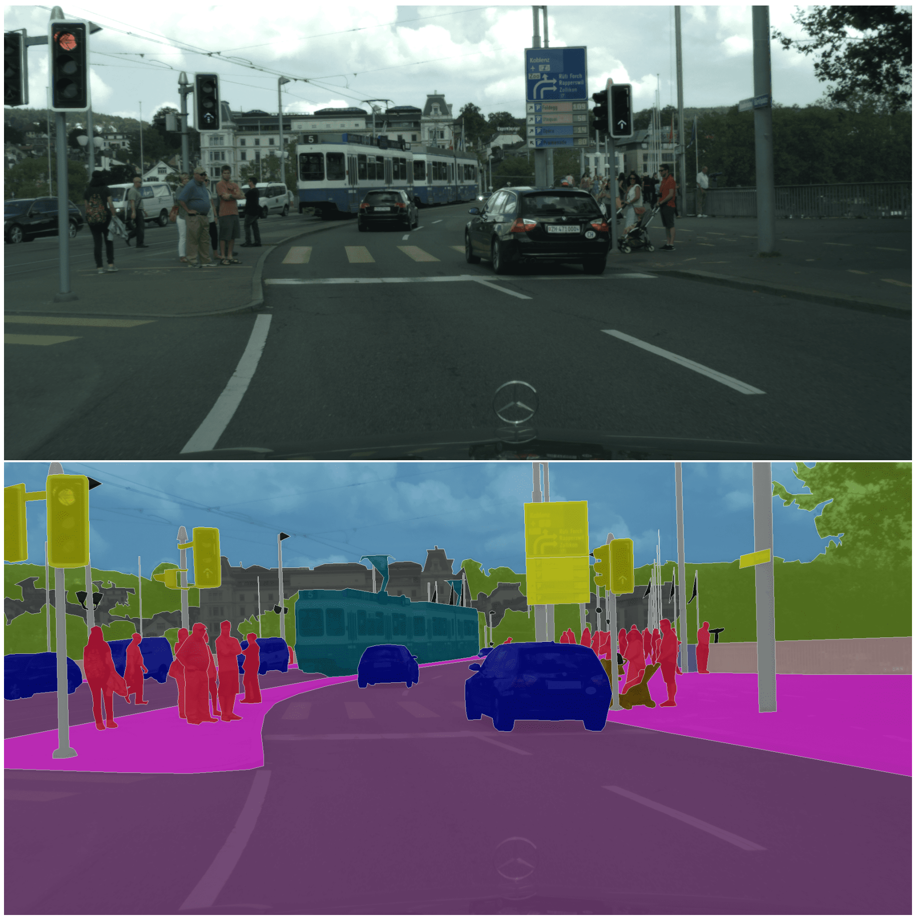


Figure 2.10: Semantic segmentation example. The top image is the input image, the bottom image is the ground truth segmentation. Each colour represents a separate class. A segmentation model doesn't directly produce an output shown on the bottom. Instead the problem is often modeled as a pixel-wise classification problem, where the output is of the shape $(C \times W \times H)$, with C being the number of classes. The images were taken from the Cityscapes dataset [1].

2.4. Model Architecture

2.4.1. SwiftNet

SwiftNet [13] is a model for efficient real-time semantic segmentation. It uses a fully convolutional asymmetric encoder-decoder architecture (Figure 2.13) with a ResNet-18 [7] backbone pretrained on ImageNet [14]. The encoders are connected to the decoders with skip connections and the model employs SPP (2.4.1) to increase the effective local receptive field. This thesis uses the single-scale variant in all experiments (also known as SwiftNet-RN18). The model consists of three basic blocks: the recognition encoder, the upsampling decoder and the SPP block.

Recognition encoder

The recognition encoder consists of 4 encoder blocks with decreasing spatial resolutions. The first encoder block reduces the spatial resolution to $H/4 \times W/4$, while the subsequent blocks each reduce the resolution by a factor of 2. Each encoder block is comprised of a sequence of individual ResNet blocks (Figure 2.7), aside from the last block in the sequence where there is an added outgoing lateral skip connection that projects the computed features and pipes them to the corresponding decoder. When using the ResNet-18 backbone, the number of blocks per encoder block is 2.

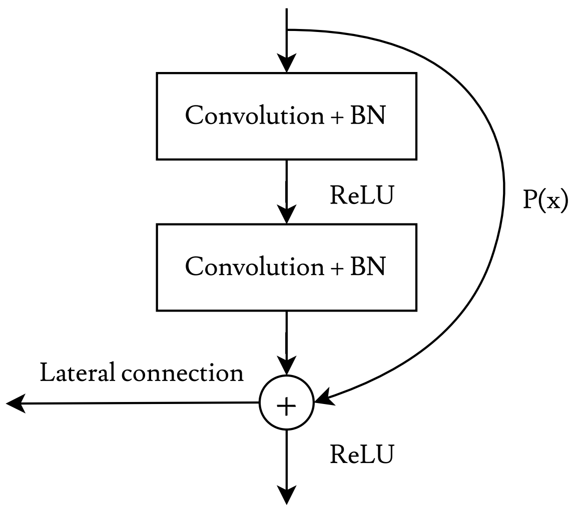


Figure 2.11: The last block in an encoder block with an outgoing lateral connection and a linearly projected residual connection to accommodate differing spatial sizes (due to subsampling). The lateral connection branches off before the activation.

Upsampling decoder

The upsampling decoder is tasked with upsampling the computed visual features back to full spatial resolution. Just like the encoder, the decoder consists of a sequence of individual decoder blocks. However, each block in the decoder block is roughly half the size of the usual ResNet block (only a single Conv-BN-ReLU layer instead of two). Additionally, there is only a single block per decoder block. The decoder blocks have two inputs, the encoder features from the corresponding lateral skip connection and the output of the preceding decoder block. The output of the decoder blocks is upsampled with bilinear interpolation to match spatial resolution of the lateral encoder blocks. It is then combined with the features from the lateral connection using element-wise summation, and subsequently processed by 3×3 convolution.

SPP block

The Spatial Pyramid Pooling [15] (SPP) module is used as an inexpensive way to increase the effective receptive field. This is done by pooling the input features over a range of coarse spatial grids, from 1×1 to 8×8. A difference worth noting is that the original SPP paper used max-pooling, while SwiftNet uses average pooling.[3]

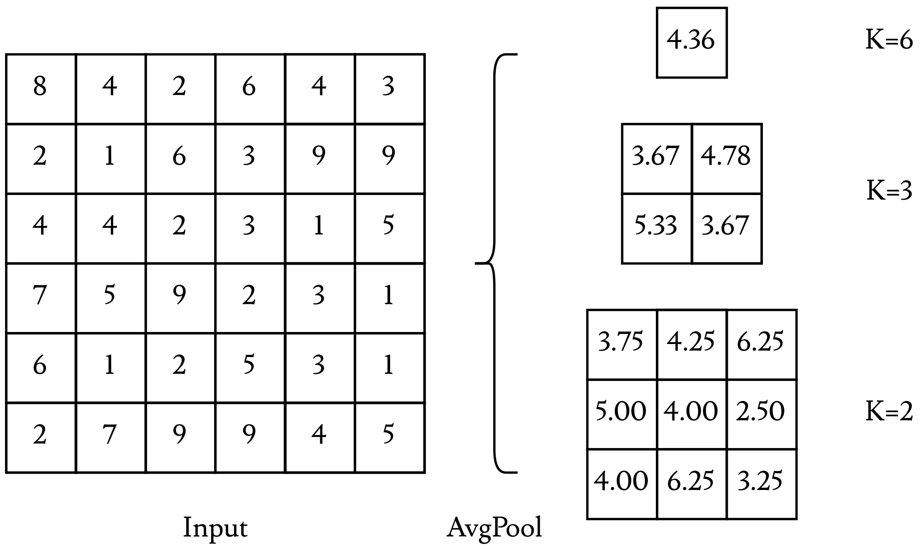


Figure 2.12: Spatial Pyramid Pooling with 3 levels of grids. The resulting pooled representations are then upsampled with nearest-neighbour back to original resolution, and concatenated with the input (channel-wise) to produce the resulting output.

[3]Additionally, SwiftNet transforms the pooled features by passing each map through individual BN-ReLU-Conv layers. The concatenated features (along with the input) are then fused together with an additional BN-ReLU-Conv layer.

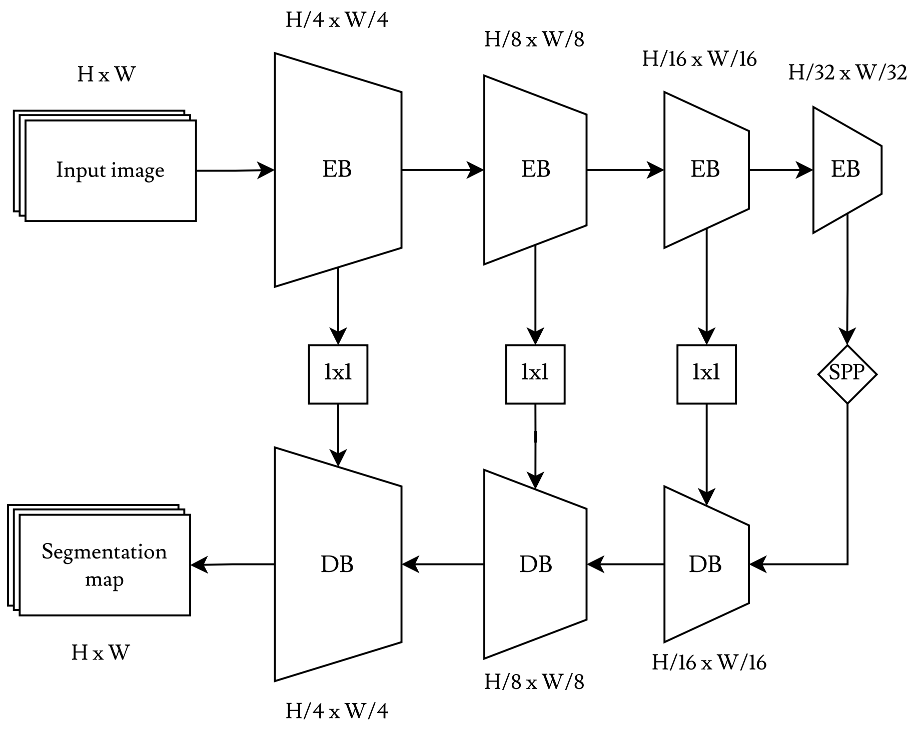


Figure 2.13: Block diagram of the single-scale SwiftNet architecture. EB designates the previously mentioned recognition encoder blocks, while DB designates the upsampling decoder blocks.

3. Datasets

Datasets are collections of data that are used to train deep learning models. A dataset consists of a set of inputs and a corresponding set of outputs. In a computer vision context, the inputs are images. Among other things, the outputs can be image labels or segmentation maps. This thesis focuses on two semantic segmentation datasets, Cityscapes and PASCAL VOC. However, the methods described are general and as such can also be applied to other semantic segmentation datasets as well.

3.1. Cityscapes

The Cityscapes dataset [1] is a dataset of dashcam footage of urban street scenes across many different German cities (plus Zürich and Strasbourg). The images from this dataset are rather distinct, having been taken with the same camera (from the same car), in favorable conditions. The dataset consists of 5027 images with a resolution of 2048x1024 pixels. The dataset contains ground truth data for several vision tasks including semantic segmentation. The train/test/validation split is done on a per-city basis, with no overlapping cities between the three sets. For the purposes of this thesis, the dataset was preprocessed in two ways. Due to computational constraints, the Cityscapes dataset was downsampled to quarter resolution (1024x512) using bilinear interpolation for the images, and nearest-neighbour for the segmentation maps. Additionally, to lower CPU load during decompression, the images were converted to the portable pixmap format (PPM).

Table 3.1: Cityscapes train/test/validation split.

Subset	**Subset size**
train	2993
test	1531
val	503

Figure 3.1: 4 images from the Cityscapes [1] dataset.

3.2. PASCAL VOC

The PASCAL VOC dataset [16] is a general purpose image dataset with a diverse set of classes. Since its inception in 2005 it has been extended multiple times with the last version being VOC2012, which is the version used in this thesis. The dataset consists of 20 classes, covering both indoor scenes (dining table, TV/monitor, ...) and outdoor scenes (car, train, ...). The ground truth includes per-pixel segmentation maps of 2913 individual images, excluding the test data. Unlike images from Cityscapes, PASCAL VOC images have varying sizes (generally smaller than Cityscapes) and varying aspect ratios.

[1]However, the test labels are not publicly available.

Table 3.2: PASCAL VOC train/test/validation split.

Subset	**Subset size**
train	1464
test	1456[1]
val	1449

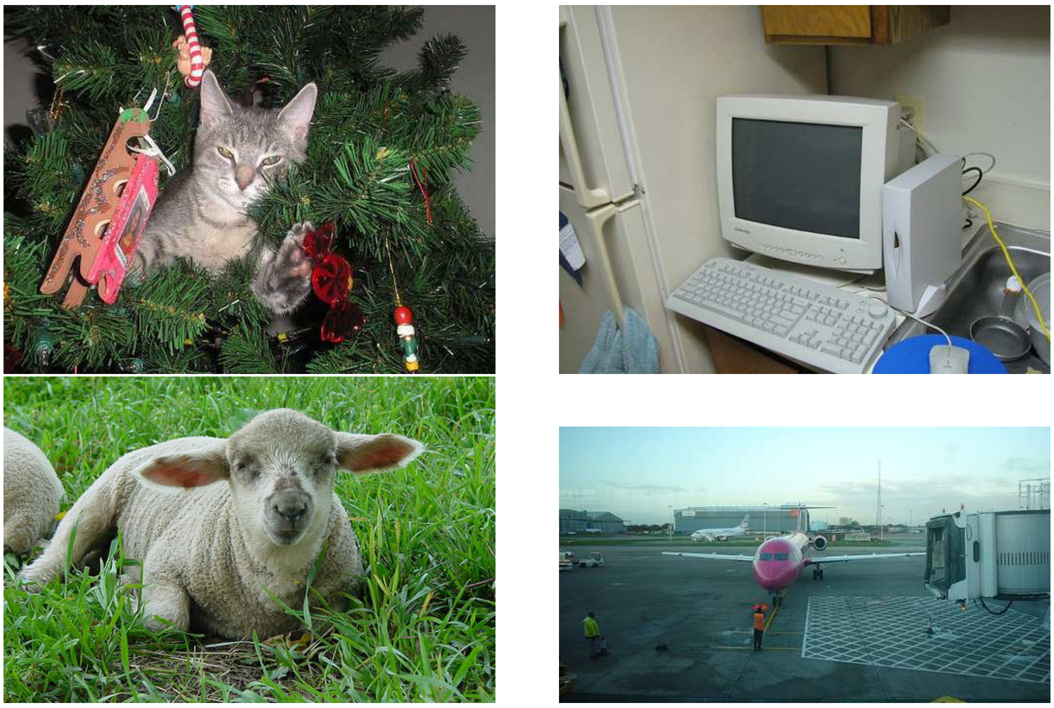


Figure 3.2: 4 images from the PASCAL VOC [16] dataset showcasing the large variety of scenes and image dimensions.

3.3. Augmentations

We use two[2] augmentations on the training set. The first augmentation is RandomFlip, which performs a horizontal flip with probability $p = 0.5$. The second augmentation used is ScaleAndCrop. The scale and crop operation operation resizes the image by a factor of r using bilinear interpolation (nearest-neighbour for the segmentation maps), with r being a random number generated from a uniform distribution between 0.7 and 1.3. The resulting image is randomly cropped with a crop size of $W \times H$. If the resized images are smaller than the crop size, square padding is added with the padding pixels

[2]Three if the image is not square.

being equal to the mean RGB colour of the image.

4. Semantic Segmentation On Multiple Datasets

4.1. Previous Work

Training on multiple datasets is an active area of research. Successful training requires adequate handling of differing label sets. Some aim to solve the problem by exploiting the embedding power of large language models [17]. This works by embedding an arbitrary set of labels and training the image encoder to align its pixel-wise outputs with the embedding of the target class for that pixel. However, the main drawback of such approaches is the fact that the embeddings rely solely on the symbolic representation of the words. An approach as outlined in [17] cannot discern between classes that are represented by the same word but correspond to a different visual concept. Synonyms are a good example of labels where this approach would struggle, as they have the same symbolic representation (and thus the same word embedding), but have different visual meaning. An alternative approach [18] uses a manually constructed *universal taxonomy* that models the sub/superset relationships between classes of differing datasets. The goal of this thesis is to find that ordering automatically.

4.2. The Concat Model

The Concat model is a simple baseline that produces a universal taxonomy by naively concatenating dataset-specific labels, without doing anything to resolve the potential overlap. The model produces predictions for the labels of the two datasets simultaneously, over a single softmax vector. In this thesis I will train such a model on a concat dataset containing the Cityscapes and PASCAL VOC datasets.

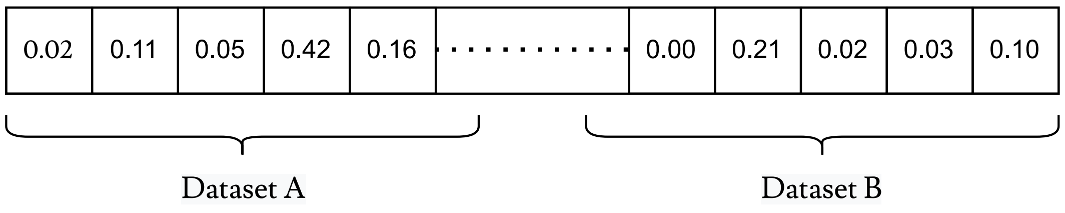


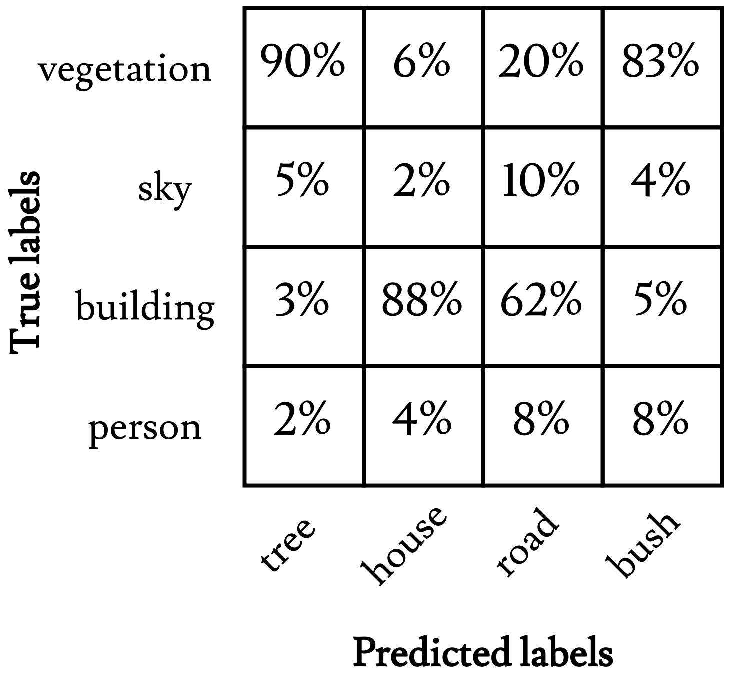
Figure 4.1: The output vector of a Concat model. The labels for dataset A and B populate the same vector and undergo a joint softmax operation. As we will see later, this can be detrimental if the two datasets share overlapping labels.

4.3. Universal Taxonomy

The procedure to create an universal taxonomy outlined in this thesis is based on confusion matrices of models cross-evaluated on opposing datasets. The models in question are standard semantic segmentation models, one for each dataset. Applying this taxonomy will allow us to minimise the unwanted effects caused by overlapping classes.

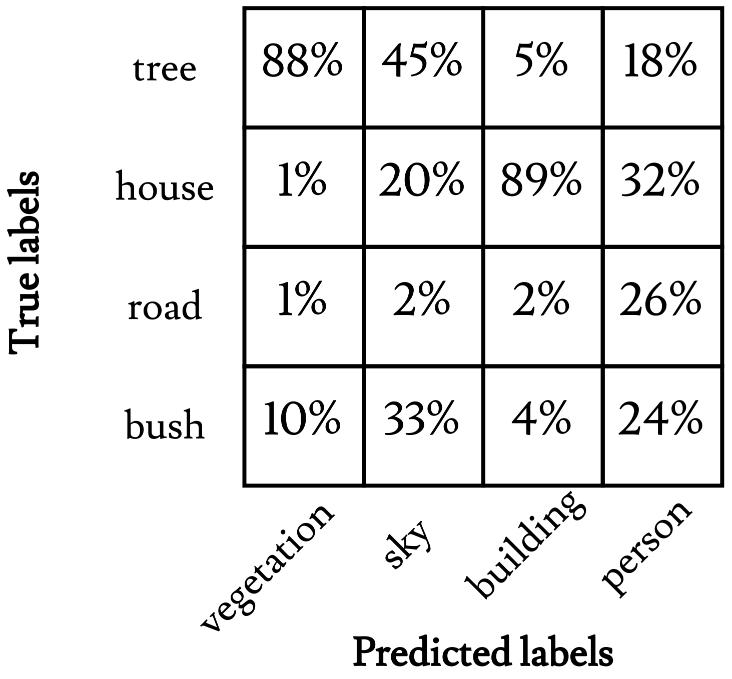
4.3.1. Cross-Dataset Confusion Matrices

To collect the confusion matrices we evaluate each model against the dataset it was not trained on. We count the number of times the model predicted $a \in A$ while the ground truth was $b \in B$, where A is the dataset the model was trained on, and B is the dataset it was not trained on. The gathered confusion matrices are then normalized column-wise. Normalizing column-wise makes the cell (B, A) reflect the proportion of occurrences when the model predicted A but the actual label was B.



True labels \ Predicted labels	tree	house	road	bush
vegetation	90%	6%	20%	83%
sky	5%	2%	10%	4%
building	3%	88%	62%	5%
person	2%	4%	8%	8%

Figure 4.2: Example of a column-wise normalized confusion map where the X axis is the original model's dataset (dataset A), and the Y axis is the target dataset (dataset B). This is seen 'from the perspective' of dataset A, and this perspective will be written as $A \rightarrow B$ from now on.



True labels \ Predicted labels	vegetation	sky	building	person
tree	88%	45%	5%	18%
house	1%	20%	89%	32%
road	1%	2%	2%	26%
bush	10%	33%	4%	24%

Figure 4.3: Just like in Figure 4.2, but from the perspective $B \rightarrow A$.

4.3.2. Finding Overlapping Labels

We can now analyse the two confusion matrices in order to discover relations between labels from the two datasets. The amount of overlap of two classes from opposing datasets can be seen in the magnitude of the two corresponding cell values. The magnitudes can be looked at as the strengths of the "connections" between the two labels. An asymmetric pair of connections indicates a sub/superset ordering, while connections of similar strengths might imply equality. For example, the connection bush $\rightarrow$ vegetation (Figure 4.2) is very strong. It shows that when the model predicted "A-bush", the actual label was "B-vegetation" 83% of the time. This observation indicates either a subset relationship (bush is a subset of vegetation, bush $\subset$ vegetation), or equivalence. This can be written down succinctly with: bush $\subseteq$ vegetation. However, when we observe the connection vegetation $\rightarrow$ bush (Figure 4.3), we get the hypotheses: vegetation $\supset$ bush or "no overlap" due to the weak connection. The intersection of the two observations is vegetation $\supset$ bush, so we conclude that vegetation is a superset of bush.

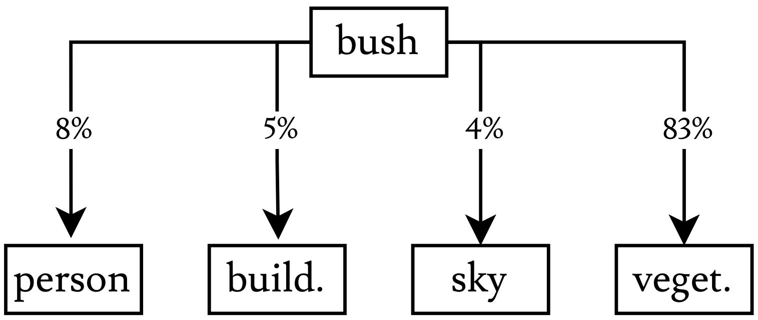


Figure 4.4: Decomposition of predictions into class bush. A strong connection (incidence in pixels labeled as vegetation) can indicate equivalence. If the labels are perfectly equivalent, the connection would be 100%. It can also indicate bush $\subset$ vegetation, because the model recognises learned features inherent to bush instances in areas labeled as vegetation.

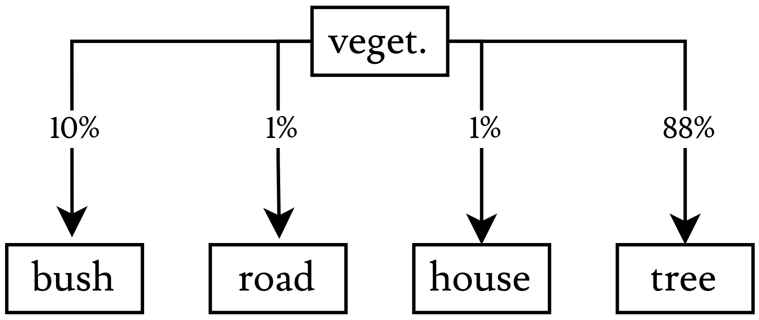


Figure 4.5: Decomposition of predictions into class vegetation. The weak connection towards the class bush suggests that bush and vegetation are not equivalent, and that they might even have no meaningful overlap. However, the option vegetation $\supset$ bush is still possible. To understand why, consider a dataset $\mathbb{X}$ with many kinds of vegetation and a perfectly uniform label distribution. In that case, assuming perfect models, the connections $(\text{vegetation} \rightarrow \text{x}, \ \ (x \in \mathbb{X}))$ will be equal to $1/|\mathbb{X}|$. If $\mathbb{X}$ is large, the connection could be very small. However the reverse connections (x $\rightarrow$ vegetation) are all expected to be very large (i.e. close to 1).

4.3.3. Constructing The Taxonomy

Constructing the taxonomy is an iterative process. We have to keep in mind that the labels of a single dataset are mutually disjunct. For example, a single label (A-a) cannot be both a superset of some label (B-b) and a subset of some other label (B-c). Such a relationship would imply (B-c $\supset$ A-a $\supset$ B-b), which would mean that some labels in dataset B are not mutually disjunct. As a consequence of this, a label can only have multiple overlaps if it is a superset of multiple labels. Any other configuration leads to a contradiction with the assumption that labels (within the same dataset) are mutually disjunct. With this in mind, we can define an algorithm for constructing the universal taxonomy as follows:

1. Initialize an empty taxonomy T and remove hypotheses with a low total number of incidences [1].
2. Find the most overlapping pair of labels (a, b) from datasets A and B respectively. This can be determined by a scoring function that takes in the strengths of the connections and returns a single value, for example $f(x, y) = x + y$ or $f(x, y) = xy$.

[1]In the confusion matrix $A \rightarrow B$: if the dataset the model is trained on (A) has a label that is disjunct with every label in the target dataset (B), the total number of classifications as that label will be very low and subject to noise. The algorithm eliminates such connections because they are overwhelmingly noise and can lead to spurious conclusions.

3. Determine which relationships are feasible given the current taxonomy, and determine the ordering of the two labels. Add the result to the taxonomy T.

4. Repeat until there are no more pairs of labels from A and B whose computed score is over some threshold ϵ.

5. Collect the leaf nodes of the constructed taxonomy tree(s) into a universal taxonomy set S.

The greedy best-first search can be replaced with smarter searching strategies if the size of the label sets allows. Additionally, we can perform regression testing of each hypothesis (against the concat model) by either training a new model using the constructed taxonomy (as in [18]), or by applying the taxonomy on top of the concat model.

4.3.4. Comparison with the Concat-only approach

The outlined taxonomy algorithm can be compared to one that uses confusion matrices given by the Concat model. The idea is similar, we evaluate the Concat model on two datasets, A and B, and while evaluating on dataset A we collect the corresponding confusion matrix of predictions of labels from dataset B, and vice versa. One possible advantage of such an approach would be better resilience to the problem of domain shift. This is because the model is trained on both datasets. However, the concat model suffers from the problem of overlapping labels which causes worse performance when evaluated on individual datasets (Table 6.1).

5. Implementation

The programming language used in this thesis is Python 3. It is designed for readability and has many supporting libraries for data processing and visualisation. It is also an interpreted language, which makes it very easy to perform experiments in an interactive manner.

5.1. Supporting Libraries

Our implementation is based on PyTorch [19]. PyTorch has a growing set of built-in modules for standard tasks in deep learning, and is thus easy to work with. The transparent use of GPU hardware makes training very efficient, and its built-in autograd feature allows for easy construction of custom models. NumPy [20] is used to save and manipulate the confusion matrices. The confusion matrices are visualised with seaborn [21], which is a statistical data visualisation library built on top of Matplotlib [22].

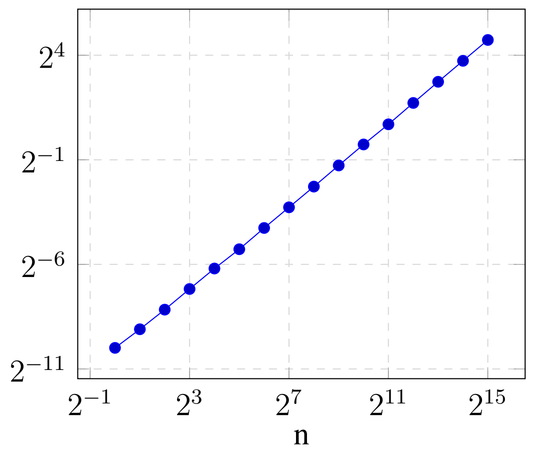
5.2. Models

All experiments use the aforementioned SwiftNet-RN18 architecture [13]. Since this work was built on top of the official implementation[1] of the model, it also follows the same learning procedure. We optimise our models with Adam, with a starting learning rate of $\eta = 4 \times 10^{-4}$ and a batch size of 14. The learning rate decays on a cosine annealing schedule down to a minimum of 1×10^{-6}. We use weight decay, with a constant hyperparameter $\lambda = 1 \times 10^{-4}$. The models were trained for 250 epochs. The augmentations are described in Chapter 3. As mentioned before, the Cityscapes model is trained on quarter resolution due to computational constraints. The crop size is 448×448.

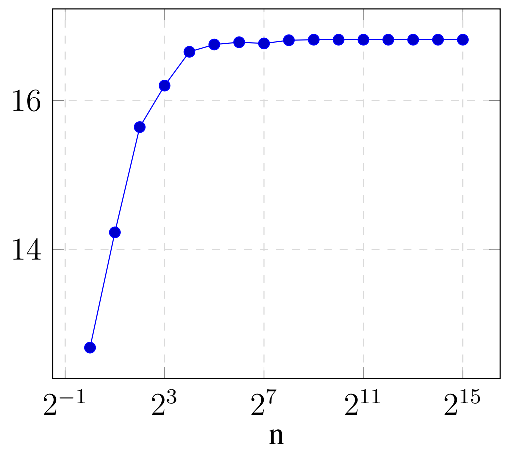
[1]`https://github.com/orsic/swiftnet`

5.3. Taxonomy Algorithm

The taxonomy algorithm follows the same structure as described in 4.3.3 with minor changes to step 2. The scoring function is a simple sum: $f(p_a, p_b) = p_a + p_b$, and the relationship between the two labels is determined by comparing the absolute difference of the connections to a fixed limit δ. If $|x - y| < \delta$, we conclude that the labels are equivalent, otherwise the source of the connection with the larger magnitude serves as the subset (i.e. if p_a is from the perspective a $\rightarrow$ b, then $p_a > p_b \implies a \subset b$). The hyperparameter values used are $\epsilon = 0.33$ (lower limit of the score of the pairs being considered), $\delta = 0.5$ and min$_n$ $= 5000$ (the minimum total number of incidences). These hyperparameters were chosen manually. Future work could try to train them, however the correct goal function to use in the training process remains unclear. Step 2 is modified by replacing greedy search with a random sampling strategy. The whole process is repeated n times. The taxonomy with the largest sum of the scores is picked as the final taxonomy. This provides better results than greedy search for a small performance tradeoff. In the Cityscapes/PASCAL context, the algorithm appears to settle after around $n = 5000$ simulations.



(a) Execution time (in seconds) with respect to n.



(b) Computed taxonomy score with respect to n.

Figure 5.1: Score and time plots of the taxonomy algorithm.

6. Experimental Results

6.1. Baseline Models

The baseline quarter-resolution Cityscapes model achieved a mIoU of 70.26% on epoch 216. The PASCAL VOC model achieved a mIoU of 59.86% on epoch 249. The poor performance of the PASCAL model can be explained by two factors. The first is that the model was only trained on the training set. It is now common practice to also train the model on the augmentation set, which contains an additional 10582 images (example [23]). The second factor is the use of a relatively weak backbone, ResNet18. Better results could be achieved with the larger ResNet34. The naive Cityscapes-PASCAL concat model achieved a mIoU of 62.09%.

Model	**Cityscapes**	**PASCAL VOC**	**Concat**
PASCAL VOC model	-	59.86	-
Cityscapes model	70.26	-	-
Concat model	65.84	58.69	62.09

Table 6.1: Comparison of the three baseline models. Here we see a significant degradation in performance of the concat model. This is most likely due to insufficient model capacity and overlapping classes.

6.2. The Constructed Taxonomy

We apply the algorithm presented in 5.3 using our two single-dataset baseline models. Table 6.2 shows the resulting relationships. The last column shows the score given by the used scoring function (sum). From Table 6.2 we can construct a universal taxonomy set:

$$\begin{aligned}\mathcal{S} = \{&\mathrm{vegetation}, \mathrm{l}_1, \mathrm{pole}, \mathrm{fence}, \mathrm{car}, \mathrm{trafficlight}, \mathrm{bicycle}, \mathrm{aeroplane}, \mathrm{motorcycle},\\ &\mathrm{building}, \mathrm{horse}, \mathrm{trafficsign}, \mathrm{bus}, \mathrm{terrain}, \mathrm{l}_2, \mathrm{sidewalk}, \mathrm{train}, \mathrm{wall}, \mathrm{sky}\}\end{aligned}$$

where $l_1 = \mathrm{person/rider}$ and $l_2 = \mathrm{road/dining\_table}$. There are multiple approaches in the case where a label has only as single subset (in our case person-horse). Here we always take the leaf node, but there is an argument for treating such relationships as equivalence. If we allow the use of the set difference operation, the most accurate way would be to model it as $\{\mathrm{horse}, \mathrm{person}\backslash\mathrm{horse}\}$.

PASCAL VOC	**Cityscapes**	**Relationship**	**Score**
background	vegetation	super	0.96
person	rider	equal	0.5
background	pole	super	0.86
background	fence	super	0.82
car	car	sub	1.18
background	traffic light	super	0.54
bicycle	bicycle	equal	0.65
aeroplane	car	sub	0.72
motorbike	motorcycle	equal	0.69
background	building	super	0.97
horse	person	sub	1.02
background	traffic sign	super	0.62
bus	bus	equal	0.56
background	terrain	super	0.93
diningtable	road	equal	0.43
background	sidewalk	super	1.01
train	train	equal	0.56
background	wall	super	0.72
background	sky	super	1.03

Table 6.2: The generated Universal taxonomy tree represented as a table with the corresponding scores. Most relationships are sensible, with the notable exception of (horse $\subset$ person). Considering the Cityscapes dataset contains no animals and the PASCAL model is bad as-is, some bad inferences are to be expected.

6.3. Applying The Universal Taxonomy

The universal taxonomy can be applied to the concat model at inference time by merging relevant logits. In the case of a sub/superset relationship, we add the logits belonging to the less general label to the more general one. If the relationship is equivalence, we merge the labels into one. This should in theory improve the performance of the concat model, since we'd be fixing the situations where the overlapping labels are starving eachother and causing an incorrect label to win. However, since the concat model is very good at distinguishing between the two datasets (Figure 6.1), the expected improvement is marginal. Indeed, if we apply the taxonomy to our concat model we can observe a mIoU of 62.24%, which is only slightly better than our previous mIoU of 62.09%. As mentioned before, an alternative to this is learning a new model on the taxonomy as outlined in [18].

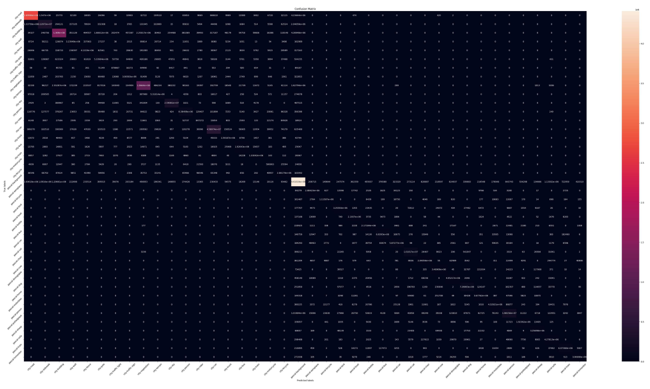


Figure 6.1: The confusion matrix of the concat model evaluated on both datasets. Small values in the first and third quadrants indicate that the model is very good at discerning between the two datasets.

7. Conclusion

This thesis proposed a method to build a universal taxonomy of dataset labels that can be applied in the context of multidataset learning. The taxonomy is constructed using a pair of confusion matrices that are generated by training two segmentation models on two separate datasets and testing each on the opposing dataset. This taxonomy is then used to establish an ordering between overlapping labels of the two datasets. We evaluate our method by augmenting the predictions of a Concat model trained on the Cityscapes and PASCAL VOC datasets. The proposed method slightly outperforms the baseline and shows a promising approach for training on multiple datasets.

Due to computational constraints, we used the ResNet18 variant of the single-scale SwiftNet model. This, along with not training on the special augmentation set, caused rather poor baseline PASCAL VOC performance. The concat model (also SwiftNet-RN18) was unexpectedly good at discerning between the two datasets (Figure 6.1). In hindsight this makes sense, since the Cityscapes dataset is very homogeneous with the camera always in the same position[1]. This limits the possible effects of applying the taxonomy during inference, since the expected contention between the labels of the two datasets is minimal. Additionally, since the baseline Cityscapes and PASCAL VOC models are used to create the taxonomy in the first place, the strength of the models directly affects the quality of the resulting taxonomy.

There are a number of ways in which this work can be extended. The simplest direction is to apply the approach using more powerful backbones, and potentially with more similar datasets (for example Cityscapes and Vistas [24]). Another approach with a lot of potential is to use the taxonomy to guide the training process, by training a model to output labels from the universal taxonomy rather than dataset-specific labels. This can be done by learning on partial labels as shown in [18]. Furthermore, it might be possible to also train the δ and ϵ taxonomy algorithm parameters[2]. Another

[1]Furthermore, the images were taken with the same camera model.

[2]The correct goal function to use in the training process is unclear. One alternative would be to perform a grid search. However, this would be quite expensive.

possibility is to extend the process to involve more than two datasets, which would require a more refined way of building the taxonomy — the current approach would have the number of confusion matrices grow exponentially. The "holy grail" of this line of research would be to find out a way to train the taxonomy and the model simultaneously. One approach[3] is to train a model with Q universal logits by attaching dataset-specific heads. The heads would have binary weights that transform the universal maps into dataset-specific maps, with the restriction that one universal logit cannot be used in multiple logits of the same dataset. The main problem of this idea is how to train the model, since binary weights cannot be trained with SGD. One potential approach is to binarize the weights in the forward pass, but keep them in floating point format for backpropagation (and constrain them to the unit interval). This is analogous to quantisation aware training (QAT). There is also the potential to use the taxonomy to improve the performance of other types of models such as object detectors or image classifiers.

[3]Proposed by Siniša Šegvić and Ivan Grubišić.

A. Confusion Matrices

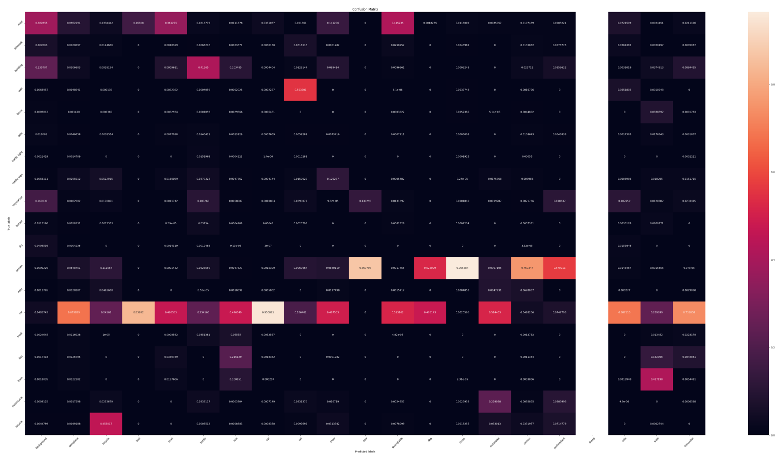


Figure A.1: Column-wise normalised confusion matrix Pascal->Cityscapes.

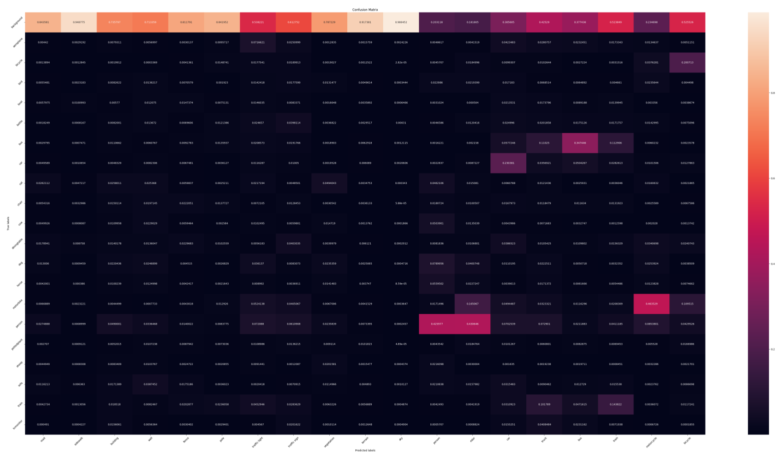


Figure A.2: Column-wise normalised confusion matrix Cityscapes->Pascal.

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," pp. 3213–3223, 2016.

[2] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, June 2010.

[3] T. van Laarhoven, "L2 Regularization versus Batch and Weight Normalization," Tech. Rep. arXiv:1706.05350, arXiv, June 2017. arXiv:1706.05350 [cs, stat] type: article.

[4] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, "Dying ReLU and Initialization: Theory and Numerical Examples," *Communications in Computational Physics*, vol. 28, pp. 1671–1706, June 2020. arXiv:1903.06733 [cs, math, stat].

[5] 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*, pp. 448–456, PMLR, June 2015. ISSN: 1938-7228.

[6] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How Does Batch Normalization Help Optimization?," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada* (S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 2488–2498, 2018.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, June 2016. ISSN: 1063-6919.

[8] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[9] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts," Tech. Rep. arXiv:1608.03983, arXiv, May 2017. arXiv:1608.03983 [cs, math] type: article.

[10] B. Gavranović, "Automatic differentiation," tech. rep., Faculty of Electrical Engineering and Computing, University of Zagreb, Dec. 2017.

[11] C. H. Martin and M. W. Mahoney, "Rethinking generalization requires revisiting old ideas: statistical mechanics approaches and complex learning behavior," *CoRR*, vol. abs/1710.09553, 2017. arXiv: 1710.09553.

[12] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," Tech. Rep. arXiv:1207.0580, arXiv, July 2012. arXiv:1207.0580 [cs] type: article.

[13] M. Orsic, I. Kreso, P. Bevandic, and S. Segvic, "In Defense of Pre-Trained ImageNet Architectures for Real-Time Semantic Segmentation of Road-Driving Images," pp. 12607–12616, 2019.

[14] 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," Tech. Rep. arXiv:1409.0575, arXiv, Jan. 2015. arXiv:1409.0575 [cs] type: article.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," vol. 8691, pp. 346–361, 2014. arXiv:1406.4729 [cs].

[16] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results."

[17] B. Li, K. Q. Weinberger, S. Belongie, V. Koltun, and R. Ranftl, "Language-driven Semantic Segmentation," Tech. Rep. arXiv:2201.03546, arXiv, Apr. 2022. arXiv:2201.03546 [cs] type: article.

[18] P. Bevandić, M. Oršić, I. Grubišić, J. Šarić, and S. Šegvić, "Multi-Domain Semantic Segmentation With Overlapping Labels," pp. 2615–2624, 2022.

[19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, 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," Tech. Rep. arXiv:1912.01703, arXiv, Dec. 2019. arXiv:1912.01703 [cs, stat] type: article.

[20] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020. Number: 7825 Publisher: Nature Publishing Group.

[21] M. L. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, p. 3021, Apr. 2021.

[22] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, pp. 90–95, May 2007. Conference Name: Computing in Science & Engineering.

[23] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," Tech. Rep. arXiv:1802.02611, arXiv, Aug. 2018. arXiv:1802.02611 [cs] version: 3 type: article.

[24] G. Neuhold, T. Ollmann, S. R. Bulo, and P. Kontschieder, "The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes," in *2017 IEEE International Conference on Computer Vision (ICCV)*, (Venice), pp. 5000–5009, IEEE, Oct. 2017.

Učenje semantičke segmentacije na slikama iz različitih domena

Sažetak

Semantička segmentacija na više skupova podataka je izazovan problem zbog mogućnosti preklapanja oznaka. U ovom radu predlažemo metodu za izgradnju univerzalne taksonomije oznaka koja se može primijeniti u kontekstu učenja s više skupova podataka. Taksonomija je konstruirana korištenjem para matrica zabune koje se generiraju treniranjem dva modela segmentacije na dva odvojena skupa podataka i testiranjem svakog na suprotnom skupu podataka. Ova se taksonomija zatim koristi za uspostavljanje redoslijeda između preklapajućih oznaka dvaju skupova podataka. Metodu validiramo na Concat modelu naučenom na skupovima podataka Cityscapes i PASCAL VOC. Predložena metoda nadmašuje osnovu i pokazuje obećavajući pristup za učenje dubokih modela na više skupova podataka.

Ključne riječi: računalni vid, semantička segmentacija, duboko učenje, swiftnet, učenje na više skupova podataka, univerzalna taksonomija

Training semantic segmentation on images from different domains

Abstract

Semantic segmentation on multiple datasets is a challenging problem due to the possibility of overlapping labels. In this thesis, we propose a method to build a universal taxonomy of dataset labels that can be applied in the context of multidataset learning. The taxonomy is constructed using a pair of confusion matrices that are generated by training two segmentation models on two separate datasets and testing each on the opposing dataset. This taxonomy is then used to establish an ordering between overlapping labels of the two datasets. We evaluate our method by augmenting the predictions of a Concat model trained on the Cityscapes and PASCAL VOC datasets. The proposed method slightly outperforms the baseline and shows a promising approach for training on multiple datasets.

Keywords: computer vision, semantic segmentation, deep learning, swiftnet, multi-dataset learning, universal taxonomy