UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER'S THESIS no. 2602

Incremental open-set recognition for semantic segmentation

Marijan Smetko

Zagreb, September 2021.

Zahvaljujem se svim osobama iz mog života koje su me pratile, bodrile i bili podrška kroz ovo putovanje do diplome, ponajviše užoj obitelji (ocu Damiru i majci Dubravci), parnerici Amaliji, te mentoru Siniši Šegviću, te svim ostalima koji će do kraja života biti dio mojih priča iz studentskih dana.

TABLE OF CONTENTS

1. Introduction

Introduction of deep learning to the field of computer vision brought unprecedented success and continues to move the boundaries of possible, surpassing the points deemed unreachable just a few years back. Nowadays, with sufficient data, it is possible to develop systems that classify images, detect objects, recognize faces, hallucinate dreams, paint pictures in predetermined styles and so on. Some of these problems are basically solved, and some are on the sole threshold of solving, mostly due to the advances in tools, computing power and, most importantly, available data.

It is relatively easy to label data for the classification: it takes one look to the picture to identify the dominant object and assign it a class. If this is done some million times, a very good learning dataset is obtained, and rather quickly. Face recognizers work just the same, usually with a twist that there are less data but more classes (identities). Object detection systems require a special box around the object determining its location, together with a class of an object. There can be multiple objects in an image so special care should be taken not to miss anything important.

All these problems are relatively common, so naturally, there are a lot of organized, annotated and publicly available data to help solve them. However, there is a special hard problem in image processing which is far from solved due to the sheer unavailability of quality and/or vast data - that problem is image segmentation.

Image segmentation is an old problem that has been gaining some traction lately, largely because of the interest for the segmentation of camera feeds of autonomous vehicles for the automotive industry. Image segmentation can be considered *pixelwise classification*: for **every** pixel the system should decide which class does it belong to. Immediately we arrive at several difficulties. First and foremost, there isn't a simple and straightforward to determine a class for a pixel based solely on its color or position. A green pixel can be a part of the grass, green car, green shirt, a frog, or possibly even a part of a socially frequent representation of a particular visitor from outer space. What the image segmentation system needs for the successful segmentation is a broad view of the image for it to be able to divide the input signal into regions that are

specific enough, yet not too small and unimportant. Furthermore, it is not enough to reduce an image to a single class or several axis-aligned bounding boxes. To train an image segmentation system, it is required to obtain data which has every single pixel annotated. This operation is very time consuming, error prone, and resource hungry. During inference, the system battles against having too little representation power coming from too small inputs or not enough data, and too large inference time, coming from too large inputs or a lot of computation.

There are still difficulties. The specific one this thesis is concerned with is the implication that all the classes for all pixels are known beforehand. For example, consider the segmentation of road vehicle camera feeds. There are some classes which are present almost always, such as the road, the sky, other vehicles, road asphalt, trees, pedestrians and so on. After spending some time, one could come up with more classes for, say, dogs and cats crossing the road, and maybe road signalization such as semaphores and road signs. If we built the image segmentation system with these classes it would (mostly) correctly segment the camera feeds until some rare events happened, such as a car crash (which shouldn't be labeled as another car!) or road work, where the bagger would either be recognized as something else (which is dangerous), or wouldn't be recognized at all as the system never saw it (which is even more dangerous).

There are two canonical solutions to this rather serious problem. The first one is to try to come up with every single class there can exist in the images. This would work in theory, but at the expense of costly obtaining of a large and diverse enough dataset, very prolonged computation time and radically increased resource demand, as the resources for the image segmentation scale with both image dimensions and the number of classes. There may even be laws against collecting the data of some class, such as government or military vehicles, or car crashes. The other solution is to admit that collecting every single possible class, along with appropriate annotations is unrealistic, and the effort should concentrate on the collection of the most probable classes, together with a special class representing "other" or "unknown". This setup is the most reasonable with respect to the cost and resources.

This thesis follows the second approach in detail, from defining and obtaining the appropriate dataset, defining models, metrics and results, all applied to the problem of road vehicle camera feed segmentation. It is mostly based on the work [3] but brings a few advancements.

2. Elementary concepts

2.1. Image segmentation

Image segmentation[1] is a computer vision processing technique which divides (segments) the input image into some disjunct regions.[2] These regions usually represent some semantic categories, e.g. various classes which can be found while driving a vehicle for the autonomous vehicle case. Mathematically, the process of image segmentation is described by the eq. (2.1)

$$\mathbf{Y} = \mathbf{f}_{IS}\left(\mathbf{X}\right), \mathbf{X} \in \mathbb{R}^{C \times H \times W}, \mathbf{Y} \in \mathbb{N}_0^{H \times W} \tag{2.1}$$

where $\mathbf{X}$ represents an input image of width W, height H and the number of channels C (most oftenly $C = 3$ for RGB images, but not necessarily), $\mathbf{Y}$ represents the output called the segmentation map where for every input pixel $\mathbf{X}_{ij}$ there is a corresponding scalar $\mathbf{Y}_{ij}$ unambiguously representing some class, and $\mathbf{f}_{IS}$ is the image segmentation function which is, in this case, represented with a deep learning model.

[1] There are two main types of image segmentation, semantic segmentation and instance segmentation. They differ in the range they operate: semantic segmentation classifies every pixel in the input image, while instance segmentation separates some instances of some classes from the image, leaving other pixels unprocessed. From this point on, only semantic segmentation is considered, and is referred to as *image segmentation*

[2] There are some instances of image segmentation where the regions need not be disjunct. However, this form of multi-label image segmentation is not considered here at all

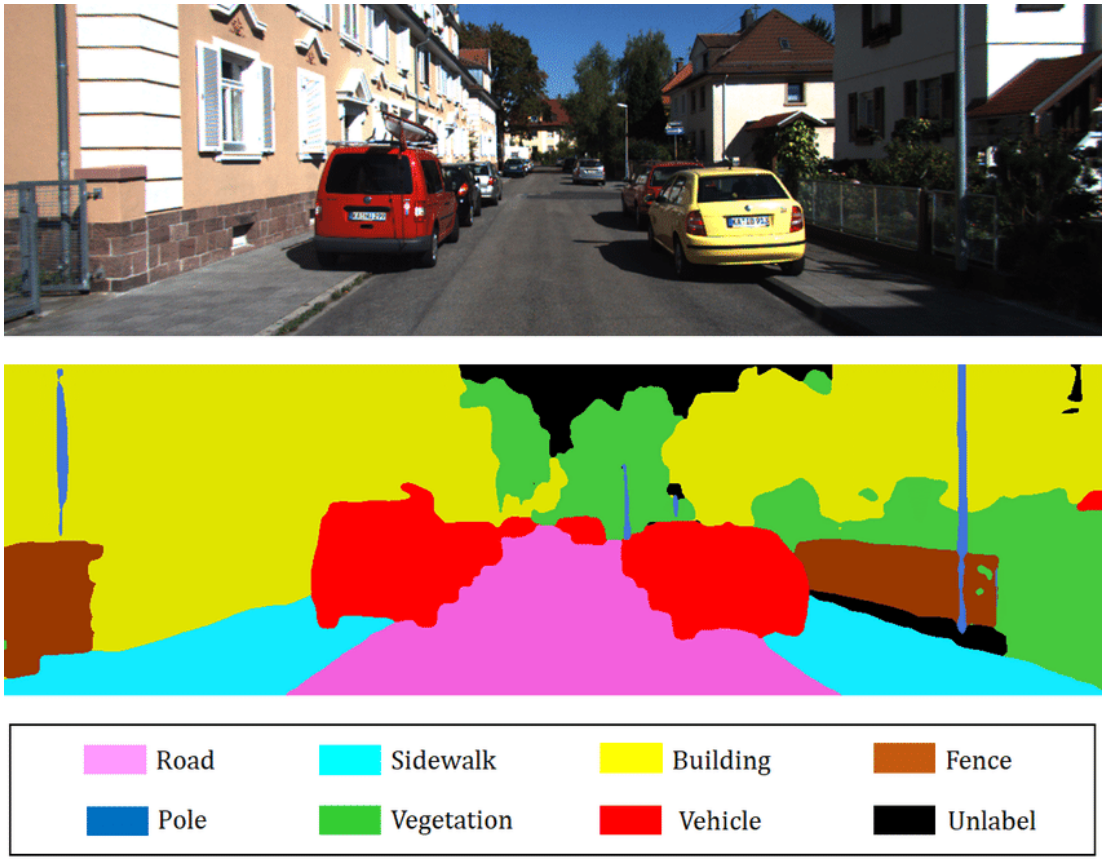

Figure 2.1: An example of image segmentation in the area of vehicle dashboard camera feed processing. This image pair comes from the KITTI semantic segmentation challenge and dataset. The upper image is a real world image of a street in Karlsruhe which serves as an input to the model. The lower image is the model's target output, a semantic map in which every pixel is assigned an integer denoting a class. In this image, there are 7 classes as per legend, which is a small subset of all the classes contained in the dataset. The classes are usually denoted by integers, and here every integer was assigned a random color. Image taken from [19]

Since the image segmentation is an old problem, there are a lot of different approaches developed, differing in the assumptions they assume. Most of techniques employ a notion of similarity between pixels. For instance, the pixels can be segmented using histogram approaches, where the pixels are implicitly grouped by their light intensity or color components[26]. The image can be first segmented in one way, and then every segment can be further split into more regions in another way, possibly even merging some of the clusters. Similarly, if a difference between pixels is defined in terms of their position, color, texture, intensity etc., a clustering approach can be used to cluster the pixels, resulting in groups of similar pixels[25][1]. The drawback is that the defined similarity of pixels may not really reflect reality, so the resulting

segments won't represent semantic categories. Another drawback is that changing the lighting, shadows, contrast, or camera focus can completely change the clustering output, rendering such methods unstable or limited to very specific problems.

Completely other family of algorithms replaces the assumption about the similarity of the pixels with the assumption that the resulting segments are usually separated in input image with well defined edges[20]. This in turn delegates the problem to the well-developed field of edge detection algorithms in image processing, and focusing on the smart way to combine the regions in between the detected edges. There is a plethora of edge detection algorithms that won't be covered in detail, so the interested reader is instructed to learn about them elsewhere, but the two most basic algorithms are described here for the completion. Gradient methods work by computing the magnitude of the color gradient for every pixel in the image. The gradient is defined as the difference in the color between neighboring pixels. The edges usually have a property that they are areas with large color gradient. Most important gradient edge detection algorithm is Canny edge detector [7]. This approach has a problem with relatively wide edges. To improve this, second order gradient information can be used, because maximal gradient usually correspond to the inflection point of the function, which can be found by finding the root of the second order gradient [23]. This approach, however, is quite sensitive to the noise. No matter the algorithm, the detected edges are usually augmented to form closed loops, and then the regions are combined to form semantic segments.

With the rise of deep learning techniques these "hand-assembled" algorithms fell out of favor solely due to their inferiority. It can be empirically shown that deep convolutional neural networks can learn to reproduce the annotated targets in much greater accuracy than any approach before. This is done by collecting and precisely annotating a large amount of images. By converting the $\mathbf{Y}_{ij}$ from $\mathbb{R}^{H\times W}$ to $\mathbb{R}^{C\times H\times W}$ where C stands for the number of classes there are in the problem (not related to the number of channels in the input), which means expanding every pixel from an integer to the vector representing one-hot encoded class, the convolutional neural net model can be trained to output a class distribution for every pixel which minimizes some measure of distribution difference such as cross entropy [3][2]. And while these systems are generally inexplicable in how they make a decision about segmentation, it turns out this is not usually a constraint, so thanks to the superiority in segmentation quality of deep learning models, this is the approach taken in this thesis.

2.1.1. Image segmentation metrics

While there are $C^{H \cdot W}$ different segmentation maps[3], only a small subset of them is actually any good to the human eye. Since the eye is not much of a precise instrument, there is a need for more numerical measure of the goodness of segmentation.

Pixel accuracy

The easiest metric to come up with is a classification equivalent of accuracy, called *the pixel accuracy*. In simplest terms, it's a ratio of correctly classified pixels and the total number of pixels in the image. Its virtues are that it is easy to understand, easy to implement, and it does not depend on the number of classes. When that number is low, the model definitely performs horribly. But, high values do not necessarily mean the segmentation system performs well. To understand why, imagine a segmentation system in a vehicle which only outputs the upper half as a **sky** class and the lower half as a **road** class. This system could easily obtain 80% of accuracy by being nothing more than a dumb baseline, which is fundamentally flawed. The core problem of the accuracy metric is the class imbalance. The sky and the road are in the largest patches of the images, so small patches do not get enough opportunity to bring the accuracy down as all pixels are considered the same. There is no penalty for the size of the predicted patch. While improvements for the pixel accuracy do exist (usually in the form of pixel-weighted or class-weighted accuracy), there is a better solution.

[3]Short proof: for each of the $H \cdot W$ independent pixels, C classes can be chosen

Intersection-over-Union and F_1

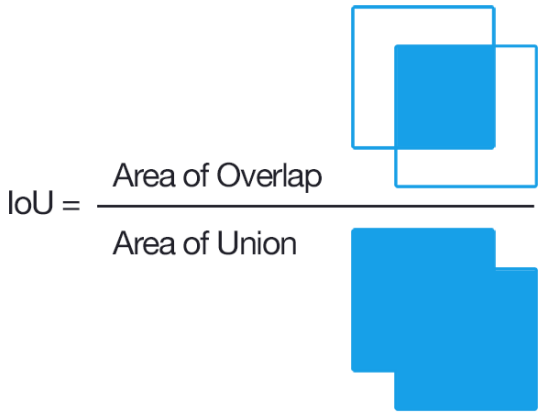

Figure 2.2: Visual aid in understanding the intersection over union. This visualization illustrates the hypothetical calculation of Intersection-over-Union with two square regions. The intersection is then the set of pixels contained in both square regions, and the union is, naturally, the set of pixels contained in either square region. Image taken from [8].

If we compare a patch of image of some class C and a different patch of image where the model predicts the same class C, we can consider how much correspondence there is between those two patches. This correspondence is a ratio between the number of pixels which are correctly predicted (the true positive count) and the total number of pixels both actually, and, predicted as C, the number which can be decomposed into three different numbers: the true positive count, the false positive count, and the false negative count. If we consider the image patches as sets of pixels, correctly predicted can be thought of as an intersection of the two sets, the metric can be represented as in Figure eq. (2.2) called *Intersection over Union*, or historically, a Jaccard index [18]. It's a much better metric because, unlike accuracy, it penalizes the size of the predicted patch. Furthermore, it just as easy to understand, easy to implement, but it also reflects the human segmentation grading more closely. One little flaw is that it's not agglomerative as the pixel accuracy since it can really only be defined in a per-class basis. The usual remedy is to take a (weighted) mean[4] of all the IoU's for all the unique target classes, a single metric called *mIoU*. Alternative to an IoU metric is a slight variation called a segmentation F_1 metric, shown in the equation eq. (2.3). F_1 metric is really similar to IoU (and is, in fact, positively correlated with it), but the

[4]The means of the pre-class IoU's should not ever be weighted by the patch size, since this formulation is actually an alternate definition of pixel accuracy. Usually all the weights are 1 or are fixed per-class over all the images.

main difference is that IoU tends to penalize bad classes more than F_1, so optimizing IoU tends to produce better results.[5]

$$IoU_i = \frac{|A \cap B|}{|A \cup B|} = \frac{TP}{TP + FP + FN} \tag{2.2}$$

$$F_1 = \frac{2|A \cap B|}{|A| + |B|} = \frac{2TP}{2TP + FP + FN} \tag{2.3}$$

Panoptic quality

However, even with a robust metric such as Intersection-over-Union defined in section 2.1.1, the numbers can sometimes seem too high in comparison with what we expect. Therefore, yet stronger metric is needed, and that is Panoptic Quality[21]. It is designed to correspond more to how a human would rate a prediction, which boils down to measuring the IoU value of all TP pixels, weighted down by the size of TP, FP, and FN sets.

While the equation shown in eq. (2.4) seems simple, it demands explanation. TP, FP and FN sets are now defined as *sets of pairs* (p, q) where p is a feature map for some class C the model outputs, and q is a target map for the same class C. Pair (p, q) is in TP set if and only if $IoU(p, q) \geq 0.5$[6]. Summing the IoU's of patches over all the classes and dividing it by something connected to the number of patches is akin to taking an average. But, just with pixel accuracy, big patches are not penalized, so the metric is weighted down by the sizes of FP and FN sets of corresponding pairs. An astute reader will recognize a connection with the F_1 metric, and it can mathematically be shown that PQ is actually an F_1 metric scaled down by the mean of all IoU's in the TP set. The final PQ metric is obtained by averaging out all the PQ_i over all the classes.

$$PQ_i = \frac{\sum_{(p,q) \in TP_i} IoU(p, q)}{|TP_i| + \frac{1}{2}|FP_i| + \frac{1}{2}|FN_i|} \tag{2.4}$$

While simple at first, and possibly convoluted underneath the surface, PQ is actually a good measure of semantic segmentation performance which aligns more closely

[5] F_1 metric is actually a family of metrics F_β, parameterized by β which denotes a tradeoff between the precision and the recall. F_1 in particular weights them equally[29]

[6] It can be mathematically proven that there can be only one pair of p and q for any class C such that the IoU is greater than or equal to 0.5; this proof is out of scope of this thesis but interested reader is instructed to study the details of [21]

with how humans rate semantic segmentation output. The part of a reason is that is was tailored for that by the authors of [21].

2.2. Transfer learning

Due to the complexity of functions represented by neural networks, there is usually no (tractable) way to compute the optimal set of weights in a closed form, mostly due to the nonconvexness of the problem formulation. Therefore, all deep learning models are almost universally trained by gradient descent. In all generality, gradient optimization procedures take into the consideration the information about the gradient of a function they optimize. The gradient, which is in this case computed by a special algorithm called backpropagation, is a vector which points in the direction of biggest increase of a function. Since the function optimized is usually a loss function we attempt to minimize, all the weights are modified in such a way that a step is taken in the opposite direction from the gradient, as to minimize the loss function by the greatest magnitude.

The problem is, however, where to start optimizing, that is, what initial set of weights to use. In the past, deep learning models were trained for a specific problem from scratch, usually from randomly initialized weights. However, a starting point for the model's weights can have a huge impact on the model's characteristics such as maximal performance and convergence speed (or convergence in general!). Naturally, the best starting point is in the optimum itself, but it's highly unlikely to aprori know where the optimum is; if that was the case, no learning procedure is needed at all. The second best starting point is, then, in a vicinity of the optimum. Now, it is still hard to tell where the optimum lays for a specific problem that is being solved, but in practice in turns out different problems can be averaged out in the parameter space so as to obtain a probability distribution of local optima. Starting somewhere in this region will perform good in a variety of problems. Some of the initialization methods are Xavier uniform/normal[12] and Kaiming uniform/normal[16], both of which sample the starting weights from a probability distributions the weights are observed to follow in practice.

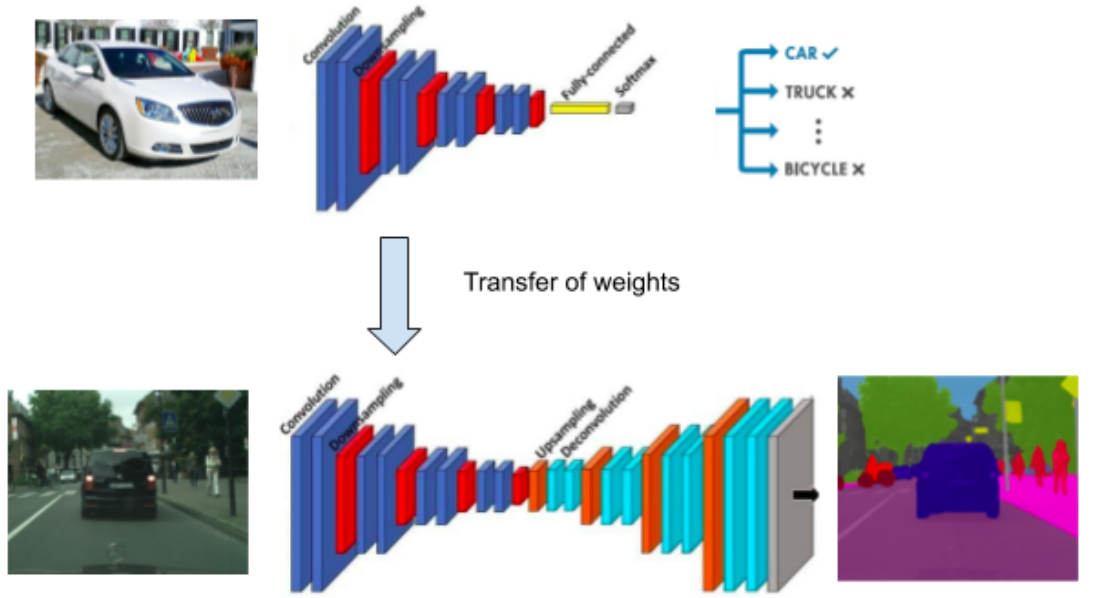

Figure 2.3: An illustration of transfer learning. The model which is well trained for image classification or object detection is being reused as a feature extractor for the semantic image segmentation. As both problems are in the domain of traffic, it is expected that the weights of a feature extractor will extract good features for the segmentation. This, however, should not be confused with incremental learning from section 2.3 as the tasks (classification / detection vs. segmentation) are different. Image taken from [13]

Remarkably, it also turns out we can use a model already pretrained to solve one problem as a starting point of solving another, similar problem. It has been shown in practice that using a pretrained model as a backbone of a model for the current problem improves both the rate of the model improvement and the end performance. This is easily explained if we look at the pretrained backbone as a feature extractor for the problem at hand. Instead of learning all the low-level features from scratch and slowly combining them into higher order features, the pretrained backbone is already yielding higher order features, which helps the model. Usually, the backbone itself is not trained at all until a certain convergence plateau is reached (also called locking the backbone), as that would destroy the learned features. A few extra points of performance can, then, be obtained by unlocking the weights of the backbone after the plateu so as to finetune the weights of the backbone to extract features more aligned for the problem. Alternatively, the backbone can be given substantially lower learning rate to change its weights more slowly, in the hope it'll align its features with the problem at hand

simultaneously with the untrained model segment.

Transfer learning is especially useful tool for solving problems that have scarce data, such as the segmentation of medical images in the search for malign tumors, which are hard to obtain due to patient privacy laws, or the segmentation of aerial images of the ground, which are hard to obtain due to the price required for their capture. In both cases, and much more, using a pretrained backbone usually performs much better and generalizes well in comparison with training the model from scratch.

2.3. Incremental learning

Transfer learning solved quite a few problems in the period when both data and models were scarce. However, with time, the number and availability of both data and models rapidly increased. People started solving various problems, producing a spectrum of solutions. Yet, most of these solutions broke under new data. A trivial solution is to retrain the model from scratch, but one can do better with *incremental* learning. The idea behind the incremental learning is similar to transfer learning: to take a pretrained model which solves a problem with the hope it'll help solve the problem at hand. The main difference is that, in incremental learning, the problem the pretrained model solves is *exactly* the problem at hand. The pretrained model is often times the same model, it's just provided with more data, which helps fill the gaps in the knowledge it had.

Training the models incrementally is powerful as it can be regarded as the usage of a really good feature extractor, one which misses the local optimum by a small margin. It is especially useful in the case where the original data is no longer available, just the recent data. One peculiarity is that models trained in this way sometimes have a tendency to forget the knowledge they had, a phenomenon which should ideally be avoided at all costs, usually by decreasing learning rate or by using dataset augmentations.

The incremental learning is somewhat similar to curricular learning, in which the model is presented with easier examples at first, and then the examples get harder and harder. Just as with incremental learning, the model can still be occasionally shown easier examples to slow down the forgetting. The curricular learning can be thought of as a subset of incremental learning as curricular learning doesn't ever change the model architecture, but incremental can, and does sometimes.

The core idea in this thesis is applying incremental learning for image segmentation. First, using transfer learning, a model pretrained for classification is used as a starting point of a model performing image segmentation on several datasets containing only inlier classes, providing estimates for the pixel class probabilities using a prediction head. Then, the trained model is further incrementally trained for prediction on outlier classes using a separate prediction head, but it is forced to retain the knowledge it had by exploiting the fact the old data is still present, and feeding the model data it should already know how to solve.

2.4. Public datasets

All machine learning systems are in need of a large quantity of (quality) data, and semantic segmentation systems take this requirement to the next level. Therefore, it is not surprising how the attempts of solving this generated several publicly available datasets, containing images and labels for both training and validation. Some of the below-mentioned datasets are showed in fig. 2.4. One of the oldest datasets is Pascal VOC (*Visual Object Classes*) [10], usually used for classification and detection, but which obtained the segmentation labels in 2009. The latest version of the dataset is from the 2012 and contains 1464 images in the train set and 1449 images in the validation set, with the test set hidden in private. The dataset has started as an official dataset for the VOC object recognition competition of the PASCAL organization from the University of Southampton. It is a general purpose dataset with 20 classes of the general world, such as people, animals (cat, dog, bird etc.), vehicles (car, boat, train etc.) and indoor objects (tv, chair, bottle etc.).

Another general purpose image segmentation dataset is Microsoft COCO (*Common Objects in Context*) [22]. It's a quite large dataset with more than 118K labeled images in the training set and 5K images in the validation set in more than 80 diverse classes, covering everything in the Pascal VOC and much more. It has multitude of labels: the same dataset can be used for instance segmentation, panoptic segmentation, human pose estimation and human keypoint localization.

There is a lot of interest for the semantic segmentation of images in the scope of self driving vehicles, so naturally some public datasets containing dash camera feeds appeared. One of the most famous representative of such is the CityScapes dataset[9]. Its fine grained image segmentation collection contains 2975 training images, 500 validation images and 1525 testing images. The segmentation maps cover 30 classes found in usual traffic (such as person, road, car, wall, traffic light etc).

There is no doubt that CityScapes is invaluable to the semantic segmentation field, but with time appeared the need for larger and more comprehensive datasets. One of such is definitely Mapillary Vistas[24] dataset with 66[7] classes over 25,000 images. And while the CityScapes isn't really as diverse as it should be (only 50 cities, mostly German, good weather conditions, similar cameras), the Vistas creators went out of their way to make it as diverse as they can (images from all 6 inhabited continents, in various lightning and weather conditions, and taken by mobile camera, action camera, professional equipment etc). It also features both coarse and fine annotations.

[7] for version 1.2; this number climbs to 124 in v2.0 od the dataset

There are other datasets as well. KITTI [11] dataset is a another small German dataset. It contains 8008 frames in the training set and 10173 frames in the test set over the 19 CityScapes' classes. It's relatively new. CamVID [5][6], on the other hand, is one of the oldest car dashcam video datasets. While historically important, it only has $\tilde{7}00$ labeled images which is way too small for any important real world usage. Another relatively small dashboard camera segmentation dataset is WildDash[30]. It has only 4256 labeled frames, but instead of the quantity, the authors rather focused on the variety in the images, such as weather conditions, locations, vehicle types, road types, camera distortions etc.

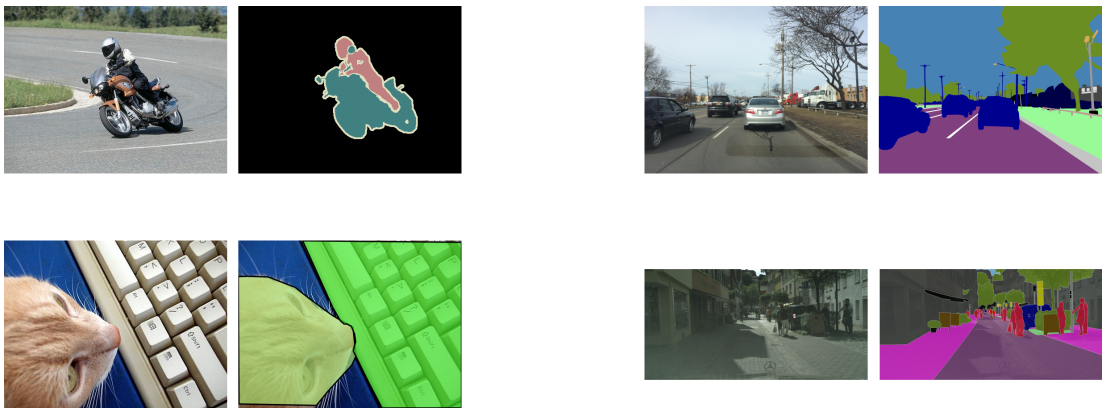

Figure 2.4: Example images (left in any pair) and their target segmentation maps (right in any pair) of various semantic segmentation datasets described in this section. Top left is an example image from Pascal VOC[10] featuring two classes, namely `person` and `motorcycle`, whereas everything else is labeled as `background` which denotes the absence of a particular class. Top right is one of the tamer images from Vistas[24] dataset featuring several classes, the largest of which are `road`, `sky`, `vehicles` and `vegetation`. Bottom left comes from the COCO[22] dataset, and contains a cute `cat` near a `keyboard`. The bottom right is wide-resolution and fine-grain labeled image of a German city Tübingen from CityScapes[9]. The image is populated by more than 10 classes, but the largest are `road`, `person`, `building`, `vegetation` and `sidewalk`. These examples show how the differences between, but also some similarities across the datasets.

3. Method

The goal of this thesis is to successfully train a deep learning system which will segment the dashboard camera feed in real time, but also provide an estimate of its uncertainty. This is accomplished by extracting features and then feeding them to two separate heads, one which performs the segmentation of the input image, and the other which assesses the confidence of the segmentation predictions.

3.1. Model architecture

3.1.1. Feature extractor

The feature extractor is based in the DenseNet-121. DenseNets[17] can be thought of as an upgrade of Residual Networks (ResNet's [15]). ResNet architecture introduced a novel concept of skip connections in the models, which facilitate information flow deeper in the model. These skip connections manifest as elementwise addition between the layer's input and output - instead of the features themselves, the layer in ResNet learns the difference in the features, also called the residual. This seemingly simple trick is very effective accross all deep learning fields, including computer vision and natural language processing.

However, a layer in the ResNet only ever takes the previous layer's input, which still doesn't let the features flow deep enough . To promote the information flow even further, one layer of DenseNet (in one dense block) takes as input all the outputs of the previous layers, and its output is feeded to all the next layers. An important note is that the previous inputs aren't summed in an elementwise fashion. They are, instead, concatenated. Along with the information flow, DenseNets are more parameter efficient as the convolution layers don't have to have as much channels as they had to have before. The only downside is that DenseNets use more memory as they have to store all the layer outputs of a block.

Nevertheless, DenseNets are powerful feature extractors. The architecture of one

DenseNet block is shown in fig. 3.1. Every layer takes as input the concatenated activations of all the layers before it. Despite the need to retain all the activations, DenseNet block layers actually have less parameters since they don't have to have as much kernels - it's usual to have a ResNet with 256 kernels, but here every layer in a block adds only 4 more features. DenseNet architectures, such as DenseNet-121, usually have 4 dense blocks, with 3 transition layers between them consisting of 1×1 convolution and a pooling layer, which downsamples the signal by half.

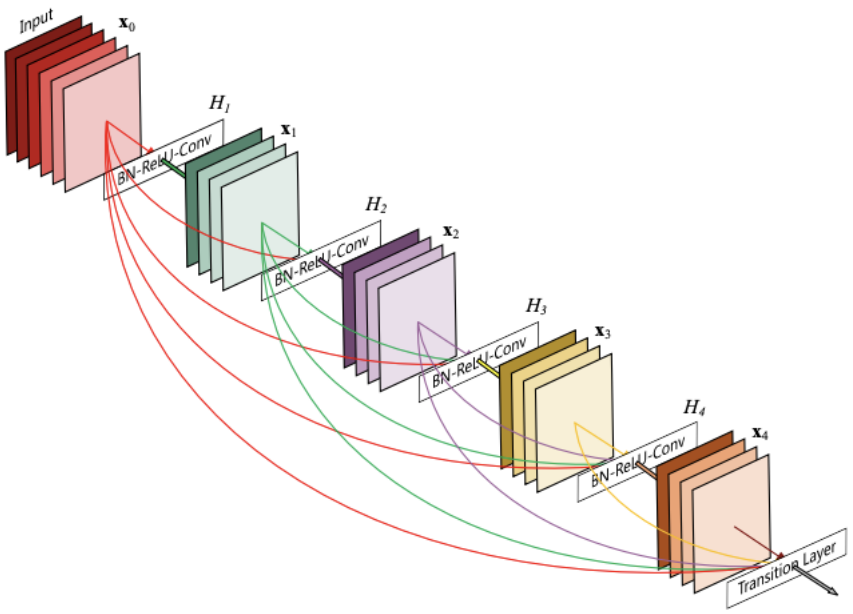

Figure 3.1: The architecture of a DenseNet block. The i-th layer takes as input all the activations of layers $0..i-1$. In the original paper, every layer performed the BatchNorm-ReLU-Conv operation. Every layers adds k new features, usually $k = 4$. Here in the figure, there are 5 layers in a block, but in the paper that number can be up to 64. The resulting tensor is then transformed and scaled down for the input into the next dense block. Image taken from [17]

3.1.2. Spatial pyramid pooling

The output of the last dense block DB4 is then fed into a special layer performing spatial pyramid pooling. Spatial pyramid pooling was first introduced in [14] as a simple way to represent the whole input signal with a fixed number of features. One simple way to do this is to split the signal into N^2 equal patches and perform a pooling

operation (e.g. a max pool) over every patch[1]. This can usually be performed for every input dimension, provided that N divides the height/width dimension of the signal. Doing this several times with different N_i corresponds to different "attentions": a small N_i means splitting the image into less patches which are larger, providing the large scale features, and large N_i means splitting the image into more small patches, providing the small scale features.

This procedure is very appropriate for the image segmentation problem due to its side effect: it increases the receptive field of the model. Image segmentation models have to be aware of both the high level context of the image, and the low level features such as edges and shapes. The SPP block fulfills this requirement perfectly.

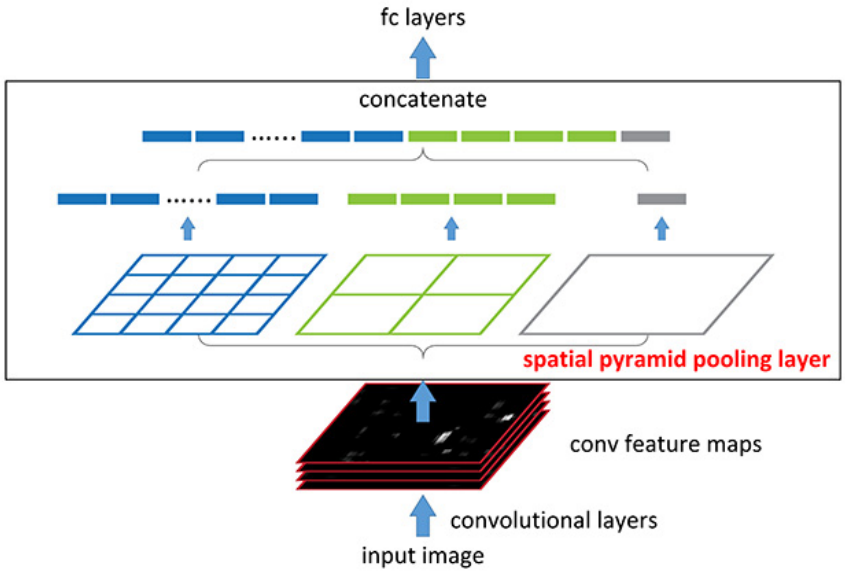

Figure 3.2: The schema of the Spatial Pyramid Pooling (SPP) layer from [14]. The input signal is split into patches in multitude of ways (1^2, 2^2 and 4^2 patches in the figure) and a pooling operation is performed upon every single patch. Since there is a constant number of patches w.r.t the scale factors, the SPP always produces the output of the same size. Also, it summarizes the input signal across several scale levels, increasing the receptive field of the model, which is necessary for successful image segmentation. Image taken from [14]

The models in this thesis use SPP with $N_i = 1, 2, 4, 8$. For every input image the model outputs $\sum_{i=1}^{4} N_i^2 = 85$ features *per input channel*.

[1]While this operation summarizes the activations in the patch to a single number, it keeps the channels separated, meaning a $C \times H' \times W'$ dimensional patch is always going to be reduced to $C \times 1 \times 1$.

3.1.3. The upsampling

While there are dozens of layers in the downsampling and feature extraction path, there are only few upsampling layers, concretely 3. These upsampling layers are structured similarly as U-net[27] architecture, which is most easily explained with a ladder analogy: the activation after every **block** in the DenseNet feature extractor is saved, and used in reverse order during upsampling. In this way, the model extracts high order features at its end, but is reminded of the lower order features with the corresponding block activation while restoring the input resolution, which is used to restore the segmentation maps in significantly higher detail.

To speed up the training process, three auxiliary losses are defined after every upsampling block. They act as a regularizers and help propagate the training signal deeper into the network. These auxiliary losses target the downsampled segmentation distribution at lower resolution, to ease up the upsampling. The details are in fig. 3.3.

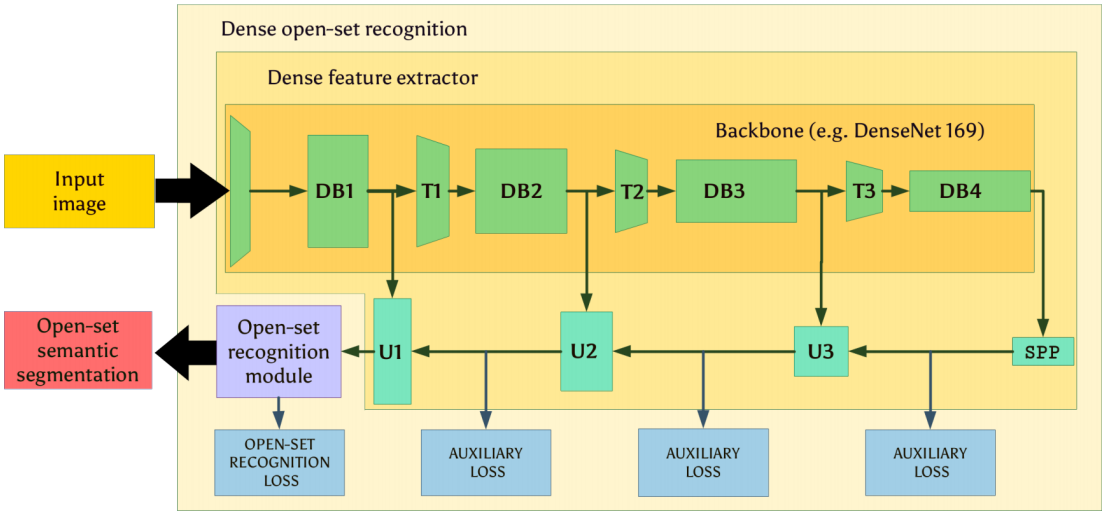

Figure 3.3: The architecture of the two headed model

3.1.4. The segmentation and confidence heads

The output of the last upsampling layer is a tensor of logits for pixel level classification. These logits are usually fed into a special layer which outputs a distribution across the classes, called a segmentation layer, or a segmentation head. This approach, however, doesn't really perform when there are objects of an outlier class in the image.

Therefore, another layer is introduced, with the same input, parallel to the segmentation head, which has a sole purpose of determining if the pixels in the image are a part of outlier objects or not. The output of this head is $\mathbf{O} \in [0,1]^{H \times W}$ which is inter-

preted as a posterior probability that the given pixel at some location is an outlier. It is important to note that sharing the input is not a strict requirement but an inductive bias that the output upsampled tensor contains enough information for both the segmentation and outlier detection. Luckily, that bias turns out to be correct, and the results will show that effective outlier detection is obtained *"for free"* when performing semantic segmentation .

3.2. Experiments

Usually, the weights used for transfer learning are the weights learned after training some model architecture on the ImageNet[28] classification tasks. The hypothesis this thesis tests is that this may, in fact, be suboptimal. Since the classification and the segmentation tasks are quite different, it could happen that the weights in the feature extractor sometimes specialize in extracting unimportant or unnecessary features from the image, resulting in lower accuracy, mIoU and other important metrics.

To combat this, the idea is to pretrain the feature extractor on a different image segmentation task, and then use this as a (hopefully) better basis for the task at hand. These pretraining segmentation tasks should be simpler, to speed up the learning process, but also diverse, as the model should not be overfit for that task, but should extract important and useful features. This thesis achieves that by pretraining the model on CityScapes and Vistas features for the inlier classes, which is assumed will boost the performance for the outlier class detection.

4. Experiments

4.1. Experiment setup

All experiments were performed on the faculty's train machine containing nVidia GeForce GTX1070 (8GB) and nVidia GTX980 (4GB) graphics cards. Batch size was tuned to decrease the training time as much as possible without hurting the metrics, and typically ranges between 32 and 64.

For the first part of the incremental learning training process, the only augmentation is the input size jitter. From the input images with varying dimensions, random patches were cropped out and rescaled such that the longer side is 768 pixels, which makes the other side in range of 384 to 960, analogous to [3] with a smaller resolution. The training on CityScapes and Vistas datasets ran for 50 epochs which took 1 day and 3 days, respectively. The starting learning rate was $4 \cdot 10^{-4}$ for the untrained upsampling path and $1 \cdot 10^{-4}$ for the pretrained feature extracting backbone. No learning rate tuning was performed in any time. An epoch in this part was a simple pass through a training dataset while updating the model, and a simple pass through the validation dataset while not updating the model but only evaluating it.

For the second, incremental learning step of the training process, a different approach is taken. After augmenting the model with another head which is to specifically predict outliers, the whole model is then trained on a combination of CityScapes, WildDash and Vistas datasets. In addition to standalone negative images, mixed-content images are obtained by pasting random ImageNet outliers into random patches within an inlier image (together with appropriate scaling of course). However, due to the issues related with the training resources and extremely large training time which would render this thesis indefensible, the resolution was further downsampled by a factor of 2. This type of training took 4-7 days for 10 epochs. The epoch here is a simple pass of the CityScapes and Vistas datasets together with the ImageNet portion, the one that has bounding boxes.

4.2. Pretraining results

The results of the CityScapes- and Vistas-pretraining phase are shown in table 4.1 and figures fig. 4.4 and fig. 4.1, respectively. The feature extractor weights were pretrained on an ImageNet classification task and then finetuned for two segmentation tasks by means of transfer learning.

Dataset	batch size	resolution	Max validation mIoU	At epoch
CityScapes	**64**	**W/4**	**64.28%**	**50**
CityScapes	32	W/4	63.89%	50
CityScapes	32	W/8	63.2%	49
Vistas	**32**	**W/4**	**60.99%**	**47**
Vistas	64	W/4	60.3%	50
Vistas	32	W/8	59.7%	48

Table 4.1: The table of the experiments performed for the pretraining step. Validation metrics were performed on the validation sets of the respective datasets, since these models were only trained for the semantic segmentation. The total of 6 pretraining experiments successfully finished with varying hyperparameters. The resulting validation mIoU is shown with respect to the training process parameters. Bolded rows are the best experiments, which are then used for the incremental learning step.

4.2.1. CityScapes pretraining results

The training metrics progression of the CityScapes experiment is shown in fig. 4.1. The shape of the curves is quite peculiar, with two distinct performance degradation of unknown origin in epochs 8 and 24. Nevertheless, the model ends up having more than 93% pixel accuracy and more than 64% validation mIoU at the end of the training. Another curiosity is the extent of the bad performance at the beginning of training, at least in comparison with fig. 4.4. This could indicate that the starting feature extractor isn't really well suited for the extraction of features on the CityScapes dataset. Luckily, the performance at the start improved rapidly.

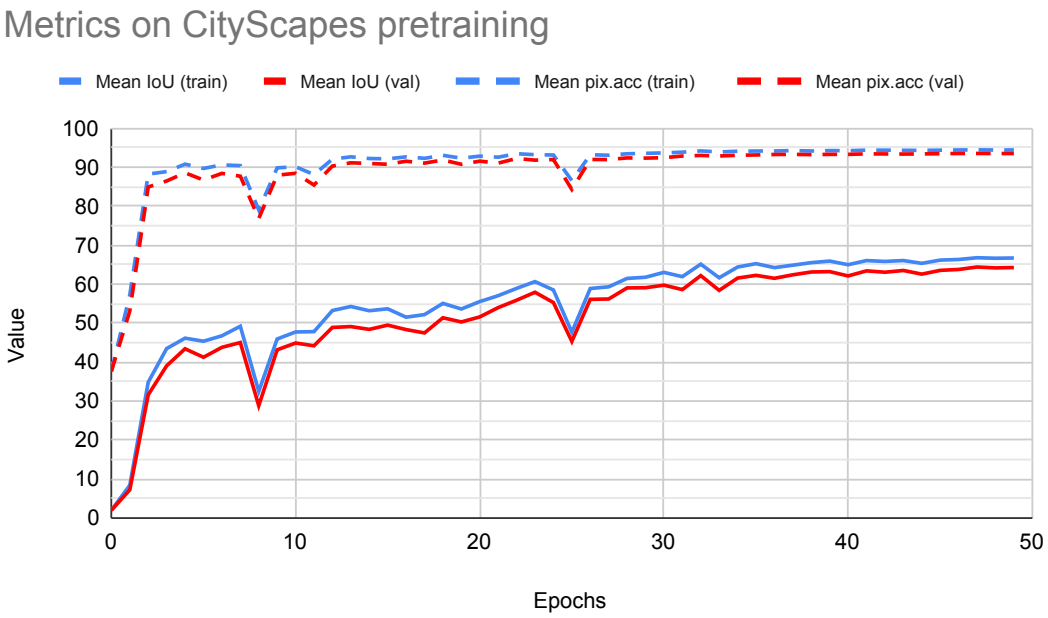

Figure 4.1: Training progression while training on the CityScapes dataset. The metrics show a quite low starting point, but a large raise over 50 epochs. The pixel accuracy (dashed lines) is at 94.4% in train time (93.5% in validation time), and the mean Intersection-over-Union (solid lines) is 66.7% in train time (and 64.3% in validation time) at its best, which is in epoch 50. Small relative differences between training and validation metrics also show the model was not overfitting and appears to retain the ability to generalize well. However, this training progression implies that the feature extractor wasn't particularly good for this dataset, but luckily the data of the CityScapes dataset itself is easier to learn than the data of Vistas.

Good examples of the CityScapes pretrained model are shown in fig. 4.2. Examples are row-wise, where the left column is the input image, the middle column is the target segmentation map, and the right column is the predicted segmentation map. The model seems to be recognizing the class of the small objects, such as semaphore poles and distant objects. Model also seems to have an understanding of common scenes, such as that the light green color (denoting grass vegetation) is often times near a deep purple color (denoting the road). The model is also very good at recognizing the car class (dark blue). The model shows signs of a mild overfit in the fourth example, where the Mercedes sign is confused with a bicycle (deep red).

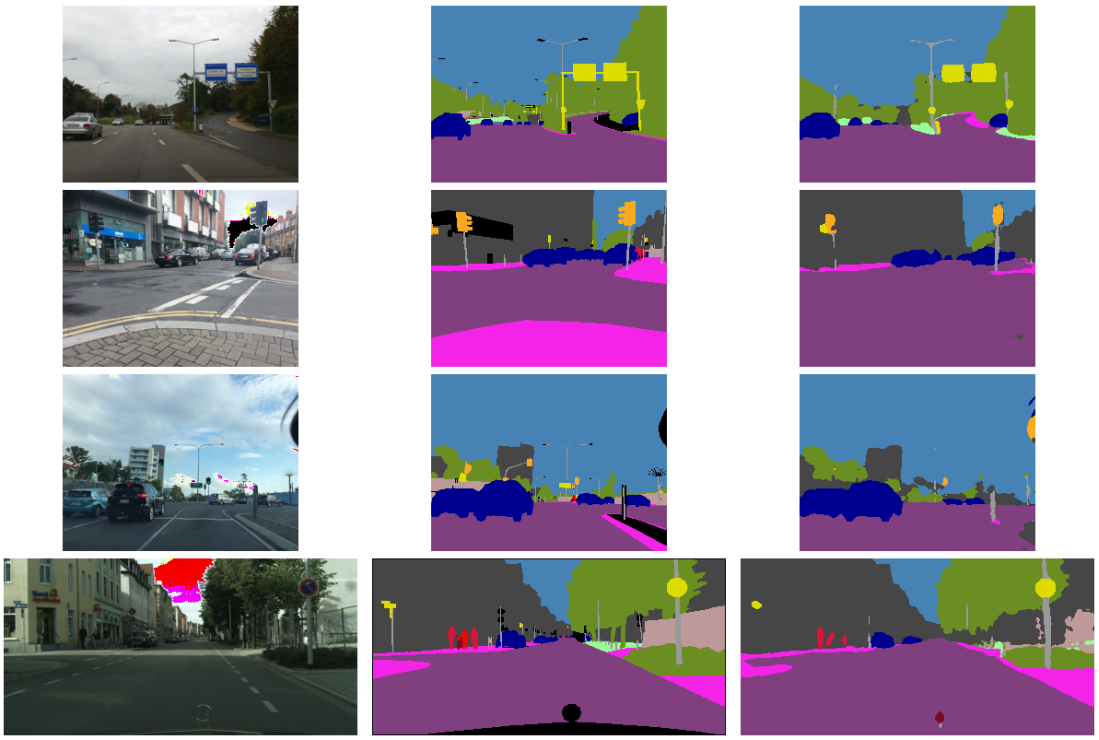

Figure 4.2: Hand picked examples of the CityScapes pretraining task, on which the model performed well. It is worth paying attention to how the model successfully segments tiny image patches, such as traffic light poles, people and cars in distance and some small vegetation. Interestingly enough, second and fourth examples were segmented quite well despite the visible corruptions.[1]

Bad examples of the CityScapes pretrained model are shown in fig. 4.3. They mostly reveal the instability of the model's predictions such as emergence of nonexistent classes (false positives) and the disappearance of existent classes (false negatives). For example, a gray color (denoting a building) and a blue color (the car) appear in the middle of the road, which isn't really physically possible. This is due to the dashboard's reflection on the windshield, which interferes with the model's predictive capabilities. Other mistakes are the disappearance of the sidewalk (pink color) in the second example, hallucination of a car in the third example, and a total overlook of humans (red), which renders such model as dangerous.

[1]These corruptions aren't really much of a problem since they originate from a decoding mistake. The only way to obtained the original image the model saw was to "denormalize" the input tensor with the RGB mean and variance. However, the original image mean and variance were lost so the dataset mean and variance were used as a proxy. While generally similar enough, sometimes the decoded values exited the 0-255 range, which tends to cause these artefacts.

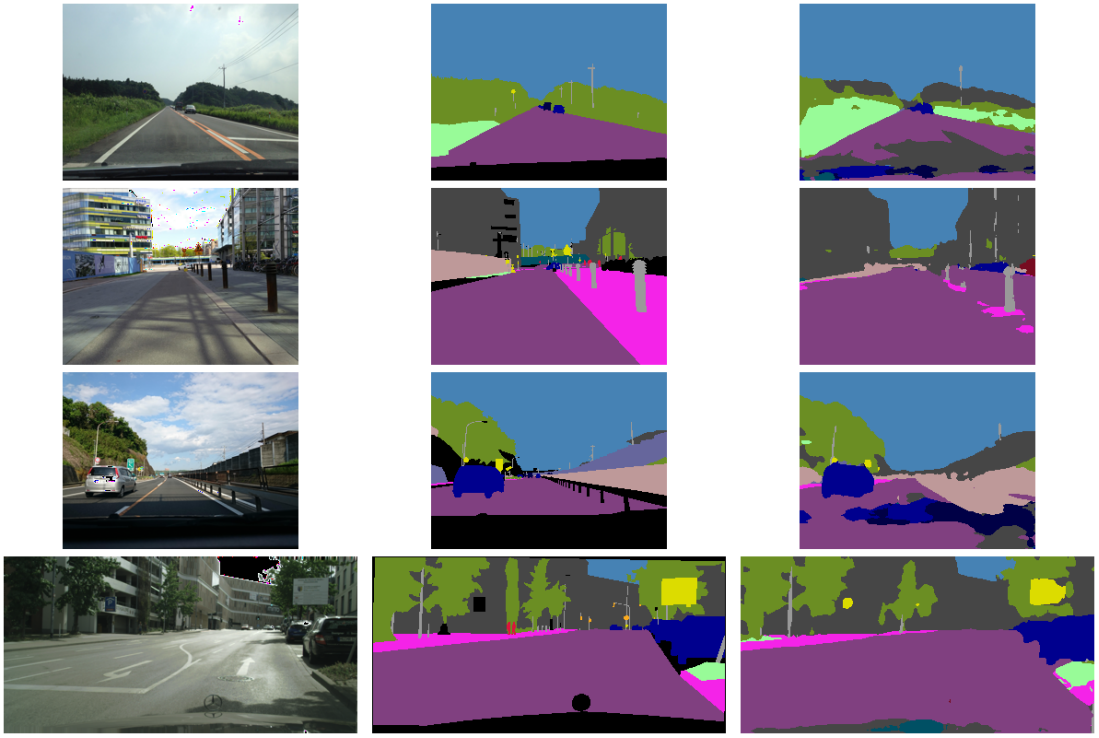

Figure 4.3: Hand picked examples of the CityScapes pretraining task, on which the model performed poorly. The model seems to be having problems with the hallucination of nonexistent classes, such as cars and buildings, and disregarding other classes, such as sidewalk and human. This is, however, mostly due to the interference of reflections (first example), colors(second example) and shadows (third example).

4.2.2. Vistas pretraining results

The metrics on the Vistas segmentation dataset show a slow and steady rise in performance over 50 epochs. The pixel accuracy (dashed lines) is at 94% in train time (93% in validation time), and the mean Intersection-over-Union (solid lines) is 62.7% in train time (and 61% in validation time) at its best, which is ubiquitously in epoch 47. Small relative differences between training and validation metrics show the model was not overfitting and still retains the ability to generalize well. The starting mIoU of 40% and the uplift of 20% hint towards the conclusion that the feature extractor is well suited for this problem, but that the dataset is not particularly easy to learn.

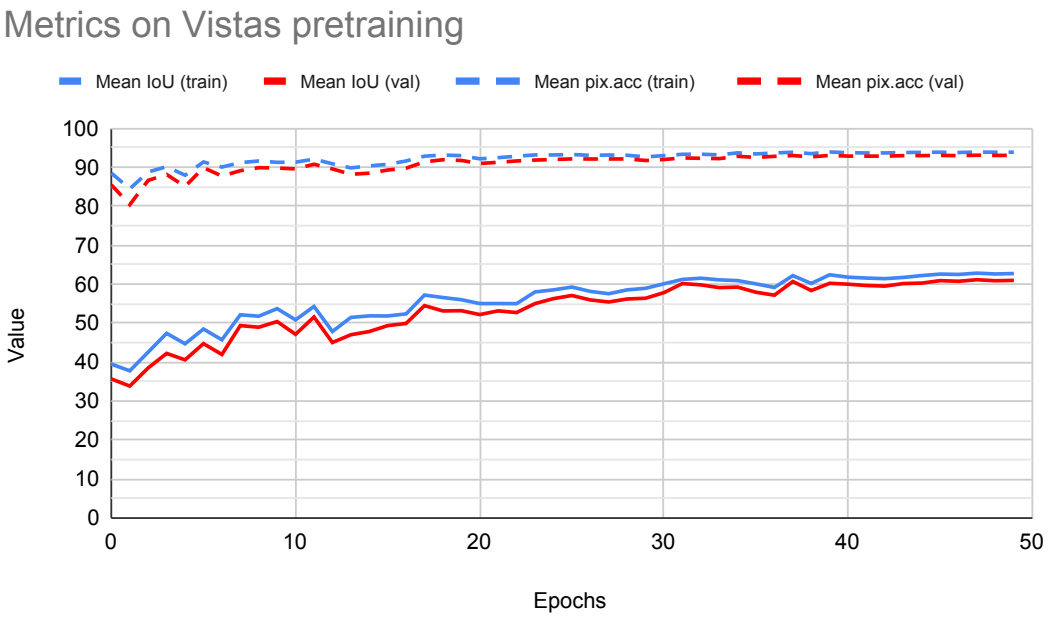

Figure 4.4: Training progression while training on the Vistas dataset. The results indicate that, unlike 4.1, the pretrained prior weights extract good enough features in the beginning of the training. This can be seen from the rather high starting point in the training process with around 40% of mIoU. The best epoch across all metrics was epoch 47 with 93% of validation pixel accuracy (dashed lines) and 63% of validation mIoU (solid lines). This graph as well shows no visible signs of overfitting, just like 4.1.

The examples on which the model performs well are shown in fig. 4.5, in which the model seems to have a considerable understanding of the scene, as it consistently labels the pixels correctly, even the small image portions, similar to CityScapes model, with a specially good performance at segmenting other road vehicles such as cars and trucks. The largest differences seem to be some false negatives on the vegetation class

at Example 4 and some sidewalk uncertainties at Example 3.

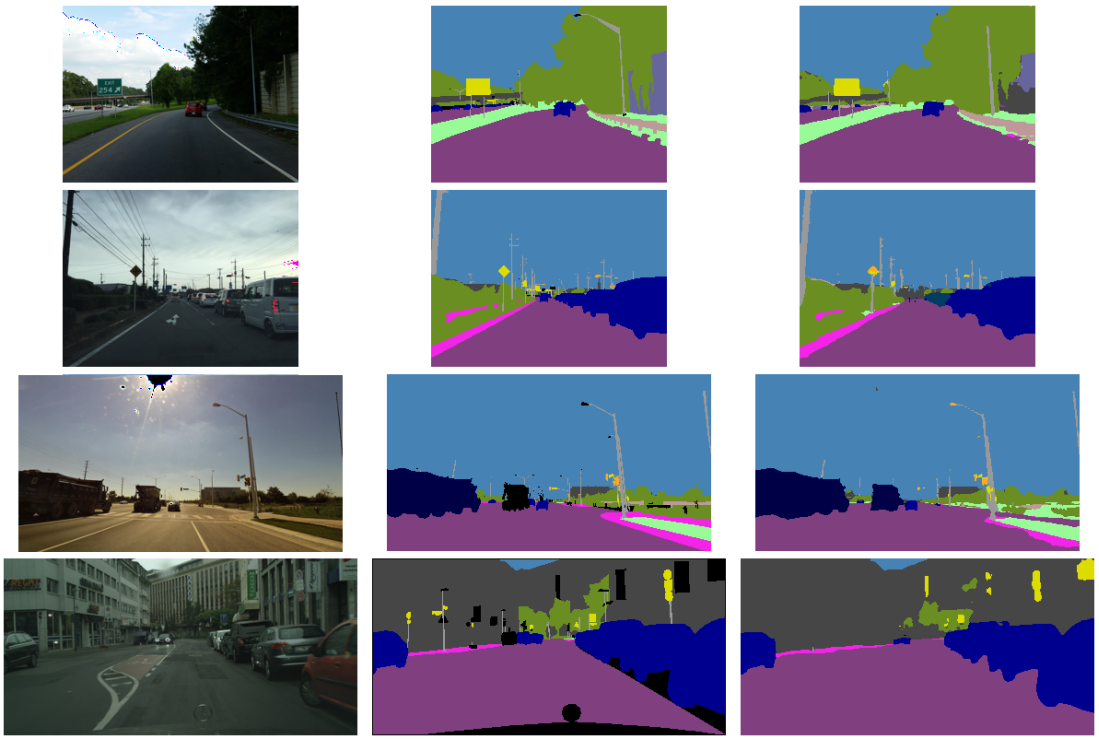

Figure 4.5: Hand picked examples of the Vistas pretraining task, on which the model performed well. Similar to the model pretrained on CityScapes, this model also segments the road vehicles very well, along with extra fine segmentation of road signs. The only large visible differences are the lack of recognition for a sidewalk in the Example 3 and some false negatives in the vegetation in Example 4. Note that the decoding artifacts appeared once again (e.g. the sun in Example 3) yet they proved not to be a challenge to the model.

Bad examples of the Vistas pretraining tasks are shown in fig. 4.6. Although the results superficially look good enough, closer look reveals serious drawbacks. For instance, the model mistook the bicycle bell and the handlebar for a car. Then, in Example 3, the model completely confused the tramway with a truck. The worst of all is a complete disregard of people in the Example 4, together with loss of the details, which is in contrast with the capabilities shown fig. 4.5. All in all, the resulting model has some degree of scene understanding but, just like the resulting model from the CityScapes pretraining task, lacks prediction stability and does not concentrate on details enough.

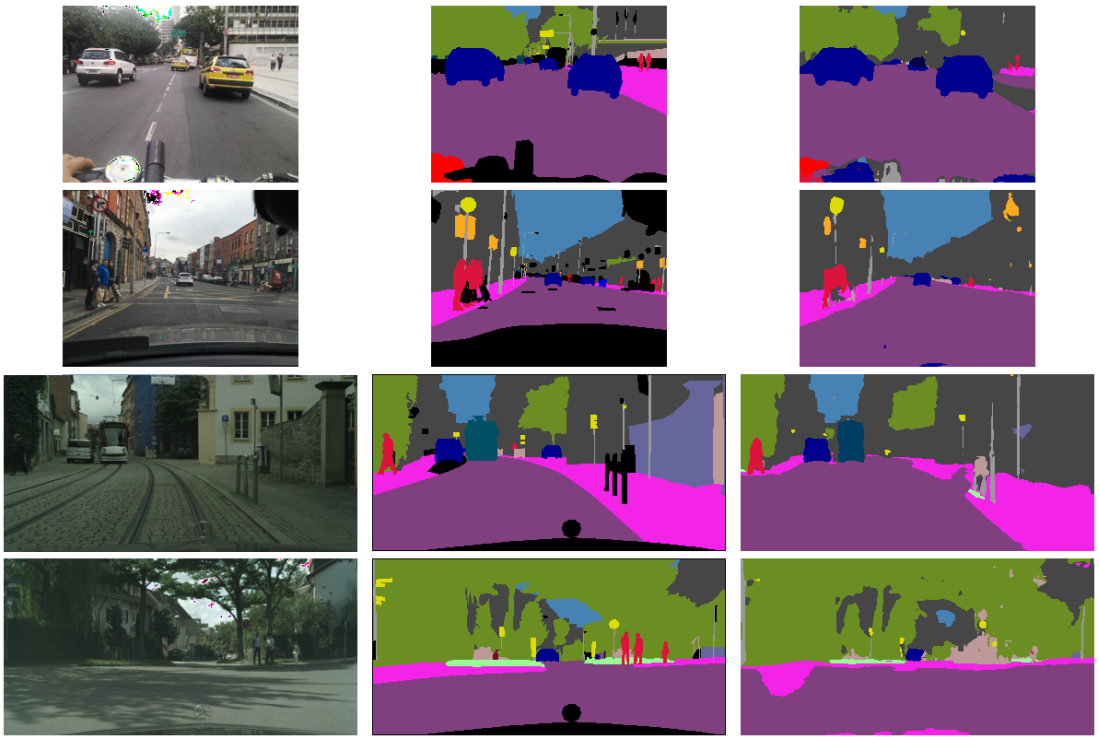

Figure 4.6: Hand picked examples of the Vistas pretraining task, on which the model performed poorly. The same model as in fig. 4.5 exhibits unwanted behaviors, such as not recognizing people at Example 4, disregarding all the small details across all the examples, having trouble discerning the road from the sidewalk, and total misclassification of the incoming tramway (dark cyan) for a truck (grayish blue).

4.3. Two head model results

Base model	batch size	resolution	Max validation mIoU	At epoch
CityScapes @ 50	**64**	**W/8**	**65.15%**	**10**
Vistas @ 47	**64**	**W/8**	**65.31%**	**10**

Table 4.2: The table of the experiments performed for the two head step. Due to the hardware constraints, only two experiments finished successfully in reasonable time. Another consequence is the reduced input image resolution from W/4 to W/8. The "@" sign denotes the epoch from the pretraining task from which the model was taken.

4.3.1. CityScapes

The results of the two head model pretrained on CityScapes are shown in fig. 4.7. After only 10 epochs the mIoU jumped for more than 15 points, and already high pixel

accuracy of 90.67% climbed further up to 93.69%. The model seems to be performing quite well as the validation curve is almost indiscernible from the training curve. This points to the hypothesis that this incremental learning pipeline could be really potent.

Figure 4.7: The progress of the second phase of incremental learning of the model pretrained on CityScapes. The models gets a boost of more than 15 points of mIoU in 10 epochs, resulting with 65.15%. The validation curve closely follows the training curve which is indicative of a potent model.

Some good examples are shown in fig. 4.8. The behavior of the model after incrementally training it on the ImageNet seems similar to the behavior before (the details, the road vehicle segmentation quality), with an emphasis on a correct and well behaved segmentation of a jaywalking person, which is of utmost importance in real life.

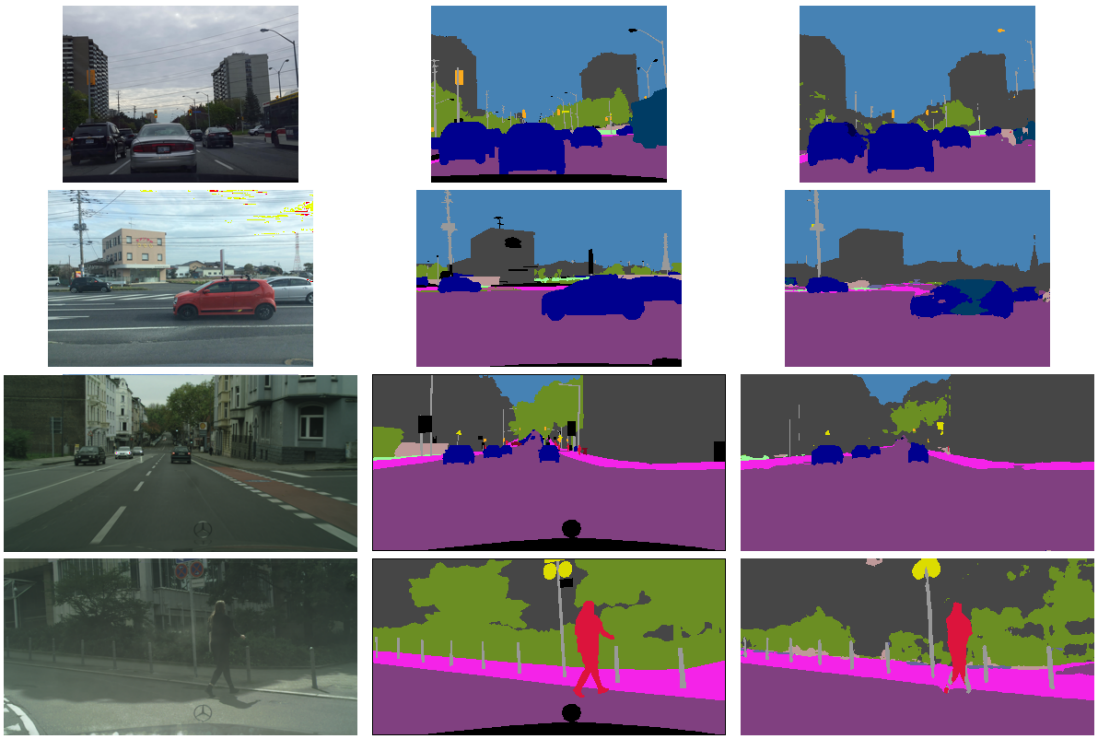

Figure 4.8: Hand picked examples of the two head model pretrained on CityScapes, on which the model performed well. The model continues to show great performance at segmenting road vehicles and picking up small details of traffic lights. However, it shows increased mistakes in detecting the vegetation class.

However, bad examples in fig. 4.9 show a peculiar degradation of performance. The number of false positives and negatives skyrocketed. It seems that the introduction of a different dataset for the incremental learning step confused the model completely, which now sees buildings (gray) and sidewalks (pink) all over the image, even in areas they do not usually appear. The most disturbing is the apparent loss of the model's ability to identify people (red), which are now predicted only sporadically.

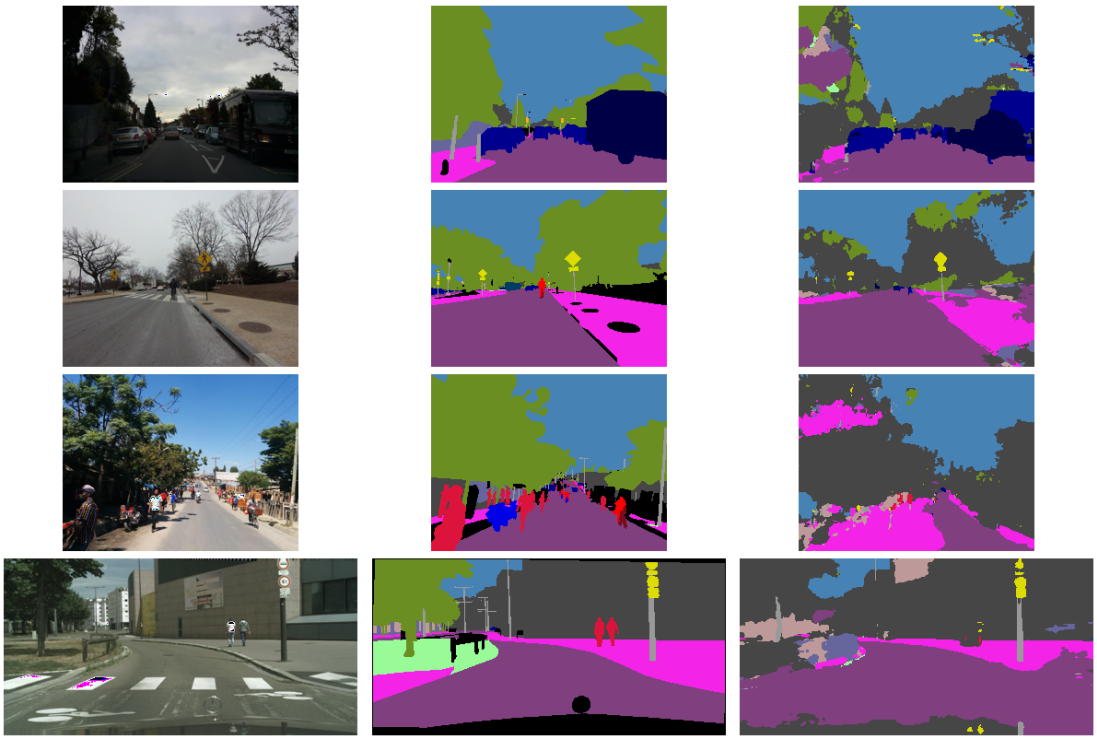

Figure 4.9: Hand picked examples of the two head model pretrained on CityScapes, on which the model performed poorly. In contrast with what this model presented earlier, it seems that it had some its knowledge destroyed, especially the vegetation recognition and human recognition, together with a severely degraded sidewalk prediction performance.

This degradation of performance after introducing outlier images indicates that the features the model had learned weren't, in fact, expressive enough. One could say the model learned to extract the wrong features, in the sense that the recognition of the strongest signal in the CityScapes dataset does not generalize well to other types. The model trained on the Vistas, as it'll be shown next, has it marginally better.

4.3.2. Vistas

The results of the two head model pretrained on Vistas are show in fig. 4.10. Curiously enough, the plot itself looks a lot like the plot fig. 4.7, with only slight deviations. The origin of the similarity is uncertain. Nevertheless, this mIoU also jumped for more than 15 points to just over 65%, and the pixel accuracy obtained high 93.66%, both on the validation set.

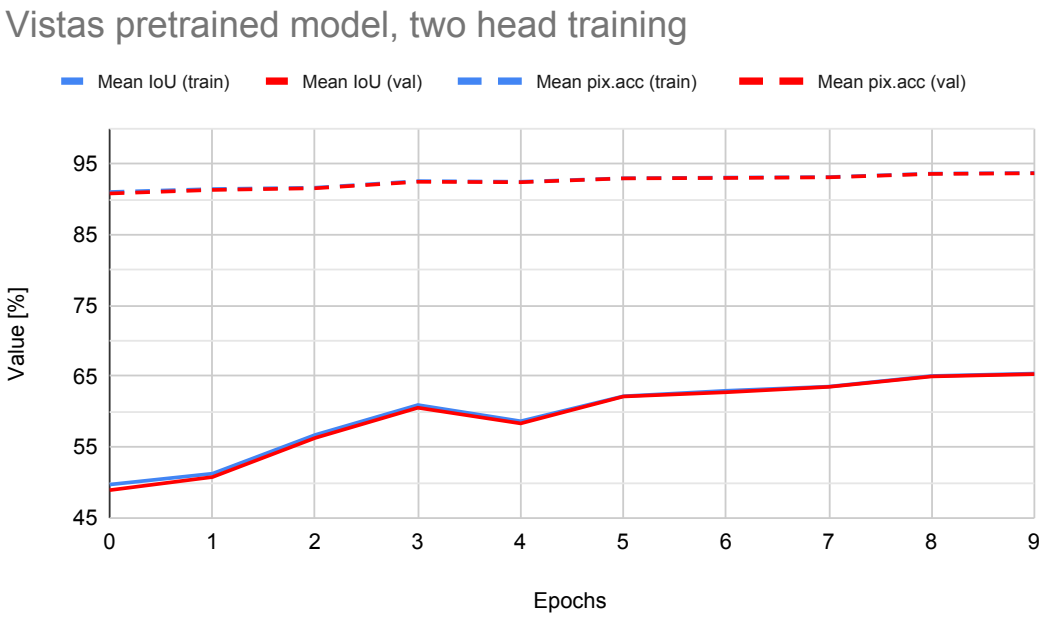

Figure 4.10: The progress of the second phase of incremental learning of the model pretrained on Vistas. Uncannily, this experiment progressed almost identically to the experiment with the model pretrained on CityScapes as shown on fig. 4.7. This model, too, gets a boost of more than 15 points of mIoU in 10 epochs, resulting with 65.31%.

Some example images on which the model performed well can be seen in figure fig. 4.11. The model shows more stable outputs and similar performance on small image patches in comparison with both the base model pretrained on Vistas, and the two-head model pretrained on CityScapes. Once again, the model does a great job segmenting the road vehicles and road signs, and slightly worse job segmenting the vegetation.

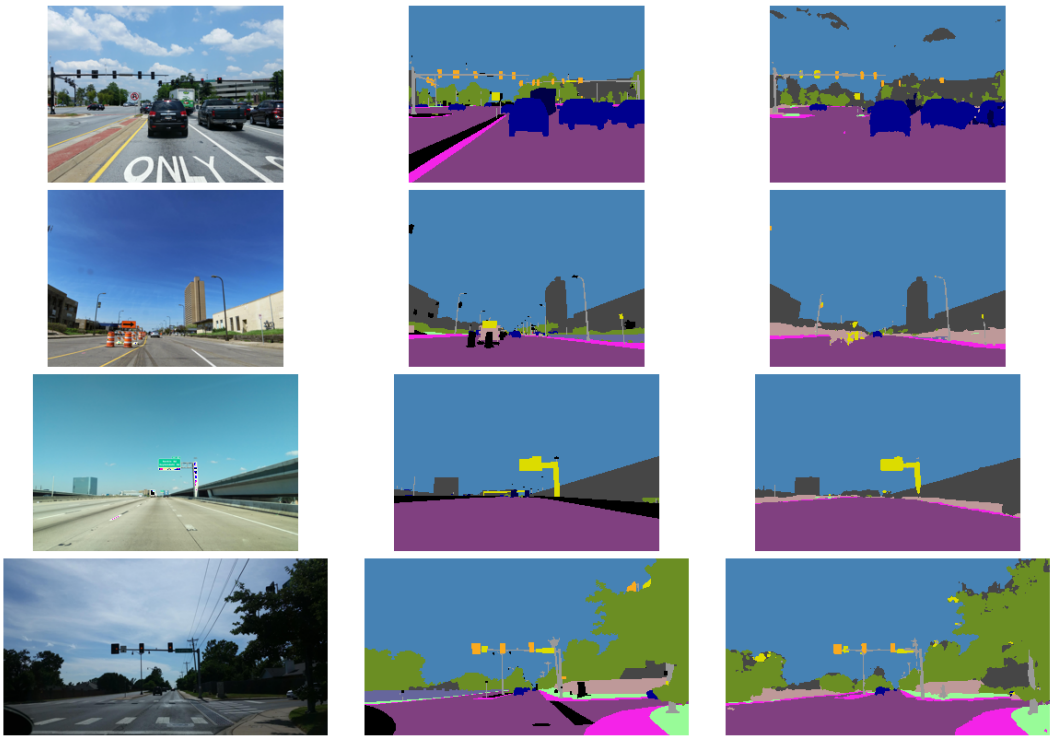

Figure 4.11: Hand picked examples of the two head model pretrained on Vistas, on which the model performed well. The model shows a competence in segmenting cars, traffic lights and building. A cute false positive appears in Example 1 in which the clouds are confused with buildings.

Bad examples are shown in fig. 4.12. Similarly to all the other models, there is some detail losing and prediction instability, especially with the vegetation class. However, after a closer look it can be asserted that these images are actually not that easy to recognized in the first place. The center right tree in Example 1 image isn't really obvious, and some humans (including the author) would've said that the label is wrong before a careful look at the image at the original resolution. Together with that, the people in the Example 4 image are not obvious at first due to the shadow, and the model reasonably predicted a building class (gray) for that image patch. Example 2 predictions are quite messy, though, and one possible explanation is that the image has more color contrast than other images, which confuses the model. Other predictions show the model does a good job at grouping semantically connected pixels. An interesting phenomenon is exhibited in the Example 3, where a car's reflection on the hood of the car with the dahsboard camera is segmented as a car (dark blue). While it's technically incorrect to label reflections of objects as the object's class, it does show a model has a somewhat generalization ability.

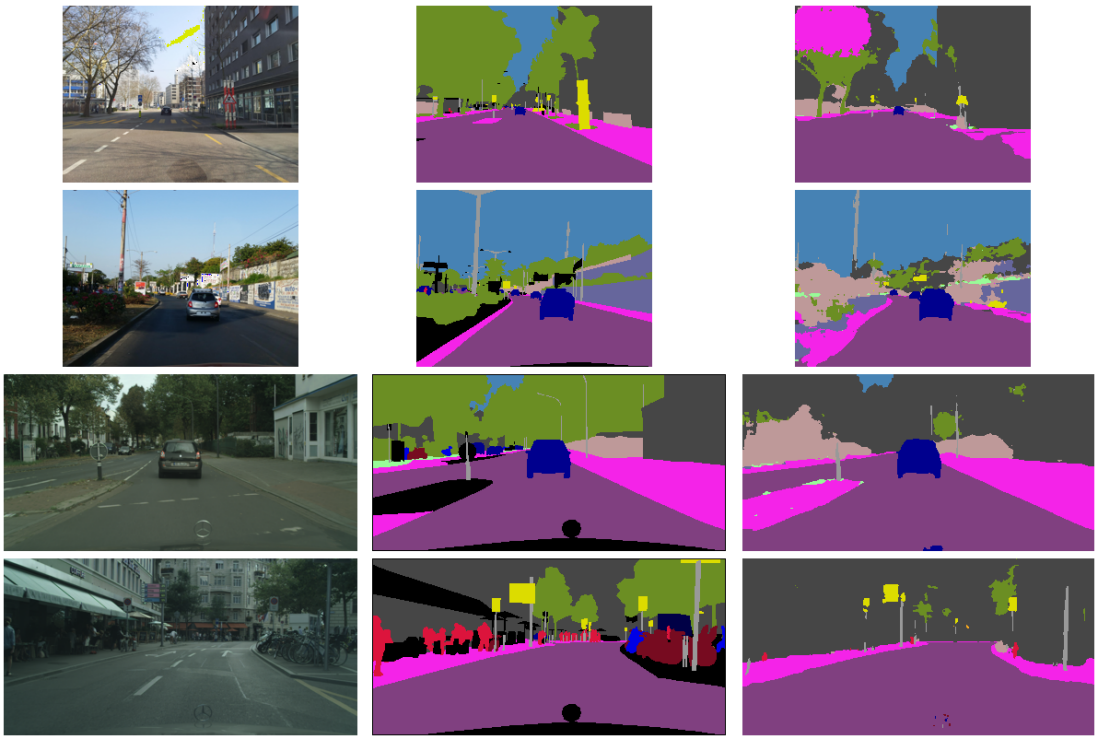

Figure 4.12: Hand picked examples of the two head model pretrained on Vistas, on which the model performed poorly. These prediction maps are, at first, riddled with false positives and negatives, however a closer look explains some of the model's decisions. For example, the missing people in the Example 4 are sort of hard to detect, just like the bicycles on the right. In the Example 1, the model should have predicted a vegetation, however the branches are hardly visible, while the building far away is quite visible. Example 2 seems challenging on its own solely due to strong color differences in the image, which trick the model. The author can not find an explanation for the sidewalk in the middle of the tree in Example 1, though. On the other hand, a cute phenomenon is exhibited in the Example 3, where a car's reflection on the hood of the car with the dashboard camera is segmented as a car (dark blue).

All in all, the two head model pretrained on the Vistas dataset shows a modest improvement in comparison with the two head model pretrained on the CityScapes dataset, which is most easily seen in the prediction stability. This points to the conclusion that the diversity of the Vistas dataset is actually useful for this task, allowing the feature extractor to learn better features which are well suited for the diversity of the ImageNet dataset.

However, these segmentation results, while important, are actually of a second importance in this thesis. The most important results are on the detection of outliers in the images, which will be shown in the upcoming section.

4.4. OOD results

4.4.1. Quantitative comparison of the models

In the table 4.3 there are 2 measurements of the average precision for the two models trained for the outlier detection. The evaluation was performed on the entire Fishyscapes lost and found dataset [4], since the model never saw them, and two syntetic datasets: WD-LSUN and WD-Pascal. They are synthetised by pasting the outliers (such as animals from Pascal, or random objects from LSUN) to the inliers of WildDash. It seems that the two head model pretrained on Vistas does significantly better than the two head model pretrained in CityScapes. One possible explanation for that is that the variety of the Vistas dataset helps the model to generalize and allows it to detect more inliers, while the specificity of the CityScapes dataset allows the model to overfit to it, hindering the ability for the model to recognize when something is an inlier.

Model	AP @ FSLF	AP @ WD-LSUN	AP @ WD-Pascal
TwoHead CityScapes @ 50	22.91%	21.47%	48.02%
TwoHead Vistas @ 47	24.15%	22.42%	49.06%

Table 4.3: The table of results of models evaluation on the Fishyscapes Lost and Found (FSLF), WildDash-LSUN combination (WD-LSUN) and WildDash-Pascal combination (WD-Pascal). WD-LSUN and WD-Pascal are both created in the same way: the outliers of LSUN and Pascal, respectively, are pasted into the inliers of WildDash. The model pretrained on Vistas seem so perform better than the model pretrained on CityScapes. These numbers inform that the second model is, on average, less eager to label inlier pixels as outliers. This can be explained by the Vistas' more diverse images. Note that the base models cannot be compared for an OOD detection as they weren't, in fact, even trained for that.

In the following images, the examples are row wise, such that the first column was the input image and the second column is the output outlier prediction, where the predicted outlier pixel is denoted with the white color, and the predicted inlier pixel is denoted with the black color. The targets mappings aren't shown as the reader is presumed to be perfectly capable of figuring the outliers out.

4.4.2. CityScapes

The predictions of the out-of-distribution head are shown in fig. 4.13. The model seems to be quite good at detecting the outliers, such as Example 1 & 2, but it sometimes it wrongly labels some inliers as outliers, for instance the reflections in Example 3 and the back of a truck in Example 4. It is possible that the model is slightly overfit to the geometry of the pasted outliers, as the back of a truck resembles axis aligned box, just as all the pasted images. A future work may include pasting geometrically transformed images as well, such as rotated and perspective-warped images.

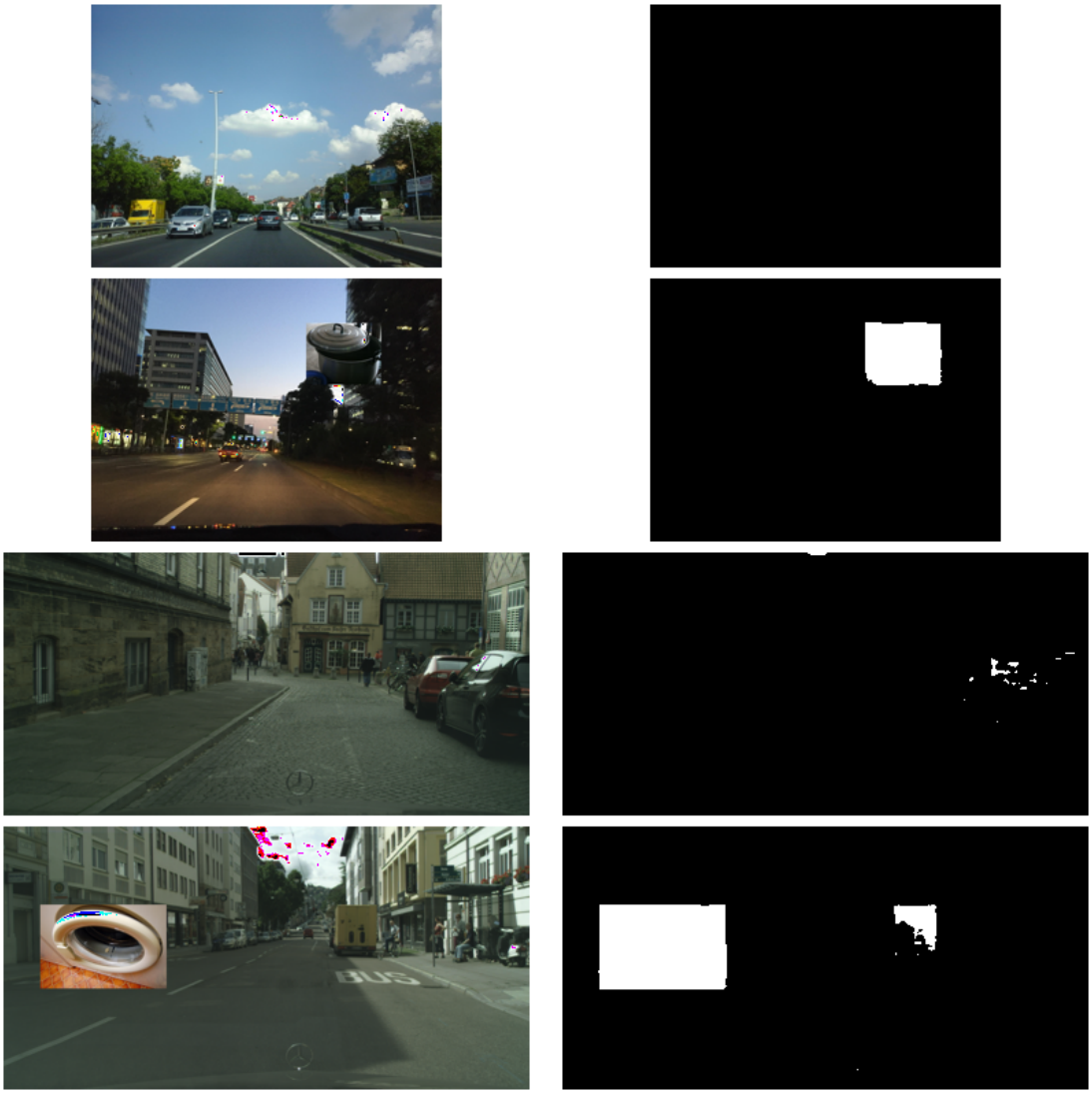

Figure 4.13: The predicted outlier maps of the model pretrained on CityScapes. The model is quite certain about the outliers but tends to produce false positives on the inliers, especially if the pixels are a part of an axis aligned rectangle, just like the pasted outliers. One possible solution is to paste geometrically transformed images.

The bad predictions of the model pretrained on CityScapes are in fig. 4.14. Similarity to the fig. 4.13, all the outliers are correctly labeled as outliers, however the number of false positives skyrocketed. A quick glance doesn't reveal any connection between the outlier label and the original image part, except that the outliers tend to follow the geometry in the images.

Figure 4.14: The poorly predicted outlier maps of the model pretrained on CityScapes. Just as in fig. 4.13, the true outliers are detected correctly and never missed, but the inliers are heavily misclassfied as outliers.

4.4.3. Vistas

The predictions of the out-of-distribution head are shown in fig. 4.15. The model does a good job segmenting the outliers, such as the entire Example 1, and obvious parts

of Examples 2 & 3. However, the model tends to produce quite a few false positives, labeling inlier pixels as outliers, such as a road sign and the dashboard in Example 2 and a traffic column in Example 4. The results are obtained by using the default decision threshold of 0.5, but some other decision threshold could in fact improve the performance.

Figure 4.15: The well predicted outlier maps of the model pretrained on Vistas. The model is quite certain about the outliers but tends to produce false positives on the inliers. The possible solution is to increase the decision threshold.

Bad predictions of the outliers of the model pretrained on Vistas are in the figure fig. 4.16. Once again, the outliers are predicted with no or small error, but the false positives count is too high. The situation seems marginally better than of the model

pretrained on CityScapes, as the outliers now follow the image geometry more (meaning the shapes in the input image and output map resemble each other). Still, that's not enough to conclude what in the input pixels pushed the model to label them as outliers. One solution is to train the models for longer than 10 epochs, which could shift the other activations towards inliers.

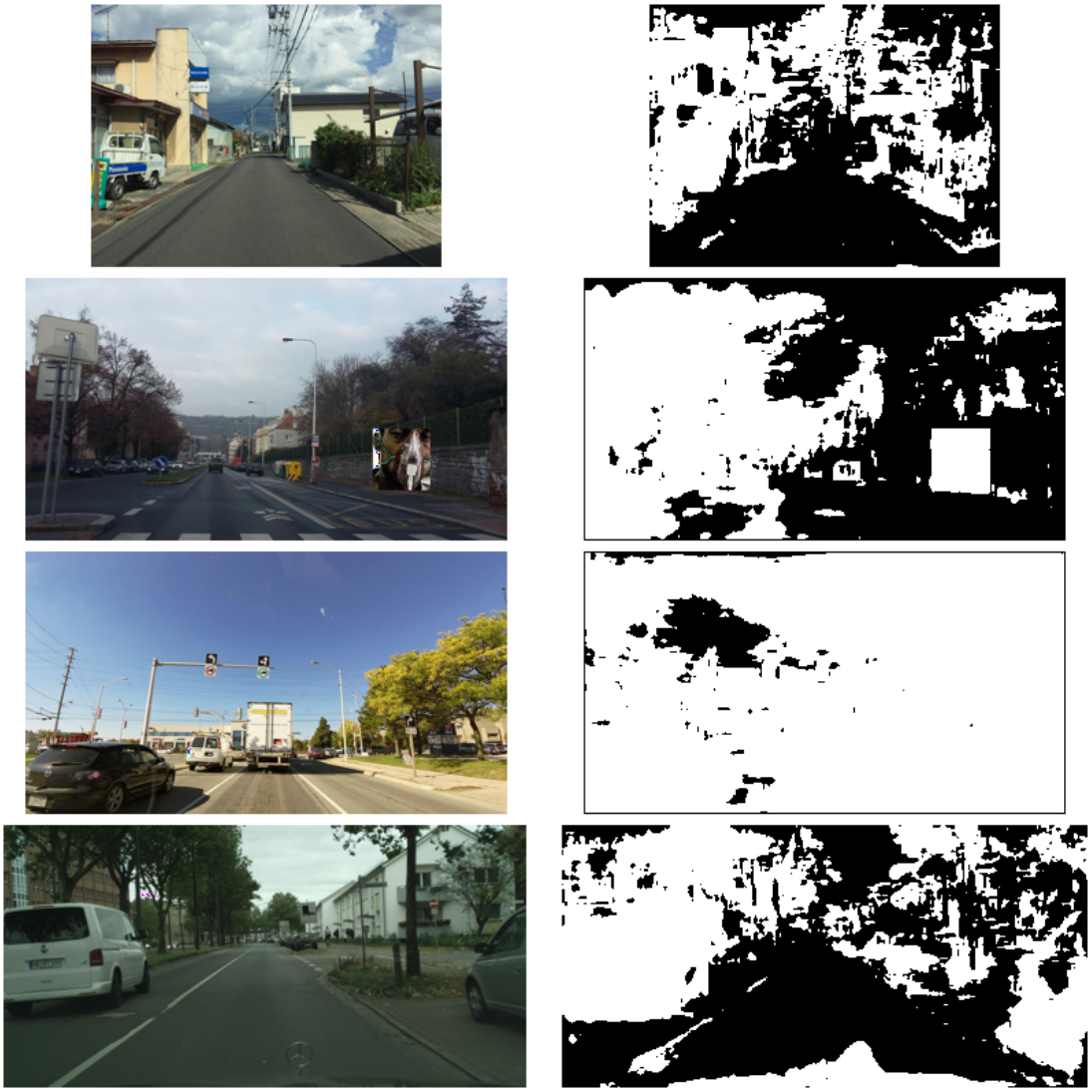

Figure 4.16: The poorly predicted outlier maps of the model pretrained on Vistas. While the performance seems slightly better than the CityScapes-pretrained model, at least in terms of label accuracy, there are still significantly more outliers falsely labeled in these images. A possible solution is to just train for more epochs.

5. Conclusion

The results of the experiments provided enough data for some conclusions to be made. First and foremost, it is possible to apply incremental learning for the task of semantic segmentation of the dashboard camera video feeds. However, once again it's been proved that the data the model is trained on is of utmost importance. CityScapes dataset is less diverse than the Vistas dataset, so, naturally, the model didn't have the opportunity to "prepare" for the upcoming task of outlier detection. Since the images weren't augmented in any way, future work may include experimenting with augmentations of images (some color augmentations, e.g. color jitter or JPEG compression simulation) or image/target semantic map pairs (some geometric augmentations, e.g. perspective transformations and warps).

Due to the insufficient resources, two head experiments were performed on a subsampled resolution in comparison with the pretraining task. This tends to, and did, hurt the performance of the semantic segmentation head, but not as much. The outlier detection head detects the outliers well, which indicates that the outlier detection on the extracted features is an easy problem. To confirm this hypothesis, the same set of experiments should be performed at a full resolution to notice if there is any significant improvement in the model performance.

And while the outlier detection head rarely misclassifies the outlier pixels as inliers, it does seem eager to unnecessary label pixels as outliers. In future, this may improve with data augmentations, larger resolution and longer training times, but a threshold tuning might be needed.

6. Summary

The goal of this thesis was to successfully train a deep learning system which will segment the dashboard camera feed in real time, but also provide an estimate of its uncertainty. This was attempted by using incremental learning. Incremental learning is a technique of training deep learning algorithms in which the model is prepared for the final task in phases. It is a form of transfer learning, in which the model trained for one task is used for another task, with the belief that such model is a good starting point for the task at hand.

After an overview of the semantic segmentation field and obsolete methods, deep learning concepts are introduced, and several of the most important datasets are described. Next, the core method and its building blocks are described in detail, following a precise description of the experiments to be performed. Next, described experiments are reviewed, first quantitatively, in terms of training progressions and maximal metrics, and then qualitatively, by showing hand picked examples of good and bad model performance and their analysis. Afterwards, the same qualitative analysis is done for the outlier maps, with the addition of proposing solutions.

BIBLIOGRAPHY

[1] Lauren Barghout i Jacob Sheynin. Real-world scene perception and perceptual organization: Lessons from computer vision, Jul 2013. URL https://doi.org/10.1167%2F13.9.709.

[2] Petra Bevandić, Ivan Krešo, Marin Orši, i Siniša Šegvić. Discriminative out-of-distribution detection for semantic segmentation, 2018.

[3] Petra Bevandić, Ivan Krešo, Marin Oršić, i Siniša Šegvić. Dense outlier detection and open-set recognition based on training with noisy negative images, 2021.

[4] Hermann Blum, Paul-Edouard Sarlin, Juan Nieto, Roland Siegwart, i Cesar Cadena. The fishyscapes benchmark: Measuring blind spots in semantic segmentation. *International Journal of Computer Vision*, Sep 2021. ISSN 1573-1405. doi: 10.1007/s11263-021-01511-6. URL http://dx.doi.org/10.1007/s11263-021-01511-6.

[5] Gabriel J. Brostow, Julien Fauqueur, i Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, xx(x):xx–xx, 2008.

[6] Gabriel J. Brostow, Jamie Shotton, Julien Fauqueur, i Roberto Cipolla. Segmentation and recognition using structure from motion point clouds. U *ECCV (1)*, stranice 44–57, 2008.

[7] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986. doi: 10.1109/TPAMI.1986.4767851.

[8] Wikimedia Commons. File:intersection over union - visual equation.png — wikimedia commons, the free media repository, 2020. URL https://commons.wikimedia.org/w/index.php?title=File:

Intersection_over_Union_-_visual_equation.png&oldid=466295636. [Online; accessed 27-August-2021].

[9] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, i Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. U *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[10] Mark Everingham, Luc Van Gool, C. K. I. Williams, J. Winn, i Andrew Zisserman. The pascal visual object classes (voc) challenge, 2010.

[11] Jannik Fritsch, Tobias Kuehnl, i Andreas Geiger. A new performance measure and evaluation benchmark for road detection algorithms. U *International Conference on Intelligent Transportation Systems (ITSC)*, 2013.

[12] Xavier Glorot i Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. U Yee Whye Teh i Mike Titterington, urednici, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, svezak 9 od *Proceedings of Machine Learning Research*, stranice 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/glorot10a.html.

[13] Divam Gupta. A beginner's guide to deep learning based semantic segmentation using keras, Jun 2019. URL https://divamgupta.com/assets/images/posts/imgseg/image4.png.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *Lecture Notes in Computer Science*, stranica 346–361, 2014. ISSN 1611-3349. doi: 10.1007/978-3-319-10578-9_23. URL http://dx.doi.org/10.1007/978-3-319-10578-9_23.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Deep residual learning for image recognition, 2015.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[17] Gao Huang, Zhuang Liu, Laurens van der Maaten, i Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[18] Paul Jaccard. The distribution of the flora in the alpine zone.1. *New Phytologist*, 11(2):37–50, 1912. doi: https://doi.org/10.1111/j.1469-8137.1912.tb05611.x. URL https://nph.onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-8137.1912.tb05611.x.

[19] Jongmin "Jeong, Tae Yoon, i Jin" Park. "towards a meaningful 3d map using a 3d lidar and a camera". *"Sensors"*, "18":"2571", "08" "2018". doi: "10.3390/s18082571".

[20] R. Kimmel i A.M. Bruckstein. Regularized laplacian zero crossings as optimal edge integrators, 2003. URL https://link.springer.com/article/10.1023%2FA%3A1023030907417.

[21] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, i Piotr Dollár. Panoptic segmentation, 2019.

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, i Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[23] D. Marr, E. Hildreth, i Sydney Brenner. Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 207(1167):187–217, 1980. doi: 10.1098/rspb.1980.0020. URL https://royalsocietypublishing.org/doi/abs/10.1098/rspb.1980.0020.

[24] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulò, i Peter Kontschieder. The mapillary vistas dataset for semantic understanding of street scenes. U *2017 IEEE International Conference on Computer Vision (ICCV)*, stranice 5000–5009, 2017. doi: 10.1109/ICCV.2017.534.

[25] Ron Ohlander, Keith Price, i D. Raj Reddy. Picture segmentation using a recursive region splitting method. *Computer Graphics and Image Processing*, 8(3):313–333, 1978. ISSN 0146-664X. doi: https://doi.org/10.1016/0146-664X(78)90060-6. URL https://www.sciencedirect.com/science/article/pii/0146664X78900606.

[26] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979. doi: 10.1109/TSMC.1979.4310076.

[27] Olaf Ronneberger, Philipp Fischer, i Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, i Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[29] Wikipedia contributors. F-score — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=F-score&oldid=1030531397`, 2021. [Online; accessed 23-August-2021].

[30] Oliver Zendel, Katrin Honauer, Markus Murschitz, Daniel Steininger, i Gustavo Fernandez Dominguez. Wilddash - creating hazard-aware benchmarks. U *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

Incremental open-set recognition for semantic segmentation

Abstract

Semantic segmentation is an important task of computer vision with many interesting applications. Currently, the state of the art in the field is achieved by discriminative convolutional models. Unfortunately, discriminative models are prone to unwarranted optimism. Consequently, outliers are often misclassified with very small uncertainty. This thesis attempts to solve the problem by incremental learning with respect to a noisy negative set that approximates distribution of the entire visual world.

Keywords: Image segmentation, open set, out of distribution, incremental learning, deep learning

Inkrementalno učenje semantičke segmentacije nad otvorenim skupom razreda

Sažetak

Semantička segmentacija slika važan je zadatak računalnog vida s mnogim zanimljivim primjenama. U posljednje vrijeme najbolji rezultati u tom području postižu se diskriminativnim konvolucijskim modelima. Međutim, diskriminativni modeli su skloni neopravdanom optimizmu, što znači da primjerci izvan domene ekspertize modela često bivaju neispravno klasificirani s vrlo malom nesigurnošću. Ovaj rad proučava mogućnost rješavanja tog problema diskriminativnim učenjem s obzirom na šumoviti negativni skup koji aproksimira distribuciju cjelokupnog vizualnog svijeta.

Ključne riječi: Segmentacija slike, otvoreni skup, izvandistribucijski primjeri, inkrementalno učenje, duboko učenje