CITY UNIVERSITY
LONDON

Deep Learning for Image Analysis

Written Report

Martin Fixman and Grigorios Vaitsas

2023/2024 Term

1 Introduction

1.1 Overview

In this study we have deployed the freely available Cityscapes dataset [1] to perform semantic segmentation analysis on urban scene imagery using a variety of Deep Learning architectures. Semantic segmentation is a computer vision technique used to identify and label specific parts of an image at pixel level. The main goal is to partition an image into areas (segments) that correspond to different objects or classes of interest (for example person, car, tree, road, building etc.). This technique is widely used in various fields like for instance, in autonomous driving, where it helps cars understand the environment around them, as well as in medical imaging, remote sensing, and robotics. Semantic segmentation is typically performed using deep learning methods, particularly convolutional neural networks (CNNs) and their variants as well as more recent attention-based models.

1.2 Dataset

The Cityscapes dataset is a large-scale dataset widely used for training and evaluating algorithms in the fields of computer vision, particularly for semantic understanding of urban street scenes. It consists of high resolution images captured from a vehicle-mounted camera while driving through 50 different cities in Germany and neighbouring countries. The capturing apparatus used was equipped with a dual camera for stereoscopic capability, allowing for the possibility of applying stereo vision techniques useful for tasks like depth estimation, 3D reconstruction and scene understanding.

The images that are used in our analysis are located in the folder **leftImg8bit**. These, as the name suggests, are taken from the left camera of the stereo camera system. They form the main high-resolution images (2048×1024 pixels) with an 8-bit color depth that are used for all semantic and instance segmentation tasks using this dataset. Each image is stored in a PNG file format with a size that varies between 2.0 and 2.5 MB. The images have an RGB color space, which means that there are 3 color channels for each image.

The dataset comes with two different types of image annotations:
Fine annotations: Figure 1(b) These provide detailed, pixel-level annotations for 5,000 images which have labels for 30 classes such as road, car, pedestrian, building, and traffic light. This dataset is split into 2,975 annotations for training, 500 for validation and 1,525 for testing . It is worth mentioning that the ground truth annotations are not available to the users and this happens in order to allow for fair evaluation and benchmarking of models. Users instead need to submit their code to an online evaluation server which provides the quantitative performance metrics.

Coarse annotations: Figure 1(a) These include a much larger set of 20,000 images with less precise annotations intended for data augmentation and for tasks where granularity is not critical. The coarse annotations are split into a training and validation set. No

test set is provided since this is only done using the fine annotations set.

(a) Coarse mask **(b)** Fine mask

Figure 1: Examples of coarse (a) and fine (b) mask annotations superimposed on the corresponding original images

The scenes in the CityScapes dataset represent a variety of urban settings, seasons, daylight conditions, and weather scenarios, providing robust, real-world environments for training models that need to perform under varied conditions. This dataset has been widely used in research for developing and testing algorithms on tasks such as object detection, semantic segmentation, and instance segmentation in urban settings as well as advancing the state-of-the-art in visual perception for autonomous driving systems. The Cityscapes dataset can be accessed in the following address:

`https://www.cityscapes-dataset.com`

Our particular approach in this study has been to start off by training a fairly simple model, that forms our baseline model, and then experiment with more complex architectures. We have first started off by demonstrating that our models are able to perform semantic segmentation on the chosen dataset. Our second goal was to investigate how the choice of various hyper-parameters and tweaks in architecture can affect the accuracy and performance of the models. The architectures that we have deployed in our study are briefly outlined in the section below.

1.3 Architectures

Fully Convolutional Network

Modern, state-of-the-art techniques for image analysis involve convolutional neural networks (CNNs) in their implementation. CNNs can be used to extract vital information from spatial features, which allow us to classify and segment objects in images. A type of neural network architecture designed specifically for semantic segmentation is the Fully Convolutional Network (FCN). This architecture, developed by researchers from UC Berkeley in 2014 [2], has been influential in advancing the field of computer vision, particularly in tasks requiring dense prediction, like semantic segmentation.

Unlike standard convolutional neural networks used for image classification, which typically end with fully connected layers, FCNs are composed entirely of convolutional layers. This design allows them to take input of any size and output segmentation maps that correspond spatially to the input image, providing a per-pixel classification. FCNs

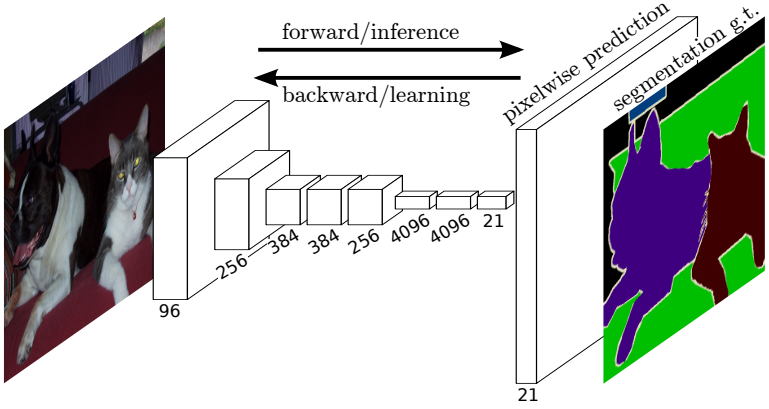


Figure 2: Example of a fully convolutional network used for semantic segmentation [2]

transform the fully connected layers found in traditional CNNs (like those in AlexNet or VGGNet) into convolutional layers (Figure 2). This is done by treating the fully connected layers as convolutions with kernels that cover the entire input region. For example a fully connected layer that accepts an input of size 16×16 can be reimagined as a convolutional layer with a 16×16 filter size. To generate output segmentation maps that match the size of the input image, FCNs use transposed convolution layers (also known as deconvolutional layers) for upsampling. This process helps in recovering the spatial dimensions that are reduced during the pooling or convolutional operations in earlier layers. FCNs often utilize skip connections to combine deep, semantic information from lower layers with the shallow, appearance information in the upper layers of the network. This helps in improving the accuracy and detail of the output segmentation maps, as it allows the network to use both high-level and low-level features.

U-net

Further improving upon the architecture of the fully convolutional network, the U-net, first introduced in a paper by Olaf Ronneberger, Philipp Fischer and Thomas Brox in 2015 [3], has become very popular due to its efficiency and effectiveness, especially where data is limited. U-Net's architecture is distinctive because of its symmetric shape, which resembles the letter "U" (see Figure 3). It consists of two main parts: the left side of the U-net is the contracting path (encoder) which is designed to capture the context of the input image. This path is essentially a typical convolutional network that consists of repeated application of convolutions, followed by rectified linear unit (ReLU) activations, and max pooling for downsampling. Each of these steps helps in capturing the features at different levels of abstraction. On the right side of the network is the expansive path (Decoder), which performs the task of precise localization using transposed convolutions for upsampling. This part of the network gradually recovers the spatial dimensions and detail of the input image. The upsampling in the expansive path is combined with the high-resolution features from the contracting path via skip connections.

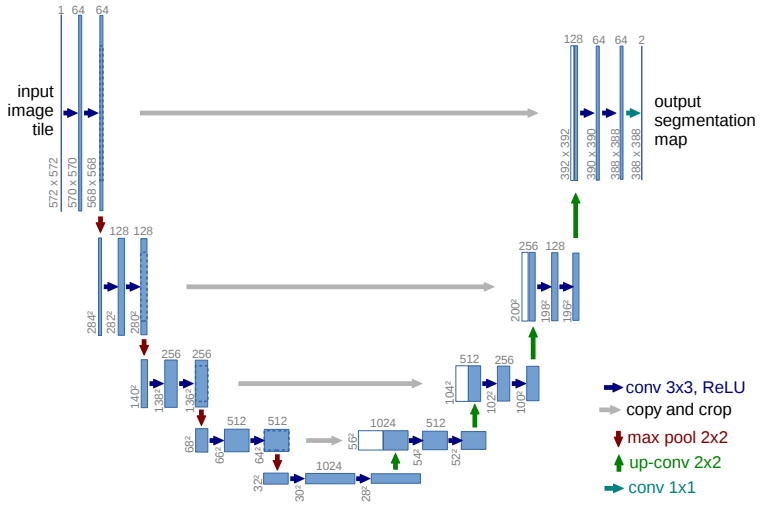


Figure 3: Example of the original U-net architecture as presented in [3]

These connections are crucial as they help in propagating context information to higher resolution layers, which enhances the accuracy of the output segmentation maps. U-Net's design is particularly effective in providing good segmentation results even with small training datasets, which has led to its widespread adoption and adaptation in different segmentation tasks beyond its initial medical imaging focus.

Vision and Swin Transformers

The last architecture we explored in our study was a Vision Transformer (ViT), which is a novel approach to applying transformer models, primarily used in Natural Language Processing (NLP) tasks, to image recognition. In the paper "An Image is worth 16×16 Words: Transformers for Image Recognition at scale" [4], the authors proposed a method that treats an image as a sequence of fixed-size patches (similar to words in NLP). Each patch is embedded similarly to how tokens are embedded in NLP, and then processed by a standard transformer encoder. An overview of this model is depicted in Figure 4.

Vision transformers can achieve excellent results, outperforming many advanced CNN architectures, particularly when pre-trained on very large datasets. However, the computational complexity especially when applied on high-resolution images increases rapidly, especially for dense prediction tasks like semantic segmentation. For this reason, an improved approach has been suggested called the Swin Transformer model [5]. In contrast to the Vision Transformer that processes the entire image at once by dividing it into fixed-size patches and applying global self-attention to all patches, the Swin transformer divides the image into smaller, non overlapping windows and applies local window-based self-attention (Figure 5(a)). This reduces the computational complexity because the number of interactions is limited to within a window rather than across the entire image. Additionally, the Swin transformer introduces a novel mechanism of shifting

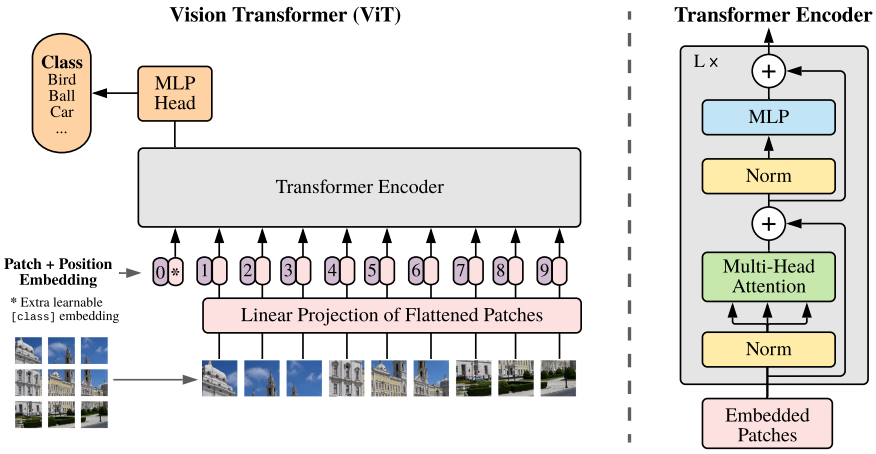
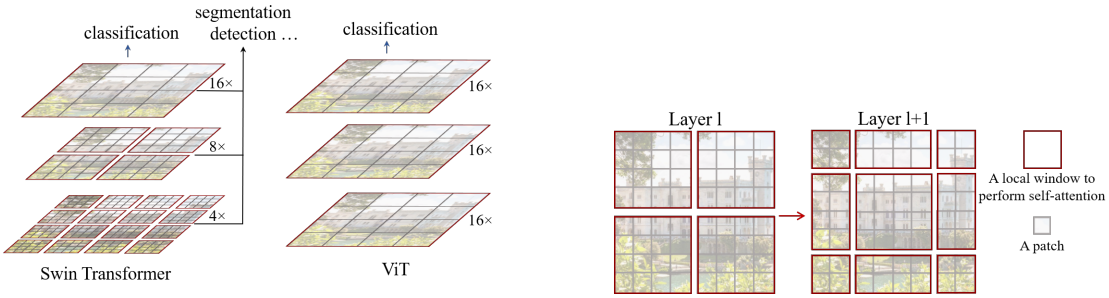


Figure 4: Model overview of a Vision Transformer [3]

these windows in alternate layers, thus enabling cross-window connections. This helps in integrating local and global contextual information more effectively (Figure 5(b)). Finally, the fact that the Swin transformer starts with smaller windows and gradually merges them, reducing resolution while increasing the depth of features, enables a hierarchical architecture that processes features at multiple scales. This is beneficial for various vision tasks like object detection and segmentation, where multi-scale features are crucial. These advantages make the Swin transformer suitable as a general purpose backbone for various vision tasks and as will be explained later, we found this model to perform very well in our particular study.



(a) ViT vs Swin transformer

(b) The shifted window technique

Figure 5: Characteristics of the Swin transformer[5]

2 Training Methodology

2.1 Transforming and Enhancing Data

In our final runs, the images in the input data are resized to 768×768. This size presents a good balance between model performance and training speed: the final results of the models were almost identical to using the entire size of 2048×1024. Additionally, this new size makes it possible to use pretrained Swin2 in the entire data instead of having to separate the images of stretch it separately.

The loss of the final mask is calculated on the small, resized image. When evaluating, the final mask is stretched to its original larger size.

Additionally, we experimented with creating random transforms of the input data, including random flips and random crops.

In different experiments these transforms either replaced or augmented the original training data. However, this is not present in the final models as their results weren't conclusive: since the original data is complete enough, adding these transforms only made the results on the validation set worse.

2.2 Loss function

The choice of loss function is crucial for the performance of our model.

In this assessment, we experimented with three different loss functions.

Categorical Cross-Entropy Loss
: All pixels (except the ones marked as background) are equally weighted.

Intersection over Union Loss
: A differentiable version of IoU score, which should ideally work to maximise it.

Dice Loss
: Categories that appear less often are weighted higher[6].

Using **Categorical Cross-Entropy** provided better results in the relevant metrics, including IoU score, when compared to both dice loss and IoU loss.

This surprising result, which can be seen in Figure 6, is likely due to the difference in depth of the loss functions. Since it contradicts some existing literature[6], it warrants more exploration later.

2.3 Optimiser and Scheduler

All mature models use the **AdamW** optimiser, which decouples weight decay from the optimisation process[7]. This model produced a more effective L_2 regularisation than other similar classifiers, and generally better results than both the Adam and Adamax optimiser.

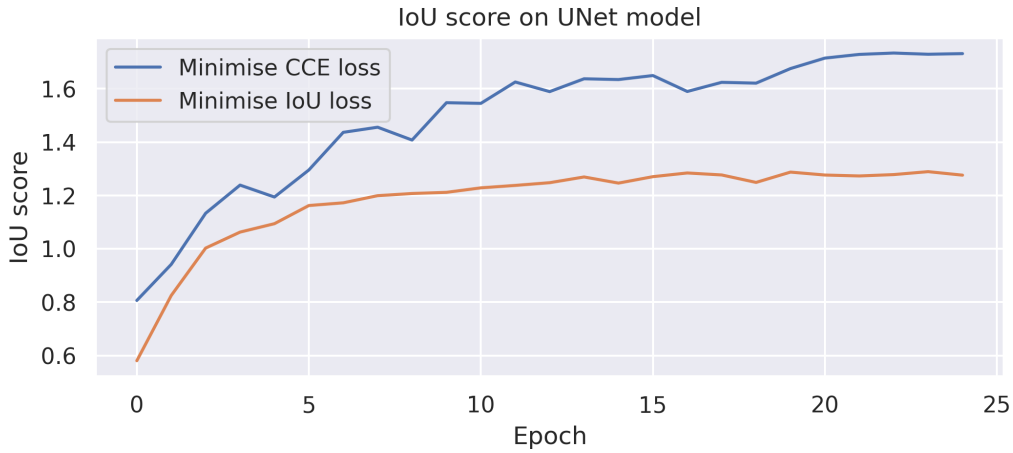


Figure 6: IoU score for UNet models trained to minimise both categorical cross-entropy and a related IoU loss depending on the epoch. Surprisingly, the one that optimises CCE has better results.

Initial experiments produced a clear overfit in most complex after about 10 steps, which became noticeable after 20. To prevent this overfitting, among many other solutions, we implemented a **Step Scheduler** which lowers the learning rate every 10 steps by a certain factor, γ. This can be seen in Figure 7.

The ideal value of γ depends on other hyperparameters, and was thus found in a hyperparameter sweep in **??**.

2.4 Early Stopping

Our input data is large and, even with many different kinds of regularisation, all models overfit after so many epochs.

To prevent this we only capture the classifier trained up to the epoch with the lowest validation loss. This returns a model that's trained on subtle details that could help identify each pixel as a different class, but ignores noise on the training data that leads it to overfit.

2.5 Halving Hyperparameter Sweep

Some hyperparameters, such as the AdamW optimiser, were set early on as they produced consistently good results.

Others can change the loss function in unpredictable ways. In order to test all of them, for the two enhanced models we run a hyperparameter sweep with the parameters in Table 1.

The entire dataset of 2994 images is very large, and making this 36-parameter sweep cost-prohibitive.

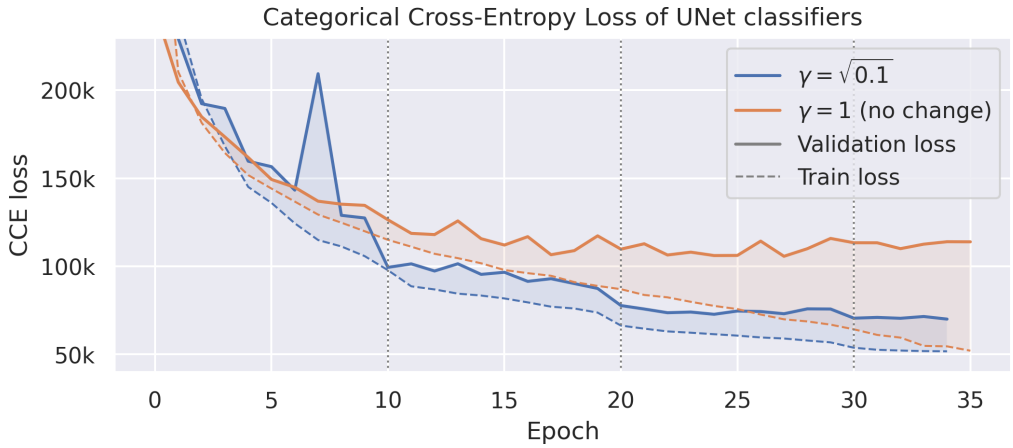


Figure 7: Differences between training and validation loss of a classifier that dynamically adjusts its learning rate every 10 steps and one that doesn't. While both classifiers overfit, the dynamic learning rate makes this process slower and the eventual validation loss reaches better values.

Hyperparameter	Options		
Initial Learning Rate	10^{-3}	10^{-4}	
Learning Rate Decay γ	1	$\sqrt{0.1}$	0.1
L_2 Weight Decay	0	10^{-4}	
Dropout	0	0.05	0.1

Table 1: Parameter list for hyperparameter sweep

To run a parameter sweep, we do a **Halving Parameter Sweep**[8].

1. Train classifiers for 20 epochs using a randomly sampled subset of the data for each combination of hyperparameters.
2. Split the results in two. Discard the half with the highest validation loss, and keep the half with the lower loss.
3. Double the amount of data used for the training, and repeat.

This method makes it possible to discard the least promising results early on, while focusing most time and computing power on the most promising candidates. The top 2 candidates are trained using the whole dataset.

The results can be found in **????**, with the best parameters in Table 2.

Model	LR	γ	L_2	Dropout	Val Loss
UNet					
Swin2-J					

Table 2: Best parameters for parameter sweep

3 Results

4 Conclusions

5 Reflections

References

[1] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016.

[2] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[5] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *CoRR*, abs/2103.14030, 2021.

[6] Carole H. Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M. Jorge Cardoso. *Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations*, page 240–248. Springer International Publishing, 2017.

[7] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[8] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. *arXiv*, 2015. [This was later implemented in SKLearn's `HalvingGridSearchCV`, from which I took the implementation idea].