CITY UNIVERSITY
LONDON

Deep Learning for Image Analysis

Written Report

Martin Fixman and Grigorios Vaitsas

2023/2024 Term

1 Introduction

1.1 Overview

Deep learning has revolutionised the field of computer vision in profound ways and has allowed many sectors that rely on image understanding to make significant progress. Computer vision tasks can be broken down into three main categories, image classification, object detection and localisation and semantic segmentaion. Semantic segmentation (also referred to as dense prediction) is a computer vision technique used to identify and label specific parts of an image at pixel level. The main goal is to precisely partition an image into areas (segments) that correspond to different objects or classes of interest (these could be 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. 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. Our goal is to train a model that will be able to receive an image of an urban landscape as input and produce as output an overlay mask that will identify the different types of objects with a high degree of accuracy. This task is of high importance, especially as many car manufacturers are trying to make their cars safer for their passengers as well as pedestrians and other road users. A system that uses this type of deep learning technology could be used for example to alert drivers when a pedestrian or a bicycle is detected in the projected path of the car.

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 lens 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 lens 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 for the testing set are not available to the users 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, testing and benchmarking new algorithms for computer vision tasks, gaining a place alongside datasets as iconic as ImageNet, COCO and Pascal VOC. 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 relatively simple model, that forms our baseline model, and then experiment with increasingly complex architectures. We aim to demonstrate that all these models are able to perform the task and further than that 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.

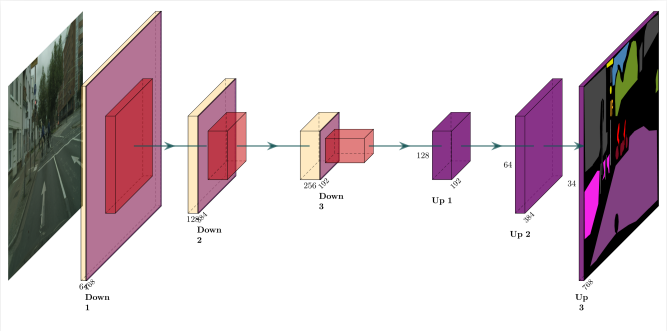

Figure 2: The architecture of the baseline model

2 Model Architectures

2.1 Baseline model: Fully Convolutional Network `models/SimpleFCN.py`

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 [4], has been influential in advancing the field of computer vision, particularly in tasks requiring dense prediction, like semantic segmentation.

Our baseline model, illustrated in Figure 2, is a simple fully-convolutional network with 3 encoding and 3 decoding steps.

Each encoding step

1. Conv2D layer with kernel size 3×3 and simple padding to double of channels.
2. ReLU layer to add some non-linearity.
3. Max-pooling layer with kernel size 2×2 to ensure the following encoding layer gathers less local data.

Each decoder step

1. Transposed convolution with kernel size 2×2, which halves the amount of channels and doubles the size of the image.

This model produces a result that's *good enough*, but it can be improved by adding a bit of complexity.

Following are two enhanced models, both with a significantly better result.

2.2 Enhanced Model 1: UNet `models/UNet.py`

The first enhanced model for our image segmentation task is a UNet model[?].

This produces an encoder-decoder architecture that's not too dissimilar to our baseline model, but it has the addition of skip connections so that the transposed convolutions get data with different resolution as seen in Figure 3.

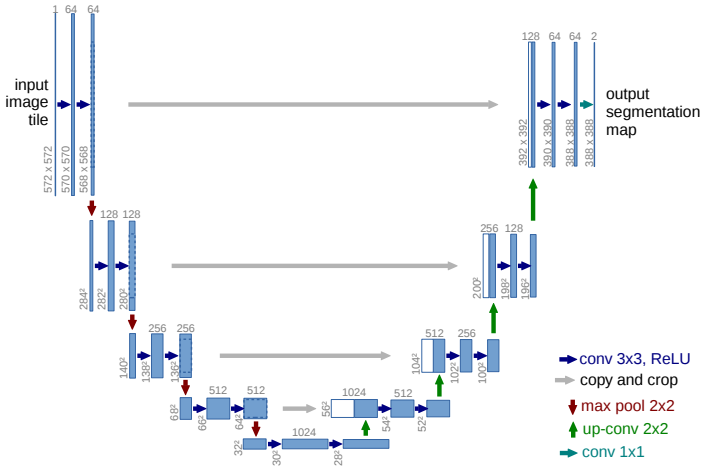

Figure 3: UNet architecture

In practice, these skip connections give the model a significant improvement in image segmentation, where precise localisation is critical and getting both high-resolution and low-resolution images in each layer can help it identify each part better.

The best hyperparameters for this model and the following one were found in a parameter sweep in Section 3.5.

2.3 Enhanced Model 2: Swin transformer + JNet `models/Swin2Base.py`

Vision Transformers were introduced by the Google Brain team in 2020[?]. To train this model an image is split into 16×16 patches, each of which is flattened and sent to an attention head. Each attention head makes its own image segmentation prediction, and all the predictions are finally patched together.

Vision transformers have had good results in many image analysis tasks[?]. However, many tests in our codebase of Cityscapes didn't produce results on models using them, so an alternative was needed.

Instead, we use the attention mechanisms in the Swin2 transformers[?] to encode the relevant features of the image. Swin2 uses a more refined architecture mechanism that ViT, such as shifted windows to ensure that the attention heads from one of the patches can be used in the next.

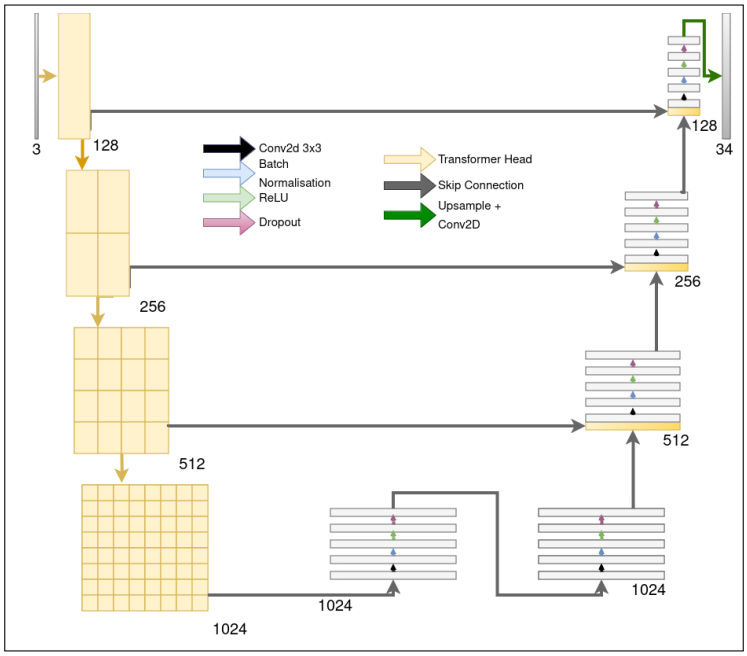

Figure 4: Rough outline of Swin2-JNet architecture in use for the Enhanced Model 2. The encoder is a Swin2 tranformer with pre-trained weights; the decoder is a series of upscaling convolutions

Like most transformers architectures, Swin2 requires a lot of data and time to be trained into something relevant. Instead of doing this, we leverage transfer learning by using pre-trained weights in the encoder.

The base model of Swin2 encodes the data in 4 attention heads, each one with a smaller image containing more channels of data. We can leverage this data by getting each of the 4 output feature maps at each step of the way, and adding them as skip connections to up-convolutions in a similar way to UNet.

Additionally, we noticed that this model tends to overfit quicker than other UNet model. To prevent this, we added a both a *Dropout* and a *Batch Normalisation* layer to each convolutional block; these normalisations tend to improve the performance of the model.

The ideal ratio of dropout was found in a parameter sweep, along with most other parameters, in Section 3.5.

3 Training Methodology

3.1 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.

3.2 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[**?**].

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 5, is likely due to the difference in depth of the loss functions. Since it contradicts some existing literature[**?**], it warrants more exploration later.

3.3 Optimiser and Scheduler

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

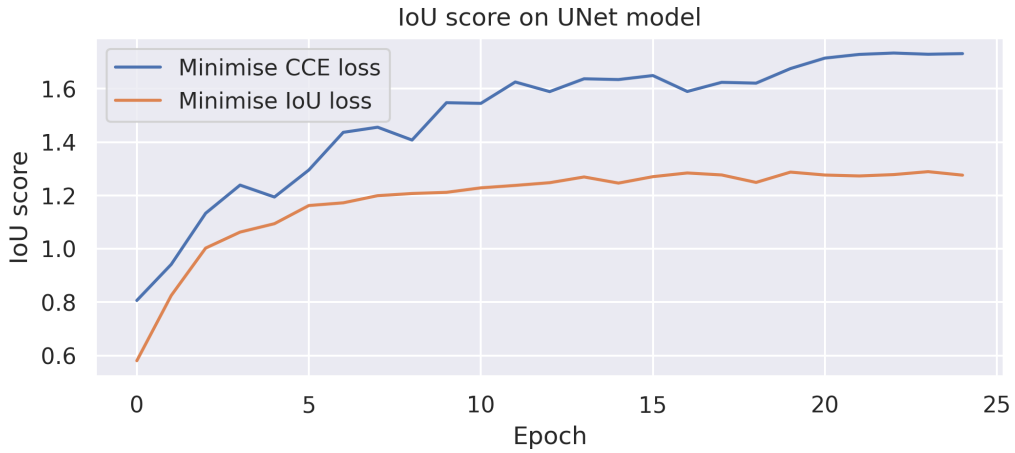

Figure 5: IoU score for UNet models trained to minimise both categorical cross-entropy and a related IoU loss depending on the epoch. Surprisingly, the one that optimises CCE has better results.

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 6.

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

3.4 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.

3.5 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.

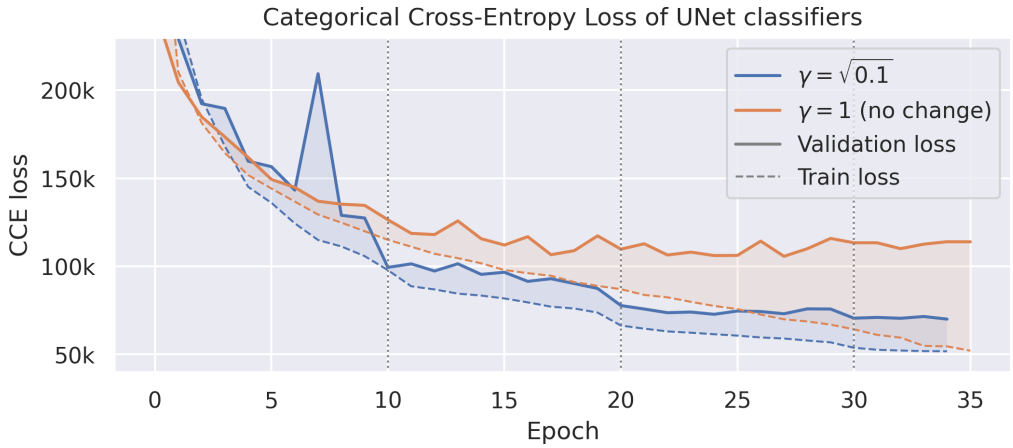

Figure 6: Differences between training and validation loss of a classifier that dynamically adjusts its learning rate every 10 steps and one that doesn't. While both classifiers overfit, the dynamic learning rate makes this process slower and the eventual validation loss reaches better values.

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**[**?**].

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

4 Results

5 Conclusions

6 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] 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.

[3] 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.

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

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