Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Andreas Hammer Håversen

QT-UNet

A Self-Querying All-Transformer U-Net for 2D and 3D Segmentation Augmented by Self-Supervised Learning

Master's thesis in Computer Science
Supervisor: Frank Lindseth & Gabriel Kiss
June 2022

NTNU
Norwegian University of
Science and Technology

Andreas Hammer Håversen

QT-UNet

A Self-Querying All-Transformer U-Net for 2D and 3D Segmentation Augmented by Self-Supervised Learning

Master's thesis in Computer Science
Supervisor: Frank Lindseth & Gabriel Kiss
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

In 2017, the Transformer model revolutionised the Natural Language Processing field, bringing large-scale models capable of understanding complex long-range dependencies in text at a manageable computational cost. In 2020, the Vision Transformer brought a similar revolution to Computer Vision, with similar scaling benefits.

The development of linear time complexity Vision Transformers like the Swin transformer further aided the adoption of Vision Transformers, leading to a large number of applications in Autonomous Driving and Medical Image Computing. Models like VT-UNet melded traditional UNets with Swin transformers to create a strong volumetric segmentation model for brain tumour segmentation, introducing a novel Encoder-Decoder Cross-Attention concept.

Parallel to these revolutions, Self-Supervised Learning saw a similar revolution and uptake in use within several Computer Vision subdomains, particularly Medical Image Computing where training data is often scarce. Notably, Swin-UNETR pre-trained a strong Swin-based encoder with a large CT dataset utilising contrastitive, reconstructive, and rotation tasks, demonstrating strong performance in downstream Medical Segmentation Decathlon (MSD) and Beyond The Cranial Vault (BTCV) tasks.

Our research melds these advances together to produce the Querying Transformer UNet (QT-UNet): A all-Swin Transformer UNet with Encoder-Decoder Cross-Attention, enhanced by Self-Supervised Learning (SSL). QT-UNet is tested with several Medical Image Computing datasets to evaluate its efficacy as a general volumetric segmentation model. We also collect a large CT pretraining dataset dubbed CT-SSL with 3,597 CT scans. A 2D version, QT-UNet-2D, is spun out to evaluate the techniques in a 2D Autonomous Driving context.

Our best model is competitive with State of the Art in BraTS2021 despite a 40% reduction in FLOPs against our baseline VT-UNet, with an average Dice score of 88.61 and Hausdorff Distance of 4.85mm. We find weaker results with BTCV and Medical Segmentation Decathlon, but demonstrate the effectiveness of both our new Cross-Attention mechanism, and our SSL pipeline when pre-training with our CT-SSL dataset. We transfer the model to a 2D context with CityScapes, finding that our new Cross-Attention mechanism and SSL pipeline are effective without modification.

Keywords: Medical Image Segmentation, Autonomous Vision, Deep Learning, UNet, Self-Supervised Learning, Encoder-Decoder Cross-Attention, Vision Transformer, Swin Transformer

Sammendrag

I 2017 revolusjonerte Transformer-modellen naturlig språkbehandling ved å tilgjengeliggjøre store modeller som var i stand til å forstå komplekse sammenhenger over store avstander i tekst med en håndterbar beregningskostnad. I 2021 brakte introduksjonen av Vision Transformere med seg en liknende revolusjon i bildebehandling, med liknende skaleringsfordeler.

Utviklingen av effektive Vision Transformere med lineær tidskompleksitet som Swin Transformeren bidrog til ytteligere opptak i bruk av Vision Transformere, spesielt i felter som Autonomt Syn og Medisinsk Bildeanalyse. Modeller som VT-UNet smeltet sammen tradisjonelle UNet med Swin Transformere for å skape en ny sterk volumetrisk segmenteringsmodell for hjernesvulster, med et nyskapende Enkoder-Dekoder Cross-Attention konsept.

Parallelt med disse revolusjonene opplevde Self-Supervised Learning en lignende revolusjon og økning i bruk innen flere datasynsdomener, spesielt domener der det er knapt med treningsdata. Swin-UNETR forhåndstrente en sterk Swin-basert koder med et stort CT-datasett og kontrast-, rekonstruksjons- og rotasjonsbaserte treningsoppgaver, og viste sterk nedstrøms ytelse i Medical Segmentation Decathlon (MSD) og Beyond The Cranial Vault (BTCV).

Dette prosjektet smelter disse fremskrittene sammen med modellen Querying Transformer UNet (QT-UNet): Et all-Swin Transformer UNet med Enkoder-Dekoder Cross-Attention, forsterket med Self-Supervised Learning. QT-UNet testes med flere Medical Image Computing datasett for å evaluere modellens effektivitet som en generell volumetrisk segmenteringsmodell. Vi samler et stort datasett kalt CT-SSL med 3.597 CT-skanninger til pretrening. En 2D-versjon, QT-UNet-2D, spinnes ut av hovedmodellen for å evaluere effektiviteten til teknikkene i en 2D Autonomt synskontekst.

Vår beste modell er konkurransedyktig med "state of the art" i BraTS2021 med 40% færre FLOPs enn vår baseline VT-UNet, med en gjennomsnittlig Dice score på 88,61 og Hausdorff Distance på 4,85 mm. Vi finner mindre gode resultater med BTCV og MSD, men demonstrer effektiviteten til både vår nye Cross-Attention mekanisme og vår SSL-pipeline ved pretrening på CT-SSL. Vi overfører også teknikkene til en 2D-kontekst med CityScapes, og finner at vår Cross-Attention mekanisme og SSL-pipeline er effektiv uten endringer.

Preface

This Master's thesis in Computer Science was completed as part of the Computer Science master programme at the Norwegian University of Science and Technology.

Ever since starting my studies in artificial intelligence, I have found it to be a deeply interesting and satisfying field of study. I recognise how lucky I have been to be able to work in a field currently experiencing intense development and research. For this thesis specifically, I feel especially fortunate to have been able to work with Vision Transformers and Self-Supervised Learning, both fields that have experienced intense research interest and progress in the last three years.

I would like to thank my supervisors Frank Lindseth and Gabriel Kiss, whose valuable insight and feedback made this thesis possible.

Finally, I would also like to thank my fellow masters students at the Fiol masters hall, who kept me sane throughout the thesis writing process.

Contents

Figures

Tables

Acronyms

AD Autonomous Driving. iii, ix–xi, xv, 1–3, 5, 10, 14, 15, 49, 55, 64, 70, 78, 97–99, 105, 109

ADAM ADAptive Movement estimation. 23

AI Artificial Intelligence. 15

ANN Artificial Neural Network. ix, 5, 15, 17, 19–21, 26, 27, 32

BraTS Brain Tumour Segmentation. iii, 3, 11, 49, 53, 64, 67, 68, 89, 90, 92, 94, 106

BTCV Beyond The Cranial Vault. iii, v, xiii, 3, 12, 13, 45, 46, 48, 53, 87, 89, 90, 92–95, 106–108

BYOL Bootstrap Your Own Latent. x, 42, 65, 67, 91

CA Cross-Attention. iii, xi, xii, 2, 3, 45, 60, 61, 70, 71, 78, 81, 82, 85, 86, 92–95, 97–99, 106–109

CE Cross Entropy. 20, 62

CNN Convolutional Neural Network. x, 1, 5, 27, 29, 30, 32, 46

CV Computer Vision. iii, ix, 1, 2, 5, 8, 10, 12, 14, 15, 30, 35, 36, 50, 51, 62

DL Deep Learning. 15

DPT Dense Prediction Transformer. 38

DSC Dice Similarity Coefficient. 10, 20, 45, 46

FN False Negative. 9, 10

FP False Positive. 9, 10

HD Hausdorff Distance. iii, 10, 45, 48, 68, 73, 86, 89, 90, 93, 94, 100, 106

IoU Intersection over Union. 9, 10, 70, 71, 78, 81, 82, 95, 97, 98

MIC Medical Image Computing. iii, ix–xi, xv, 1–3, 5, 10–12, 20, 48, 49, 52, 64, 68, 73, 90, 99, 101, 105, 106, 109

ML Machine Learning. 5, 8, 15

MLP Multi-Layer Perceptron. xiii, 17–19, 35, 42, 43, 50, 58, 61, 65

MSD Medical Segmentation Decathlon. iii, v, xiii, 3, 11, 46, 48, 49, 53, 67, 69, 73, 76, 87, 89, 90, 92–95, 107, 108

MSE Mean Squared Error. 20

NLP Natural Language Processing. iii, 1, 26, 30, 32, 35

ReLU Rectified Linear Unit. 18

RQ Research Question. 55, 87, 95, 105–107

SA Self-Attention. 48

SotA State of the Art. iii, 1–3, 36, 39, 42, 43, 46, 74, 76, 78, 90, 95, 97, 106, 107

SSL Self-Supervised Learning. iii, xi, xv, 2, 3, 5, 17, 41, 46–48, 51, 52, 55, 64–69, 87, 90, 91, 95, 98, 99, 103, 105–107

TCIA The Cancer Imaging Archive. 65

TN True Negative. 9

TP True Positive. 9, 10

ViT Vision Transformer. x, xiii, 1, 35–38, 46, 58

Chapter 1

Introduction

1.1 Background and Motivation

Transformers, introduced by Vaswani *et al.* [1], revolutionised the Natural Language Processing (NLP) field by introducing a model that can effectively model long-range dependencies while maintaining a manageable computational cost. Transformers are now the dominant model in that field, with notable examples being BERT [2] and GPT-3 [3]. The computational efficiency of the Transformer has enabled NLP models of unprecedented size, with the largest variant of GPT-3 having approximately 175 billion trainable parameters.

The attention mechanisms that power Transformers have also inspired the adoption of similar mechanisms in models for Computer Vision (CV), with some models incorporating self-attention mechanisms instead of convolution or using a Transformer in conjunction with a convolutional backbone.

Dosovitskiy *et al.* [4] made significant progress in 2020 by introducing the Vision Transformer (ViT), a near end-to-end Transformer model for image classification. They showed that Transformers can be used for vision tasks without significant backbones and the inductive bias that Convolutional Neural Networks employ. They also showed that these models can achieve State of the Art (SotA) performance for image classification, whilst still requiring less computational resources and parameters to train than conventional CV models.

A major challenge for ViTs is that standard Transformer models suffer a quadratic increase in memory and time complexity in terms of input length. This hinders the application of ViTs on high-resolution images and volumes such as those used for Autonomous Driving (AD) and Medical Image Computing (MIC) tasks without the usage of CNN backbones to reduce dimensionality. Several approaches have been suggested to reduce the time complexity of standard Transformers [5–7], and a handful of approaches to optimise for image and volumetric data have been suggested for ViTs.

Liu *et al.* [8] introduced the Swin Transformer, which through the use of self-attention in shifted windows has a linear-time complexity. This has en-

abled its use in applications with high-resolution images, as well as volumetric data such as CT and MRI scans. The Swin Transformer has been successfully applied to several dense prediction tasks, like segmentation and depth estimation, thanks to its efficiency. Notable examples include Swin-UNet [9] and VT-UNet [10]. The latter, a all-Swin Transformer UNet used for segmenting MRI scans, introduces a novel Encoder-Decoder Cross-Attention concept, giving the decoder access to more information from the Encoder in addition to what the traditional skip-connections in a traditional UNet allows.

A common challenge in any machine learning problem is getting enough data to train the model with, to avoid overfitting the data and poor generalisation. This is even more so a challenge in Medical Image Computing (MIC), where labelling the data requires the effort of highly trained professionals in a very time-consuming task. Self-Supervised Learning (SSL) allows ML-practitioners to get more out of their data without having to label their data, by generating pseudo-labels for more accessible unlabelled data. This has been used to great effect for CV in general and in MIC especially. A notable work, Swin-UNETR [11], combines SSL with a UNet that employs a Swin Transformer as its encoder.

In this project, we attempt to build on recent advances in Vision Transformers, focusing on MIC and AD. Taking VT-UNet and Swin-UNETR as inspiration, as they currently hold SotA for several MIC datasets, we attempt to combine these methods as well as introducing improvements to them. To this end, we introduce the QT-UNet, which employs an all-Swin Transformer UNet with Encoder-Decoder Cross-Attention and Self-Supervised Learning. We also apply this model in a 2D AD context to evaluate the efficacy of these techniques in a more traditional CV context.

1.2 Research Goal and Research Questions

In this thesis, we investigate the effects of Cross-Attention as utilised in VT-UNet and SSL as utilised in Swin-UNETR when applied in a general cross-modality model, across 2D and 3D data from a wide variety of sources. We also seek improvements to their original approaches to enhance model performance. Our research goal can be stated as follows:

> To explore the efficacy of a cross-domain all-Transformer UNet segmentation model based on the Swin Transformer, self-supervised pre-training, and Encoder-Decoder Cross-Attention on Medical Image Computing and Autonomous Driving datasets.

To achieve this goal, we pose the following research questions (RQs):

- **RQ1:** What is the effect of using self-supervised pretraining of the encoder in an all-Transformer UNet on the performance of the overall network in segmentation tasks?

- **RQ2:** What is the effect of using encoder-decoder Cross-Attention on the overall performance of an all-Transformer UNet?
- **RQ3:** How can these techniques be applied effectively for both 2D and 3D segmentation tasks?

1.3 Contributions

We introduce QT-UNet, a Swin-based all-Transformer U-Net for semantic segmentation of 3D and 2D data, augmented by Self-Supervised Learning. We perform an extensive battery of tests with Medical Image Computing (MIC) and Autonomous Driving (AD) datasets to examine its efficacy, comparing it with recent state-of-the-art models for the respective datasets.

We introduce a novel Cross-Attention mechanism inspired by VT-UNet [10], coupled with a new decoder block design that allows the decoder blocks in the model to query the output of the same-stage encoder for information at each stage of the decoding process. We also employ Self-Supervised Learning (SSL) for the encoder, based on the procedure developed for Swin-UNETR [11], the first application of the technique across 2D images and 3D MRI and CT volumes to an all-Transformer UNet known to the authors at the time of writing. We collect a large dataset consisting of 3,597 CT scans, dubbed CT-SSL, to pre-train the encoder for CT-based tasks.

We find strong performance with the BraTS2021 dataset when trained from scratch, and an even stronger result in terms of Hausdorff Distance when trained with weights pre-trained on the dataset. Both models are competitive with current SotA with a 40% reduction in FLOPs compared to our baseline VT-UNet.

We find weaker results with the BTCV and MSD, but validate the effect of our Cross-Attention mechanism and our SSL pipeline when pre-training with CT-SSL.

We transfer our techniques to a 2D Autonomous Driving context, demonstrating the positive effects of Encoder-Decoder Cross-Attention and our SSL pipeline in CityScapes and CityScapes by categories, although the model itself is weaker than the current SotA. We stress that the SSL pipeline was not changed between the 3D and 2D context, highlighting its generality. We also find that the effectiveness of Cross-Attention is related to the number of classes in the targets.

1.4 Report outline

Chapter 1 - Introduction introduces the project, its motivation, research goals, and contributions.
Chapter 2 - Background theory and Related Work describes relevant background knowledge, as well as work related to the project and its experiments.

Chapter 3 - Methodology describes in detail the methodology of the project. It contains information about the different experiments that have been run.
Chapter 4 - Results contains the results of our experiments.
Chapter 5 - Discussion contains our reflections on the results achieved in the previous chapter, as well as on our method.
Chapter 6 - Conclusion and Future Work concludes the report describing the answers this project gave to the research questions and the achievement of the overall project goal. A discussion of potential directions for future work and improvement is also presented.

Chapter 2

Background theory and Related Work

This chapter will introduce the theoretical background for the work done in this project and works related to it. The chapter starts with a short introduction to CV tasks in Section 2.1 followed relevant MIC and AD datasets in Section 2.2, to introduce the problem domain. Section 2.3 then describes Machine Learning fundamentals. Section 2.4 describes Artificial Neural Networks, while Section 2.5 describes Convolutional Neural Networks. Section 2.6 describes Transformers in general, whilst Section 2.7 describes Vision Transformers specifically. Section 2.8 describes Self-Supervised Learning before Section 2.9 rounds of the chapter by presenting work relevant to this project. The contents of this chapter draws heavily upon work done for the preparatory project for this thesis [12].

2.1 Computer Vision

Computer Vision (CV) as a field deals with how computers can understand digital images, volumes, and video. The field is in its entirety rather broad. For brevity, this section will introduce common tasks in the field and introduce relevant evaluation metrics, focusing on segmentation.

2.1.1 Classification

Classification is the task of correctly assigning an element to the correct class [13]. Specifically, for CV, the tasks concern taking an image and correctly classifying its content into one of multiple classes. For example, a model might be fed a 256×256 RGB image of a cat and asked to predict which breed of cat is present in the image. An example of image classification can be seen in Figure 2.1.

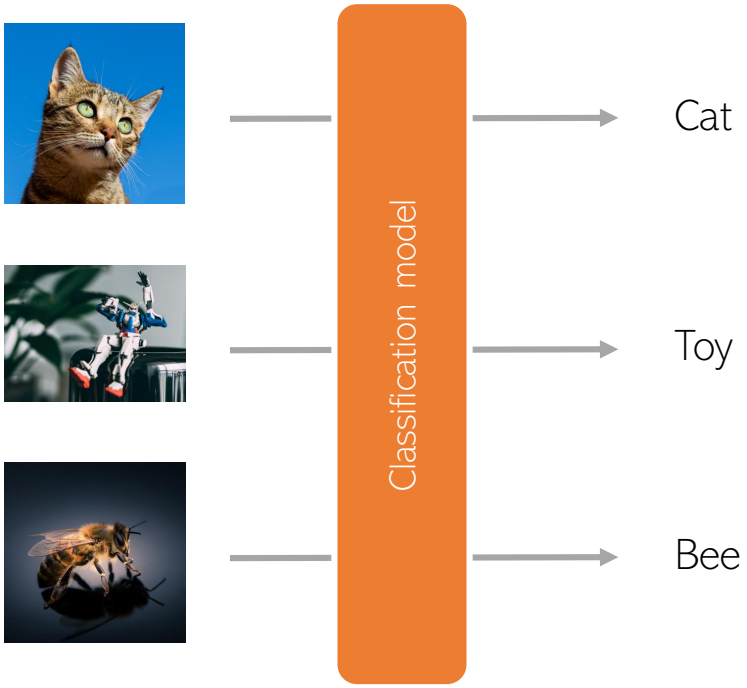


Figure 2.1: An example of a classification task.

2.1.2 Object detection

Object detection can be considered an amp-ed up version of the classification task. Rather than classifying a single object in an image, the goal is now to correctly classify *and* locate multiple objects in the image. This allows us to, as an example, detect the presence of multiple cats in an image and where they are located in the scene. The output in an object detection task is bounding boxes that indicate the position and dimensions of each object, as well as the object class. An example can be seen in Figure 2.2.

In addition to being trained on the labels the model outputs, it is also trained on how well they place the bounding boxes compared to the ground truth boxes in the training set.

2.1.3 Segmentation

Segmentation takes object detection a step further and detects which class or object *each pixel* belongs to. Pixels that are collectively assigned to the same object or class are often referred to as a segmentation mask.

We can broadly divide the world into two distinct classes of objects: Things – which are countable objects such as cars, people and animals – and stuff – which are amorphous regions of similar texture or material such as grass, sky,

Figure 2.2: An example of a object detection task.

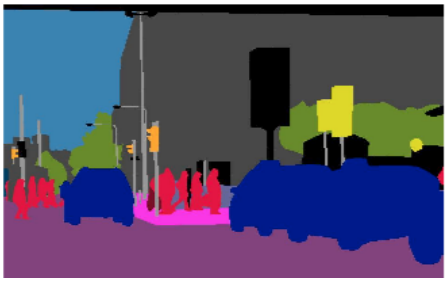
or road.

The three different types of segmentation can be characterised by this divide. The study of *stuff* can be formulated as the task of doing *semantic segmentation*, where the goal is to assign each pixel a class label [14]. Note that things are considered a subset of stuff in the context of semantic segmentation and thus are still classified. Say we have an image of a busy road. Then, cars, people, and other classes of entities on the road would each belong to their respective class segmentation mask.

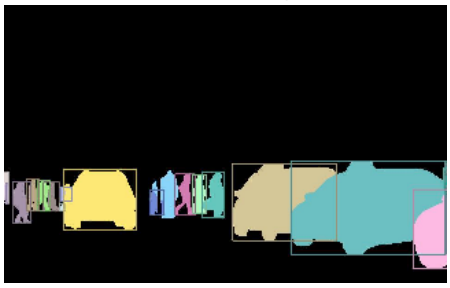
In contrast, the study of *things* can be formulated as the task of *instance segmentation*, where the goal is to detect each object and delineate it with a segmentation mask, classifying each object individually [14]. With our example from above, each car and person on the road will have its own segmentation mask.

Finally, the study of stuff *and* things together can be formulated as the task of doing *panoptic segmentation*, where the goal is to assign both class and instance labels to each pixel [14]. In our example, the road and the sky would be classified as stuff and thus be segmented semantically, while the cars and the people would each have their own instance segmentation mask. Figure 2.3 shows the difference between the three segmentation tasks.

(a) Raw image



(b) Semantic segmentation



(c) Instance segmentation

(d) Panoptic segmentation

Figure 2.3: A comparison of segmentation tasks. Illustration from [14].

.

2.1.4 Evaluation metrics for CV

Good quantitative metrics are needed to compare the performance of one model to another, or against that of a human. There are several metrics in use in the ML community in general and within the CV community specifically. Different metrics will emphasise different aspects of model performance. Some metrics will highly reward correct predictions of positives, whilst others will emphasise the correct classification of a negative sample. This section will give a brief overview of metrics terminology, some commonly used CV metrics, and the tasks with which the metrics are used.

Positives and negatives

Several metrics operate with the notion of positives and negatives relative to some class, where a sample with a positive label is a member of the class, while a sample with a negative label is not. When predicting membership of the sample, either yes or no, we get four classes of predictions. If the model correctly predicted that the element does indeed belong to the class, we have a *true positive*. Likewise, if the model correctly predicted that the element is *not* a member of the class, we have a *true negative*. On the other hand, if the model predicted membership in the class while the label of the sample indicates that it is not, we have a *false positive*. Vice versa, a prediction of non-membership for a sample that is indeed a member of the class is a *false neg-*

Label / Prediction	**Positive**	**Negative**
Positive	True Positive (TP)	False Positive (FP)
Negative	False Negative (FN)	True Negative (TN)

Table 2.1: Overview of classification types with abbreviations.

ative. See Table 2.1 for a summary.

Precision and Recall

Using these primitives, we can define more complex metrics. Two of the most fundamental metrics are *precision* and *recall*.

Precision, defined as in Equation (2.1), measures the probability that a sample classification predicted by the model is actually a member of the class. In other words, it measures the percentage of predictions that were true positives compared to the total number of true positives and false positives.

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

Recall, as defined in Equation (2.2), measures how many elements belong to the class that the model is capable of correctly predicting, as a ratio of True Positives and False Negatives.

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

Ideally, both precision and recall should both be high. That is, one would like to have a high percentage of true positives and a high number of correctly classified samples. However, in reality, increasing either measure will often result in a decrease in the other. A trade-off must be made between high recall and high precision. There is no silver bullet here. Different tasks will have different requirements and require different trade-offs.

Intersection over Union

A commonly used metric for object detection and segmentation is Intersection over Union (IoU) [15], also known as the Jaccard Index, as defined in Equation (2.3). Essentially, IoU measures the intersection between the prediction and the ground truth (that is, the true positives) divided by the area of the union of the prediction and the ground truth (that is, the true positives, the false positives and the false negatives). The metric is defined in the range $[0, 1]$, with 0 indicating that there is no overlap between the ground truth and the prediction and 1 indicating perfect overlap. For multi-class prediction, IoU is calculated for each class and then averaged. This is called mean IoU, denoted as mIoU.

$$IoU = \frac{TP}{TP + FP + FN} \tag{2.3}$$

Dice Similarity Coefficient

Another metric commonly used for segmentation is Dice Similarity Coefficient (DSC) [16, 17], also known as the F1 score, as defined in Equation (2.4). It is quite similar to IoU, but it is more rewarding of true positives than IoU. Nonetheless, they are positively correlated, meaning that a higher IoU score also means a higher Dice score and vice versa. This metric has its roots in biology and is used widely in the MIC field.

$$DSC = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{2.4}$$

Hausdorff Distance

This metric [18], although less commonly used for standard CV tasks, is quite often used for segmentation tasks in the MIC field. In a nutshell, Hausdorff Distance measures how far two subsets of a metric space are from each other. More concretely, it is informally the longest distance one can travel from any point in one of the sets to any points in the other. Its interpretation can be understood as a measure of how far the worst part of the predicted mask was from the label. For MIC, it is usually denominated in millimetres.

Formally, the Hausdorff Distance is defined as follows: Let X and Y be two non-empty subsets of a metric space M with distance measure d. Then we define their Hausdorff Distance $d_H(X, Y)$ as in Equation (2.5), where $d(a, B) = \inf_{b \in B} d(a, b)$ is the distance from a point $a \in X$ to the subset $B \subseteq X$, sup is *supremum* and inf is *infimum*. The supremum and infimum are the smallest and greatest elements of a set, respectively.

$$d_H(X, Y) = max \left\{ \sup_{x \in X} d(x, Y), \sup_{y \in Y} d(X, y) \right\} \tag{2.5}$$

2.2 Datasets

There are several different datasets in the Medical Image Computing and Autonomous Driving domains. What follows is a short description of a few significant datasets in these domains.

2.2.1 Medical Image Computing (MIC)

MIC is a subfield of CV that deals with the processing and analysis of medical image data for the purposes of aiding medical personnel with diagnosis and

Figure 2.4: Example ground truth from MSD Task 6, Lung Tumour segmentation in a CT-scan.

decision making in the treatment of diseases. It has its own set of interesting challenges, including high-resolution – often volumetric – data, with high demands upon the accuracy of the predictions produced. Incorrect predictions can have dangerous consequences, although it is uncommon for MIC algorithms to make direct decisions about patient care, instead acting as an aid and decision support to qualified medical personnel.

Medical Segmentation Decathlon

The Medical Segmentation Decathlon (MSD) dataset [19] is a collection of ten semantic segmentation tasks from different organs and image modalities (CT and MRI scans), segmenting organs, tumours, and cancer primaries depending on the task. Each task has a unique set of challenges, with the collection as a whole aiming to highlight common challenges with medical data, such as small training sets, unbalanced classes, multi-modality data, and small segmentation targets. An example segmentation from Task 6 can be seen in Figure 2.4.

BraTS2021

The Brain Tumour Segmentation (BraTS) challenge [20–22], organised by the Radiological Society of North America (RSNA), the American Society of Neuroradiology (ASNR), and the Medical Image Computing and Computer Assisted Interventions (MICCAI) society, is a 3D MRI dataset for tumour segmentation.

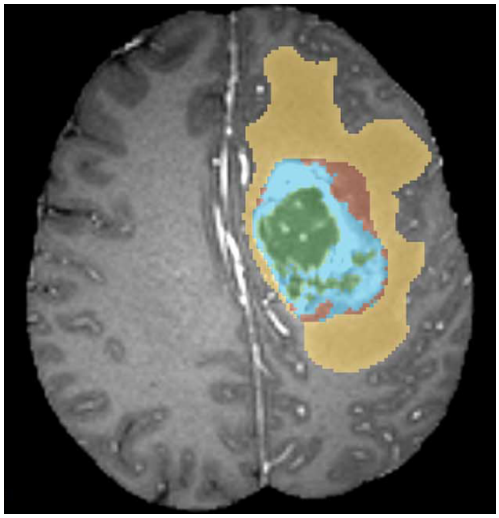


Figure 2.5: Example ground truth from BraTS2021.

It includes 1251 samples, each with four 3D MRI modalities: native (T1), post-contrast T1 weighted (T1Gd), T2-weighted (T2), and T2 Fluid-attenuated Inversion Recovery (FLAIR). Its ground truth labels were annotated by physicians, dividing the tumour into four regions: The enhancing tumour, the peritumoral edema, the necrotic tissue, and the non-enhancing tumour core. An example can be seen in Figure 2.5.

Beyond The Cranial Vault

The Beyond The Cranial Vault (BTCV) abdomen challenge dataset [23] is a multi-organ segmentation dataset consisting of 50 samples. The samples are portal venous phase CT scans collected from various collaborating institutions. 13 organs were annotated by trained raters and reviewed for label accuracy by a radiologist or radiation oncologist. Being a small dataset with many classes and with certain small target organs, this dataset poses a challenging segmentation task. An example can be seen in Figure 2.6.

PanNuke

The PanNuke dataset [24] is a semantic nuclei segmentation dataset, containing 7,904 cases across 6 classes in three folds from 19 different tissue types. In total, the set contains 205,343 labelled nuclei. Each fold contains a little over 2,500 samples. Whilst the dataset is relatively small in terms of number of cases when compared to other recent CV datasets, is it quite large for a MIC dataset. An example can be seen in Figure 2.7.

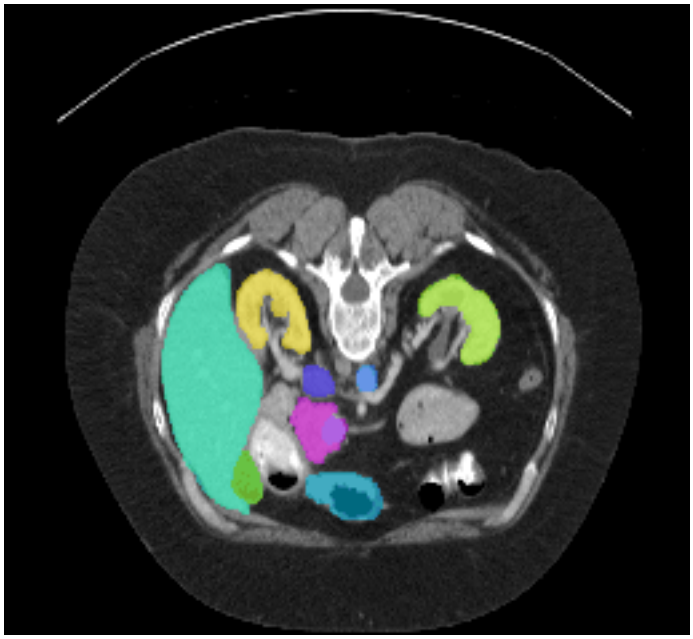
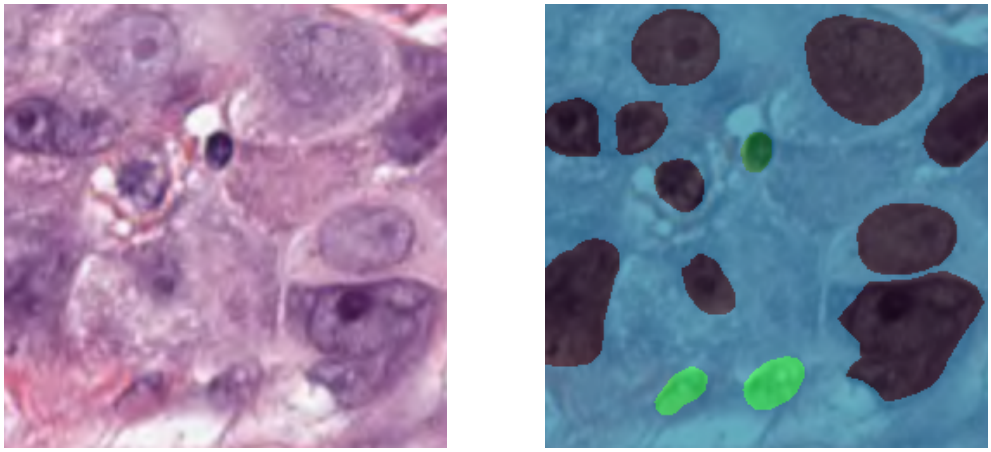


Figure 2.6: Example ground truth from BTCV.



(a) Raw image **(b)** Ground truth segmentation

Figure 2.7: Example of PanNuke image and ground truth.

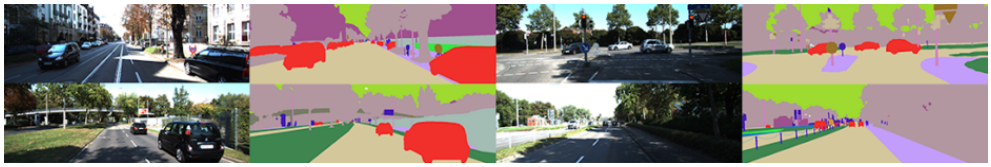


Figure 2.8: Examples from the KITTI semantic segmentation dataset, taken from [30].

2.2.2 Autonomous Driving (AD)

Autonomous Driving is a sub-field of CV that deals with the processing and analysis of images for the purposes of independent computer-driven navigation in an environment, typically a city street or a highway. It poses an interesting set of challenges, as it requires accurate real-time analysis of complex high-resolution scenes with many interacting elements in order to produce intelligent driving decisions. The demands upon the algorithms used in this space are high, as the consequences of late or incorrect predictions can lead to dangerous manoeuvres in the real world, causing material damage and human injury. A notable example of a CV application in the AD field is Tesla's full self-driving system [25].

KITTI

The KITTI dataset [26–29] is a collection of several different datasets, compiled into a Vision Benchmark Suite. It includes datasets for vision flow, depth estimation, visual odometry, 3D object detection, 3D object tracking, and semantic and instance segmentation. Though it represents for many of these domains a seminal collection of datasets, it is relatively small with, for instance, only 200 train images and 200 test images in the semantic and instance segmentation datasets. Examples from the dataset can be seen in Figure 2.8.

CityScapes

The CityScapes dataset [31] is a large-scale segmentation dataset of city scenes, collected from 50 different German cities. It contains in total 5,000 frames at 1024×2048 pixels with pixel-level annotations across 30 classes, with an additional 20,000 frames with coarse annotations. It can be used for semantic, instance, and panoptic segmentation. The dataset is widely used in the AD community. An example from the dataset can be seen in Figure 2.9.

nuImages

The nuImages dataset [32] is a large-scale AD dataset for semantic and instance segmentation and 2D object detection collected from Singapore and

Figure 2.9: An example segmentation mask from CityScapes.

Boston. It contains in total 93,000 images with annotations. Although the nu-Images dataset is relatively new, it has gained some traction in the AD community. An example can be seen in Figure 2.10.

2.3 AI fundamentals

Whilst Computer Vision (CV) depends heavily upon Artificial Intelligence (AI), it is difficult to establish a specific and clear definition of AI. AI can, however, in general, be interpreted as a field concerned with building systems that make intelligent decisions in response to input from its environment. This broad definition does, of course, contain a wealth of diversity in approaches. These range from the purely algorithmic to the purely statistical. Subfields that have recently emerged include Machine Learning (ML) and Deep Learning (DL) using Artificial Neural Networks.

2.3.1 Machine Learning

Machine Learning focuses on the construction of algorithms in which the agent learns to solve a task through experience. Formally, we can say that an agent (A) that tries to solve a task (T) with a performance measure (P) is learning if P increases with experience (E). Within this definition, there exist several paradigms, such as supervised and unsupervised learning.

Supervised and Unsupervised learning

Supervised learning is most relevant for this project, but both paradigms are complementary and thus deserving of treatment in this section. For some ma-

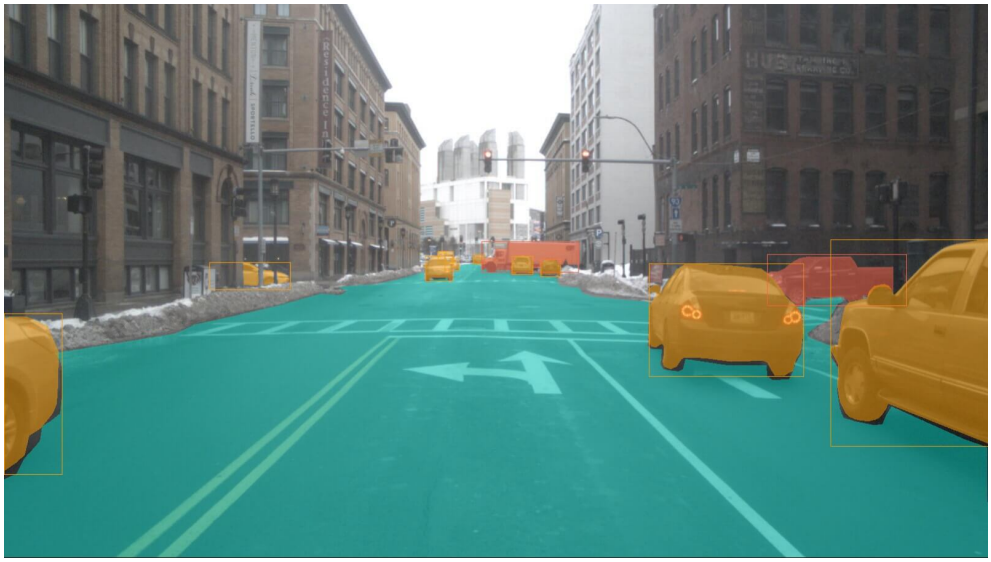


Figure 2.10: Example segmentation from nuImages, from [32].

chine learning problems, the set of correct input-output pairs for the task may already be known. For others, such a set may not be available. For instance, for a system that tries to predict the topic of a newspaper article, a human can go through it in advance and label the articles completely and correctly. In contrast, a system that is trying to identify clusters in large volumes of data might not have answers available due to a large amount of data to label.

Tasks in which the expected output is known in advance can be solved using supervised learning techniques. Formally, given a training set of N data pairs,

$$(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)$$

where each y_i was generated by some function $y = f(x)$, the task is to discover a function h that approximates f [13].

In contrast, tasks where the expected output is unknown can be solved using unsupervised learning techniques. Generally, unsupervised learning attempts to discover similarities and relationships in the data. Returning to our example from before, a typical unsupervised technique is to use clustering algorithms to discover entities in the data set that are similar, say good and bad weather days measured by rainfall and the number of rides in a taxi on a scatter plot [13].

Variants of supervised and unsupervised learning also exist. Semi-supervised learning is a paradigm that uses both labelled and unlabelled data to train the model. For example, one approach of several approaches to semi-supervised learning is to train a model on the limited data available, use that model to generate new labels for the unlabelled dataset, and then re-train the model over both the old and new labels.

Another paradigm of specific interest to this thesis is Self-Supervised Learning (SSL), where the training procedure uses pseudolabels that are automatically generated from unlabelled data. SSL is described in more detail in Section 2.8.

2.4 Artificial Neural Networks

Some of the first machine learning models recognised today were intended to be computational models of biological learning [33]. One such model is the Artificial Neural Network (ANN), inspired by the biological brain. This model was inspired by two insights: Firstly, that the brain provides a proof by example that a system that can create intelligent behaviour can exist, and secondly that it provides a conceptually straightforward path for duplicating that model by reverse engineering the computational principles of the brain [33].

2.4.1 The Artificial Neuron

Similarly to how the basic building block of a biological brain is the biological neuron, the basic building block of an artificial neural network is the artificial neuron.

A basic artificial neuron consists of nothing more than a numerical bias term and associated weights. Upon activation, it takes the weighted sum of its inputs plus a bias term, feeds that to some activation function, and returns the output. In practice, we deal with the input and the weights as vectors, which lends us to an approach where we take the dot product of the transposed weights with the input, add the bias, and then apply an activation function. The bias allows the neuron to shift its zero point up or down. A mathematical formulation of the artificial neuron can be seen in Equation (2.6).

$$\begin{aligned} \boldsymbol{z} &= \boldsymbol{w}^\intercal \cdot \boldsymbol{x} + \boldsymbol{b} = \sum_i \boldsymbol{w}_i \boldsymbol{x}_i + \boldsymbol{b}_i \\ \boldsymbol{h} &= g(\boldsymbol{z}) \end{aligned} \tag{2.6}$$

We follow the notation of Goodfellow *et al.* [33] and denote the sum as z, the layer weights as w, the inputs as x, and the bias as b, with vectors set in bold font, over the number of inputs i. Furthermore, we denote the activation function as $g(x)$ and the final output as h.

The simplest possible ANN consists of a single neuron with the Heavyside step function as its activation function and is called a *Perceptron* [13]. While a Perceptron can be useful on their own, they suffer from a fundamental flaw: They can only learn linearly separable functions. A Multi-Layer Perceptron (MLP), consisting of several layers of neurons, can express more complex linear functions, but will still produce linear decision boundaries due to the linear nature of the activation function [33].

2.4.2 Activation functions

In order to allow MLPs to express non-linear functions, we need to introduce non-linear activation functions. There are several to choose from. We list a handful of common choices here for reference.

Sigmoid

The Sigmoid function outputs a value in the range $[0, 1]$, and can be seen in Equation (2.7)

$$\sigma(z) = \frac{1}{1 + e^(-z)} \tag{2.7}$$

Hyperbolic Tangent

The hyperbolic tangent function outputs a value in the range $[-1, 1]$, and can be seen in Equation (2.8)

$$tanh(z) = 2\sigma(2z) - 1 \tag{2.8}$$

Rectified Linear Unit

The Rectified Linear Unit (ReLU) function outputs a value in the range $[0, \infty]$, and can be seen in Equation (2.9)

$$ReLU(z) = max(0, z) \tag{2.9}$$

2.4.3 Network architecture

As noted in Section 2.4.1, we can arrange several artificial neurons together into layers. We can then compose several layers into a network that, with the use of non-linear activation functions, can approximate functions that are far more complex and non-linear. An example of a multi-layered perception can be seen in Figure 2.11. Neurons in the middle layers, the "hidden layers", are called "hidden units". Notice that each neuron in a layer is connected to every neuron in the previous layer. Each neuron in a layer maintains a bias term, as well as a weight vector where each entry in the vector corresponds to the weight associated with the input from each connected neuron in the previous layer. Consequently, each layer can be represented with a bias vector $\boldsymbol{b} \in \mathbb{R}^n$ and a weight matrix $\boldsymbol{W} \in \mathbb{R}^{m \times n}$, where n and m correspond to the number of neurons in this layer and in the previous layer, respectively.

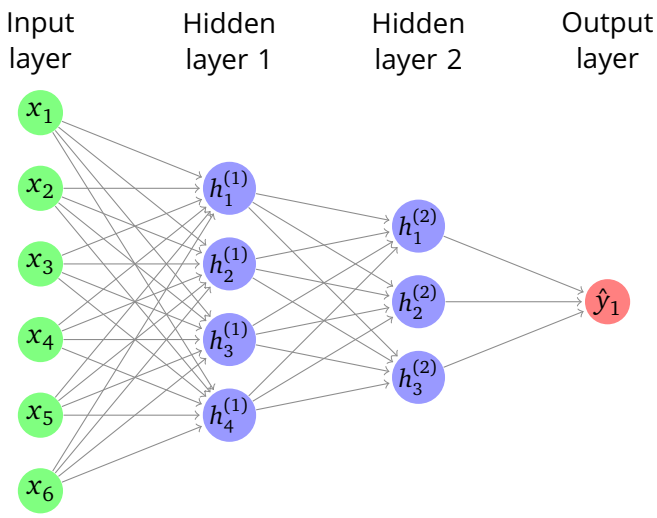


Figure 2.11: A Multi-Layer Perceptron (MLP).

The Universal Approximation Theorem

This theorem states that a feedforward network with a linear output layer and at least one hidden layer with any "squashing" activation function (that is, the function must saturate for large positive and negative values) can approximate any continuous function on a bounded and closed subset of $\mathbb{R}^n$ with any desired non-zero amount of error, provided that the network has enough hidden units [33].

2.4.4 Forward pass in a ANN

A ANN produces its output by forwarding its input through its layers. Using the matrix and vector definition of the layers defined at the end of Section 2.4.3, we can formulate a general expression for the output of a n-layer neural network with input x in Equation (2.10).

$$h^n = g^n(z^n) = g^n((\mathbf{W}^n)^\intercal h^{n-1} + b^n) \tag{2.10}$$

Where the superscript denotes the layer from which the term or function belongs, and $a_0 = x$. We can see how the output from each previous layer is fed as input to the next layer, with each layer applying its weights, biases, and activation functions all the way forward until we reach the last layer in the network.

2.4.5 Loss functions

An essential part of learning in general is to figure out where we got it right and where we got it wrong. It is no different for machine learning in general or

ANNs specifically. In machine learning, the mechanism we use to discover our errors and successes is called a *loss function*. As with the activation functions described in Section 2.4.2, there are several functions to choose from, each well suited for different types of tasks. A hand-full of important loss functions are summarised in the following paragraphs. We denote the predicted output of our model as $\hat{y}$, the ground truth labels as y, and the number of samples as n.

Mean Squared Error (MSE) Loss

MSE loss is commonly used for regression tasks. It is defined as in Equation (2.11), where y_i is the ground truth and $\hat{y}_i$ is the prediction produced by the model.

$$\mathcal{L}_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.11}$$

Cross Entropy (CE) Loss

CE loss is commonly used for classification and segmentation tasks, and is defined in Equation (2.12) for multi-class problems where $\boldsymbol{y}$ is the label vector and $\hat{\boldsymbol{y}}$ is the model prediction vector.

$$\mathcal{L}_{CE} = -\frac{1}{n}\left(\sum_{i=1}^{n}\boldsymbol{y}_i \cdot log(\hat{\boldsymbol{y}}_i)\right) \tag{2.12}$$

Dice Loss

Dice Loss is commonly used for segmentation tasks in the MIC field and is defined by Equation (2.13), where DSC is Dice Similarity Coefficient. This is possible since DSC has a range between zero and one, where one represents a perfect overlap between the label and the prediction, and zero represents a label and prediction that do not overlap at all. Minimising DSC loss thus maximises the overlap between labels and predictions.

$$\mathcal{L}_{Dice} = 1 - DSC \tag{2.13}$$

For segmentation, this represents an important change in perspective. Whereas minimising CE loss will maximise pixel-wise accuracy, minimising Dice loss will maximise the overlap of the prediction and the label as a whole. That is, Dice loss considers both global and local information, whereas CE loss considers only local pixel information.

2.4.6 Backward pass in a ANN

A fundamental part of how ANNs learn is the backward pass using *back-propagation*. When a neural network is first instantiated, its weights are set randomly according to some probability distribution, for example the Xavier or He distributions due to Glorot and Bengio [34] and He *et al.* [35] respectively. Once the weights have been properly initialised, our network can produce a prediction, but it will most likely not be any good and will have a high loss value. Our goal is now to adjust the weights so that the loss decreases.

How much each weight should change is determined by the overall gradient of the network, in other words a measure of how much each weight contributes to the total loss of the network. This can be determined by taking the partial derivative of the loss with respect to the weights and biases. We perform these calculations and weight adjustments in a layer-wise fashion, passing the remaining error that cannot be explained in a layer back to the layer before it. In essence, the error gradient, or *delta*, *back-propagates* through the network, which names this algorithm *the back-propagation algorithm*.

First, we find the gradient of the loss with respect to the weighted input z in the last layer, denoted δ^N. Adopting the same notation as in Section 2.4.1, we get Equation (2.14) for each output neuron j with a network with N layers.

$$\delta_j^N = \frac{\partial \mathcal{L}}{\partial h_j^L} g^{N\prime}(z_j^N) \tag{2.14}$$

We can rewrite Equation (2.14) to matrix form in Equation (2.15), where $\odot$ refers to the Hadamard Product.

$$\boldsymbol{\delta}^N = \nabla_h \mathcal{L} \odot g^{N\prime}(\boldsymbol{z}^N) \tag{2.15}$$

With this final delta, we can now calculate the delta for each layer n using the delta of the previous layer, with Equation (2.16).

$$\boldsymbol{\delta}^n = \boldsymbol{W}^{n+1}\boldsymbol{\delta}^{n+1} \odot h^{n\prime}(\boldsymbol{z}^n) \tag{2.16}$$

The delta of each layer can then be used to calculate the contribution of each weight and bias in that layer to the total loss. The contribution of weight $\boldsymbol{W}_{jk}^n$, that is the weight connecting neuron k in layer $n-1$ and neuron j in layer n, is given in Equation (2.17). Similarly, the contribution of the bias can be calculated by Equation (2.18).

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_{jk}^n} = h_k^{n-1}\boldsymbol{\delta}_j^n \tag{2.17}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_j^n} = \boldsymbol{\delta}_j^n \tag{2.18}$$

2.4.7 Gradient descent

Broadly, the equations derived in the previous section described the *gradient* of the parameter space in our network, with respect to the loss. As previously noted, the goal is to navigate this parameter space in such a way that the loss decreases. This is called *gradient decent*, interpreting the loss as a measure of "height" in this parameter space.

Gradient decent works in several iterations, following these steps:

1. Feed the model an input and attain an output.
2. Calculate the loss of the output with respect to some label or performance measure.
3. Calculate the gradients of each layer in the network.
4. Use those gradients to update the weights and biases of the network.

This sequence of operations is repeated either for a specified number of iterations or until a convergence criteria for the model is reached. There are several variations upon gradient decent, depending on how much data is sent through the model at each iteration. The most common variant, called *mini-batch gradient decent*, passes a subset of the dataset at each iteration. The weights and biases are then updated as specified in Equation (2.19) and Equation (2.20), respectively.

$$\boldsymbol{W}^n_{jk} \rightarrow \boldsymbol{W}^n_{jk} - \frac{\epsilon}{m} \sum^{m} \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^n_{jk}} \tag{2.19}$$

$$\boldsymbol{b}^n_{j} \rightarrow \boldsymbol{b}^n_{j} - \frac{\epsilon}{m} \sum^{m} \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^n_{j}} \tag{2.20}$$

Here, ϵ denotes the *learning rate* and m denotes the *mini-batch size*, both of which are *hyper-parameters* that are set manually. The learning rate determines how large of a step we make in the direction of the gradient for each iteration, whilst the minibatch size determines how much data is used for each training step. How large these mini-batches should be is often task-dependent and an area of intense research. Often, a mini-batch size between 2 and 32 is used. Using mini-batches increases the stability of the training and aids in convergence, since the model makes updates to its parameters based on the average of several gradients rather than one, increasing the chances of descending the parameter space in a globally sensible direction. Once all the data points in our training set have passed through the network, the model has completed one *epoch* of training. Usually, several training epochs are needed for the model to converge.

2.4.8 Optimisers

A major issue when training models using gradient decent is navigating the parameter space in such a way that the model avoids getting stuck in *local min-*

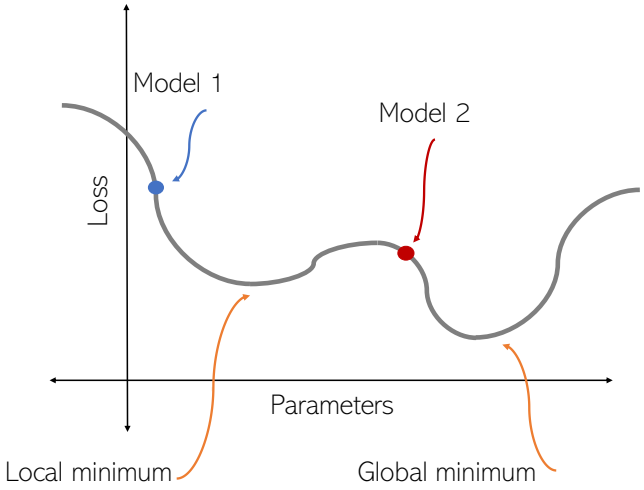


Figure 2.12: Illustration of local and global minima. Without momentum, model 1 will get stuck in a local minimum, while model 2 will reach the global minimum.

ima, as illustrated in Figure 2.12. Although there may exist a more optimal and perhaps a globally optimal solution (a *global minimum*), the model might fail to reach it, as the cost of escaping the valley in which it has settled is too large. One way to escape such local minima is to use a form of *momentum*, usually denoted as ρ. The intuitive idea here is to have the gradient decent algorithm move across the parameter space in a similar way to how a heavy ball would roll down a hill. With sufficient momentum, such a ball would be able to roll over local minima and potentially discover more optimal minima elsewhere in the parameter space. Momentum can also help to deal with saddle points in the space, for much the same reasons that they help with local minima [33].

A typical optimiser with momentum is ADAptive Movement estimation (ADAM) [36]. It computes first- and second-order estimates of momentum for each parameter in the network, using the estimated momentum to adjust the updates to the weights in the model. It also maintains a bias term for the first- and second-order estimates.

2.4.9 Overfitting

Provided sufficient model learning capacity, a model can in theory train until the training loss is zero, effectively memorising the entire training set. Whilst this might sound like an attractive proposition, it is not. The model would most likely not generalise well when exposed to new data that is not present in the training set. This problem is known as *overfitting* [13].

There are several ways to deal with and mitigate overfitting. One approach

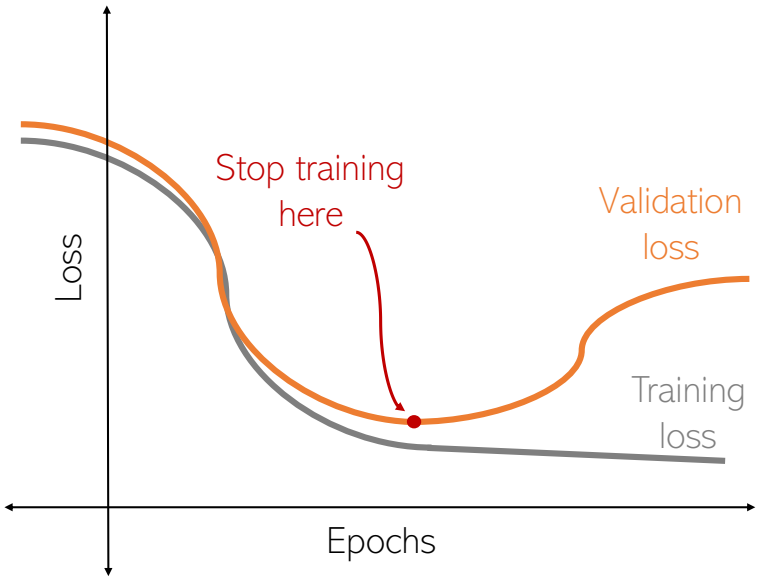


Figure 2.13: Early stopping example. Training stops where validation loss begins to increase.

is to use *early stopping*. First, the training set is divided into two parts: a training set and an validation set. The new, smaller training set is used as before. The validation set is held out during training and only applied when the training epoch is complete. Then, the loss of the model is evaluated on the validation data. This validation loss is *not* used for training directly, but rather compared with validation losses calculated over previous epochs. If the validation loss is less than the previous calculated validation loss, training continues. If, on the other hand, the validation loss has increased, we interpret this as the start of overfitting and terminate training. In practice, validation losses can fluctuate during training, so a patience parameter is often used to determine how many subsequent epochs of increasing validation loss are tolerated before model training is terminated. An illustration of this approach can be seen in Figure 2.13.

Another approach is to apply *regularisation*. Specifically, for neural networks, we apply *parameter regularisation*. In essence, this technique punishes the model for being overly complex by increasing the loss as complexity increases. In this context, high weigh values are equated to high complexity. Two common techniques for regularisation are L1 and L2 regularisation [33]. When regularisation is applied, our loss function is slightly modified, as can be seen in Equation (2.21).

$$\hat{\mathcal{L}}(\boldsymbol{\phi}; \boldsymbol{X}, \boldsymbol{y}) = \mathcal{L}(\boldsymbol{\phi}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\phi}) \tag{2.21}$$

Here, $\boldsymbol{\phi}$ denotes the parameters of the network (the weights), Ω the chosen regularisation function, and α the contribution of the regularisation to the loss.

When applying L1 regularisation, Ω is as seen in Equation (2.22).

$$\Omega(\boldsymbol{\phi}) = ||\boldsymbol{w}||_1 = \sum_i |w_i| \tag{2.22}$$

When applying L2 regularisation, Ω is as seen in Equation (2.23).

$$\Omega(\boldsymbol{\phi}) = \frac{1}{2}||\boldsymbol{w}||_2^2 = \frac{1}{2}\sum_i w_i^2 \tag{2.23}$$

Both of these techniques work by pushing the weights towards zero in an effort to reduce the number of features in a network and thus make the network focus more on essential features in the input. Their effect on the network differs slightly. Whilst L1 regularisation pushes weights towards zero in order to get rid of features in general, L2 regularisation keeps weights small in general.

A third option for dealing with overfitting is to apply *dropout*. Dropout is a technique where the activations of hidden nodes in the network are randomly set to zero in each training step. The effect of this is that instead of training one network, an ensemble of sub-networks is trained instead [33]. Essentially, we encourage the network to become more robust by forcing it to learn how to route information through the network across several different paths, discouraging specialised routes through the network that might lead to overfitting.

2.4.10 Batch Normalisation

A batch normalisation layer [37] normalises the mini-batch input to an internal layer of the network. That is, it redistributes the values of the input so that the mean is centred around zero. This can be done by calculating the mean μ and standard deviation σ across the minibatch. Let $\boldsymbol{X}$ be a batch of inputs to an internal layer in the network. Then, the normalised mini-batch $\hat{\boldsymbol{X}}$ can be computed as in Equation (2.24).

$$\hat{\boldsymbol{X}} = \frac{\boldsymbol{X} - \mu}{\sigma} \tag{2.24}$$

Using the two learned parameters γ and β, the final output of the batch normalisation layer is computed as in Equation (2.25). These parameters exist to preserve the expressive power of the network [33].

$$\gamma\hat{\boldsymbol{X}} + \beta \tag{2.25}$$

Applying batch normalisation allows the use of higher learning rates, saturating activation functions and makes network performance less sensitive to weight initialisation. It also seems to act as a form of regularisation, it reduces the effect of vanishing or exploding gradients and increases training speed [37]. The exact reason why batch normalisation works so well is poorly understood. Ioffe and Szegedy [37], the creators of batch normalisation, claim in their paper that it works by reducing *internal covariate shift*, where parameter initialisation and changes in the distribution of the inputs of each layer affect the learning rate of the network. More recent research by Santurkar *et al.* [38] suggests that the gains can be attributed to batch normalisation smoothing out the objective function, thus increasing performance.

2.4.11 Layer Normalisation

Proposed by Ba *et al.* [39], Layer Normalisation attempts to address some of the drawbacks of Batch Normalisation. Since the calculation of the mean μ and standard deviation σ is dependent on the size of the minibatch, their quality as statistical estimators of the true values of μ and σ across the whole dataset degrades as the size of the minibatch decreases. Batch Normalisation is also difficult to apply to sequence-focused data since the length of each sequence in the dataset can vary, making calculation of the mean and standard deviation non-obvious.

Layer Normalisation attempts to deal with this by taking the mean and standard deviation along the feature dimension rather than the batch dimension. That is, Layer Normalisation takes the mean and standard variation of the features of a single case and uses it to normalise the input, rather than taking the mean of each feature across a batch. This means that Layer Normalisation can work independent of batch size. This method of normalisation works especially well with Recurrent Neural Networks [39], which is why it has seen extensive use in models for NLP.

2.4.12 Gradient Accumulation

A challenge that can arise when dealing with ANNs is to fit all the data in a training mini-batch into memory. This is particularly challenging in Computer Vision tasks, where the inputs are large multidimensional arrays and training is often done on GPUs. As previously noted, larger mini-batch sizes can aid in training speed and stability, but such gains are lost if we have to reduce the batch size in order to fit our data onto our computing accelerators.

One technique that is often used to attain the advantages of a larger mini-batch size whilst keeping it small enough to fit all the data on the accelerator is *gradient accumulation*. Instead of updating the weights immediately after a mini-batch has been sent through, we send another mini-batch through and accumulate the gradients of the two mini-batches. We can keep accumulating

the gradients of say k mini-batches. Only when the k mini-batches have been sent through do we update the weights. If the mini-batch size is m, the effective batch size of the weight update is $k \cdot m$. Whilst some of the speed gains of larger batch-sizes might be lost, the training still enjoys the stability of the larger mini-batch size.

2.5 Convolutional Neural Network (CNN)

Although Artificial Neural Networks can form strong predictive models, they scale poorly to large inputs since each neuron in each layer is connected to every neuron in the next layer. This leads to very high memory requirements in order to store all the weights in the model. For example, for a 1024×1024 pixel input image, the first layer would need to have 1024^2 input neurons. With a hidden layer of size n, we would get $1024^2 * n$ weights to connect the two layers.

A technique commonly applied to deal with this is parameter sharing, often expressed through Convolutional Neural Networks. Instead of regular neural layers, CNNs employ so-called *convolution layers*. Convolutional layers can be applied to any type of data organised in a grid-like topology in a n-dimensional input, for example 1D time series, 2D images or 3D imaging scans.

To achieve this, convolutional layers utilise the *convolution operation* for which they are named. For a 2D input, it is defined as in Equation (2.26).

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{2.26}$$

In Equation (2.26), I is a 2D input and K is a $m \times n$ kernel of weights. The equation defines an operation where the kernel K is slid over the input I , where the value of each cell in I is multiplied with the corresponding cell in K and then summed, as illustrated in Figure 2.14. The output is a feature map S, where $S(i,j)$ is the result of the convolution operation over I at row i and column j of S. In some convolutional layers, an activation function is applied element-wise over S after the convolution operation has been applied. Additionally, a bias may also be applied.

The spatial size of the feature map S depends heavily upon the sizes of K and I, as well as the stride, padding, and dilation of the convolution. The stride of the convolution refers to how far the kernel will slide over the input I between each application. The padding refers to whether or not the input has had zeros added around it (padded) in each spatial dimension. Finally, dilation refers to how much distance there should be between each value in the input to be applied with the kernel, if any (see Figure 2.15).

A convolutional layer can employ several kernels, each with its own set of weights, producing a separate feature map S. That is, a convolutional layer with n kernels produces as output a stack of n feature maps.

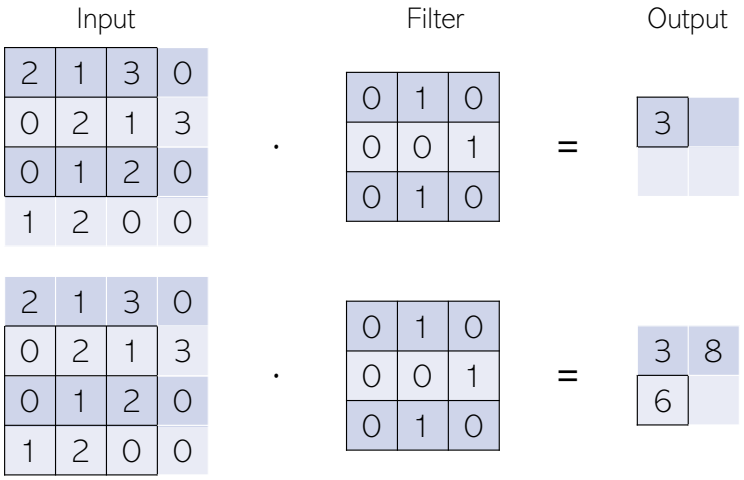


Figure 2.14: Illustration of a 2D convolution.

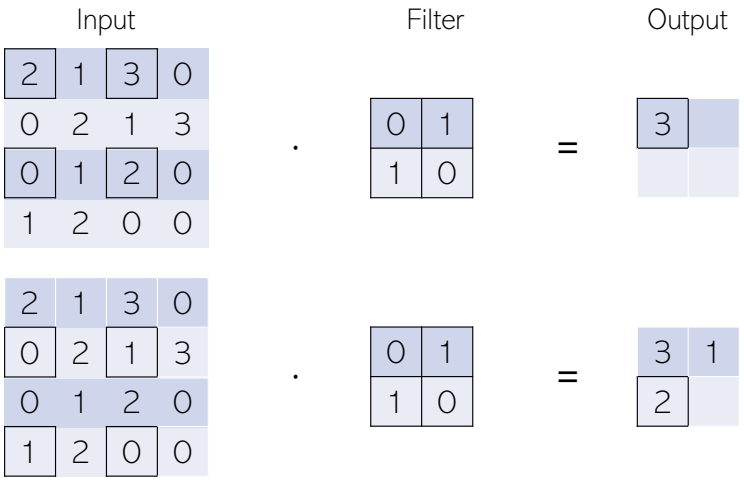


Figure 2.15: Illustration of dilated convolution.

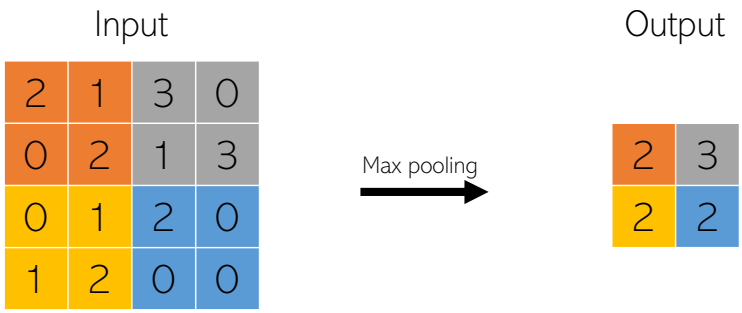


Figure 2.16: Max pooling example.

Crucially, since the kernels involved in the convolution operation only connect a $m \times n$ subset of neurons together between layers, significant parameter savings can be achieved. Additionally, since the kernel is simply sled across the input, the weights are shared between each input subset, which further contributes to the reduction in memory usage. During training, each filter learns to recognise patterns in the input. As the kernels are slid across the input, we can recognise these patterns anywhere in the input, a property known as equivariance [33].

Pooling

A commonly applied operation in CNNs is pooling. This is an operation that reduces the input resolution whilst retaining important information. Whilst there exist many variants of pooling, the most common are average-pooling and max-pooling. The latter is described here.

The max-pooling operation simply selects the maximum value within the pooling filter and forwards it to the next layer. An example can be seen in Figure 2.16. This operation essentially summarises the input, letting only the strongest signals through, and reducing the spatial resolution of the input in the process.

Transposed Convolution

The previous two sections have looked at operations that scale down their inputs. A similar set of operations also exists to increase the spatial resolution of an input, called up-scaling. Traditional approaches employ purely computational approaches like nearest-neighbour interpolation, bilinear interpolation, and other approaches. Another approach, called Transposed Convolution, introduces learnable parameters to this up-scaling operation in a similar fashion to traditional convolution. Instead of using kernels to reduce the resolution of the input, they scale the image up. An illustration of the operation can be seen in Figure 2.17.

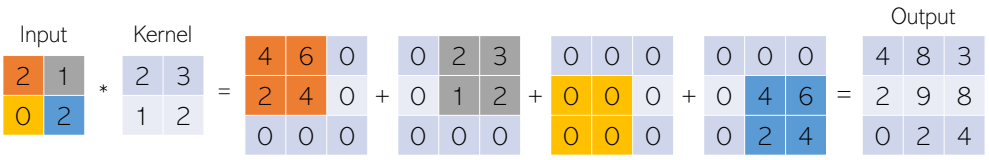


Figure 2.17: Transposed convolution example.

2.5.1 U-Net

Whilst there exists a wealth of different CNN architectures that take advantage of the advantageous properties of convolutional layers in CV, this section will focus on a specific architecture that has seen successful application on a broad spectrum of different CV tasks: The U-Net [40].

A U-Net commonly consists of three distinct components: An encoder, a bridge (also called a bottleneck), and a decoder. In the encoder, the input is down-sampled whilst the number of channels is increased in stages through the successive application of convolutional and pooling layers with an increasing number of kernels for each convolution. In the bridge, more convolutions are applied, but the resolution is maintained. The bridge is where the input resolution is at its lowest, whilst the number of channels is the highest. This component is placed at the end of the encoder and at the start of the decoder, joining them. In the decoder, the input is up-scaled back to the original input resolution in stages through the successive application of transpose convolutional layers with a decreasing number of kernels. The encoder and decoder are, in addition to the bridge, connected at each stage by skip connections in order to preserve the spatial information of the input [40].

In a nutshell, the encoder *encodes* the spatial information of the input into a dense feature space represented in the channels of the input. The decoder then takes this densely encoded input and *decodes* it back into the original resolution, providing a representation of the input with high semantic value. This output can then be fed to a classifier to produce a final prediction.

2.6 Transformers

Transformers were introduced by Vaswani *et al.* [1] in their seminal paper, "Attention is all you need". They proposed a novel model for Natural Language Processing (NLP) based on an attention mechanism called self-attention. This model, and subsequent evolutions, have set new records in the NLP field and currently dominate the state of the art for many tasks in that field [1–3]. In this section, we provide a brief overview of how the Transformer model works. Interested readers who wish to explore the details of the model are referred to the original paper by Vaswani *et al.* [1].

There are several key ideas to explore when addressing Transformers, foremost of which is self-attention, multi-head self-attention, and the Encoder-

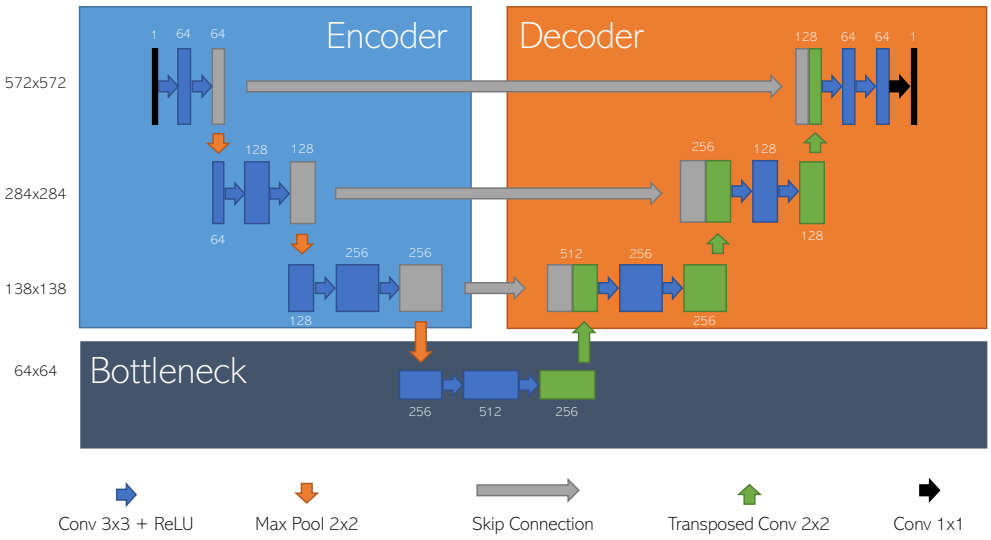


Figure 2.18: A typical U-Net architecture.

Decoder structure of the model. Each aspect is explored in turn, before an overview is presented in Section 2.6.3. We also present efforts to reduce the computational complexity of Transformer models in Section 2.6.4.

2.6.1 Self-Attention

The essence of self-attention is this: Given a sequence of tokens[1], how relevant is each token to every other token in the sequence? Self-Attention tries to answer this, which in turn explicitly models the interactions between all the entities in the sequence. In essence, a self-attention layer weights up and down the different tokens in the sequence based on how important, i.e. worthy of attention, they are relative to the other tokens in the sequence. Let $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ denote a sequence of n entity vectors of length d. We then define three learnable weight matrices to transform $\boldsymbol{X}$ into a queries matrix, a keys matrix, and a values matrix as seen in Equation (2.27), where $d_k = d_q$.

$$\boldsymbol{W}^Q \in \mathbb{R}^{d \times d_q}, \boldsymbol{W}^K \in \mathbb{R}^{d \times d_K}, \boldsymbol{W}^V \in \mathbb{R}^{d \times d_v} \tag{2.27}$$

The queries ($\boldsymbol{Q} \in \mathbb{R}^{n \times d_q}$), keys ($\boldsymbol{K} \in \mathbb{R}^{n \times d_k}$), and values ($\boldsymbol{V} \in \mathbb{R}^{n \times d_v}$) are then calculated by projecting X onto these matrices, as seen in Equation (2.28).

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q, \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K, \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V \tag{2.28}$$

Then, the final output $\boldsymbol{Z} \in \mathbb{R}^{n \times d_v}$ can be calculated as seen in Equation (2.29).

[1] Say, a sentence of words, embedded using a word embedding.

$$\boldsymbol{Z} = Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = Softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\intercal}{\sqrt{d_q}}\right)\boldsymbol{V} \tag{2.29}$$

Intuitively, we can understand the queries as "what token i is interested in knowing from the other tokens", and the keys as "what token i can provide as information to the other tokens". For a given token in the sequence, the operation above the fraction ($\boldsymbol{Q}\boldsymbol{K}^\intercal$) computes the dot product between the key and query for that token. This can be interpreted as evaluating the agreement between the query and key vectors. Similar vectors will receive a high attention value, whilst dissimilar vectors receive a low attention value. Attention values are scaled by the square root of the query dimension in order to improve training stability [1], before they are normalised using $Softmax$. The final output is then calculated by multiplying these normalised attention scores by $\boldsymbol{V}$.

It should be noted that self-attention is, in contrast to standard ANNs and CNNs, invariant to the position of each token. That is, self-attention cannot capture the positional information of each token in a sequence. In many domains, however, the position of each token can have a profound impact on the overall meaning of the sequence, say for example the order of words in a sentence. In order to address this, Transformers adds a positional encoding to its inputs, either concatenating or adding the encoding to the input. Typically, the positional encoding is a learnable encoding or some function of the position of the token.

A variation of standard self-attention, called masked self-attention, is also used in the standard transformer model [1]. When performing word prediction, for example, the model should not attend to "future" tokens in the sequence before these tokens have actually been predicted. Masked self-attention is achieved by element-wise multiplication with a mask $\boldsymbol{M} \in \mathbb{R}^{n \times n}$, where $\boldsymbol{M}$ is an upper-triangular matrix. Masked self-attention is then defined as in Equation (2.30), where $\odot$ denotes the Hadamard product. In essence, the attention scores of future entities in the sequence are set to zero.

$$\boldsymbol{Z}_M = MaskedAttention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{M}) = Softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\intercal}{\sqrt{d_q}} \odot \boldsymbol{M}\right)\boldsymbol{V} \tag{2.30}$$

2.6.2 Multi-Head Self-Attention

A layer with Self-Attention can in itself model a relatively complex interaction between the different tokens, but often there are several interactions that could and should be modelled. For example, when processing language, the structure of how adjectives relate to different nouns, or how verbs relate to different nouns, etc., are all interesting factors to account for in a wide range of different NLP tasks. To address this, multiple self-attention *heads* are used

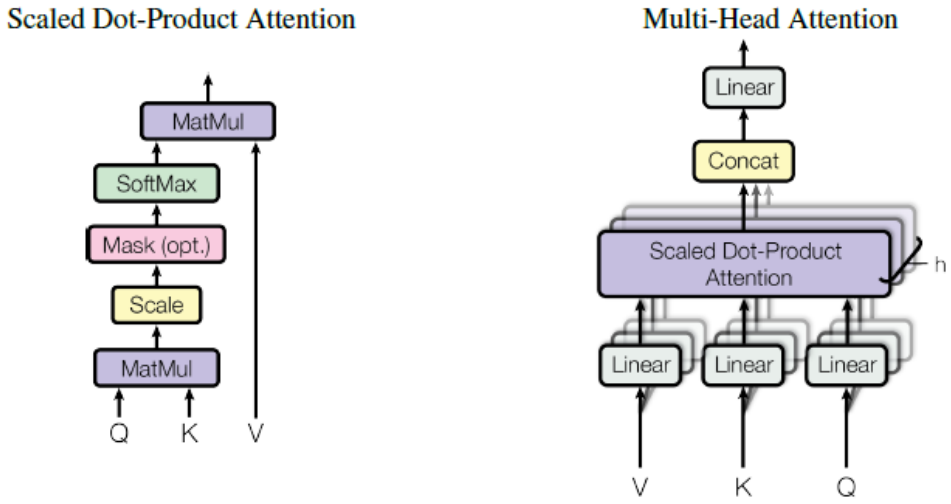


Figure 2.19: Self-attention (left) and Multi-head self-attention (right). Figure from [1].

in each block. Here, each block has its own set of query, key, and value weight matrices, denoted $\boldsymbol{W}_i^Q, \boldsymbol{W}_i^K, \boldsymbol{W}_i^V$. Given an input $\boldsymbol{X}$ and a number of heads h, three groups of vectors are calculated: The query group $\boldsymbol{Q}'$, the key group $\boldsymbol{K}'$, and the value group $\boldsymbol{V}'$, all calculated from the input and the corresponding weight matrices for each head. To clarify, each group consists of several matrices, where each matrix is calculated as seen in Equation (2.31).

$$\boldsymbol{Q}_i' = \boldsymbol{X}\boldsymbol{W}_i^Q, \boldsymbol{K}_i' = \boldsymbol{X}\boldsymbol{W}_i^K, \boldsymbol{V}_i' = \boldsymbol{X}\boldsymbol{W}_i^V \tag{2.31}$$

Multi-head self-attention can then be formulated as seen in Equation (2.32).

$$\begin{aligned} MultiHead(\boldsymbol{Q}', \boldsymbol{K}', \boldsymbol{V}') &= Concat(head_1, head_2, \ldots, head_h)\boldsymbol{W}^O, \\ \text{where } head_i &= Attention(\boldsymbol{Q}_i', \boldsymbol{K}\prime_i, \boldsymbol{V}_i') \end{aligned} \tag{2.32}$$

Here, $\boldsymbol{W}^O \in \mathbb{R}^{d_{model} \times d_{model}}$ is a learnable linear projection matrix that projects the result of the concatenated output of each head into the dimensions of the model, where d_{model} is the size of the outputs internally in the model. A visual summary of these operations can be seen in Figure 2.19.

2.6.3 Model structure

As can be seen in Figure 2.20, the model also makes extensive use of residual connections across the different subcomponents of each transformer module. This strengthens the flow of information through the model. A layer-normalisation follows each residual connection. Finally, each module is topped

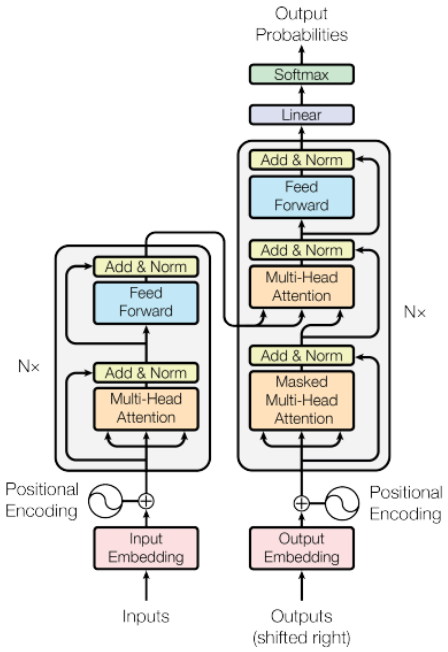


Figure 2.20: Full Transformer architecture. Figure from [1].

by a feed-forward network that consists of two linear transformation layers and a non-linear activation function between then, which can be denoted as in Equation (2.33), where $\boldsymbol{W_1}$ and $\boldsymbol{W_2}$ are the weight matrices of the linear transformations, $\boldsymbol{b_1}$ and $\boldsymbol{b_2}$ are the bias vectors of the linear transformations, and g denotes the activation function.

$$FFN(\boldsymbol{X}) = \boldsymbol{W}_2 g(\boldsymbol{W}_1 \cdot \boldsymbol{X} + \boldsymbol{b_1}) + \boldsymbol{b}_2 \tag{2.33}$$

As can be seen in Figure 2.20, the model makes use of an encoder-decoder architecture where the encoder output is fed into the decoder blocks. The use of the encoder and decoder modules of the original Transformer model varies from model to model. Some models, such as those of the BERT family [2], are based on the encoder, which can then be regarded as a type of feature extractor. Other models, such as the GPT family of models [3], are based on the decoder.

2.6.4 Speeding up the Transformer

An area of study that has received much attention is how to reduce the computational complexity of the Transformer. Recall from the description of self-attention that it performs matrix multiplication of two matrices, the queries ($\boldsymbol{Q} \in \mathbb{R}^{n \times d_q}$) and the transposed keys ($\boldsymbol{K^\intercal} \in \mathbb{R}^{d_k \times n}$), which has computational complexity $\mathcal{O}(n^2 d_k) = \mathcal{O}(n^2)$. That is, the computational complexity is quad-

ratic in the length of the sequence. All other parameters in the Transformer being constant, this means that the computational complexity of the Transformer as a whole is $\mathcal{O}(n^2)$. While this is manageable for sequences of short and medium lengths, it can become exceedingly slow for longer sequences. Motivated by this, several approaches for reducing the complexity have been explored in the literature. Tay *et al.* [41] provide a comprehensive survey of efforts to build more efficient transformers. A couple of highlights are briefly summarised below.

Reformer

Kitaev *et al.* [5] introduced the Reformer, which uses locality-sensitive hashing in several rounds of self-attention in order to quickly identify similar keys and queries that would yield non-trivial attention values. Additionally, using reversible layers and chunking of Feed Forward Network activation calculations, they are able to reduce the computational complexity to $\mathcal{O}(nlog(n))$. They demonstrate comparable performance with the standard Transformer in a English to German translation task.

Linformer

Wang *et al.* [6] introduces the Linformer, where they show that self-attention can be approximated by a low-rank matrix, reducing the computational and spatial complexity to $\mathcal{O}(n)$. They also demonstrate similar performance with BERT-based baselines on several NLP tasks.

2.7 Vision Transformers

Inspired by the success of self-attention in NLP, several works in CV attempted to combine elements from the attention mechanism from the Transformer with standard convolutional networks [42] or otherwise combine Transformers with convolutional backbones [43]. Some works have taken this a step further and replaced convolutions entirely with self-attention and related variants [44, 45].

Motivated by the scaling successes of the Transformer in NLP, Dosovitskiy *et al.* [4] experimented with the application of a standard Transformer with minimal modifications directly to images for image classification. They were able to do this by splitting each image into a sequence of flattened and projected patches, which were then fed to the model as words along with a BERT-inspired class token [2]. Their model, named the Vision Transformer (ViT), consisted of a sequence of Transformer encoder blocks, topped with a MLP head that received the class token for class prediction. As with the standard Transformer by Vaswani *et al.* [1], they injected each patch with a

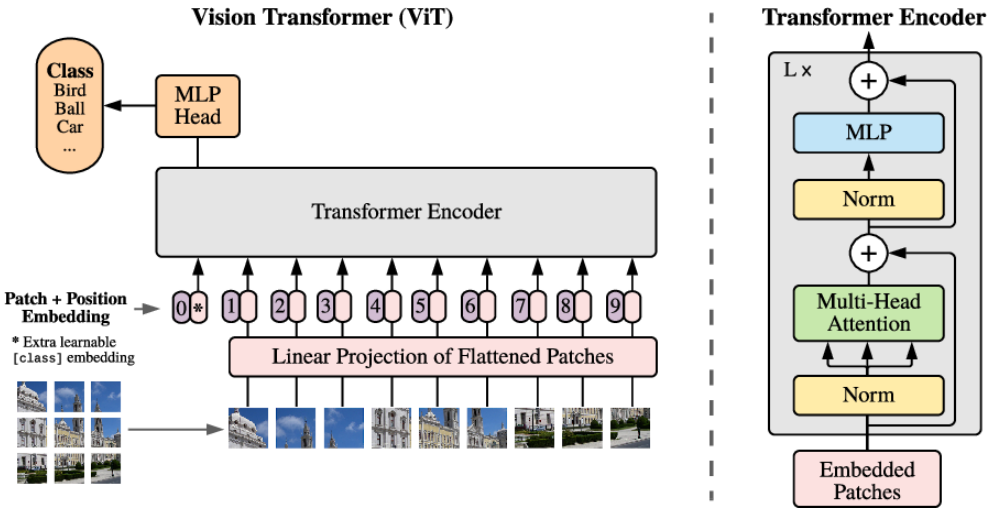


Figure 2.21: Overview of the original Vision Transformer (ViT). From [4].

position embedding, which in this case was a 1D learnable position embedding. By leveraging large scale pre-training using the JFT-300M dataset [46], they were able to match or outperform SotA at the time on ImageNet [47], CIFAR-10, CIFAR-100 [48], and VTAB [49] tasks, whilst requiring significantly less resources to train. An illustration of the model can be seen in Figure 2.21.

The success of the ViT model inspired several new CV publications using Transformers, with 4 096 publications citing the original ViT-paper by Dosovitskiy *et al.* [4] since October 2020 at the time of writing. Khan *et al.* [50] and Han *et al.* [51] have conducted extensive surveys on the use of Vision Transformer in Computer Vision. The rest of this section highlights important aspects of ViT models and trends, as well as a hand-full of general ViT models.

2.7.1 Why ViT?

In addition to the large scale of Vision Transformer models, there are other qualities that also make ViTs desirable for CV. One such attribute is the incorporation of global information. As noted previously in Section 2.6, transformer models are generally quite good at modelling long-range dependencies in their input sequences, due to their self-attention mechanism. Standard convolutional neural networks, on the other hand, are designed to attend primarily locally. Pooling layers can be introduced to increase the effective perceptive field, but information is lost as the network grows deeper and the perceptive field increases. It can be likened to how a near-sighted person might see an image: Up close, the fine details are crisp and clear, but much of the image is out of view. As the image moves further away from the person, more and more of the image becomes visible, but it becomes progressively more blurry and unclear; information is lost. This effect is illustrated in Figure 2.22. ViTs

Figure 2.22: An illustration of the blurriness.

are able to better deal with this, since they can attend to all the tokens globally in the first layer. This means that they are much better at attending to long-range dependencies in the input, even in the early layers of the model.

2.7.2 How ViTs learn

Raghu *et al.* [52] performed an interesting comparison between ResNets and ViT (the model referred to in [4]), comparing how the two models learn. As their results are relevant for the discussion in the previous subsection, a brief summary of their findings is presented in this subsection.

They find, for example, that while ViT attends globally at every stage, it starts out attending highly locally and gradually more and more globally in the higher layers. Interestingly, these patterns emerge only when the model has been trained on sufficient data. With less data, the ViT tends to attend more globally. This seems to indicate that the ViT has to *learn* to attend locally, something that CNN-based ResNets do automatically as an inductive bias owing to their structure.

When comparing the two models, the authors also find that the lower half of the ResNet layers are similar to the lower quarter of layers in the ViT, with the remaining half[2] of the ResNet layers being similar to the next third of ViT layers, with the final layers being quite dissimilar. The dissimilarity of the final layers is explained by the ViT primarily modifying the class token in these layers. The similarities in the other layers are interesting, as it seems to indicate that both ViTs and ResNets learn to compute similar representations in their

[2]Minus a handfull of layers

lower layers. However, the authors are able to demonstrate that ViT is capable of incorporating far more global information, which could explain their increased performance.

The authors also find that ViT more faithfully preserves spatial information in the input, that is, the input and output tokens are much more similar in ViT compared to ResNets, which could be promising for object detection and segmentation tasks.

2.7.3 Speeding up ViT

As with standard Transformers, Vision Transformers also struggle with the quadratic memory and time complexity of self attention, as described in Section 2.6. In fact, the issue is even bigger for Vision Transformers, which fundamentally deal with images and video. If our model is to attend to all the pixels of a $H \times W$ image, the length of the input sequence to the transformer would be $n = HW$, giving complexity $\mathcal{O}(n^2) = \mathcal{O}(W^2H^2)$. When dealing with three dimensional volumes, like video or medical imaging scans, the sequence length grows even longer to $\mathcal{O}(n^2) = \mathcal{O}(W^2H^2D^2)$ where D is the depth of the volume.

Early Vision Transformers like ViT and DPT [4, 53] deal with this by extracting and embedding patches from the image, reducing the sequence length to $n = \frac{H}{p}\frac{W}{p}$ where p is the patch size. Although this reduces n and thus makes the problem manageable, it still does not directly deal with the quadratic exponent in the complexity. Some newer works like Twins, the Swin Transformer, and the Perceiver [8, 54, 55] make architectural changes, including changes to the attention mechanism, that makes the complexity sub-quadratic. The latter two, the Swin Transformer and the Perceiver, even manage to make it linear in the size of n. Twins and the Swin Transformer primarily achieve their speedups by cleverly applying local and global attention, whilst Perceiver passes its input through a latent bottleneck that constricts the time and space complexity. Another approach, favoured by the Axial Transformer [56], replaces standard self-attention with a row- and column-wise variant with similar linear complexity.

2.7.4 Relevant general Vision Transformer architectures

This section will quickly summarise major general Vision Transformer architectures that have had major impact on the field, but that are not necessarily directly related to the experiments in this project.

Vision Transformers for Dense Prediction

Ranftl *et al.* [53] proposed the Dense Prediction Transformer (DPT) in March 2021, setting new records at the time for mono-ocular depth estimation and

competitive performance for semantic segmentation. They use a Transformer-based backbone. The backbone receives a sequence of embedded patches of the image, produced either by a linear projection or a convolutional feature extractor. The backbone itself consists of several layers with transformer encoders, each processing at a different resolution depending on the output of the previous layer. The output of each level is fused back together through a ResNet-based feature fusion module. Before fusion, they employ a reassembling module that reassembles the output tokens of each level into an image representation of each token which is then re-sampled to set patch size and dimensionality. They top their model with a task-specific head, taking the fused feature map as input. They also employ pre-training of their Transformer backbone using ImageNet-1K and 21K. For semantic segmentation, they achieve a mIoU score of 49.02% on the ADE20K [57, 58] validation dataset and 60.46% with the Pascal Context [59] validation set, using a ResNet-based feature extractor.

Swin Transformer

Liu *et al.* [8] proposed the Swin Transformer backbone in March 2021, setting new records at the time for object detection with COCO and semantic segmentation with ADE20K. They employ a hierarchical approach, taking 8×8 patches of the image and processing them at different resolutions at each stage of the encoder. The encoder consists of specialised Swin Transformer blocks that perform self-attention within each patch rather than across the entire input. Information is spread between windows by a *shifted window partitioning* mechanism that changes the partition layout in each succeeding module and computes self-attention within the new windows. This approach gives the Swin Transformer linear time and space complexity. A *cyclic shift* algorithm using padding and masked self-attention is used to deal with shifted windows that might not completely align with the size of the overall input. In addition, *relative position* bias is used instead of position embedding. An overview of the model can be seen in Figure 2.23 With pre-training on ImageNet-21K, Swin achieves 53.5% mIoU on the ADE20K validation dataset when used as a backbone for UPerNet [60].

A further improvement to the model, dubbed the Swin Transformer V2 [61], was proposed in November 2021 and further improves upon these results, scaling up the model to 3 billion parameters. These improvements include a more efficient implementation of relative bias, the use of *cosine self-attention*, and post-normalisation rather than pre-normalisation of the data in the Transformer modules. They report State of the Art performance on ADE20K for semantic segmentation, achieving 59.9% mIoU as a backbone for UperNet.

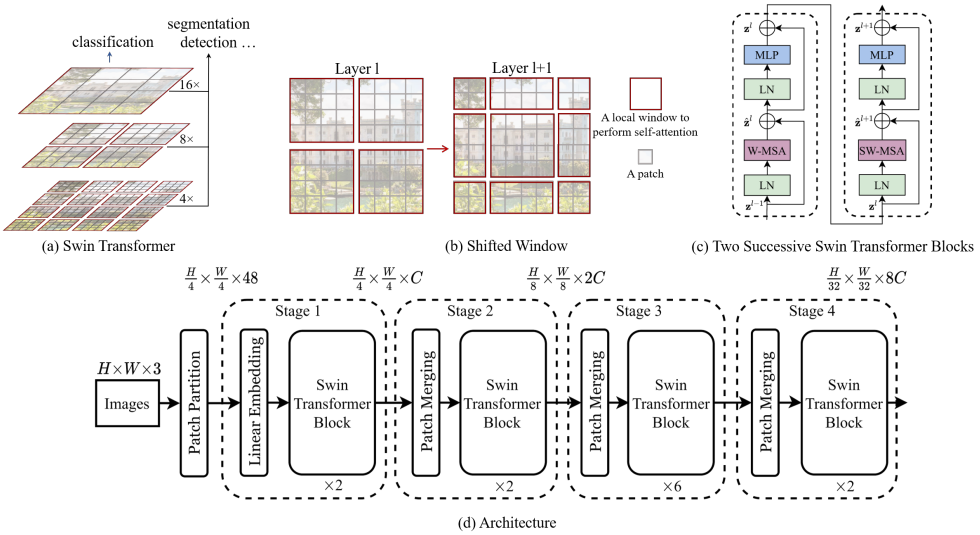


Figure 2.23: An overview of the Swin Transformer. Figures from [8].

Twins

Chu *et al.* [54] proposed the Twins Transformer backbone in April 2021, setting a new state of the art at the time for COCO object detection and showing strong performance on semantic segmentation with ADE20K. They propose two methods. One based on the Pyramid Vision Transformer (PVT) [62] and the Conditional Position Encoding Vision Transformer (CPVT) [63], dubbed the Twins-PCPVT. This method uses a hierarchical approach, where each stage of the encoder processes progressively lower-resolution feature maps. The conditional position encoding generator from CPVT is used to impart positional information in each stage of the encoder.

The other approach, dubbed Twins-SVT, uses another approach with so-called Spatially Separable Self-Attention (SSSA), which is composed of Locally-grouped Self-Attention (LSA) and Global Sub-sampled Attention (GSA). LSA is similar to how self-attention works in Swin: Self-attention is applied locally within the input windows. Unlike how Swin shares information globally with shifting windows, Twins achieves this using GSA. GSA extracts a sub-sample from each group using regular strided convolutions and computes self-attention across these sub-samples. This variant also uses the position encoding generator from CPVT. Altogether, SSSA gives this method linear complexity in time and space.

They report 48.8% mIoU with the ADE20K validation dataset, with pre-training on ImageNet-1K.

2.8 Self-supervised learning

Self-Supervised Learning (SSL) is a machine learning method that learns from unlabelled sample data, as a kind of intermediate form between supervised and unsupervised learning. It is a two-stage process: First, the model is trained using pseudo-labels that pre-condition the network parameters for the actual task. Then, the model is trained for its actual task using supervised or unsupervised learning. Crucially, for SSL, these pseudo-labels are generated from input data by the training procedure itself at train time. This allows the training of effective models in domains where labelled data is scarce and resource intensive to obtain, by pre-training the models on more extensive sets of unlabelled data or by leveraging the limited labelled data available more efficiently.

The SSL field as a whole is too broad and complex to fit within the confines of this thesis. Interested readers are referred to Jing and Tian [64] and Jaiswal *et al.* [65], who have conducted extensive surveys of the field. The remainder of this section will briefly treat significant concepts and methods in the field that are relevant for this thesis.

2.8.1 Pretext tasks

A relatively straightforward approach to self-supervised learning is to generate pseudo-labels on the data for some pretext task. Example pretext tasks include image reconstruction from a transformed version of the raw input. The raw input could, for example, have had parts of it removed, colour transformations introduced, or noise added. The pretext task is then to recover the original input image.

Another type of pretext task is geometric transformation, where the original image could have been exposed to flips, crops, and rotation. When applying rotation, a typical task could be to predict how many degrees the image has been rotated, typically in 90° increments.

2.8.2 Contrastitive Methods

Whilst training directly on pretext tasks can be effective, several newer methods utilise so-called contrastitive methods. In a contrastitive method, samples are drawn from the data set in large batches. Similar samples are considered positive samples, whilst dissimilar samples are considered negative. The objective of the pre-training is to have the model produce representations of positive sample pairs that are relatively similar, whilst representations of negative sample pairs are relatively dissimilar.

SimCLR

SimCLR [66] (A Simple Framework of Contrastitive Learning of Visual Representations) is a prominent contrastitive method used for pre-training image encoders. It works by drawing samples in large batches from the dataset. Each sample is then transformed into a pair of augmented views of the original data, by applying a stochastic augmentation chain. In SimCLR, the augmentations applied are a random crop, a colour shift, and Gaussian blur. This pair of augmented views is considered a *positive pair*. The other samples in the batch (and indeed later their augmented views) are considered negative to this positive pair. The pair is then fed into the encoder, which produces its representation of the pair. The representation is finally fed through a MLP – a projector – that projects the representation into a lower dimensional space. Finally, loss is calculated using a contrastitive loss function, as described in 2.34, where N is the batch size, i and j is the positive pair, $\mathbb{I}_{[k \neq i]} \in 0, 1$ is an indicator function that outputs $1 \iff k \neq i$, τ is a temperature parameter, and sim is a similarity measure. For SimCLR, this similarity measure is as described in 2.35.

$$\mathcal{L}_{i,j} = -log \frac{exp(sim(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{I}_{[k \neq i]} exp(sim(\mathbf{z}_i, \mathbf{z}_k)/\tau)} \tag{2.34}$$

$$sim(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^\intercal \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|} \tag{2.35}$$

SimCLR has shown itself to be a capable contrastitive pre-training method, and was SotA at the time of publication. However, it suffers from a significant drawback: It requires very large batch sizes in order to provide a proper supervision signal, using a batch size of 8192 for their strongest experiments. With augmentations, this gives a total of 16384 samples per forward pass, which can be prohibitive if the images have high resolution due to excessive memory usage.

There exists approaches, like MOCO and MOCOv2 [67, 68], that solve this by utilising a memory bank that holds a number of previous mini-batches in a queue, drawing upon that queue to provide a sufficient number of negative samples without having to push large batches through the encoder. These samples do, however, have to be kept in memory during training.

2.8.3 Bootstrap Your Own Latent (BYOL)

Bootstrap Your Own Latent [69] is another approach to self-supervised learning that is able to outperform contrastive methods that require large batch sizes and negative samples, without negative samples and with a significantly smaller batch size.

The method achieves this by using two mostly identical networks: An online network and a target network. The networks consist of the encoder to

be pre-trained and a MLP projection head, with the online network getting an additional prediction head. From an image augmented by a stochastic augmentation chain, the online network is trained to predict the output of the target network. Loss is calculated based on the similarity of the outputs. Crucially, gradients for the target network are discarded. Only the gradients for the online network is back-propagated. Instead, the weights of the target network is updated as a moving average of the weights in the online network. The dynamics of this update can be seen in 2.36, where θ are the parameters of the online network, ξ are the parameters of the target network, $\mathcal{L}_{\theta,\xi}$ is the loss, τ is the decay rate, and η is the learning rate.

$$\begin{aligned} \theta &\leftarrow optimiser(\theta, \nabla_\theta \mathcal{L}_{\theta,\xi}, \eta) \\ \xi &\leftarrow \tau\xi + (1-\tau)\theta \end{aligned} \tag{2.36}$$

Although such a setup might seem vulnerable to collapse, where both the online and target networks attain maximum similarity by outputting nil-predictions, the authors of [69] find that this does not occur in practice. Indeed, Chen and He [70] find that a potential collapse is prevented by discarding of the gradient for the target network.

The method sets new SotA results for both linear evaluation with ImageNet and other vision tasks such as semantic segmentation and object detection with the VOC2012 dataset and depth estimation using the NYU v2 dataset. Importantly, it shows strong performance with smaller batch sizes, effective even at a batch size of 128.

2.9 Related work

This section will list works of specific interest to this project. The core aspects of their contributions are highlighted.

2.9.1 Swin-UNet

In May 2021, Cao *et al.* [9] introduced Swin-UNet: A UNet-like pure Transformer for medical image segmentation in 2D. They employ Swin encoder blocks, patch merging, and patch expansion modules to build an Encoder-Decoder architecture with skip connections, as seen in Figure 2.24.

Patch merging layers use a mechanism in which the input patches are divided into four parts, concatenated, and then reduced by a linear layer in the channel dimension to produce a final output that is half the spatial size and has double the channel dimension ($H \times W \times C \rightarrow H/2 \times W/2 \times 2C$). Patch expansion layers perform a similar operation, but in reverse. It rearranges the input feature dimensions to double the spatial size and applies a linear layer to reduce the feature dimension to half the input dimension ($H/4 \times W/4 \times 4C \rightarrow$

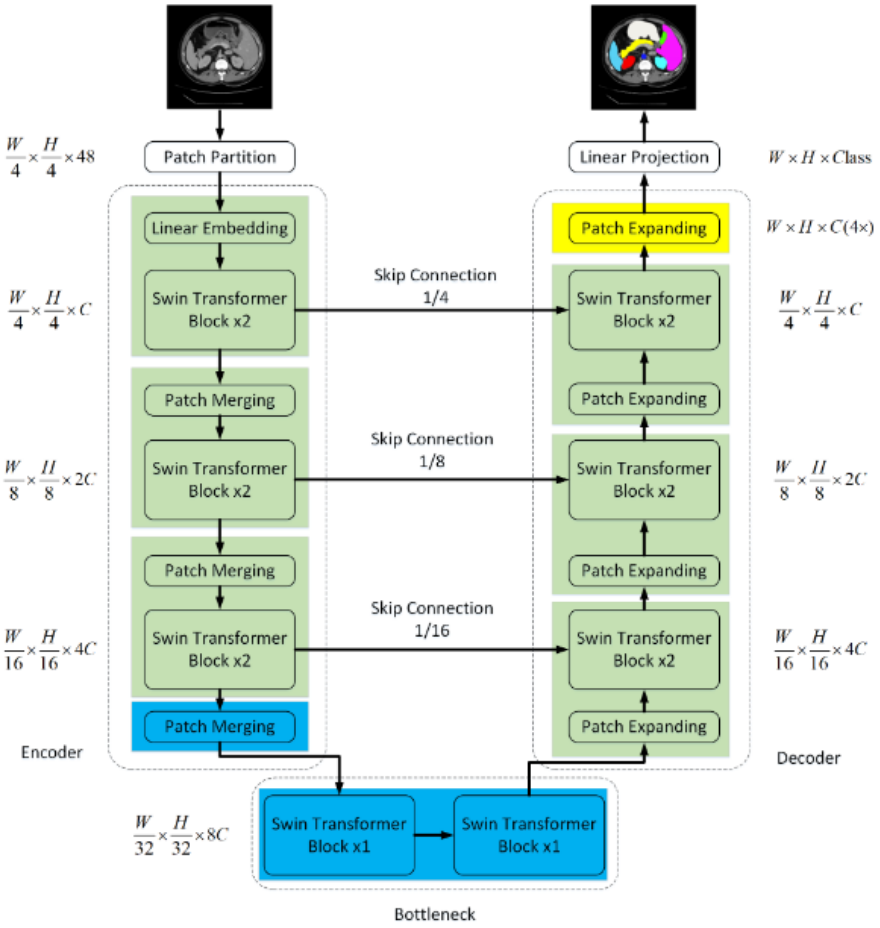
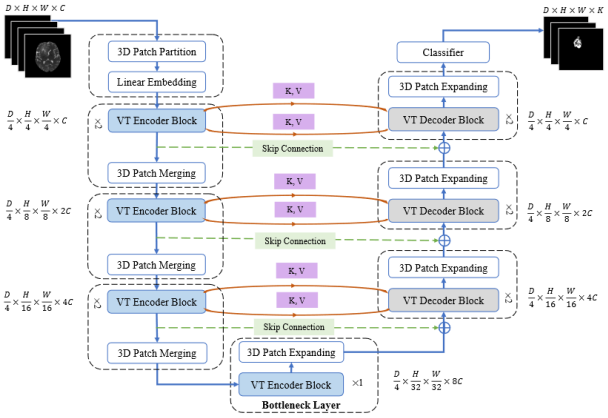
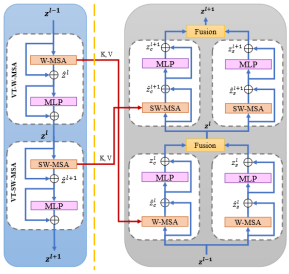


Figure 2.24: The architecture of Swin-UNet, figure from [9].



(a) The architecture of VT-UNet. Note that k corresponds to the number of output classes.



(b) The interaction between encoder and decoder modules in VT-UNet.

Figure 2.25: Overview of VT-UNet. Figures from [10].

$H/2 \times W/2 \times 2C$). Cao *et al.* [9] show empirically that this way of down- and upsampling performs better than using standard and transposed convolutions or Bilinear interpolation.

By applying the model to 2D slices of 3D volumes, Swin-UNet achieves a strong DSC score of 79.13 on the BTCV dataset [9, 23].

2.9.2 VT-UNet

Peiris *et al.* [10] introduced VT-UNet in November 2021. VT-UNet is a Transformer architecture for volumetric medical image segmentation, closely resembling Swin-UNet but in three dimensions rather than two. Indeed, the authors of VT-UNet cite Swin-UNet as an inspiration for their work. Their architecture builds upon the Video Swin Transformer, a Swin Transformer for video data. Using the Video Swin Encoder block, together with patch merging and expansion in width and height, they build a 3D UNet architecture, which can be seen in Figure 2.25a.

Notably, VT-UNet also introduces a novel Cross-Attention mechanism, where the keys and values from each stage in the encoder is shared with the corresponding decoder in the same stage. This allows the model to effectively integrate information across the encoder and decoder branches. The authors show empirically that Cross-Attention aids the performance of the model. Details of the Cross-Attention interaction between the encoder and decoder modules can be seen in Figure 2.25b. Note that the decoder module employs a parallel scheme where one branch operates solely on the input from the previous layer. Fusion is performed using a linear combination of each branch, controlled by a parameter $\alpha, S = \alpha z_1 + (1-\alpha)z_2$.

Their method sets a strong 88.07 average DSC score with an average Haus-

dorff Distance of 7.52 using the BraTS2021 dataset [10, 20–22]. They also showed that the method is robust against noise and artefacts in MRI images.

2.9.3 UNETR

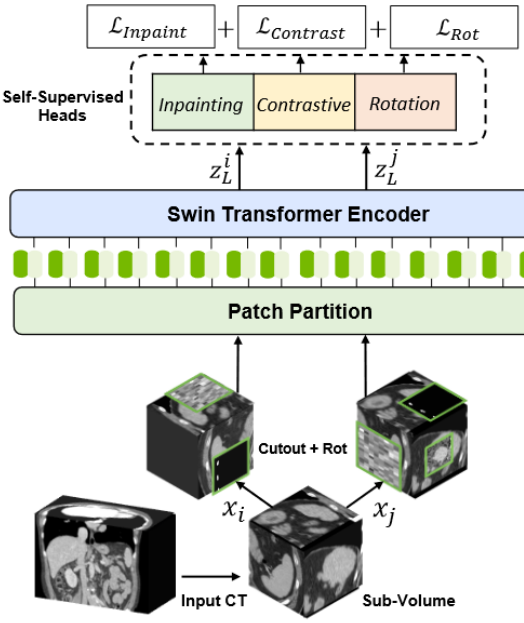
UNETR, introduced by Hatamizadeh *et al.* [71] in March 2021, is a ViT based model that uses a ViT encoder coupled with a conventional CNN decoder. They utilise no patch merging in their model, instead opting for a 12 stage encoder at the same resolution throughout. Their patch embedding layer extracts $16 \times 16 \times 16$ patches from the input volume with an embedding size of $k = 768$. They present an at the time SotA result for BTCV, with an average Dice score of 89.10.

2.9.4 Swin-UNETR

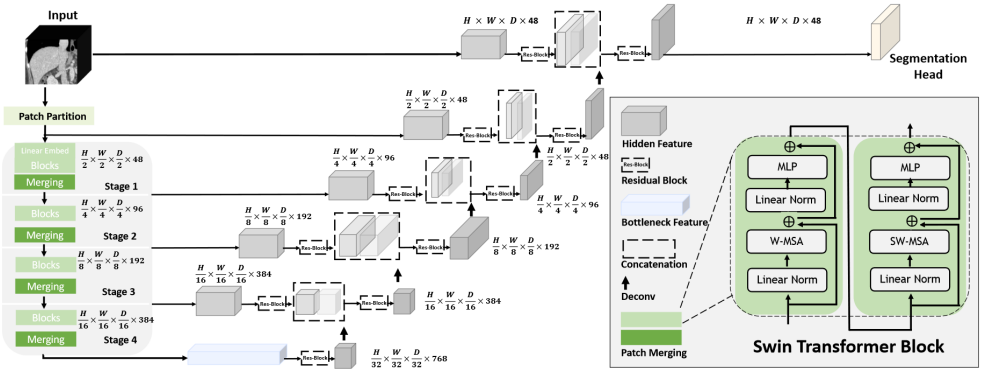
Tang *et al.* [11] introduced Swin-UNETR in November 2021, building upon their previous UNETR model [71]. They utilise a Swin Transformer Encoder, coupled with a CNN based decoder. In contrast to UNETR, they utilise a patch merging strategy to reduce the spatial dimension of the input at each stage of the model. Notably, they introduce a novel SSL framework for their encoder that includes both contrastitive learning, masked volume in-painting, and 3D rotation prediction SSL heads. It is worth noting that UNETR has 92.58 M parameters compared to 61.92 M in Swin-UNETR, highlighting significant savings in terms of model size between the two models.

Their pre-training approach works by extracting a sub-volume from a larger CT scan. This sub-volume is then subjected to stochastic masking and z-rotation augmentation, creating two views of the sub-volume. These views are then fed to the encoder. The encoded representations are then fed to each of the SSL heads and produce a loss for backpropagation to the encoder. The contrastitive head maps the representations to a latent representation. Contrastitive loss is then calculated using cosine similarity between the representations. The reconstruction head uses a single transpose convolutional layer to reconstruct the input view from the encoder representation. The loss is calculated using L1 loss. Finally, the input rotation is predicted using an MLP across 4 classes: 0°, 90°, 180°, and 270°. Cross-entropy loss is used for the rotation task. The model is trained jointly across the sum of the losses. An overview of the SSL framework can be seen in Figure 2.26a, with the whole architecture in Figure 2.26b.

They pre-train their encoder using 5,050 CT scans drawn from The Cancer Imaging Archive, and go on to set new State of the Art results for the BTCV datasets and the CT tasks in MSD. Specifically, they get a 90.8 DSC score for BTCV and similarly strong results in the CT MSD tasks.



(a) Swin UNETR SSL framework.



(b) Full Swin-UNETR architecture.

Figure 2.26: Overview of Swin-UNETR, figures from [11]

2.9.5 nnFormer

This model, introduced by Zhou *et al.* [72] in September 2021, introduces an interesting three stage all-attention UNet model that shows strong performance in several MIC datasets, despite a relatively lean and elegantly simple architecture. They interleave convolutional blocks for downsampling and Transformer blocks in their encoder, employing a local windowed Self-Attention scheme similar to that of Swin Transformer blocks, only applying global Self-Attention in the bottleneck where the spatial resolution is the smallest. They also incorporate a skip attention module instead of regular skip connections to incorporate information from the encoder into the decoder, extracting keys and values from the same stage encoder output and taking queries from the previous up-sampling layer.

They train the model using deep supervision over the output of every decoder stage, with greater emphasis given to the loss output at the higher levels of the encoder. They report an average Dice score of 86.57 on the BTCV dataset, an average Dice score of 86.4 and a 95th percentile average Hausdorff Distance of 4.05 mm in the MSD Brain Tumour task.

2.9.6 Model Genesis

Model Genesis, as introduced by Zhou *et al.* [73] in April 2020, is set of generic models for 3D medical image segmentation trained using SSL. These models, seven in total, are pretrained across a vide variety of different data and modalities, ranging from 2D X-Ray images to 3D MRI scans. Three of those models are trained using SSL over 2D and 3D CT chest data, using input reconstruction of a transformed volume as a proxy task. They demonstrate rapid convergence and strong performance of their models across a wide variety of datasets.

2.9.7 Trans VW

Transferable Visual Words (Trans VW), introduced by Haghighi *et al.* [74] in February 2021, is an approach to SSL for MIC where the model learns generalisable representations of recurring anatomical patterns across samples, the so-called Visual Words. In essence, visual words are essentially patches of the original input volume. They generate a data set of grouped similar visual words from unlabelled medical data using an unsupervised clustering strategy. An encoder is connected to a classification head and a decoder. The encoder is then trained in a self-supervised manner by taking a visual word perturbed by cutout and masking as input, feeding the encoded representation to the classification head which tries to classify which group of visual word the input came from, and also feeding the representation to the decoder which tries to reconstruct the original unperturbed input. They demonstrate

strong performance across a variety of MRI, CT, and X-Ray datasets, surpassing Model Genesis in terms of convergence speed and performance in several of them.

2.9.8 UNetFormer

In April 2022, concurrently with with thesis, Hatamizadeh *et al.* [75] introduced UNetFormer: A Unified Vision Transformer Model and Pre-Training Framework for 3D Medical Image Segmentation. Although this paper was published too late to have significant effect upon this project, it is included in this discussion to highlight concurrent and related work. They introduce two variants of their model, UNetFormer and UNetFormer+. Common to both is a 5 stage Swin-based encoder with patch merging layers. They pretrain this encoder using a masked in-painting scheme across the masked tokens and a skip-connected auto-encoder. Their decoder varies depending on variant. The standard variant uses a 6 stage conventional Convolution-based decoder, whilst the plus-variant uses a 6 stage Transformer-based decoder, both using trilinear interpolation to upscale the tokens at each stage.. They train their encoders using deep supervision, generating segmentation maps from the three last stages of their decoder. In MSD Task 3 Liver, their plus-variant attains a score of 95.06 DSC for the liver and 51.23 for the tumours, with 24.44 parameters using 39.63 GFLOPs. When training from scratch on BraTS, their plus-variant attains a average Dice score of 91.20.

2.9.9 Other models

In this section, strong and modern models in the AD and MIC field are listed. These models are distinguished from the other models as they did not directly influence the methods developed for this project and thus are not quite deserving of the title "related work". They are included nonetheless to provide context for the AD and MIC fields at large and to serve as baselines in our experiments.

nnUNet

nnUNet, introduced by Isensee *et al.* [76] in September 2018, is a strong convolutional UNet that has up until quite recently been one of the most powerful general 3D medical image segmentation models. They propose an automatic adaptive approach that adapts the model architecture to the specific task at hand, with a cascaded two stage approach being used with certain applicable datasets. They show strong performance across several tasks in the MSD dataset, and is commonly used as a standardised baseline.

SeMask

In December 2021 Jain *et al.* [77] introduced SeMask: Semantically Masked Transformers for Semantic Segmentation. They propose a incorporating specific semantic layers and decoder during training to enhance semantic information in the features generated by the feature decoder. The output of these semantic decoders is used to provide deep supervision at every stage. These semantic layers utilise a specialized form of attention classed *semantic attention*, whilst the regular encoder layers are Swin-based. They show strong performance on both ADE20K with 57.00 mIoU and CityScapes val with 83.97 mIoU, holding 3rd place for the latter.

VOLO

VOLO, also known as the "Vision Outlooker", is a Transformer-based generic CV model introduced by Yuan *et al.* [78] in June 2021 using a modified attention mechanism called "Outlook attention" to better finer level features and contexts into tokens before global dependency modelling using Transformers. Their largest model, VOLO-D5, shows strong performance across several downstream tasks, including semantic segmentation on CityScapes val with a 84.3 mIoU score and ADE20K with a 54.3 mIoU score, holding 4th place for the former.

Segformer

In May 2021, Xie *et al.* [79] introduced the SegFormer (not to be confused with the Segmentation Transformer), a simple, efficient, and powerful semantic segmentation framework with Transformer-based encoders and straightforward MLP decoders. Using an efficient formulation of self-attention and hierarchical features together with their all-MLP decoder, they show remarkable performance and efficiency on the semantic segmentation datasets ADE20K with a 51.8 mIoU score and CityScapes val with a 84.0 mIoU score, holding 5th place for the latter.

Chapter 3

Methodology

This chapter describes the methodology of our work. Section 3.1 describes the software and hardware used for the development of the project and for running our experiments. Section 3.2 describes the different datasets made available to the experiments. Section 3.3 introduces the QT-UNet in detail, while Section 3.4 briefly describes the models against which we compare it. Section 3.5 describes the experiments carried out in the project, before Section 3.6 describes the metrics by which we evaluate the models.

3.1 Software and hardware

Datasets, models, and experiments were set up using Anaconda [80] with Python 3.9.11, using PyTorch 1.11.0 [81], PyTorch Lightning 1.6.0 [82], PyTorch Lightning Bolts 0.5.0 [83], and MONAI 0.8.1 [84]. PyTorch was selected due to its wide support in the research community and the availability of its TorchVision library, which contains both wrappers for several standard CV datasets such as CityScapes [31]. It was decided to extend PyTorch with PyTorch Lightning, as that library automates much of the manual engineering work associated with writing PyTorch code. For example, setting up multi-GPU training with Lightning is relatively easy, requiring only a small change in configuration, whereas the same feat in plain PyTorch would require significant engineering effort. PyTorch Lightning Bolts was also used, as it further extends some of the datasets available in TorchVision to work with Lightning and includes standard implementations of several SSL techniques. MONAI was also used as it significantly simplifies working with 3D data volumes and provides several utilities for working with medical data.

Most of the development, debugging, and data processing for this project was carried out using a virtual machine made available by the IDI Horizon visual computing group. Some trial training was also performed on the VM, though the runs reported in Chapter 4 were run on IDUN. IDUN is a state-of-the-art compute cluster maintained by the High-Performance Computing

Machine	Device	Specs
IDUN	GPU	Up to 2× NVIDIA Tesla V100 32/16GB, or P100 16GB, or A100 80/40GB
	CPU	Intel Xenon Gold 6148, 20 cores at 2.4GHz
	RAM	768GB
Horizon VM	GPU	NVIDIA A10-24Q, 24GB
	CPU	Intel Xenon Gold 6342, 8 cores at 2.8GHz
	RAM	87GB

Table 3.1: Hardware Specifications.

Group at NTNU [85]. The cluster is made up of various nodes with different hardware, using the SLURM resource manager [86] to distribute jobs to the nodes. For this project, nodes with NVIDIA Tesla A100 GPUs were used. Specifically, we used one A100 40GB card for all our standard runs, increasing to two A100 80GB cards for our SSL pre-training runs. A summary of the available hardware can be found in Table 3.1.

We describe, for the benefit of future practitioners, some particulars of our working setup with the VM and IDUN that we found productive and effective. First and foremost, we used a Git repository to sync code between the VM and IDUN cluster. Using a Git repository provided simple synchronisation between the machines and flexibility to experiment safely in branches. We also used a Anaconda environment specified in a environment yaml file, which allowed us to keep the Python environments in sync easily. Furthermore, we opted to use VSCode over SSH with both machines for development and inspection, which provided a single consolidated development environment across both machines without having to log into either machine with a remote desktop setup. Finally, we made heavy use of SLURM array jobs in order to orchestrate experiment runs across many models and model versions at the same time.

3.2 Datasets

The datasets selected and the reasoning behind their selection are described in this section. Any pre-processing of the data is also explained.

3.2.1 MIC datasets

BraTS2021

BraTS2021 is selected for our experiments because it reflects a real-world scenario and has diversity in its MRI scans, as they are acquired at different institutions with different equipment and protocols. The dataset contains 1251 MRI scans of shape $240 \times 240 \times 150$. Following Peiris *et al.* [10], the scans are

divided into sets of 834, 208, and 209 for training, validation, and testing, respectively.

Each scan in the dataset is interpolated to a isotropic voxel spacing of $[1.0 \times 1.0 \times 1.0]mm$. The foreground in the scan is cropped, before intensities are normalised in a non-zero, channel-wise fashion. During validation and testing, the augmentation ends here. For training, we use randomly selected sub-volumes of size $128 \times 128 \times 128$ voxels.

The raw labels in the dataset consist of the Enhancing Tumour (ET), Non-Enhancing Tumour (NET), Necrotic Tumour (NCR), and Peritumoral Edema (ED). Following standard pre-processing for the BraTS dataset, the ET label is used directly. NET, NCR, and ET are combined to produce the label Tumour Core (TC), and ED is combined with TC to produce the label Whole Tumour (WT). This leaves us with three meaningful labels for training and evaluation.

BTCV

BTCV is used for our experiments, as it poses an interesting challenge of 13 organ segmentation targets with few training samples available. Each CT scan consists of between 85 and 198 slices, with resolution 512×512 pixels. Of the 50 scans, 40 are available with labels. Of these 40 labelled samples, 35 are used for training and the rest are used for validation and testing.

The pre-processing pipeline for BTCV follows the pipeline used for Swin-UNETR [11], since our model shares architectural similarities with theirs. Each scan is interpolated to a voxel spacing of $[1.5 \times 1.5 \times 2.0]mm$. CT intensity is clipped between -175 and 250, before being normalised. The foreground is then cropped whilst labels are one-hot encoded. The validation pipeline stops here. For training, $96 \times 96 \times 96$ voxel sub-volumes are extracted, and padded if smaller than the requested size. Random flips are applied in each dimension with a probability of 0.1 Random 90 degree rotation is also applied with probability 0.1, before a random intensity shift is applied with offset 0.1 and probability 0.5.

MSD

The MSD dataset is included due to its diversity of modalities and challenges. This diversity allows us to comprehensively benchmark our method and evaluate its generalisability for medical image segmentation.

As with BTCV, we adopt the pre-processing pipelines used in Swin-UNETR [11] due to the similarities in architecture. The pipelines for each task are set up as follows.

Task 1 Brain Tumour The pipeline for these MRI images is identical to that described in Section 3.2.1, with the addition of random flips in each dimension

with probability 0.5, random intensity scaling with factor 0.1 and probability 0.1, and random intensity shifts with offset 0.1 and probability 0.1.

Task 2 Heart The MRI images are interpolated to an isotropic voxel spacing of 1.0 mm and cropped to remove the foreground, before channel-wise non-zero normalisation is applied. Training sub-volumes are sampled at a resolution of $96 \times 96 \times 96$ voxels, with a positive to negative class ratio of 2:1. Random flip is then applied with probability 0.5, random 90°rotation with probability 0.1, intensity scaling with factor 0.1 and prob 0.2, and intensity shift with offset 0.1 and prob 0.5.

Task 3 Liver The CT scans are interpolated to a isotropic voxel spacing of 1.0 mm and cropped to remove the foreground. The intensities are scaled to $[-21, 189]$, and then normalised. Training sub-volumes are sampled at a resolution of $96 \times 96 \times 96$ voxels, with a positive to negative class ratio of 1: 1. Random flip is applied with probability 0.2, random 90°rotation with probability 0.2, intensity scaling with factor 0.1 and prob 0.1, and intensity shift with offset 0.1 and prob 0.1.

Task 4 Hippocampus Each MRI image is interpolated to a voxel spacing of $0.2 \times 0.2 \times 0.2$ and cropped to remove the foreground. Training samples are extracted at a resolution of $96 \times 96 \times 96$ voxels. Channel-wise non-zero normalisation is applied, before random flip, rotation, intensity scaling, and shifting are applied with probability 0.1. For scaling and shifting, the factor and offset are both 0.1.

Task 5 Prostate With both channels of the MRI scan, the scans are interpolated to a voxel spacing of $0.5mm$ and cropped to remove the foreground. Training samples are extracted at a resolution of $96 \times 96 \times 96$ voxels. Channel-wise non-zero normalisation is applied, before random flip, intensity scaling, and shifting is applied with probability 0.5. For scaling and shifting, we set the factor and offset to 0.1. Additionally, a random affine transformation is applied with a scale factor of $[0.3, 0.3, 0.0]$ with a rotation range of $[0, 0, \pi]$ on each axis.

Task 6 Lung Each CT scan is interpolated at an isotropic spacing of $1.0mm$. The intensities are clipped to $[-1000, 1000]$, and then normalised. Training samples are extracted at a resolution of $96 \times 96 \times 96$ voxels, with a ratio of positive to negative classes of 2:1. Random flip is applied with probability 0.5, random rotation with probability 0.3, intensity scaling with factor 0.1 and prob 0.1, and intensity shift with offset 0.1 and prob 0.1.

Task 7 Pancreas CT scan intensities are clipped to $[-87, 199]$. Training samples are extracted at a resolution of $96 \times 96 \times 96$ voxels, with a positive-to-negative class ratio of 1:1. Random flip is applied with probability 0.5, random rotation with probability 0.25, and intensity scaling with factor 0.1 and prob 0.5.

Task 8 Hepatic Vessel Each CT scan has its intensities clipped to $[0, 230]$. Besides this change, the same augmentations as for Task 7 are used.

Task 9 Spleen The CT scans are interpolated to an isotropic voxel spacing of $1.0mm$. Intensities are clipped to $[-125, 275]$. Training samples are extracted at a resolution of $96 \times 96 \times 96$ voxels. Random flip is applied with probability 0.1.5, intensity scaling with factor 0.1 and prob 0.1, and intensity shift with offset 0.1 and prob 0.1.

Task 10 Colon The intensities of each CT scan are clipped to $[-57175]$ and then normalised. Training samples are extracted at a resolution of $96 \times 96 \times 96$ voxels, with a positive-to-negative class ratio of 1:1. Random flip is applied with probability 0.5, random rotation with probability 0.25, and intensity scaling with factor of 0.1 and probability of 0.5.

3.2.2 AD datasets

CityScapes

We opted to use CityScapes, described in Section 2.2, for this project for semantic segmentation, as it is one of the most widely used and cited semantic segmentation datasets in the AD community. There was also a wrapper available in PyTorch Lightning Bolts [83], which tied in nicely with PyTorch Lightning. It also corresponds well to RQ 3, to test our methods in 2D.

Augmentations include normalisation of the images using the mean and standard deviation of CityScapes. For training, we extract 1024×1024 pixel images from the original 2048×1024 pixel images to facilitate their use in our SSL pipeline. The classes to be ignored, as described by [31], are set as background, leaving us with 20 total classes. A table of the mapping can be found in the Appendix, in Table B.1.

CityScapesCat

In order to investigate the effect of the number of classes on QT-UNet, we create a variant dataset of CityScapes by mapping the class IDs to category IDs using the mapping described in Table B.1. This dataset, CityScapes over Categories, is dubbed CityScapesCat. We use the same augmentations as for the standard CityScapes dataset.

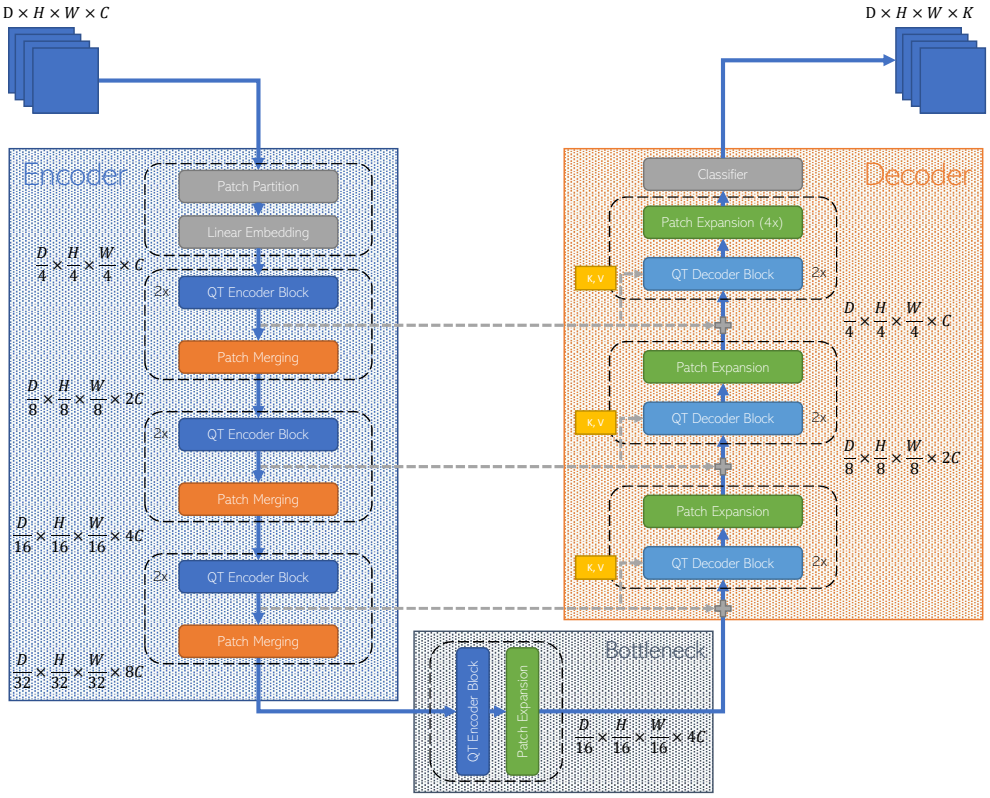


Figure 3.1: The proposed QT-UNet architecture.

NTNU Data

We utilise a small, 10 sample dataset from Trondheim, Norway provided by the NTNU Autonomous Perception Laboratory (NAPLab) to test the zero-shot transfer performance of QT-UNet-2D. The dataset is annotated with classes corresponding to the CityScapes class definition.

For this data set, we used the same augmentations as with CityScapes.

3.3 QT-UNet

Our model is inspired by VT-UNet [10] and Swin-UNETR [11], with our architecture drawing heavily upon the former and our training procedure being based upon the latter. An overview of the model and components can be seen in Figure 3.1. The model is named Querying-Transformer UNet, or QT-UNet for short. It exists in two variants: One for 3D inputs and one for 2D inputs. The 3D variant, simply named QT-UNet, takes as input a 3D volume of size $D \times H \times W \times C$ and produces as output a volume of size $D \times H \times W \times K$, where K is the number of classes. The 2D version is identical to the 3D version, but drops the depth dimension (D) in all components. The 2D version is denoted

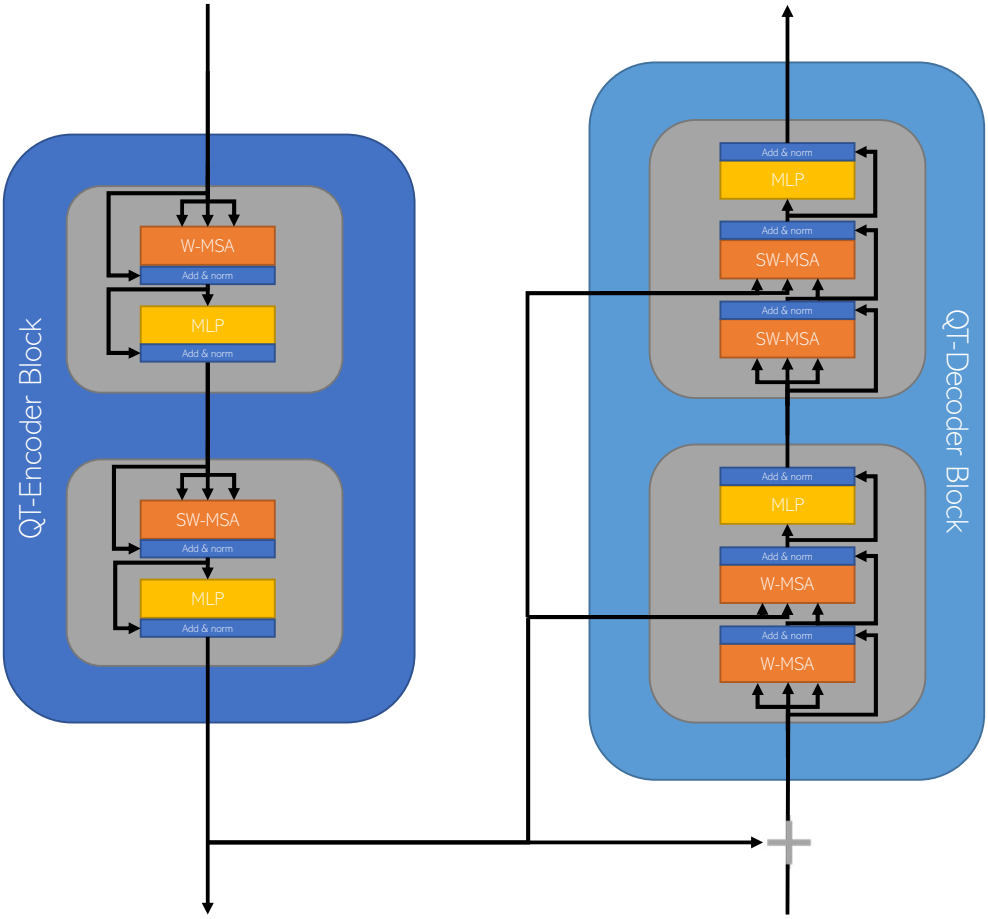


Figure 3.2: Encoder-Decoder interaction.

as QT-UNet-2D.

The model consists of the following components:

1. Patch partitioning
2. QT-UNet encoder
    - QT encoder block
    - Patch merging
3. Bottleneck
    - QT encoder block
    - Patch expansion
4. QT-UNet decoder
    - QT decoder block
    - Patch expansion
5. Classifier

This section will discuss each component in turn.

3.3.1 Patch partitioning

As with other Vision Transformers, we transform the model input into a sequence of tokens using a convolutional layer. This layer partitions the input into non-overlapping patches using a partitioning kernel. For our 3D variant, this kernel has size $M \times M \times M$, resulting in a sequence of tokens corresponding to a volume of $\lfloor \frac{D}{M} \rfloor \times \lfloor \frac{H}{M} \rfloor \times \lfloor \frac{W}{M} \rfloor$. The convolutional layer is equipped with C such kernels as to embed each patch in a C-dimensional vector. A value of 4 is set for M, whilst the value of C varies depending on the model variant used (see Section 3.3.9).

3.3.2 QT-UNet Encoder

The QT-UNet encoder consists of successive QT encoder blocks and patch merging layers. Each stage in the encoder consists of two QT Encoder blocks, followed by a patch merging layer.

3.3.3 QT Encoder Block

The design of the QT Encoder Block draws upon the design of the Video Swin Encoder blocks [87] and VT Encoder Blocks [10]. Each block consists of two sub-blocks with a 3D window-based MHSA (W-MHSA) module followed by a two-layer MLP with a GELU activation function. Layer normalisation is applied before and after the self-attention module, with skip connections between the self-attention module and the MLP. For the second sub-block in each block, the window partitioning operation is shifted two voxels in each direction to

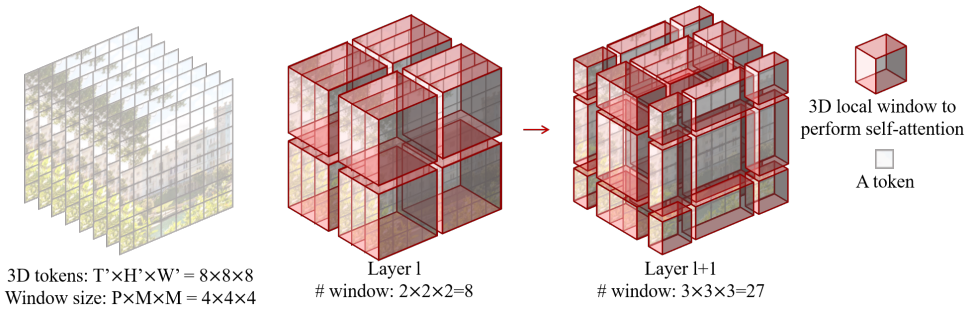


Figure 3.3: Illustration of 3D windowed self attention, from [87].

introduce cross-window connections between the blocks. Self-attention with this shifting operation is known as Shifted Window Multi-Head Self-Attention (SW-MHSA). Additionally, a 3D relative bias $B \in \mathbf{R}^{M^2 \times M^2 \times M^2}$ is added to each self-attention head.

Self-attention is applied to each window as in Equation (3.1), where $K, Q, V \in \mathbf{R}^{M^3 \times d}$ are the key, query, and value matrices, d is the dimension of the key and value features, and M^3 is the number of tokens in each window.

$$Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = Softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\intercal}}{\sqrt{d_q}} + \boldsymbol{B}\right)\boldsymbol{V} \tag{3.1}$$

Since the relative position along each axis lies in the range of [-M + 1, M - 1], we parameterise a smaller bias matrix $\hat{B} \in \mathbf{R}^{(2M-1)\times(2M-1)\times(2M-1)}$, taking values for B from $\hat{B}$, as in [8].

The windowing operation can be understood as injecting an inductive bias of locality into the model. The shifting operation allows successive applications of the blocks to receive information across windows, whilst the position bias informs the relative positioning of those windows.

Patch Merging

Strong feature hierarchies are an essential feature of many segmentation models [8–11, 40, 54] in order to predict dense outputs. After each QT Encoder Block, adjacent $2 \times 2 \times 2$ groups of tokens are concatenated along their feature dimension, producing a vector with $8C$-dimensional features with spatial dimensions $D/2 \times H/2 \times W/2$. A linear layer is used to project the concatenated features to a fourth of their expanded dimension. That is, the $8C$-dimensional features of each token is reduced to $2C$ dimensions, producing final tokens of size $D/2 \times H/2 \times W/2 \times 2C$.

It should be noted that our patch merging application is slightly different from the one used for the Video Swin Transformer [87] and VT-UNet[10]. As noted, the patch merging layers in QT-UNet merge adjacent tokens in all three

spatial axes. This is in contrast to the Video Swin Transformer and VT-UNet, which perform patch merging only in the height and width axes.

3.3.4 Bottleneck

The deepest stage of QT-UNet is a bottleneck layer. This layer consists of a single QT Encoder Block, followed by a patch expansion layer.

3.3.5 QT-UNet Decoder

The QT-UNet Decoder consists of successive pairs of QT Decoder blocks, patch expansion layers, and ends with a classifier.

Patch Expansion

In essence, the Patch Expansion layers work to undo the operation of the Patch Merging layers. That is, their function is to increase the spatial resolution of the tokens whilst reducing their feature dimension.

This is achieved through a two-stage process. First, a linear layer is applied to increase the feature dimension fourfold (i.e. $2C \rightarrow 8C$). Then, $2\times2\times2$ tokens with feature dimension C are extracted from the expanded token. That is, given an expanded volume $D/2 \times H/2 \times W/2 \times 8C$, we reshape the volume along the spatial axes by reducing the embedding dimension, producing a volume of tokens of size $D \times H \times W \times C$.

3.3.6 QT Decoder Block

UNet architectures typically use skip connections between stages in the encoder and decoder pipelines to produce higher detail predictions by forwarding spatial information from earlier stages in the network to later stages that have strong semantic information but weaker spatial information. These skip connections merge these spatially and semantically dense representations together, allowing us to enjoy the best of both worlds.

Inspired by this, the authors of VT-UNet [10] introduced a novel Cross-Attention mechanism that, in addition to the already well-established skip connections, fed the keys and values from the same-stage encoder to the decoder, adding another path between the pipelines. This effectively allows the decoder to query for spatial information using the spatially strong keys and values from the encoder. They employ this Cross-Attention mechanism in a two-pipeline fusion module (see Figure 2.25b), where each pipeline consists of one block each. One block receives keys and values from the encoder, whilst the other receives it from the previous block. The output is combined using a linear combination of the outputs of the two pipelines.

The QT Decoder Block iterates upon this approach by introducing two major changes. First, we allow the decoder block to generate its own keys and values from the output of the same-stage encoder, rather than directly forwarding them. This allows the model to more flexibly query the spatially dense encoder output by generating its own queries and values rather than being bound by the generation of these in the encoder. This also saves some memory usage at the cost of more parameters, since the keys and queries from the encoder stages need not be stored.

Secondly, we remove the fusion module and instead structure the block more in accordance with a standard Transformer decoder. That is, we first employ standard Windowed Self-Attention with keys, queries, and values derived from the input as normal, before applying Windowed Cross-Attention where the keys and values are generated from the output of the same-stage encoder, and queries are generated from the output of the previous Cross-Attention block. This mirrors the design of the original Transformer Decoders due to Vaswani *et al.* [1].

The general intuition of this approach is that we allow the decoder to flexibly query the output of the encoder. In our setup, the decoder can essentially decide for itself what is and is not pertinent information in the spatially dense encoder output, whilst still basing the queries upon the semantically dense decoder output. In essence, the model is querying itself, hence the name "Querying Transformer UNet".

An illustration of the block can be seen in Figure 3.2. As with the encoder blocks, there are several skip connections across the modules in the block, and a final MLP at the top. As in the encoder blocks, this MLP is a two-layer module with a GELU activation function. Again, similarly to the encoder block, windows are shifted 2 voxels in each axis for each pair of sub-blocks to produce shifted window self-attention. A relative spatial bias is also applied in the same manner as in the encoder.

3.3.7 Classifier

After the final Patch Expansion layer in the decoder, the model is topped with a convolutional classification head, mapping the C dimensional features to K segmentation classes. The final output of the model is then $D \times H \times W \times K$.

3.3.8 Common parameters

For the sub-blocks used in both the encoder and decoder, a hand-full of common parameters are set. Firstly, a window size of $7 \times 7 \times 7$ is used for the window partitioning for both W-MHSA and SW-MHSA. Secondly, an expansion ratio of 4.0 is used for the hidden layers in the MLP. Finally, the number of heads in each module increases for each stage down into the encoder and

decreases for each stage moving upward in the decoder, following the pattern given in Equation (3.2).

Additionally, the encoder modules in the Base variant of QT-UNet are pre-loaded with Swin Transformer weights pre-trained on ImageNet, with the other variants initialised randomly, following Peiris *et al.* [10].

$$3 \to 6 \to 12 \to 24^{1} \to 12 \to 6 \to 3 \tag{3.2}$$

3.3.9 Variants

Several variants of QT-UNet can be introduced by varying the parameters that control its behaviour. We introduce three variants by adjusting the number of embedding dimensions C in the Patch Embedding layer, following VT-UNet [10]. Applying the same naming convention as VT-UNet, these variants are as follows:

1. Tiny: **QT-UNet-T**, $C = 48$
2. Small: **QT-UNet-S**, $C = 72$
3. Base: **QT-UNet-B**, $C = 96$

For all models, we employ three stages of encoding and decoding, plus the bottleneck. The patch embedding uses a patch size of $M = 4$ for all experiments.

QT-UNet-2D

The model described so far is the 3D variant of QT-UNet. To test the applicability of the techniques used in 2D CV domains, we also spin out a 2D version of the model as can be seen in Figure 3.4. Mostly identical to the standard QT-UNet, the 2D version drops the depth dimension in all components. Additionally, the Patch Merging and Expansion layers work slightly differently. Since these layers only work on 2×2 spatial neighbourhoods in the 2D variant, they produce and require a lower feature dimension before reduction and expansion. That is, in the Merging layers, we get a feature dimension size sequence of: $C \to 4C \to 2C$, rather than $C \to 8C \to 2C$. In the Expansion layers, we have $2C \to 4C \to C$ rather than $2C \to 8C \to C$. Consequently, the linear patch expansion and contraction layers are milder in the 2D version than in the 3D version, by a factor of 2, due to the smaller spatial neighbourhood.

3.3.10 Training QT-UNet

QT-UNet is trained by minimising Dice Loss, as described in Section 2.4.5. Note that this is in contrast to VT-UNet, which jointly minimises both Dice Loss and CE loss. We found that our model performed better under Dice loss rather

[1] In the bottleneck stage.

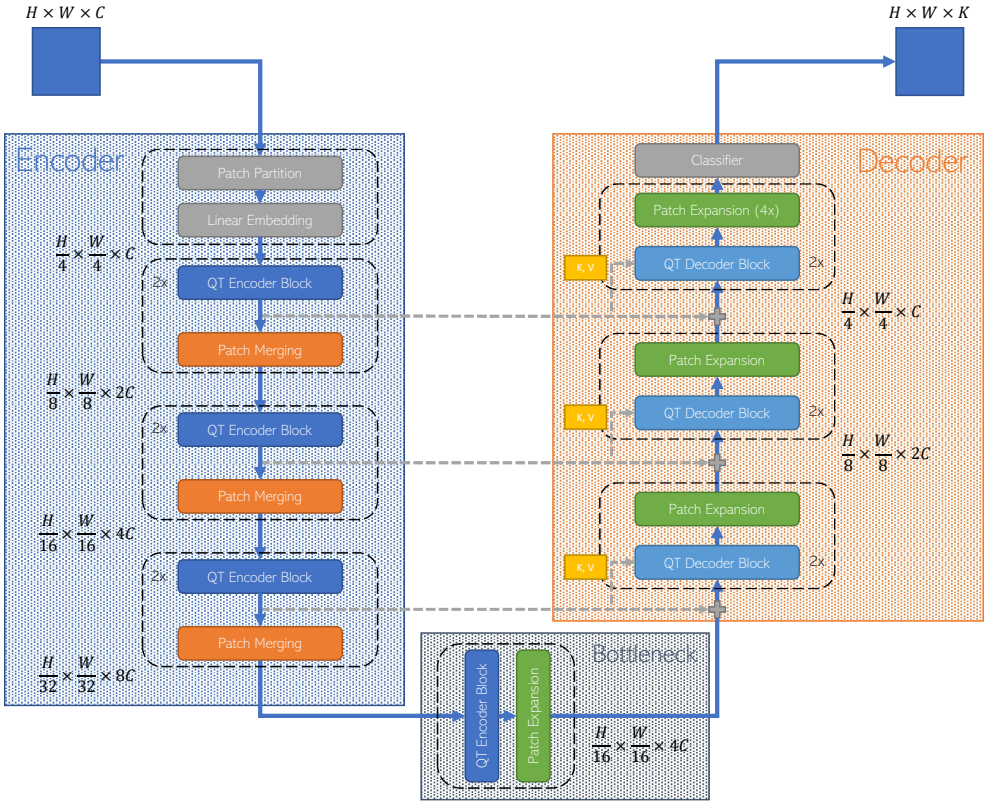


Figure 3.4: Architecture of QT-UNet-2D.

than joint loss, especially in tasks with overlapping target meshes[2], and thus decided to use it instead of joint loss.

3.3.11 Inference with QT-UNet

In both MIC and AD tasks, the input sizes are often larger than what the model can handle. This is also the case for QT-UNet and other models in this project. There are several approaches to handle this problem, but an approach of specific interest to this project is *sliding window inference*. With sliding window inference, the model is simply sled across the larger input to produce prediction windows that collectively represent a segmentation of the whole input. Often, an overlap parameter is set to specify how much each prediction window should overlap. These overlapping predictions are then combined by averaging the predictions. Two common methods of averaging are constant averaging[3] where each prediction window is given equal weight, and Gaussian averaging[4] where predictions in the edges of windows are given less weight than those at the centre.

In this project sliding window inference in constant mode with an overlap of 0.5 is used during validation and testing unless otherwise noted.

3.3.12 SSL in QT-UNet

We employ Self-Supervised Learning (SSL) on the encoder, training it to produce strong semantic representations of the input before fine-tuning. Our approach builds upon the one favoured by Tang *et al.* [11] for Swin-UNETR, with a handful of improvements.

Similarly to Swin-UNETR, we pre-train QT-UNet using a augmented multi-view multi-head approach. First, a sub-volume $x \in \mathbf{R}^{d \times h \times w \times C}$, where d, h, and w are the spatial dimensions of the volume, is extracted for the larger input volume $X \in \mathbf{R}^{D \times H \times W \times C}$. From this sub-volume x, two augmented views of the data are generated with two independent applications of an augmentation pipeline consisting a random sub-volume masking and random 90° rotation along the z-axis. These augmented views are then fed to the encoder, which outputs its representation of them. These representations are then fed to each of the three task heads:

Reconstruction head This head consists of a single transposed convolution layer that takes as input the view representation and attempts to reconstruct the un-augmented sub-volume x. We denote its reconstruction $\hat{x}$, and use L1 loss between it and x as the objective.

[2] Such as those for BraTS, where the label "tumour core" and "enhancing tumour" are both wholly contained in the label "whole tumour".

[3] Referred to as "constant mode".

[4] Referred to as "Gaussian mode".

Image rotation head This head consists of a standard one layer MLP with Batch Norm and a ReLU activation function. It attempts to predict how much the augmented volume was rotated in one of four classes: 0°, 90°, 180°, or 270°. This task is optimised using a soft-maxed cross-entropy loss between the true rotation k and the prediction $\hat{k}$.

Bootstrap Your Own Latent (BYOL) head Noting that the contrastitive SimCLR-based approach favoured by Swin-UNETR requires prohibitively large batch sizes to be properly effective [66], we opt instead to use BYOL as described in Section 2.8.3 due to its stronger performance with smaller batch sizes.

We base our implementation of BYOL on the one provided by PyTorch Lightning Bolts [82], modifying it to fit our augmentation scheme. Loss is calculated using cosine similarity between the outputs of the augmented views from the online and target branches of BYOL.

Joint Loss

Formally, the encoder is optimised over the joint loss of all head losses, with equal weight given to each following Tang *et al.* [11].

Modes of operation for QT-UNet SSL

We use this SSL setup in two different modes, depending on the task, the modality, and the domain for which we train. Some modality and domains have large, readily available corpuses of unlabelled data that can be used for SSL. In these domains, we collect a large out-of-task dataset with relevant unlabelled data and use this dataset for pre-training. We refer to this as out-of-task pre-training. In other domains, where data access is more scarce, we utilise the SSL setup on the task data directly in order to extract more learning from the limited corpus, referring to it as in-task pre-training.

CT-SSL dataset

Using The Cancer Imaging Archive (TCIA) [88], we have been able to compose a large dataset of CT scans of the abdomen, pelvis, and chest by composing data available in several of the datasets available in the archive. An overview of the datasets used can be found in Table 3.2. The dataset, which we name CT-SSL, consists of 3 597 CT scans, downloaded through the TCIA API and converted from DICOM to Nifti format. Note that some scans were discarded during conversion due to errors or inconsistencies in the data. 100 scans are held out as a validation set during training.

Dataset	Region	#of scans	Source
CT Lymph Nodes [89]	Abdomen/Lungs	175	wiki.cancerimagingarchive.net/display/Public/CT+Lymph+Nodes
CT Colonography [90]	Abdomen/Pelvis	1706	wiki.cancerimagingarchive.net/display/Public/CT+COLONOGRAPHY
COVID-19-AR [91]	Lungs	149	wiki.cancerimagingarchive.net/pages/viewpage.action?pageId=70226443
MIDRC-RICORD-1A [92]	Lungs	121	wiki.cancerimagingarchive.net/pages/viewpage.action?pageId=80969742
MIDRC-RICORD-1B [93]	Lungs	90	wiki.cancerimagingarchive.net/pages/viewpage.action?pageId=80969771
Pelvic Reference Data [94]	Pelvis	116	wiki.cancerimagingarchive.net/display/Public/Pelvic+Reference+Data
Stage II Colorectal CT [95]	Abdomen/Pelvis	230	wiki.cancerimagingarchive.net/pages/viewpage.action?pageId=117113567
LiDC [96]	Chest	1 010	wiki.cancerimagingarchive.net/display/Public/LIDC-IDRI

Table 3.2: Overview of datasets used for pre-training.

Parameter	Value
Learning Rate	0.4×10^{-4}
Weight decay	1.5×10^{-6}
Optimiser	Adam
Learning rate scheduler	Linear Warm-up Cosine Annealing
Mini-batch Size	Varies with experiment, see Table 3.4
Epochs	Varies with experiment, see Table 3.5
Warm-up epochs	10

Table 3.3: Parameters for SSL runs.

3.4 Comparison to other models

We compare QT-UNet with several other models to evaluate its performance. The models against which we compare are the models listed in Section 2.9 - Related work, including those listed under "Other models". Models are used for comparison where relevant scores are available.

3.5 Experiments

This section describes the experimental setup for each experiment, describing what models were ran against what data and the model hyperparameters.

3.5.1 Preparatory SSL

Before all experiments could be run, we ran our SSL against several datasets to prepare weights to use in later runs. This section will describe what data we trained on, with hyper-parameters listed in Table 3.3. A full description of our SSL pipeline can be found in Section 3.3.12.

The size of our pre-training datasets and whether they are used for in-task or out-of-task pre-training informs our choice of number of epochs, as can be seen in Table 3.4. In experiments with large out-of-task datasets, such as the CT-SSL dataset and CityScapes Coarse, we opt for a smaller number of epochs due to the sheer size of the datasets and time constraints. For leaner, in-task

Dataset	Batch size	Gradient accumulation	Num. GPUs	Effective batch size
CT-SSL	32	2	2	128
BraTS2021	8	4	2	64
MSD Task 2/4/5	32	2	2	128
CityScapes Coarse	16	2	2	64

Table 3.4: Batch sizes used for each experiment in SSL.

Dataset	Num epochs
CT-SSL	150
BraTS2021	350
MSD Task 2/4/5	350
CityScapes Coarse	150

Table 3.5: Epochs used for each experiment in SSL.

datasets, we opt for a larger number of epochs to extract as much learning as possible from the data.

Furthermore, batch sizes were tuned to fit as many samples as possible on the GPUs to obtain a sufficient batch size for BYOL to be effective, also using gradient accumulation to obtain an even larger batch size. Note that the effective batch size is computed as: $nGPU \cdot$ batch size $\cdot$ gradient accumulation.

CT dataset

Using the dataset described in Section 3.3.12, we pre-train all variants of the QT-UNet to initialise weights for downstream CT-based tasks.

In the augmentation pipeline for this dataset, we first interpolate all scans to an isotropic voxel spacing of $[1.0 \times 1.0 \times 1.0]mm$. We then crop out foreground and normalise the scans, before passing a random $96 \times 96 \times 96$ crop of the volume to the SSL pipeline.

BraTS 2021

Restricted by the limited availability of relevant data and by time, we perform in-task pre-training for BraTS for all variants of QT-UNet. Augmentations before the SSL pipeline are identical to those used for regular training.

MSD

Due to the limited availability of relevant out-of-task data for the MRI tasks in MSD (Tasks 2, 4, and 5), we perform in-task pre-training for each of these tasks with all variants of QT-UNet. As for augmentations as they are described in

Parameter	Value
Learning Rate	0.4×10^{-4}
Weight decay	0
Drop path rate	0.2
Optimiser	Adam
Learning rate scheduler	Cosine Annealing
Mini-batch Size	1
Epochs	350

Table 3.6: Common parameters for MIC experiments.

Section 3.2.1, we drop them all apart from spacing, foreground cropping, clipping, normalisation, and sample extraction. The final sample is then passed to the SSL pipeline.

CityScapes

For CityScapes, we pre-train the 2D variants of QT-UNet using the 20 000 samples large coarse extension for CityScapes (CityScapes Coarse). Labels are discarded, although the remaining preprocessing of the images is identical to that described in Section 3.2.2.

Due to the high dimensionality of the output from the encoder when running QT-UNet-2D against CityScapes, we bilinearly interpolate the output of the encoder to reduce its size by a factor of four before feeding the output to the task head. Without this reduction, the SSL setup does not fit into memory.

3.5.2 Experiment 1: Medical Image Computing

Our Medical Image Computing experiment consists of three subexperiments, as detailed below.

Subexperiment 1.1: BraTS 2021

For this experiment, we train all versions of both QT-UNet and VT-UNet against our own split of the data, with common training hyperparameters as described in Table 3.6. We also train all versions of QT-UNet initialised with weights pre-trained on the BraTS data with our SSL setup with otherwise identical parameters. The exact split of the samples and the augmentations used are described in Section 3.2.1. For validation and testing, we segment whole volumes using sliding window inference in constant mode with an overlap of 0.5. We report Dice score and 95th percentile Hausdorff Distance, in addition to the common metrics described in Section 3.6.

Task	Modality	Dataset used for pretraining
1 Brain Tumour	MRI	BraTS2021
2 Heart	MRI	MSD Task 2
3 Liver	CT	CT-SSL
4 Hippocampus	MRI	MSD Task 4
5 Prostate	MRI	MSD Task 5
6 Lung	CT	CT-SSL
7 Pancreas	CT	CT-SSL
8 Hepatic Vessel	CT	CT-SSL
9 Spleen	CT	CT-SSL
10 Colon	CT	CT-SSL

Table 3.7: Mapping between MSD task and weights used for pre-trained QT-UNet.

Subexperiment 1.2: BTCV

For this experiment, we train all versions of both QT-UNet and VT-UNet against our own split of the data, with common training hyperparameters as described in Table 3.6. We also train all variants of QT-UNet initialised with weights pre-trained on our CT-SSL dataset using our SSL setup, with otherwise identical parameters. The exact split of the samples and the augmentations used are described in Section 3.2.1. For validation and testing, we segment whole volumes using sliding window inference in constant mode with an overlap of 0.5. We report Dice score whilst ignoring the background label, in addition to the common metrics described in Section 3.6.

Subexperiment 1.3: MSD

For this experiment, we train all versions of both QT-UNet and VT-UNet against all 10 tasks, with common training hyperparameters as described in Table 3.6. We additionally train all variants of QT-UNet initialised with relevant weights for each task, an exact mapping is given in Table 3.7. In general, for tasks using MRI data we initialised the pre-trained models with in-task pre-training, whilst we for CT used the weights generated by out-of-task pre-training using our CT SSL dataset.

We use the default MSD data splits for each task as provided by MONAI [84], reporting results on the validation set whilst ignoring background. The augmentations used for each task are described in detail in Section 3.2.1. For validation and testing, we segment whole volumes using sliding window inference in constant mode with an overlap of 0.5. We report Dice score whilst ignoring the background label, in addition to the common metrics described in Section 3.6.

Parameter	Value
Learning Rate	0.4×10^{-4}
Weight decay	0.001
Drop path rate	0.2
Optimiser	Adam
Learning rate scheduler	Cosine Annealing
Mini-batch Size	4
Epochs	150

Table 3.8: Common parameters for AD experiments.

3.5.3 Experiment 2: Autonomous Driving

Our Autonomous Driving experiment consists of three subexperiments, as detailed below.

Subexperiment 2.1: CityScapes

We train all variants of QT-UNet-2D with CityScapes, using training parameters as described in Table 3.8 and data augmentations as in Section 3.2.2. We also train all variants of QT-UNet-2D with weights pre-trained on CityScapes Coarse as described in Section 3.5.1 with the same parameters as in Table 3.8. To better examine the effects of the Cross-Attention module in QT-UNet-2D, we additionally train a variant model QT-UNet-2D-A under the same parameters as the standard QT-UNet-2D, but with the Cross-Attention module disabled. We report results on the validation set, ignoring class 0 as prescribed by 4.4. For validation and testing, we segment whole images using sliding-window inference in constant mode with an overlap of 0.5. We report the average Dice score and the mean IoU (mIoU) across all classes except the ignore class 0, in addition to the common metrics described in Section 3.6. We additionally report inference speed, recorded by taking the average inference time over the testing epoch.

Subexperiment 2.2: CityScapesCat

Observing that CityScapes itself has a high number of classes, and finding during development that QT-UNet struggles with tasks with a high number of classes, we train all variants of QT-UNet-2D on our CityScapesCat variant, mapping the CityScapes training IDs to category IDs. Details of this mapping can be found in the Appendix, in Table B.1. The training parameters in Table 3.8 are used. We report results on the validation set, ignoring the void class 0. To highlight the effects of the Cross-Attention module in QT-UNet-2D, we also train the variant model QT-UNet-2D-A under the same parameters as the standard QT-UNet-2D, with the Cross-Attention module disabled. For validation and testing, we segment whole images using sliding-window inference

Model	Depth-wise	New CA module
VT-UNet	X	X
VT-UNet-A	✓	X
QT-UNet-A	X	✓
QT-UNet	✓	✓

Table 3.9: Overview of enabled features for the ablation models.

in constant mode with an overlap of 0.5. We report the average Dice score and the mean IoU (mIoU) across all classes in addition to the common metrics described in Section 3.6. We additionally report the inference speed, recorded by taking the average inference time over the testing epoch.

Subexperiment 2.3: NTNU Data

To evaluate the generality of the weights trained for QT-UNet-2D in subexperiments 2.1 and 2.2, we evaluate the trained QT-UNet-2D models on the NTNU dataset, described in Section 3.2.2. We apply the models directly, without fine-tuning, ignoring class 0 as void. We modify the normalisation step in our preprocessing to use the mean and standard deviation over this dataset rather than CityScapes. Whole images are segmented using sliding-window inference in constant mode with an overlap of 0.5. We report the average Dice score and the mean IoU (mIoU) across all classes except the ignore class 0, in addition to the common metrics described in Section 3.6. We additionally report inference speed, recorded by taking the average inference time over the testing epoch.

3.5.4 Ablation study

In order to disentangle the effects of adding depth reduction and our new Cross-Attention module on the performance of QT-UNet over our baseline VT-UNet, we perform a short ablation study with BraTS2021 to examine the effect of each component on the overall performance of the model when trained from scratch. We create a version of QT-UNet without depth-wise reduction and expansion in the Merging and Expansion layers and denote this model as QT-UNet-A. We also create a variant of VT-UNet with depth-wise reduction and expansion in its Merging and Expansion layers, and denote it as VT-UNet-A. Together with the two standard models, these two variant models give us four total models with which we can examine the effect of the depth-wise reduction and expansion, as well as the new Cross-Attention design. An overview of the models with enabled features is given in Table 3.9. These models are trained with the same parameters as their counterparts in Subexperiment 1.1 (see Section 3.5.2).

3.6 Model evaluation

In addition to the metrics listed for each experiment, we also include two common metrics across all experiments. These are the number of FLOPs required for a forward pass and the number of parameters in the model. FLOPs are recorded using the counter in fvcore [97] over a forward pass in training mode.

Chapter 4

Results

The results of the experiments in Chapter 3 are described in this chapter, using the metrics defined for each experiment. Experiment 1 can be found in Section 4.1, with subexperiment 1.1 BraTS 2021 in Section 4.1.1, 1.2 BTCV in Section 4.1.2, and 1.3 MSD in Section 4.1.3. Experiment 2 can be found in Section 4.2, with subexperiment 2.1 CityScapes in Section 4.2.1, 2.2 CityScapesCat in Section 4.2.2, and 2.3 NTNU in Section 4.2.3.

Results in the tables above the double line are from our experimental runs on our data split. Those below the double line are taken from the relevant leaderboards unless otherwise noted, to provide context for our results. Qualitative examples from the experiments are also provided.

In the tables, ↑ indicates that higher values are better, while ↓ indicates that lower values are better. When used with QT-UNet, the "/scratch" suffix indicates that the model was trained without pretrained weights. The omission of the "/scratch" suffix to QT-UNet indicates that the model was trained with pretrained weights. Numbers listed in bold are the best scores for that column, whilst numbers listed with an underline are the second best. Scores are presented in a 0-100 scale rather than the original 0-1 scale by multiplying the score 100×, for ease of reading and interpretation.

4.1 Experiment 1: Medical Image Computing

4.1.1 Subexperiment 1.1: BraTS 2021

We report the results of subexperiment 1.1 in Table 4.1 and qualitative results in Figure 4.1. We highlight that QT-UNet across the board requires less FLOPs than the models against which we compare, with comparable Dice results. The Base variant of QT-UNet trained from scratch attains the 2nd best average Dice score. We also note that the pretrained variants of QT-UNet attain a lower Hausdorff Distance than those trained from scratch.

Qualitatively, we observe that the edges of the segmentation masks produced by the QT-UNet variants is lightly more fine than VT-UNet, better con-

Model	#Params (M) ↓	FLOPs (G) ↓	Dice score ↑				Hausdorff Distance ↓			
			ET	TC	WT	Avg.	ET	TC	WT	Avg.
VT-UNet-T	**5.4** M	52.0 G	85.71	88.40	91.94	88.68	**4.12**	4.73	5.12	**4.69**
VT-UNet-S	11.8 M	100.8 G	**86.58**	88.01	91.85	88.82	4.16	**4.68**	5.35	5.10
VT-UNet-B	20.8 M	165 G	86.11	87.88	91.89	88.63	4.22	5.13	4.53	4.71
QT-UNet-T /scratch	6.4 M	**32.5** G	85.38	87.00	92.06	88.15	4.32	6.24	6.48	5.79
QT-UNet-S /scratch	14.5 M	61.3 G	85.90	87.60	92.20	88.56	4.18	5.17	5.95	5.39
QT-UNet-B /scratch	25.5 M	98.5 G	86.27	87.61	92.19	88.69	4.23	5.14	5.21	4.92
QT-UNet-T	6.4 M	**32.5** G	84.58	87.42	92.09	88.03	4.40	5.74	5.54	5.31
QT-UNet-S	14.5 M	61.3 G	85.66	87.70	92.15	88.50	4.43	5.63	5.24	5.18
QT-UNet-B	25.5 M	98.5 G	85.61	87.78	92.10	88.61	4.23	4.99	5.23	4.85
Swin-UNETR[1] [11]	61.98 M	394.8 G	85.80	**88.50**	**92.60**	**88.97**	6.02	5.83	**3.77**	5.21
UNETR[2] [71]	102.5 M	193.5 G	79.78	83.66	90.10	84.56	-	-	-	-
nnFormer [2] [72]	39.7 M	110.7 G	82.83	86.48	90.37	86.56	-	-	-	-

Table 4.1: BraTS2021 results. Abbreviations: ET = Enchancing Tumour, TC = Tumour Core, WT = Whole Tumour.

forming to the outline of the ground truth. The QT-UNet variant trained with pretrained weights appears to produce a slightly tighter mask than the QT-UNet variant trained from scratch, when compared to the ground truth.

4.1.2 Subexperiment 1.2: BTCV

We report the results per organ in Table 4.2, with a summary in Table 4.3. Qualitative results can be seen in Figure 2.6.

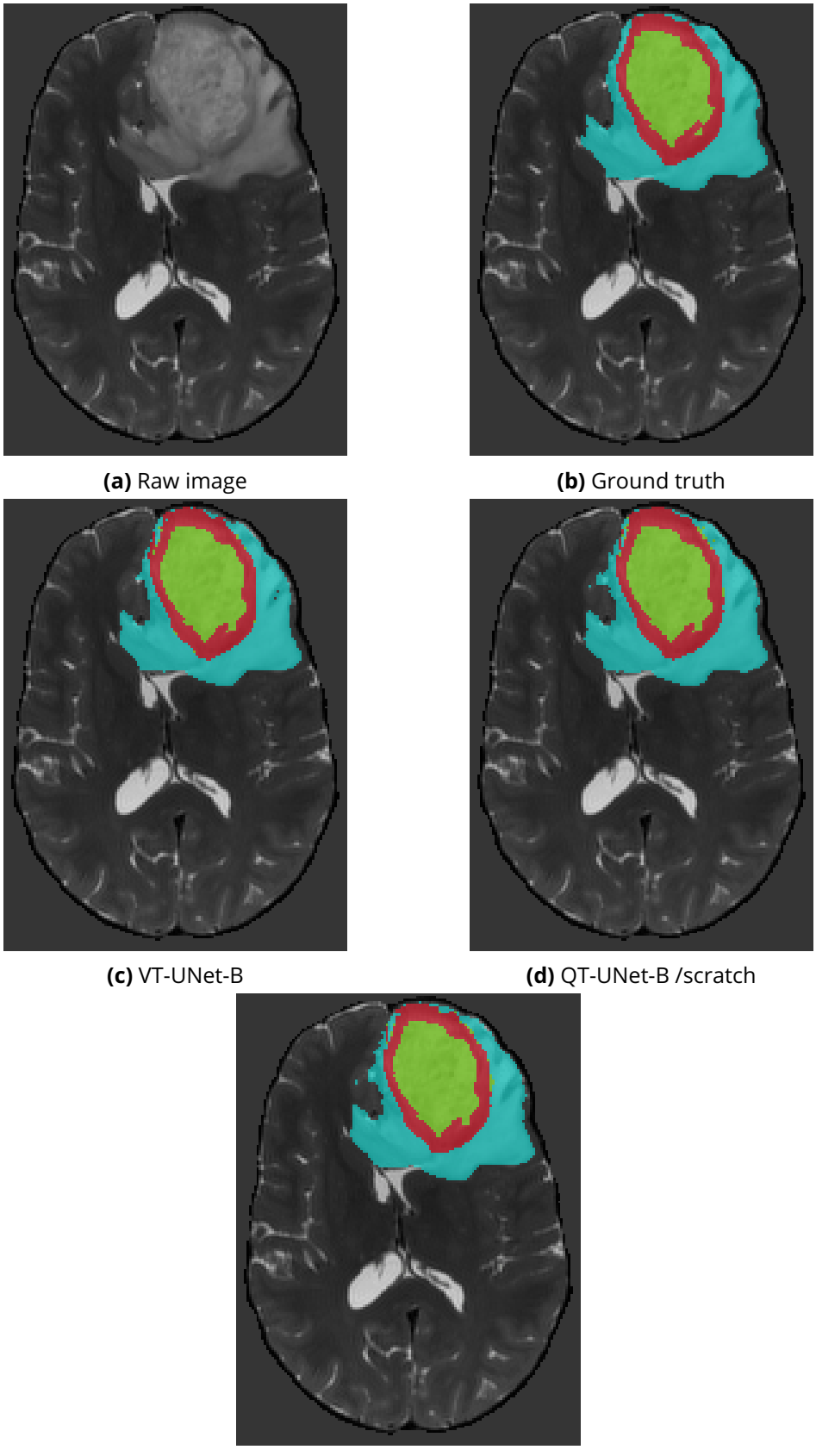
We observe that our model QT-UNet achieves significantly better results than VT-UNet, with QT-UNet showing stronger performance than VT-UNet in smaller organs such as the oesophagus, the aorta, the inferior vena cava, portal and spleinc veins, the pancreas, and adrenal glands with 23.5 Dice point margins on average between the VT-UNet and QT-UNet /scratch variants in these organs. For larger organs like the the spleen, kidneys, liver, and stomach we find smaller gains of 13 Dice points on average between VT-UNet and QT-UNet /scratch variants. We also note that QT-UNet is better able to identify the left kidney, with a 20.39 Dice point margin between VT-UNet and QT-UNet /scratch variants on average, compared to a 3 Dice point average difference for the right kidney.

We also observe that the Tiny variant of QT-UNet experiences a significant 14 DSC point performance boost and the Base variant sees a smaller 3 point boost when trained with weights pre-trained on CT-SSL, though the Small variant sees little to no change. However, both VT-UNet and QT-UNet are far weaker than the current SotA for this dataset.

Note that the results for QT-UNet and VT-UNet are based on our split of the data, whilst the results for the remaining models are taken from the BTCV leaderboard. We were unable to score our models on the BTCV test dataset, since the competition evaluation servers were offline.

[1] As reported by [11]

[2] As reported by [10]



(a) Raw image

(b) Ground truth

(c) VT-UNet-B

(d) QT-UNet-B /scratch

(e) QT-UNet-B

Figure 4.1: Example results from Experiment 1, BraTS2021.

Model	Spl	RKid	LKid	Gall	Eso	Liv	Sto	Aor	IVC	Veins	Pan	AG
VT-UNet-T	21.53	53.73	14.50	27.20	0.0	83.75	2.49	24.55	0.0	0.0	1.16	0.0
VT-UNet-S	51.27	76.28	43.23	30.97	0.0	84.65	11.16	45.81	14.41	2.21	22.46	8.69
VT-UNet-B	58.23	71.40	53.60	20.00	0.0	84.71	24.61	37.62	37.62	12.71	16.10	0.0
QT-UNet-T /scratch	62.84	60.70	50.20	24.21	23.70	79.29	24.91	53.03	21.96	39.76	26.06	16.98
QT-UNet-S /scratch	67.56	74.76	57.35	35.54	38.06	82.97	35.70	68.24	46.87	46.47	32.15	34.29
QT-UNet-B /scratch	68.11	75.65	64.95	35.11	41.97	83.80	42.28	68.08	45.95	49.75	38.99	34.46
QT-UNet-T	79.19	78.49	64.49	40.66	42.97	87.77	36.59	71.11	50.45	46.69	35.75	32.22
QT-UNet-S	71.31	73.86	59.42	35.97	39.24	84.37	35.28	66.92	44.30	48.64	29.94	32.14
QT-UNet-B	73.37	80.77	65.76	39.27	45.54	83.69	45.96	70.63	48.24	52.66	35.71	36.45
Swin-UNETR [11]	**97.60**	**95.80**	95.60	**89.30**	**87.50**	**98.50**	**95.30**	**94.90**	**90.40**	**89.90**	**89.80**	**84.60**
UNETR [71]	97.20	94.20	95.40	82.50	86.40	98.30	94.50	94.80	89.00	85.80	85.20	81.20
nnFormer [72]	90.51	86.25	86.57	70.17	-	96.84	86.83	92.04	-	-	83.35	-
nnUNet [76]	96.70	92.40	**95.70**	81.40	83.20	97.50	92.50	92.80	87.00	83.20	84.90	78.40
Swin-UNet [9]	90.66	79.61	83.28	66.53	-	94.29	76.60	85.47	-	-	56.58	-

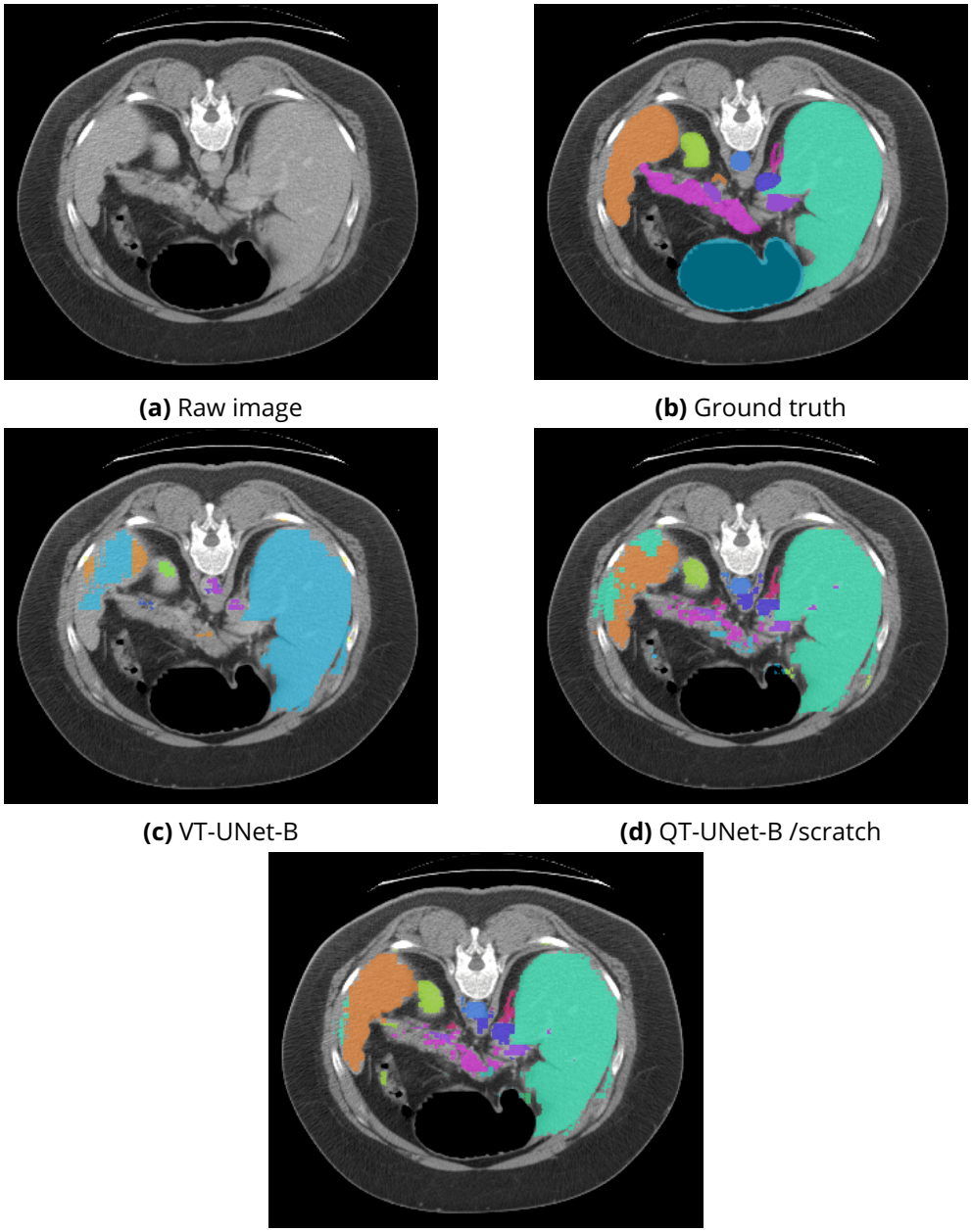
Table 4.2: BTCV Dice scores (↑) per organ.
Abbreviations: Spl: spleen, RKid: right kidney, LKid: left kidney, Gall: gallbladder, Eso: esophagus, Liv: liver, Sto: stomach, Aor: aorta, IVC: inferior vena cava, Veins: portal and splenic veins, Pan: pancreas, AG: Average of left and right adrenal glands.

Qualitatively, we observe that VT-UNet struggles to classify both the liver and the spleen, misclassifying both as stomach. None of the models are able to correctly classify the fluid-filled stomach; tough the QT-UNet variants are able to correctly segment several organs. The QT-UNet variant with pre-trained weights produces significantly better masks than the variant trained from scratch.

4.1.3 Subexperiment 1.3: MSD

We report the results of the VT-UNet and QT-UNet variants per task in Table 4.4 and a summary in Table 4.5. Select qualitative results can be found in Figure 4.3, and the remainder in the Appendix in Figure A.1. We observe that VT-UNet shows the strongest performance of all the models listed in Task 1 - Brain Tumour with QT-UNet variants being a close second. However, both VT-UNet and QT-UNet trail the SotA in all other tasks by a significant margin. Furthermore, the QT-UNet variants in tasks trained with weights pre-trained in-task (see Table 3.7) show no change or degradation in performance between the pre-trained and from scratch variants, although the QT-UNet variants in tasks trained with weights pre-trained on CT-SSL see an increase in performance in some tasks. We also observe that all QT-UNet variants produce a nil result in Task 7. All variants of VT-UNet outperform their equivalent variant of QT-UNet in the summary Table 4.3, though the margin is not very large.

The results from the models below the double line are pulled from the MSD leaderboard. Note that the submissions for this leaderboard were closed for good at the time of writing this thesis, which means that we could not make



(a) Raw image

(b) Ground truth

(c) VT-UNet-B

(d) QT-UNet-B /scratch

(e) QT-UNet-B

Figure 4.2: Example results from Experiment 1.2, BTCV.

Model	#Params (M) ↓	FLOPs (G) ↓	AVG. Dice score ↑
VT-UNet-T	**5.4** M	19.7 G	17.61
VT-UNet-S	11.8 M	38.2 G	30.76
VT-UNet-B	20.8 M	62.6 G	28.46
QT-UNet-T /scratch	6.4 M	**12.7** G	38.51
QT-UNet-S /scratch	14.5 M	23.6 G	50.33
QT-UNet-B /scratch	25.5 M	37.7 G	52.58
QT-UNet-T	6.4 M	**12.7** G	53.50
QT-UNet-S	14.5 M	23.6 G	50.27
QT-UNet-B	25.5 M	37.7 G	54.96
Swin-UNETR [11]	61.98 M	394.84 G	**91.80**
UNETR [71]	92.58 M	41.19 G	89.10
nnFormer [72]	- M	- G	86.57
nnUNet [76]	19.07 M	412.65 G	88.80
Swin-UNet [9]	- M	- G	79.13

Table 4.3: BTCV results summary.

a submission of our own. Comparisons between our results with QT-UNet and VT-UNet and those of the other models should therefore be taken with a grain of salt.

Qualitatively, we observe decent segmentation across all models in the selected tasks, though the QT-UNet variants seem a smidge better in task 6, 9, and 10. The results are pretty even in the other tasks. The pretrained variant of QT-UNet seems to produce a slightly better segmentation mask than QT-UNet /scratch in tasks 6, 9, and 10.

4.2 Experiment 2: Autonomous Driving

4.2.1 Subexperiment 2.1: CityScapes

The results for this experiment can be found in Table 4.6, with the mean Dice and IoU score for each model. The results for the models below the double line are sourced from their respective papers. Qualitative results can be seen in Figure 4.4.

We observe that the QT-UNet variants with Cross-Attention (QT-UNet-2D) show better performance in terms of both mIoU and Dice score than the variant without Cross-Attention (QT-UNet-2D-A). The Tiny and Small variants of QT-UNet see a boost in IoU and Dice scores when trained with pre-trained weights from CityScapesCoarse, seeing a 2-point increase in Dice score and a 4-point increase in mIoU. All variants of QT-UNet and VT-UNet trail SotA in terms of mIoU by a wide margin and to a lesser degree in inference speed, although QT-UNet-2D-A-T uses the fewest FLOPs at 33.6 GFLOPs.

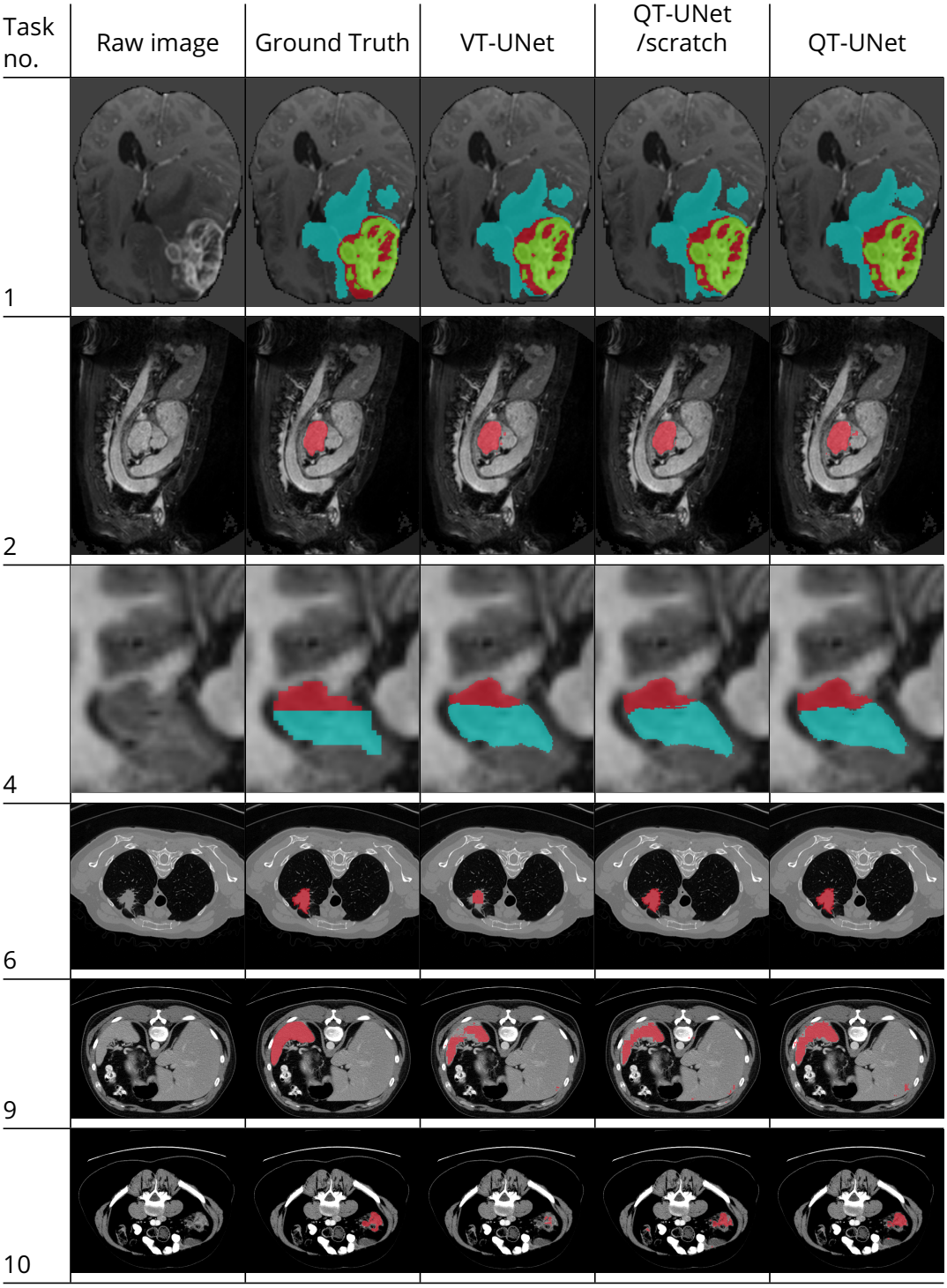


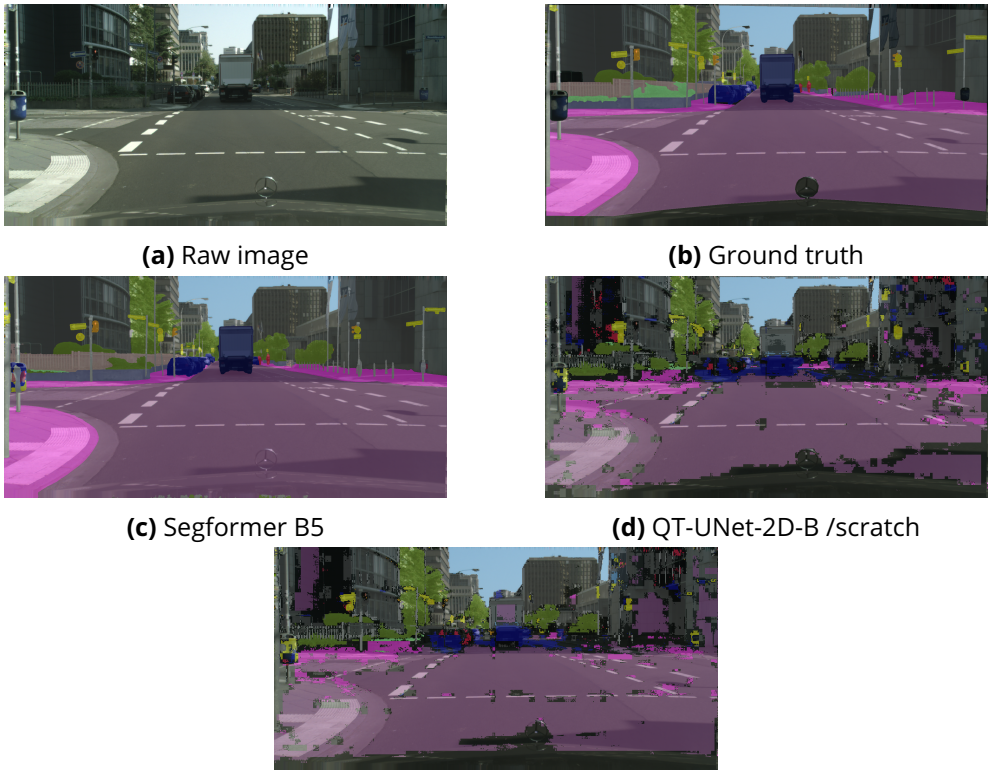
Figure 4.3: Qualitative results for select MSD tasks.

Model	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Task 10
VT-UNet-T	78.70	87.60	49.06	85.93	25.32	40.17	25.85	33.38	64.02	12.14
VT-UNet-S	78.20	86.77	47.58	85.18	24.73	45.78	23.02	33.11	69.25	7.69
VT-UNet-B	**78.61**	87.10	48.57	85.82	26.53	33.38	21.85	33.47	59.40	8.76
QT-UNet-T /scratch	77.37	85.57	36.00	85.45	30.62	26.02	0.0	42.41	49.30	14.11
QT-UNet-S /scratch	77.79	85.81	46.09	82.37	30.90	19.55	0.0	37.07	51.94	13.02
QT-UNet-B /scratch	77.81	86.52	39.31	82.92	32.26	23.21	0.0	33.77	48.10	15.76
QT-UNet-T	77.41	85.02	51.65	85.23	27.79	27.23	0.0	45.01	49.86	14.04
QT-UNet-S	77.68	85.40	37.14	82.19	31.07	19.91	0.0	36.98	53.31	13.52
QT-UNet-B	77.86	86.70	38.56	83.77	34.08	23.85	0.0	38.01	56.45	12.87
Swin-UNETR [11]	66.35	92.62	85.52	89.19	82.40	76.60	**70.71**	68.95	96.99	**59.45**
nnUNet [76]	61.10	93.30	85.86	89.46	**83.11**	73.97	67.21	**69.12**	**99.89**	58.33
Model Genesis [73]	61.14	**93.33**	**86.61**	**89.53**	81.29	74.54	65.86	68.62	97.35	51.47
Trans VW [74]	61.14	**93.33**	86.04	**89.53**	81.29	74.54	66.25	68.62	97.35	51.47

Table 4.4: MSD Dice scores (↑) per task.

Model	AVG. Dice score ↑
VT-UNet-T	50.22
VT-UNet-S	50.13
VT-UNet-B	48.35
QT-UNet-T /scratch	44.69
QT-UNet-S /scratch	44.45
QT-UNet-B /scratch	43.97
QT-UNet-T	46.32
QT-UNet-S	43.72
QT-UNet-B	45.22
Swin-UNETR [11]	**78.88**
nnUNet [76]	78.14
Model Genesis [73]	76.97
Trans VW [74]	76.96

Table 4.5: MSD results summary.



(a) Raw image

(b) Ground truth

(c) Segformer B5

(d) QT-UNet-2D-B /scratch

(e) QT-UNet-2D-B

Figure 4.4: Example results from Experiment 2.1, CityScapes.

Qualitatively, we observe that QT-UNet-2D-Base struggles to produce segmentation masks comparable to those of the Segformer, regardless of whether QT-UNet was trained from scratch or with pre-trained weights. QT-UNet-2D appears to struggle particularly with shadow borders and light patches of the road, misclassifying the light patches as a sidewalk, and classifying the shadow border into the void class. Both variants QT-UNet-2D also fail to completely segment cars and trucks in the scene, although the boundaries of buildings in the distance seem nearly correctly segmented.

4.2.2 Subexperiment 2.2: CityScapesCat

We report the results for our experiment with CityScapes over class categories (CityScapesCat) in Table 4.7, with mean Dice and IoU score for each model. Qualitative results can be seen in Figure 4.5.

We observe that the QT-UNet variants with Cross-Attention (QT-UNet-2D /scratch) show worse performance in terms of both mIoU and Dice score than

[3] As reported by Xie *et al.* [79] on a Tesla V100 card, a weaker card than the A100 cards used in this thesis.

Model	#Params (M) ↓	FLOPs (G) ↓	Inference (s) ↓	AVG. Dice score ↑	AVG. IoU score ↑
QT-UNet-2D-A-T /scratch	5.0 M	**33.6** G	0.22 s	38.82	19.82
QT-UNet-2D-A-S /scratch	11.3 M	72.5 G	0.23 s	38.68	19.97
QT-UNet-2D-A-B /scratch	20.1 M	126.3 G	0.20 s	39.10	20.25
QT-UNet-2D-T /scratch	5.4 M	38.8 G	0.20 s	39.12	20.74
QT-UNet-2D-S /scratch	12.2 M	83.3 G	0.21 s	39.35	21.04
QT-UNet-2D-B /scratch	21.6 M	144.6 G	0.23 s	39.50	21.15
QT-UNet-2D-T	5.4 M	38.8 G	0.20 s	41.53	25.86
QT-UNet-2D-S	12.2 M	83.3 G	0.21 s	**41.97**	25.97
QT-UNet-2D-B	21.6 M	144.6 G	0.23 s	39.03	21.00
SeMask [77]	211 M	455 G	- s	-	84.98
VOLO-D4 [78]	- M	- G	- s	-	84.30
Segformer-B5 [79]	84.7 M	183.3 G	0.102^3 s	-	84.00
Segformer-B0 [79]	**3.8** M	125.5 G	$\mathbf{0.0658}^3$ s	-	76.20

Table 4.6: CityScapes val results.

Model	#Params (M) ↓	FLOPs (G) ↓	Inference (s) ↓	AVG. Dice score ↑	AVG. IoU score ↑
QT-UNet-2D-A-T /scratch	**5.0** M	**33.0** G	**0.12** s	61.31	57.15
QT-UNet-2D-A-S /scratch	11.1 M	71.6 G	0.13 s	61.71	57.69
QT-UNet-2D-A-B /scratch	20.1 M	125.0 G	0.16 s	56.04	51.00
QT-UNet-2D-T /scratch	5.4 M	38.2 G	0.13 s	55.46	50.91
QT-UNet-2D-S /scratch	12.2 M	82.4 G	0.15 s	56.40	51.17
QT-UNet-2D-B /scratch	21.6 M	143.4 G	0.19 s	57.45	52.45
QT-UNet-2D-T	5.4 M	38.2 G	0.13 s	63.23	60.37
QT-UNet-2D-S	12.2 M	82.4 G	0.15 s	**64.21**	**61.55**
QT-UNet-2D-B	21.6 M	143.4 G	0.19 s	58.45	54.26

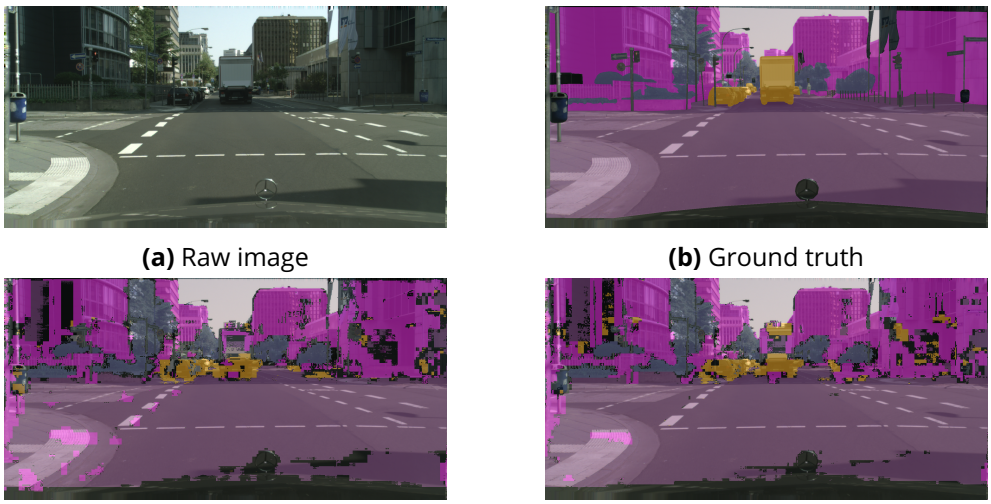
Table 4.7: CityScapesCat val results.

the variant without Cross-Attention (QT-UNet-2D-A), the opposite of subexperiment 2.1. The Tiny and Small variants of QT-UNet-2D see a large boost in performance in terms of Dice score and mIoU when trained with weights pre-trained on CityScapes Coarse, with the Base variant seeing a smaller roughly single point increase in both metrics. QT-UNet-2D-A-T uses the fewest FLOPs at 33.0 GFLOPs.

Qualitatively, we observe that QT-UNet-2D-Base trained from scratch struggles significantly with shadow borders, while QT-UNet-2D trained with pre-trained weights struggle less so. However, both variants struggle with light patches of road. Both models fail to correctly segment the truck ahead, though they both are able to identify and roughly segment cars parked to the left in the scene. Buildings further away in the distance are also nearly correctly segmented.

4.2.3 Subexperiment 2.3: NTNU data

We report the results in Table 4.8, with the mean Dice and IoU score for each model. Qualitative results can be seen in Figure 4.6. We also report results



(a) Raw image **(b)** Ground truth

(c) QT-UNet-B /scratch **(d)** QT-UNet-B

Figure 4.5: Example results from Experiment 2.2, CityScapesCat.

Model	#Params (M) ↓	FLOPs (G) ↓	Inference (s) ↓	AVG. Dice score ↑	AVG. IoU score ↑
QT-UNet-2D-T /scratch	**5.4** M	**38.2** G	**0.13** s	30.66	7.54
QT-UNet-2D-S /scratch	12.2 M	82.4 G	0.15 s	28.39	7.21
QT-UNet-2D-B /scratch	21.6 M	143.4 G	0.19 s	29.71	7.28
QT-UNet-2D-T	**5.4** M	**38.2** G	**0.13** s	30.09	**9.29**
QT-UNet-2D-S	12.2 M	82.4 G	0.15 s	29.12	8.50
QT-UNet-2D-B	21.6 M	143.4 G	0.19 s	**31.61**	8.79

Table 4.8: NTNU results.

when predicting over class categories[4] in Table 4.9, with qualitative results in Figure 4.7.

Quantitatively, we see that our model struggles to transfer to this domain, with a significant reduction across both metrics for all model variants of QT-UNet-2D compared to subexperiment 2.1 CityScapes. The same applies when looking at the results by class category and subexperiment 2.2 CityScapesCat. However, the quantitative results should be interpreted somewhat cautiously due to the small size of the dataset and consequently the small number of examples per class.

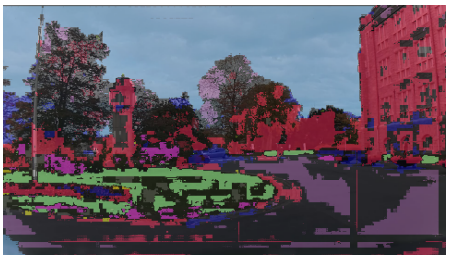
Qualitatively, we observe that the QT-UNet trained from scratch on CityScapes struggles with the NTNU data and produces a quite disjointed and error-prone mask. The QT-UNet trained on CityScapes with pre-trained weights seems to produce a more coherent mask but still has errors. A similar pattern is also present when looking at NTNU by categories, though both models struggle here.

[4]Like in subexperiment 2.2 CityScapesCat.

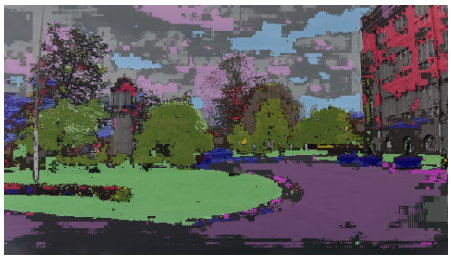
(a) Raw image

(b) Ground truth

(c) SegFormer B5



(d) QT-UNet-B /scratch

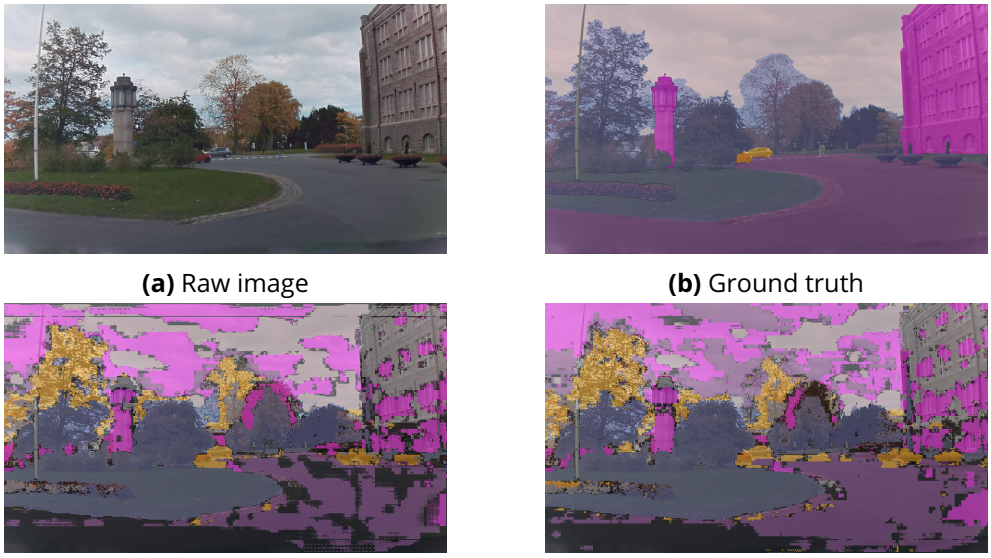


(e) QT-UNet-B

Figure 4.6: Example results from Experiment 2.2, NTNU.

Model	#Params (M) ↓	FLOPs (G) ↓	Inference (s) ↓	AVG. Dice score ↑	AVG. IoU score ↑
QT-UNet-2D-T /scratch	**5.4** M	**38.2** G	**0.13** s	30.12	20.78
QT-UNet-2D-S /scratch	12.2 M	82.4 G	0.15 s	32.30	22.59
QT-UNet-2D-B /scratch	21.6 M	143.4 G	0.19 s	27.47	18.36
QT-UNet-2D-T	**5.4** M	**38.2** G	**0.13** s	33.60	23.52
QT-UNet-2D-S	12.2 M	82.4 G	0.15 s	**34.04**	**23.77**
QT-UNet-2D-B	21.6 M	143.4 G	0.19 s	29.47	19.90

Table 4.9: NTNU by categories results.



(a) Raw image **(b)** Ground truth

(c) QT-UNet-B /scratch **(d)** QT-UNet-B

Figure 4.7: Example results from Experiment 2.2 NTNU, by categories.

4.3 Ablations

The results of our ablation study can be seen in Table 4.10. We observe that the model variants in general produce quite similar results, but with a handful of important differences. We observe that the reduction in FLOPs between the model without and with depth-wise reduction and expansion (VT-UNet → VT-UNet-A and QT-UNet-A → QT-UNet) is 32%, 33%, and 33.7% for the Tiny, Small, and Base variants, respectively, with equivalent numbers between both pairs of compared models. Comparing models without and with the new Cross-Attention module (VT-UNet → QT-UNet-A and VT-UNet-A → QT-UNet), we observe a reduction in FLOPs of 7.5%, 9%, and 9.5% for the Tiny, Small, and Base variants, respectively, with equivalent numbers between each pair of models.

Considering parameters, we observe that the increase in the number of parameters for VT-UNet models without and with depth-wise reduction and expansion (VT-UNet → VT-UNet-A) is 15% across all size variants. For the QT-UNet models (QT-UNet-A → QT-UNet), we observe a 9%, 13%, and 14 % increase in parameter count for the Tiny, Small, and Base variants.

Comparing models without and with the new Cross-Attention module (VT-UNet → QT-UNet-A and VT-UNet-A → QT-UNet), we observe an increase in parameters by 9%, 8%, and 8% between VT-UNet and QT-UNet-A[5] for the Tiny, Small, and Base variants, respectively. Between VT-UNet-A and QT-UNet[6], we

[5]That is, between models with Cross-Attention but without depth-wise reduction and expansion.

[6]That is, between models with both Cross-Attention *and* depth-wise reduction and expansion.

Model	#Params (M) ↓	FLOPs (G) ↓	Dice score ↑				Hausdorff Distance ↓			
			ET	TC	WT	Avg.	ET	TC	WT	AVG.
VT-UNet-T	**5.4** M	52 G	85.71	**88.40**	91.94	88.68	4.12	4.73	5.12	**4.69**
VT-UNet-S	11.8 M	100.8 G	**86.58**	88.01	91.85	**88.82**	4.16	**4.68**	5.35	5.10
VT-UNet-B	20.8 M	165 G	86.11	87.88	91.89	88.63	4.22	5.13	**4.53**	4.71
VT-UNet-A-T	6.2 M	35.0 G	85.16	88.00	91.64	88.27	**4.04**	4.98	4.93	4.74
VT-UNet-A-S	13.5 M	67 G	85.62	87.62	91.62	88.29	4.25	5.08	5.24	4.92
VT-UNet-A-B	23.9 M	108.7 G	86.07	87.62	91.60	88.43	4.23	4.93	4.82	4.74
QT-UNet-A-T /scratch	5.9 M	47.7 G	86.15	87.95	92.01	88.70	4.77	5.98	6.18	5.69
QT-UNet-A-S /scratch	12.8 M	91.2 G	85.66	87.96	92.17	88.60	4.68	4.93	5.91	4.96
QT-UNet-A-B /scratch	22.4 M	147.8 G	86.36	87.86	**92.24**	**88.82**	4.45	5.88	7.01	5.97
QT-UNet-T /scratch	6.4 M	**32.5** G	85.38	87.00	92.06	88.15	4.32	6.24	6.48	5.79
QT-UNet-S /scratch	14.5 M	61.3 G	85.90	87.60	92.20	88.56	4.18	5.17	5.95	5.39
QT-UNet-B /scratch	25.5 M	98.5 G	86.27	87.61	92.19	88.69	4.23	5.14	5.21	4.92

Table 4.10: Ablation study results, on BraTS2021.

observe an increase of 3%, 7%, and 6% in parameters for the Tiny, Small, and Base variants respectively.

Overall, QT-UNet has a reduction in FLOP usage of up to 40% compared to VT-UNet, at the cost of 23% more parameters.

We find that QT-UNet-A-Base, that is, a model *with* Cross-Attention and *without* depth-wise reduction and expansion, has the strongest average Dice score, sharing the top spot with VT-UNet-B. However, it should be noted that the performance differences in terms of the Dice score are relatively small. VT-UNet-T has the lowest average Hausdorff Distance, with variants with Cross-Attention and depth-wise reduction and expansion performing worse.

Chapter 5

Discussion

This chapter will discuss the project at large and our research questions. Section 5.1 through Section 5.3 discuss the results against each research question. Section 5.4 will briefly discuss other pertinent insights that can be taken from our results that are not directly related to any of the Research Questions. Finally, Section 5.5 closes this chapter with a retrospective evaluation of the project process.

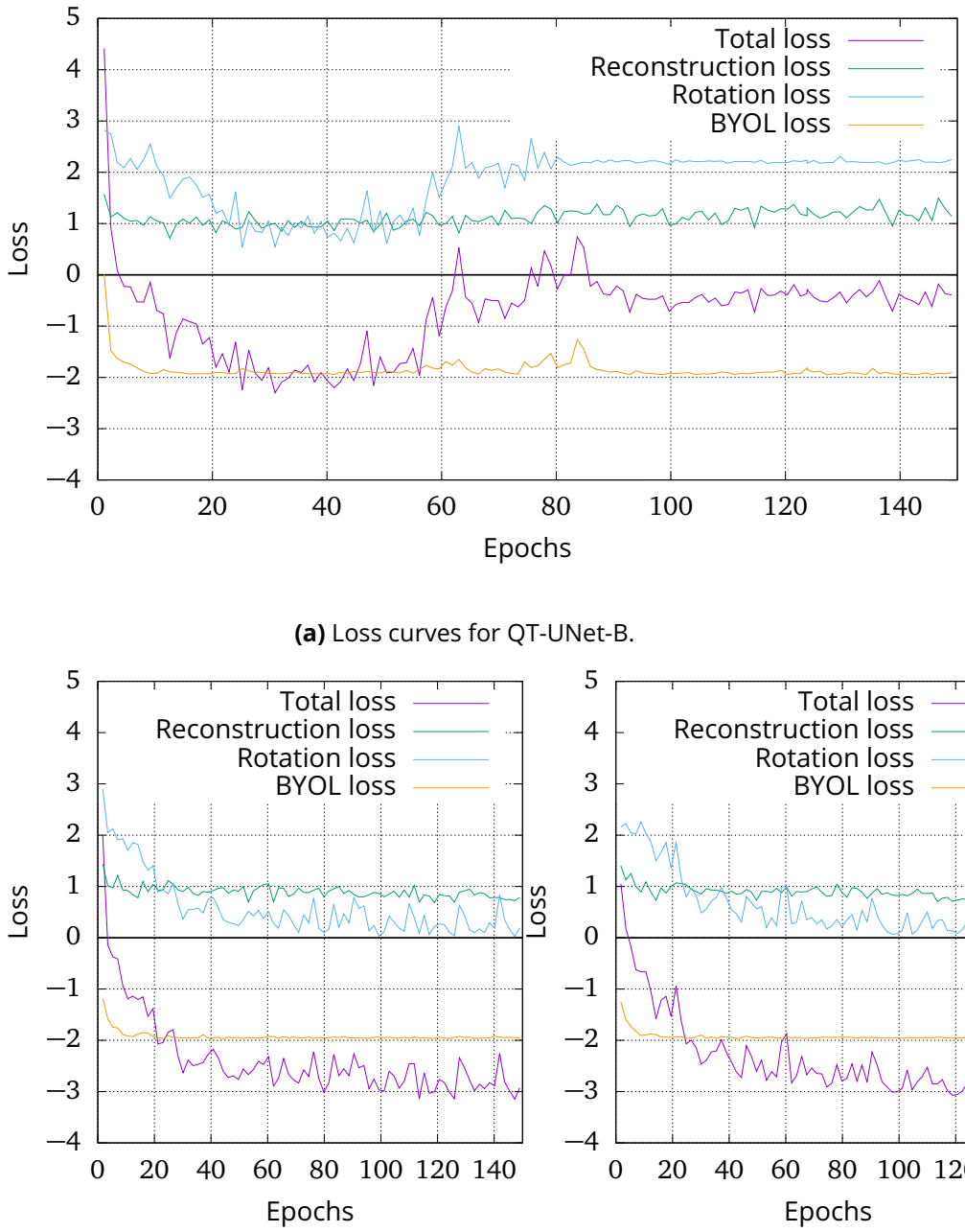
5.1 RQ1: The effect of SSL

We find that the application of SSL to our model gives mixed results, depending on what experiment it was used for and with what model. Since the experiments employ two distinct types of SSL – in-task and out-of-task – this section is split accordingly.

5.1.1 Effect of out-of-task pretraining

Looking at experiments carried out with out-of-task pre-training on CT-SSL first, we observe in BTCV (Table 4.2) per organ results that the variants of QT-UNet pre-trained on CT-SSL achieve a significantly better result for smaller organs such as the gallbladder and aorta than the variants trained from scratch, while the Base variant only sees minor changes and in some tasks significant deterioration. On average (Table 4.3) the Tiny variant of QT-UNet sees a jump of more than 15 DSC points, a slight decrease of around 0.05 points for the Small variant and a small increase of 2 points for Base variant. A similar pattern emerges when looking at the MSD scores (Table 4.4, Table 4.5), where the Tiny model sees significant improvements in some of the CT tasks, with smaller adjustments for the Small and Base variants.

This outcome is rather peculiar, as the loss curves from our CT-SSL pretraining runs illustrate. For the Tiny and Small variants (see Figure 5.1b), we observe that both the BYOL loss, the rotation loss, and reconstruction loss fall



(a) Loss curves for QT-UNet-B.

(b) Loss curves for QT-UNet-S.

(c) Loss curves for QT-UNet-T.

Figure 5.1: Loss curves for CT-SSL pretraining.

throughout the training process, whilst the Base variant sees a sharp increase in all losses around mid-way through the training process (see Figure 5.1a). After this increase in loss for the Base variant, the BYOL loss returns to its previous lower level, whilst the rotation and reconstruction loss never recovers. We theorise that the pretraining process for the Base variant somehow collapsed midway through training. This is a risk that the authors of BYOL [69] warn of, claiming that a collapse of BYOL where the model outputs only zero-vectors as projections since that also would provide a minima of loss. However, Chen and He [70] claim that this type of collapse should not be possible as long as gradients to the target network are stopped.

Another theory that could explain the collapse is the interaction between BYOL and the other SSL tasks. Could it be possible that the gradients created by the other task heads upset the balance of the model and forced it to collapse? If so, why was the same type of collapse not observed in the other variants, Tiny and Small? Given that there is not much research on the use of BYOL together with other pretext tasks, this result could indicate that more research on the interaction between BYOL and other SSL tasks is warranted.

As we shall discuss in detail in Section 5.1.5, another potential cause of the training collapse could be the rate at which the target weights were updated. Due to the nature of how the PyTorch Lightning Bolts BYOL implementation works, the weight update was applied at the end of each batch, regardless of gradient accumulation. This could have caused instability during training, especially when a large number of gradients were accumulated before they were applied to the online network.

However, despite this collapse in CT-SSL-pre-training for the Base variant, we find that it is the Small variant that experiences little to no benefit from the pre-training in both the BTCV and the MSD CT tasks. Although it is difficult to point to an exact cause, it is possible that the Small variant simply had little to no benefit from the pre-training, despite the low loss during pre-training. That is, the pre-training could have ended in a local minima that was of little use to the model. However, this seems unlikely due to the reduction in loss across all tasks, indicating that at least somewhat useful representations were learnt. More research is warranted on the effect of pretraining on the model variant.

5.1.2 Effect of in-task pretraining

Turning our attention to the in-task trained MRI tasks, we also observe mixed results. In the BraTS dataset (Table 4.1) we observe negligible changes in the Dice score, but a not insignificant decrease in the average Hausdorff Distance. This seems to indicate that pretraining aided the model in producing spatially accurate segmentation masks, which is important if the segmented volumes are to be used to guide medical personnel. We can also qualitatively observe in Figure 2.5 that the pre-trained model produces a slightly smoother mask

than the model trained from scratch.

For MSD, the in-task trained tasks 1, 4, 5 and 6 show mostly mild changes between the QT-UNets trained for scratch and those trained with pre-trained weights in-task. Apart from a 3-point increase for QT-UNet-T and a 2-point increase for QT-UNet-B in task 5, the other in-task trained tasks have equivalent or slightly degraded Dice scores. This indicates that our SSL scheme was unable to learn strong representations from the limited data available. A larger dataset for pre-training might have mitigated this.

5.1.3 Overall effect

Overall, despite pretraining QT-UNet, it still falls short of SotA in all but the BraTS dataset. Although comparisons between models above and below the double line should be taken with a grain of salt, the margin up to SotA in the BTCV and MSD datasets is considerable. For BraTS, we dare to propose that our model is competitive, though we again make such comparisons with great caution due to the nature of the results above and below the double line. Overall, we find that pretraining out-of-task with our CT-SSL dataset is the most effective, with negligible changes in performance when training in-task.

5.1.4 Implications

In our experiments, we observe that SSL seems to work much better when performing out-of-task pretraining on large datasets rather than in-task pretraining on smaller task datasets. This is, frankly, not surprising. Most pretraining datasets, particularly in SSL, are designed to take advantage of large unlabelled out-of-task datasets, as performance has been shown to scale with the size of the pre-training sets [98].

That is not to say that the in-task approach is without merit. It can be argued that there are essentially two approaches to dealing with data scarcity for model training: Either extending the dataset by adding more data to it and pretraining on related, perhaps unlabelled data, or by extracting as much learning as possible from the limited data available. In-task pretraining attempts to achieve the latter. More research to find effective techniques for in-task pretraining could greatly mediate data scarcity issues in low data availability domains such as MIC.

It is also worth noting that our approach saw some success when performing in-task pre-training on BraTS, reducing the Hausdorff Distance. This could imply that our approach could be extended to perform better in an in-task pretraining environment.

5.1.5 Error sources

Batch sizes in BYOL

Compared to the batch sizes used by Grill *et al.* [69] for their original implementation of BYOL, the batch sizes used for our experiments (see Table 3.4) are smaller than the ideal batch size 256 reported in that study due to memory constraints on the GPU accelerators. This reduced batch size could have negatively impacted performance.

Furthermore, our use of gradient accumulation to boost batch sizes was not ideal due to how the Pytorch Lightning Bolts [83] implementation of BYOL worked. Gradient accumulation instructs the model trainer to collect and average the gradients over several forward passes, only applying the backward method (and thus completing a "full" batch) only when the specified number of gradients has been collected. At the same time, the BYOL implementation from Lightning Bolts was hooked in to update the weights at the end of each forward pass. This meant that the weight update was applied after each batch, rather than after every application of the backward method. This might have caused an improperly frequent update of the weights in the target network, increasing the chances of a collapse. It should, however, be noted that the gradient accumulation used in our runs is rather moderate, for most experiments using only two batches.

Possible model collapse

As previously noted in Section 5.1.1, a possible collapse of the models in the pretraining of QT-UNet-B on CT-SSL was observed. Naturally, this could negatively impact the results from models trained with these weights, since the encoder learnt representations that could be less useful in a downstream task.

Imperfect adaption of SSL pipeline

As noted in Section 2.8, we adapt our SSL pipeline from the pipeline described by Tang *et al.* [11] for Swin-UNETR. Seeing as their code was not made public until 27.05.2022, long after the implementation phase of this project, we had to make due with their descriptions of the pipeline in their paper. Although we were fairly confident that our adaption was sound, it was hard to verify without access to the source code of the original pipeline. An imperfect adaptation might have caused a degradation in pipeline performance. We have been able to confirm, post factum, that our pipeline is a close but not faithful adaption of the one used for Swin-UNETR. Notably, they use direct linear layers for their rotation and contrastive heads rather than the MLPs we used with QT-UNet. Furthermore, they used a multilayer Transposed Convolutional Network for their reconstruction head rather than the single Transposed Convolution layer head used with QT-UNet.

5.2 RQ2: Encoder-Decoder Cross-Attention

Observing that our experiments essentially consist of three types of models – models without Cross-Attention, the original Cross-Attention module used by VT-UNet, and the new Cross-Attention module used by QT-UNet – this section is split accordingly, first discussing the effect of employing Cross-Attention (CA) at all in Section 5.2.1 and then discussing the effect of the new Cross-Attention module in Section 5.2.2.

5.2.1 Cross-Attention versus no Cross-Attention

Noting that the results in Section 4.1 are presented with our own experimental results above the double line and with leaderboard results below the double line, this section is presented bearing in mind that the results have different sources and consequently comparisons should be taken with a grain of salt. It does not make sense to present a detailed analysis between the models above and below the double line because of the differing sources, though we allow ourselves to comment on larger and more general trends between the models.

The results of subexperiment 1.1, BraTS2021, appear to indicate that models *with* Cross-Attention perform slightly better than those without it, specifically compared to UNETR and nnFormer in Table 4.1. However, this suggestion must be taken with a considerable amount of salt, given that the results above and below the double line are sourced differently. The suggestion that Cross-Attention is better than no Cross-Attention is, however, weakened by our results in the other experiments (1.2 BTCV and 1.3 MSD), where the CA-enabled models QT-UNet and VT-UNet trail all the other models by a considerable margin. It is, however, also possible that this could be an expression of other factors such as model size and data preprocessing rather than the effect of our Cross-Attention mechanism. Furthermore, the size of the crops supplied to the models could have had a not insignificant impact on the performance of the models, with a smaller $96 \times 96 \times 96$ spatial crop being used in BTCV and MSD[1] compared to the larger $128 \times 128 \times 128$ crop in BraTS, contributing to a loss of context.

5.2.2 The effect of the updated Cross-Attention module

Observing Table 4.10, specifically comparing VT-UNet with QT-UNet-A and VT-UNet-A with QT-UNet – that is, comparing models with and without the new CA module – it can be seen that the new module reduces the computational burden by 8.27%, 9.51%, and 10.42% for the Tiny, Small, and Base variants, respectively, at the cost of an increase in parameters of 9.26%, 8.47%, and 7.69%.

[1] Except for MSD Task 1, which used the same $128 \times 128 \times 128$ crop as BraTS.

At the same time, the impact on Dice score observed in Table 4.10 is mixed, with some variants seeing a boost in performance and others a slight degradation. However, the Hausdorff Distance is negatively affected across all variants. There could be a couple of possible reasons for this. One reason could simply be that this is the result of less computation happening in the decoder blocks. The fusion modules VT-UNet uses in its decoder modules allow for double the computations per stage compared to the decoders in QT-UNet due to the dual-stream design of the VT-UNet decoders, at the cost of more computational effort. Another issue could be that our new Cross-Attention module is simply not able to properly integrate information from the encoder. It is difficult to conclusively say which is the most influential factor and if there are others. A deeper quantitative and qualitative examination of the information flow through the Cross-Attention-enabled decoders is warranted but is not feasible within the time frame of this thesis.

In BTCV, all QT-UNet variants outperform their corresponding VT-UNet counterparts by a margin of 18 DSC points in the average, even when trained from scratch. Observing that the QT-UNet variants show significantly better results for smaller organs such as the oesophagus, aorta, inferior vena cava, portal and spleenic veins, as well as the pancreas, leads us to speculate that the new Cross-Attention module allowed the decoder in QT-UNet to query the encoder more efficiently for the spatial location of these organs, compared to the mechanism in VT-UNet. This theory is strengthened by the fact that, although both QT-UNet and VT-UNet perform relatively well when segmenting the right kidney, QT-UNet is significantly better at segmenting the left kidney. This could indicate that QT-UNet is better able to distinguish the two, further lending credence to the theory that the new Cross-Attention module better helps the decoder locate the targets spatially.

However, though we observe that the variants of QT-UNet outperform the corresponding variants of VT-UNet in MSD tasks 5, 8, and 10, we find that VT-UNet outperforms QT-UNet in all other tasks. This could be attributed to the new Cross-Attention mechanism, but also to the depth-wise merge and expansion added to QT-UNet. Several of the tasks were QT-UNet struggles are tasks where the segmentation masks are relatively small, though this is not the case in tasks 3, 4, and 9 where they are comparatively large. It is worth noting, however, the low number of target classes in these tasks, with most having one or two target classes. Noting that Task 9 Spleen is essentially the same task as segmenting the spleen organ in BTCV, and observing that QT-UNet outperforms VT-UNet in segmenting the spleen in that task, it seems prudent to speculate that the new Cross-Attention mechanism is affected by the number of target classes, performing better in tasks with several target classes rather than a few.

We also note that QT-UNet was wholly unable to perform in MSD task 7, where all variants get a nil score. Looking at the loss curves for these runs, we observe that they all collapse mid-way during training with a consider-

able decrease in training loss coinciding with an increase in validation loss. We speculate that the model collapses because of the relatively small targets in the task, instead collapsing to predict background everywhere. It is worth noting that we observe a similar loss behaviour for our runs with VT-UNet but that VT-UNet recovers from the collapse before the end of training.

Overall, the new Cross-Attention module seems to have attained a better speed-to-parameter trade-off, significantly reducing the computational burden. Dice score is positively affected in tasks with many target classes, but appears to have a negative effect in tasks with few target classes. In BraTS, we additionally observe a slight degradation in Hausdorff Distance.

5.2.3 Implications

Observing that the use of a Cross-Attention mechanism can have positive effects on BraTS performance compared to relevant baselines, we posit that the use of Cross-Attention has a positive impact on the performance of UNet models.

Seeing as the new Cross-Attention-mechanism achieves a better speed-to-parameter trade-off with little or no negative impact on performance, and indeed outperforms the original Cross-Attention mechanism in VT-UNet on BTCV and certain MSD tasks, we suggest that further development of Cross-Attention in Transformer based UNets could take advantage of our approach. However, the systematic weakness of the new design in terms of Hausdorff Distance and in tasks with few target classes warrants further investigation. Resolving these weaknesses would further strengthen the technique, leading to a strengthened speed-to-performance trade-off.

5.2.4 Error sources

Improper dataset pre-processing

As noted in Section 3.5.2 and Section 3.5.2, the preprocessing pipelines for the BTCV and MSD datasets were adapted from those used for Swin-UNETR due to the architectural similarities between the encoders in the models. It is, however, possible that this was not an optimal choice and that better performance could have been achieved if the pipelines had been created from scratch for QT-UNet exclusively. However, it is worth noting the considerable amount of time and effort that this would have entailed. Given the time frame of this project, developing and testing individual pipelines for the total 11 datasets between the MSD tasks and BTCV combined would have taken too long.

5.3 RQ3: Application in 2D contexts

Seeing that Research Question 3 effectively asks if the SSL and Cross-Attention scheme applied in 3D is also effective in 2D, the discussion in this section treats SSL and Cross-Attention independently. We also render a treatment of the effect of the number of classes between CityScapes and CityScapesCat, as well as a discussion on our tranfer to the NTNU data.

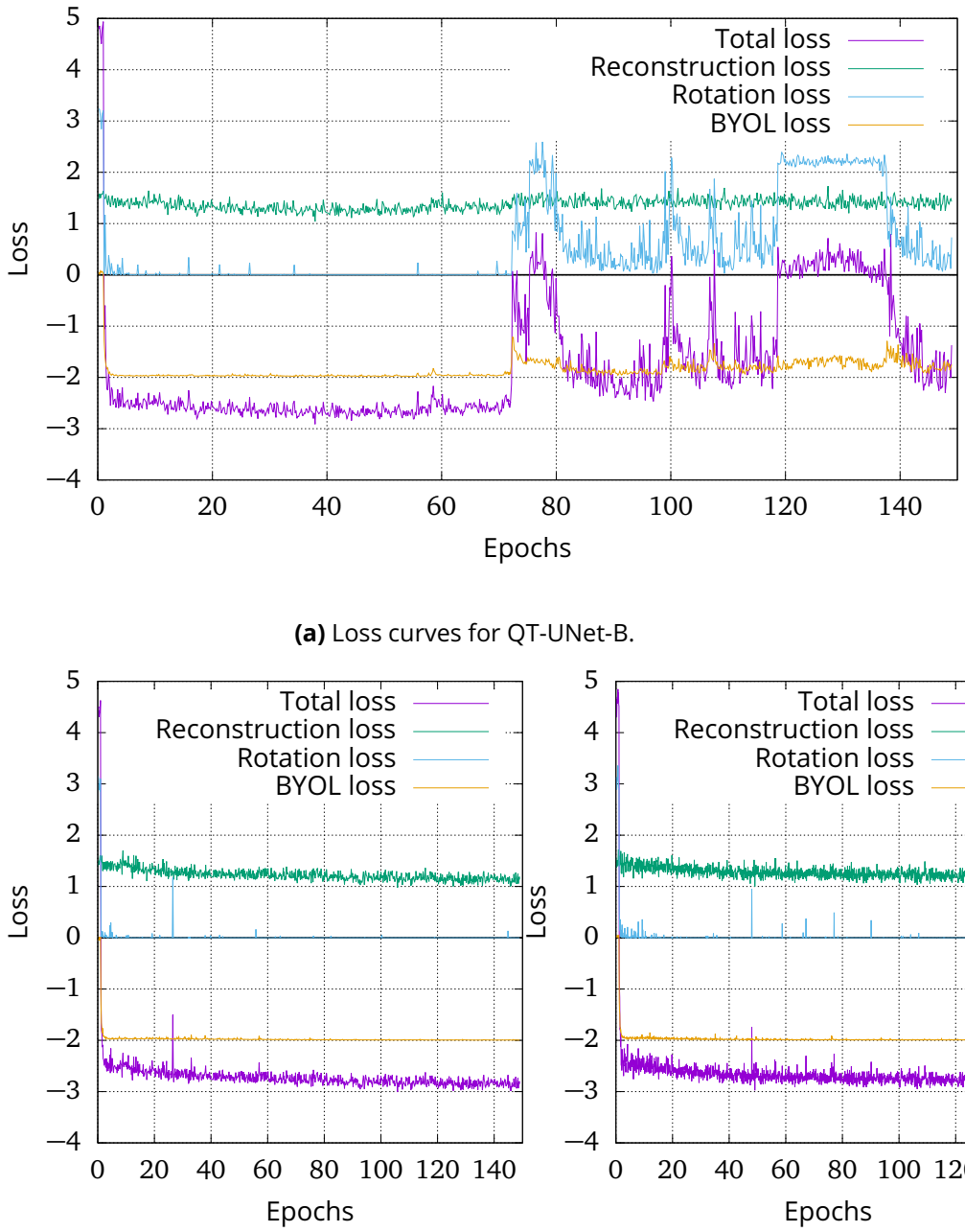
5.3.1 Effect of pretraining

As our results indicate, QT-UNet does not perform convincingly in 2D contexts like CityScapes, with a modest 41.97 point Dice score (25.97 mIoU) and is vastly outperformed by SotA. At the same time, we do observe a significant bump in Dice score between the Tiny and Small variants trained from scratch and those trained with pre-trained weights. This seems to indicate that our pre-training approach has some merit in this 2D context. At the same time, we also observe minimal changes between the pre-trained and from scratch Base variants. Looking at the loss graphs for the pre-training runs (see Figure 5.2), we observe a behaviour for each loss type similar to the collapse described for Base variant pretraining on the CT-SSL dataset, as described in Section 5.1.5. That is, the reconstruction and rotation losses suddenly increase, while the BYOL loss remains more or less the same. This collapse could explain the small difference in performance between the pre-trained and from scratch variants of QT-UNet-B.

5.3.2 Effect of cross-attention

In studying the effects of Cross-Attention on our 2D experiments with CityScapes and CityScapesCat, an interesting pattern emerges. Whilst we observe a slight increase in the Dice and IoU score in CityScapes between QT-UNet-2D-A and QT-UNet-2D across all variants (see Table 4.6), we observe the opposite in CityScapesCat with QT-UNet-2D surpassing QT-UNet-2D-A (see Table 4.7). That is, the variant with Cross-Attention beats the variant without Cross-Attention in standard CityScapes, with the reverse being true in CityScapesCat.

The only major difference between CityScapes and CityScapesCat is the number of classes to predict, 20 in CityScapes versus 8 in CityScapesCat. This disparity in the number of classes leads us to theorise that the Cross-Attention mechanism is more effective in environments with more targets, at least in a 2D context. It is, however, difficult to assert whether this effect is particular to QT-UNet-2D or if it is a general attribute of QT-UNet itself, although we also observe a similar tendency when comparing MSD task 10 and BTCV as discussed in Section 5.2.2. More experiments are warranted to explore the effect of the number of target classes on the performance of the Cross-Attention mechanism.



(a) Loss curves for QT-UNet-B.

(b) Loss curves for QT-UNet-S.

(c) Loss curves for QT-UNet-T.

Figure 5.2: Loss curves for CityScapes pretraining.

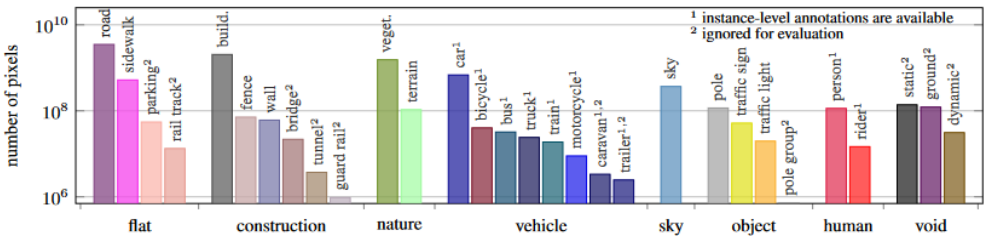


Figure 5.3: Class distribution for CityScapes in terms number of finely annotated pixels, grouped by category. Figure from [31].

5.3.3 The effect of the number of classes

Looking at the qualitative results in Figure 4.4, we see that the model indeed produces somewhat coherent segmentation masks, despite some issues such as being confused by shadows on the road. Observing that CityScapes has a high number of classes, with several of the classes such as motorbikes, caravans, and tunnels to name a few quite rare in the dataset (see Figure 5.3), it can be theorised that the model is struggling mainly due to the high number of classes and the relative rarity of several of them.

This insight led us to add CityScapesCat as an experiment. Again, we see boosts in performance between models trained from scratch and those with pre-trained weights, with minor changes for the Base variant. The qualitative results in Figure 4.5 also show slightly more coherent masks than those of regular CityScapes in Figure 4.4, noting some trouble in the void (ignored) areas.

Although the experiment with CityScapes by categories strengthens the proposition that a primary issue for the model is the number of classes, we would be remiss not to mention that QT-UNet-2D still trails far behind current SotA for the dataset. Indeed, the model is slower and less performant, though with a smaller computational and parameter budget. Although the effect of the Cross-Attention mechanism on model performance is positively correlated with the number of classes, too many classes could still overwhelm the model, especially when so many of them are so rare in the dataset. However, it can be concluded that QT-UNet-2D is too slow to be effective in any real-time AD application.

5.3.4 Transfer to NTNU data

Observing the results from our transfer from CityScapes to NTNU data in Table 4.8 and Table 4.9, we find that our models struggle to properly transfer to the new data. Although the Dice scores in Table 4.8 are quite similar to those seen for CityScapes in Table 4.6, we find that the mIoU scores are far lower for the NTNU data. Given that the Dice score metric is more rewarding of true positives than IoU, this indicates that the model is capable of generating

some true positives, but also generates many false negatives and false positives. This can also be seen in the qualitative results, where there are some areas where the model is able to correctly classify the pixels, and large areas where it produces many false positives, such as classifying the building to the right as person.

For NTNU by categories (see Table 4.9), we observe that all variants are able to transfer somewhat in terms of both Dice and mIoU scores, in contrast to NTNU by classes where the models suffer from a very low mIoU score. However, all variants are far weaker than their counterparts in CityScapes-Cat (see Figure 4.5), with more than 50% lower scores for all variants. This indicates that the weights trained for CityScapes by categories were not quite adequate for the NTNU dataset.

There could be several reasons why proper transfer could not be achieved. The most banale, but important, difference between the NTNU data and CityScapes is the season in which the images in the dataset were taken. Whilst the images in CityScapes are primarily from spring, summer, and early autumn, the images in the NTNU dataset were taken in late autumn. This difference in season has a profound impact on the appearance of objects in the image scene. For example, vegetation and shrubbery is thinner in texture and coloured brown and orange instead of green.

The NTNU data and CityScapes data are also from different countries, with the NTNU data having been sourced from Trondheim in Norway, whilst the CityScapes data is sourced from cities primarily in Germany[2]. This means that there are differences in architecture, road construction style, and signage that could have influenced the performance of QT-UNet-2D.

However, given that Segformer is able to produce quite sound segmentations of the data, as evident in Figure 4.6c, it seems more likely that the failure of our model to properly transfer is more a reflection of the properties of QT-UNet-2D rather than the exclusive effect of differences in country or the seasons.

5.3.5 Implications

As noted in the previous section, QT-UNet itself is too slow to be applied to many real-time applications. However, the individual techniques themselves do have some merits that are worth noting. The Cross-Attention mechanism seems to be positively correlated with the number of classes, seemingly better able to distinguish and locate smaller targets. This result encourages the use of the mechanism with few class tasks in Autonomous Driving, though a deeper investigation as to why the mechanism has a negative effect in tasks with fewer classes is warranted. Furthermore, the SSL-approach used seems to have had quite a strong positive effect despite its collapse for the Base variant. The effectiveness of SSL for segmentation is not novel, as several

[2]With only two foreign cities, Zürich and Strasbourg, included in the dataset.

works can testify to [64]. However, our scheme is[3] the first to share a virtually identical pipeline between the 3D MIC and 2D AD contexts. Although our experiments are too limited to elaborate on the relative effectiveness of different SSL approaches in the MIC and AD domains, it opens the door for further investigation of this type of domain- and modality-independent SSL pipeline.

5.3.6 Error sources

The ignore & void class in CityScapes and CityScapesCat

As discussed in the experiment descriptions for experiment 2.1[4] and 2.2[5], class ID 0 was ignored during evaluation as is standard for these datasets. However, we elected to include this class during training. This seems to have had a somewhat adverse effect on performance, as the model struggles to learn exactly what to ignore and classify as 0, and thus confuses classifiable objects with regions to be ignored. This phenomenon can be observed in the qualitative results in Figure 4.4 and Figure 4.5, where, for example, shadows are prone to misclassification into the ignore class. SegFormer [79], for example, ignores class 0 all-together during training. This leads us to speculate that ignoring the class during training could have helped QT-UNet-2D in both CityScapes and CityScapesCat. Naturally, better performance in CityScapes and CityScapesCat by training without class 0 could also have aided performance when transferring to the NTNU data.

5.4 Other

This section will briefly describe other pertinent insights from the experiments that are not directly related to the research questions.

5.4.1 Underutilised computational budget

The results of the ablation study (see Table 4.10) indicate a significant speedup in terms of FLOPs between VT-UNet and QT-UNet, due to the updated Cross-Attention design and the addition of spatial depth reduction. This reduction in computational burden and the consequent increase in available computational budget provide an opportunity to extend the model with another stage, a less aggressive patch embedding strategy, a higher number of embedded dimensions, and other architectural and hyperparameter changes to better exploit the released computational budget. There was not enough

[3] To our knowledge.
[4] See Section 3.5.3.
[5] See Section 3.5.3

time to carry out such experiments in this thesis, leaving it as an avenue for future exploration.

5.4.2 Depth-wise reduction and its effect on accuracy

Again looking at the results of the ablation study in Table 4.10, the only major difference in scores between versions with and without depth-wise reductions in the patch merge layers and depth-wise expansion in the patch expansion layers is in the Hausdorff Distance, increasing across all model sizes. It can be surmised that this is the price paid for the saving in FLOPs, knowing that the reason for those savings is that the model is relieved of the need to attend to long distances in the depth dimension in all stages. This means that the model is less capable of capturing certain long-range dependencies, which could negatively impact performance. Dropping the depth-wise reduction could, of course, increase performance, but at the cost of extra computation. Similarly, one could drop the reductions in height and width to help the model attend to even longer distances, but at a significantly increased computational cost. This is the essential trade-off faced in machine learning everywhere: Speed versus accuracy.

As discussed in Section 5.4.1, the reduction in the overall compute needed for the model provides a larger computational budget to extend and modify QT-UNet. Though there was not sufficient time to experiment with this in this project, it can be theorised that modifications and extensions elsewhere in the model could alleviate the performance reduction caused by the depth-wise reduction, and perhaps indeed outperform the original without depth-wise reduction. More experiments are needed to verify this theory.

5.5 Retrospective evaluation

This section will evaluate different aspects of the way this thesis was carried out, looking at positive, negative, and unfortunate aspects of the process. Overall, we are satisfied with the overall process, the effort made, and the outcome of the project, but there were also a hand-full of things that could have been done differently.

Positive aspects

- **Modular models, experiments, and pipeline**
  Building upon the experimental setup we built in the specialisation project, with modular dataset and model classes using PyTorch Lightning [82], we were able to quickly set up and test new models and datasets in our pipeline with minimal modifications to each moving part of the overall system. This enabled us to quickly set up and add new experiments quickly.
- **Using PyTorch Lightning and MONAI**
  Using these two packages greatly helped our productivity. Lightning [82] abstracted away most of the engineering challenges related to machine learning, allowing us to focus our efforts on research instead. Using MONAI [84] greatly eased the interaction with medical data, simplifying the development of MIC experiments
- **Efficient use of IDUN**
  Building upon our experience with IDUN from the specialisation project, we were able to intelligently and efficiently orchestrate the nearly 900 compute jobs completed for this project through the clever use of several job array scripts and other SLURM tricks.

Negative aspects

- **Insufficient model experimentation**
  The large number of experiments planned for this thesis meant that there was insufficient time to experiment with model configurations and hyperparameters. As noted in the discussion, QT-UNet has an underused computational budget that we were unable to tap into.
- **Slow project start**
  Though the reuse and further development of the experimental setup developed for the specialisation project helped us start this project, we still suffered a slow start to the project in which the final and proper experimental runs that were planned to begin in mid-March were not ready to go until mid-April. This set us back quite a bit from our original project timeline.
- **Missed data for CT-SSL pretraining**
  The CT-SSL dataset was intended to include 771 cardiac CT images from the TCIA COVID19 dataset [99]. Their inclusion was neglected and not discovered until pre-training on the CT-SSL dataset was well underway, in early May, but it became clear that restarted runs would take too long to complete. Therefore, it was decided to continue the runs without the extra data. This may have had a slight negative impact on the performance of the models pre-trained with this dataset. However, the general balance of the regions of interest in CT-SSL is still sound (see Table 3.2).
- **Model misconfiguration**
  We discovered in the last month of our project that we had a fatal misconfiguration in our experimental setup, with subexperiments 1.2, 1.3, 2.1 and 2.2 having been trained as if their target masks could overlap as in subexperiment 1.1, despite that not being true. We were able to redo the runs for the incorrectly configured experiments in time, though it caused considerable stress in the final stretch of the project.

Unfortunate aspects

- **File system failure on IDUN**
  Over Easter, IDUN suffered a file system failure where writing to disk would fail due to a "out of disk space" error, despite the fact that space was available. Although no data was lost, we suffered difficulties with downloading datasets to IDUN and running our experiments, as they would crash when writing to disk. This was unfortunate, as we had planned to start most of our final runs that week, setting us back another week when we were already behind schedule.
- **Swin-UNETR source code unavailable**
  As noted in our discussion of the effect of SSL, we regret the unfortunate fact that the source code for Swin-UNETR, whose pretraining scheme this project based its own scheme upon, was not published in time to affect this project, making it harder to verify that our SSL approach was valid and soundly based.
- **Disabled test servers**
  As noted in Chapter 4, several of the test servers from which we source the results for the other models did not accept new submissions when the training runs for this project ended. This meant that we were unable to submit predictions over the test set in those datasets and get results equivalent to the models against which we compare. This made it difficult to make fair comparisons between the performance of QT-UNet and that of the other models.

Chapter 6

Conclusion and Future Work

Conclusions are drawn in Section 6.1 from the discussion and results in the preceding sections, before potential avenues for future work are presented in Section 6.2.

6.1 Conclusion

Recalling the introduction to this project, the overall stated goal of the project was to:

> To explore the efficacy of a cross-domain all-Transformer UNet segmentation model based on the Swin transformer, self-supervised pre-training, and Encoder-Decoder cross-attention on Medical Image Computing and Autonomous Driving datasets.

To achieve this goal, three Research Questions were formulated and the QT-UNet model was created to incorporate them into a testable unit. QT-UNet and its variants were subjected to six experiments to answer the research questions. The first Research Question deals with the effect of Self-Supervised Learning upon the all-Transformer Swin-based UNet. For this question, we pre-trained QT-UNet using a large CT dataset consisting of 3,597 CT scans for CT tasks and using the task data directly for modalities like MRI where relevant data is more scarce.

Research Question 1

Q: What is the effect of using self-supervised pretraining of the encoder in an all-Transformer UNet on the performance of the overall network in segmentation tasks?
A: Our results show that the effect of self-supervised pre-training varies depending on whether in-task or out-of-task data is used. With out-of-task data, SSL can greatly improve the performance of the overall model. With in-task data, changes in performance are more marginal, though some improvements can be attained in certain tasks like BraTS where Hausdorff Distance was reduced.

The second Research Question asks what effect Cross-Attention has on the model. For this question, we train VT-UNet alongside QT-UNet to illuminate the effect of the updated Cross-Attention mechanism introduced in QT-UNet and compare both models to other SotA methods without Cross-Attention.

Research Question 2

Q: What is the effect of using encoder-decoder cross-attention on the overall performance of a all-Transformer UNet?
A: Our results show that the use of Cross-Attention can boost the overall performance of the model in MIC tasks. QT-UNet introduced an updated approach to the mechanism, achieving a better speed-to-performance trade-off by trading a 7.69% increase in parameters for a 10.42% decrease in FLOPs with negligible impact on Dice score compared to VT-UNet for the Base variant in BraTS. In BTCV, we find that the mechanism increases the Dice score by 18 points in the average by better capturing the position of smaller organs. However, the Cross-Attention mechanism in QT-UNet is less accurate in terms of Hausdorff Distance, indicating that it is less able to capture certain details than the original approach favoured by VT-UNet [10]. We also observe that the mechanism seems more effective in tasks with several target classes. More research is needed to explore the exact role and interaction of Cross-Attention in relation to the other components in the model and the effect of the number of classes in the target.

The third Research Question deals with the application of the techniques mentioned in RQ 1 and 2 between the 2D and 3D modalities. We spun out a 2D variant of QT-UNet by removing the depth dimension in all components and applied it to CityScapes and CityScapes by categories (CityScapesCat).

Research Question 3

Q: How can these techniques be applied effectively for both 2D and 3D segmentation tasks?
A: Our results show that the SSL pipeline can be applied directly to both 2D and 3D data without any modifications other than to account for the extra spatial dimension. Using the SSL pipeline in a similar out-of-task situation as discussed for RQ 1 leads to a significant performance increase. Cross-Attention can be applied in a similar fashion between the 2D and 3D modalities, again only needing to account for the extra spatial dimension. Its effect appears to be positively correlated with the number of classes in the target, but is negative given a restricted number of output classes. However, we find that the 2D variant of QT-UNet trails far behind the current SotA. We additionally find that the model struggles in tasks with too many target classes, as is evident by the increase in performance on CityScapesCat, although our overall results are influenced by the fact that we included the void class in our training. More research and effort is needed to bring the overall model to par with SotA.

The goal of this thesis was to explore the efficacy of a all-Transformer UNet segmentation model across domains and using Cross-Attention and SSL, giving birth to QT-UNet: The Querying Transformer U-Net. Although we find mixed results in some of the experiments, we conclude that the model and our overall approach shows promise and could potentially be developed further to fill the performance gaps discovered.

6.2 Future work

Whilst the results in this thesis are encouraging in some aspects, there is still a way to go to bring the model up to par in several of the datasets.

6.2.1 Better utilisation of computational headroom

Observing that QT-UNet has an overall reduction in FLOPs of 40% against VT-UNet, an opportunity arises to extend and modify the model within a reasonable computational budget. A handful of possibilities are listed below.

- Increase the spatial dimensions of the crop for MSD and BTCV from 96^3 to 128^3 as in BraTS2021, to add more spatial context to the model.
- Reduce the patch size in the Patch Embedding to capture more fine-grained information in the embedding and maintain a higher spatial resolution throughout the model.
- Add another stage to the UNet, deepening the network.

- Employ deep supervision, following UNetFormer [75], to strengthen decoder training.

6.2.2 Deal with ignore class in CityScapes differently

In our experiments with CityScapes and CityScapesCat, the ignore label 0 was still included during training. Retraining QT-UNet-2D without that label could be beneficial to model performance, seeing as we observed qualitatively that it reduced the quality of the predictions.

6.2.3 Deeper analysis of Cross-Attention

Our experiments uncover several interesting effects of the new Cross-Attention module, including positive and negative effects in terms of performance and speed-to-parameter trade-offs. In particular, our experiments with CityScapes and CityScapesCat uncovered a positive correlation between the number of target classes and the effect of Cross-Attention on the overall model, although the effect is negative in CityScapesCat. Furthermore, we observe that QT-UNet is much better at spleen segmentation in BTCV[1] than in MSD Task 9 Spleen segmentation[2]. This could indicate that a similar correlation also holds true in the 3D context, though our experiments are too limited to say so conclusively. Furthermore, the exact way the new Cross-Attention module incorporates the information from the encoder to the decoder is not fully understood. Experiments to illuminate these effects could inform further development of the mechanism to improve its performance.

6.2.4 Hyperparameter tuning

As noted in Section 5.4.1, our timeframe was insufficient to properly tune the hyperparameters of QT-UNet. In the future, a hyperparameter search could be conducted to potentially discover more optimal parameter values.

6.2.5 Extending CT-SSL

As noted in Section 5.5, the CT-SSL dataset is missing 717 cardiac CT images from the TCIA COVID19 dataset [99]. Adding these to the dataset and retraining the model could improve performance and reveal the impact of omitting this dataset. On a more general note, more data is generally expected to improve performance [98]. Extending the dataset with even more datasets could therefore be quite benifical.

[1] A task with 13 target classes.
[2] A task with one target class.

6.2.6 Roubustness analysis

Peiris *et al.* [10] show in their original work that VT-UNet is quite robust against data anomalies in MRI scans. A similar analysis for QT-UNet could be quite illuminating, highlighting the influence of the updated Cross-Attention module.

6.2.7 Extension to other modalities

This thesis focused on CT, MRI and RGB image data to limit the scope of the project, though QT-UNet and QT-UNet-2D are quite general in nature. They could potentially also be applied to other modalities, such as ultrasound imaging, X-ray images, histological slides, videos, and more.

6.2.8 2.5D model variant

This project produced two distinct types of QT-UNet: One for 3D modalities and one for 2D modalities. A possibility not explored in this project, a third variant that essentially merges the 3D and 2D variants, is a 2.5D variant.

In a 3D MIC context, a 2.5D model is a model that takes as input full-size (in width and height) slices of the input, as well as a hand-full of slices in either direction in the depth axis to provide context. This has the advantage of allowing the model to exploit the full spatial context of the width and height of the input and some volumetric context in the depth of the input, whilst still maintaining a manageable computational cost.

Such a model could also have uses in a 2D AD context, where time could act as surrogate for depth. For example, the model could be fed a hand-full of previous frames of either raw input or segmentations to give the model more context for the current input.

Bibliography

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, 'Attention is all you need,' in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[2] J. Devlin, M. Chang, K. Lee and K. Toutanova, 'BERT: pre-training of deep bidirectional transformers for language understanding,' in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran and T. Solorio, Eds., Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: 10.18653/v1/n19-1423. [Online]. Available: https://doi.org/10.18653/v1/n19-1423.

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever and D. Amodei, 'Language models are few-shot learners,' in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit and N. Houlsby, 'An image is worth 16x16 words: Transformers for image recognition at scale,' in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy.

[5] N. Kitaev, L. Kaiser and A. Levskaya, 'Reformer: The efficient transformer,' in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=rkgNKkHtvB.

[6] S. Wang, B. Z. Li, M. Khabsa, H. Fang and H. Ma, *Linformer: Self-attention with linear complexity*, 2020. DOI: 10.48550/ARXIV.2006.04768. [Online]. Available: https://arxiv.org/abs/2006.04768.

[7] C. Wu, F. Wu, T. Qi, Y. Huang and X. Xie, *Fastformer: Additive attention can be all you need*, 2021. DOI: 10.48550/ARXIV.2108.09084. [Online]. Available: https://arxiv.org/abs/2108.09084.

[8] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin and B. Guo, 'Swin transformer: Hierarchical vision transformer using shifted windows,' in *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, IEEE, 2021, pp. 9992–10 002. DOI: 10.1109/ICCV48922.2021.00986. [Online]. Available: https://doi.org/10.1109/ICCV48922.2021.00986.

[9] H. Cao, Y. Wang, J. Chen, D. Jiang, X. Zhang, Q. Tian and M. Wang, *Swin-unet: Unet-like pure transformer for medical image segmentation*, 2021. DOI: 10.48550/ARXIV.2105.05537. [Online]. Available: https://arxiv.org/abs/2105.05537.

[10] H. Peiris, M. Hayat, Z. Chen, G. Egan and M. Harandi, *A volumetric transformer for accurate 3d tumor segmentation*, 2021. DOI: 10.48550/ARXIV.2111.13300. [Online]. Available: https://arxiv.org/abs/2111.13300.

[11] Y. Tang, D. Yang, W. Li, H. Roth, B. Landman, D. Xu, V. Nath and A. Hatamizadeh, '[cvpr'22] self-supervised pre-training of swin transformers for 3d medical image analysis,' Mar. 2022.

[12] A. H. Håversen, 'Is general attention all you need to see? experiments on the use of general transformers for autonomous vision and medical image processing,' Specialisation/preparatory project for this thesis, Dec. 2021.

[13] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

[14] A. Kirillov, K. He, R. B. Girshick, C. Rother and P. Dollár, 'Panoptic segmentation,' in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, Computer Vision Foundation / IEEE, 2019, pp. 9404–9413. DOI: 10.1109/CVPR.2019.00963. [Online]. Available: http://openaccess.thecvf.com/content%5CCVPR%5C2019/html/Kirillov%5CPanoptic%5CSegmentation%5CCVPR%5C2019%5Cpaper.html.

[15] P. Jaccard, 'The distribution of the flora in the alpine zone.1,' *New Phytologist*, vol. 11, no. 2, pp. 37–50, 1912. DOI: https://doi.org/10.1111/j.1469-8137.1912.tb05611.x. eprint: https://nph.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-8137.1912.tb05611.x. [Online]. Available: https://nph.onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-8137.1912.tb05611.x.

[16] T. Sørenson, *A Method of Establishing Groups of Equal Amplitude in Plant Sociology Based on Similarity of Species Content and Its Application to Analyses of the Vegetation on Danish Commons*, ser. Biologiske skrifter. I kommission hos E. Munksgaard, 1948. [Online]. Available: https://books.google.no/books?id=rpS8GAAACAAJ.

[17] L. R. Dice, 'Measures of the amount of ecologic association between species,' *Ecology*, vol. 26, no. 3, pp. 297–302, 1945. DOI: https://doi.org/10.2307/1932409. eprint: https://esajournals.onlinelibrary.wiley.com/doi/pdf/10.2307/1932409. [Online]. Available: https://esajournals.onlinelibrary.wiley.com/doi/abs/10.2307/1932409.

[18] R. T. Rockafellar and R. J. B. Wets, *Variational Analysis*. Springer Berlin Heidelberg, 1998. DOI: 10.1007/978-3-642-02431-3. [Online]. Available: https://doi.org/10.1007/978-3-642-02431-3.

[19] A. L. Simpson, M. Antonelli, S. Bakas, M. Bilello, K. Farahani, B. van Ginneken, A. Kopp-Schneider, B. A. Landman, G. Litjens, B. Menze, O. Ronneberger, R. M. Summers, P. Bilic, P. F. Christ, R. K. G. Do, M. Gollub, J. Golia-Pernicka, S. H. Heckers, W. R. Jarnagin, M. K. McHugo, S. Napel, E. Vorontsov, L. Maier-Hein and M. J. Cardoso, *A large annotated medical image dataset for the development and evaluation of segmentation algorithms*, 2019. DOI: 10.48550/ARXIV.1902.09063. [Online]. Available: https://arxiv.org/abs/1902.09063.

[20] U. Baid, S. Ghodasara, S. Mohan *et al.*, *The rsna-asnr-miccai brats 2021 benchmark on brain tumor segmentation and radiogenomic classification*, 2021. DOI: 10.48550/ARXIV.2107.02314. [Online]. Available: https://arxiv.org/abs/2107.02314.

[21] B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, Y. Burren, N. Porz, J. Slotboom, R. Wiest, L. Lanczi, E. Gerstner, M.-A. Weber, T. Arbel, B. B. Avants, N. Ayache, P. Buendia, D. L. Collins, N. Cordier, J. J. Corso, A. Criminisi, T. Das, H. Delingette, Ç. Demiralp, C. R. Durst, M. Dojat, S. Doyle, J. Festa, F. Forbes, E. Geremia, B. Glocker, P. Golland, X. Guo, A. Hamamci, K. M. Iftekharuddin, R. Jena, N. M. John, E. Konukoglu, D. Lashkari, J. A. Mariz, R. Meier, S. Pereira, D. Precup, S. J. Price, T. R. Raviv, S. M. S. Reza, M. Ryan, D. Sarikaya, L. Schwartz, H.-C. Shin, J. Shotton, C. A. Silva, N. Sousa, N. K. Subbanna, G. Szekely, T. J. Taylor, O. M. Thomas, N. J. Tustison, G. Unal, F. Vasseur, M. Wintermark, D. H. Ye, L. Zhao, B. Zhao, D. Zikic, M. Prastawa, M. Reyes and K.

Van Leemput, 'The multimodal brain tumor image segmentation benchmark (brats),' *IEEE Transactions on Medical Imaging*, vol. 34, no. 10, pp. 1993–2024, 2015. DOI: 10.1109/TMI.2014.2377694.

[22] S. Bakas, H. Akbari, A. Sotiras, M. Bilello, M. Rozycki, J. S. Kirby, J. B. Freymann, K. Farahani and C. Davatzikos, 'Advancing the cancer genome atlas glioma MRI collections with expert segmentation labels and radiomic features,' *Scientific Data*, vol. 4, no. 1, Sep. 2017. DOI: 10.1038/sdata.2017.117. [Online]. Available: https://doi.org/10.1038/sdata.2017.117.

[23] B. Landman, Z. Xu, J. E. Igelsias, M. Styner, T. R. Langerak, A. Klein, D. Shen, H. Wang, J. Gee, A. Akhondi-Asl, C. Ledig and Y. Fan, *Segmentation outside the cranial vault challenge*, 2015. DOI: 10.7303/SYN3193805. [Online]. Available: https://repo-prod.prod.sagebase.org/repo/v1/doi/locate?id=syn3193805&type=ENTITY.

[24] J. Gamper, N. A. Koohbanani, K. Benet, A. Khuram and N. Rajpoot, 'Pannuke: An open pan-cancer histology dataset for nuclei instance segmentation and classification,' in *European Congress on Digital Pathology*, Springer, 2019, pp. 11–19.

[25] *Autopilot and full self-driving capability*, Apr. 2022. [Online]. Available: https://www.tesla.com/support/autopilot.

[26] A. Geiger, P. Lenz and R. Urtasun, 'Are we ready for autonomous driving? the kitti vision benchmark suite,' in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[27] A. Geiger, P. Lenz, C. Stiller and R. Urtasun, 'Vision meets robotics: The kitti dataset,' *International Journal of Robotics Research (IJRR)*, 2013.

[28] J. Fritsch, T. Kuehnl and A. Geiger, 'A new performance measure and evaluation benchmark for road detection algorithms,' in *International Conference on Intelligent Transportation Systems (ITSC)*, 2013.

[29] M. Menze and A. Geiger, 'Object scene flow for autonomous vehicles,' in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[30] H. Alhaija, S. Mustikovela, L. Mescheder, A. Geiger and C. Rother, 'Augmented reality meets computer vision: Efficient data generation for urban driving scenes,' *International Journal of Computer Vision (IJCV)*, 2018.

[31] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth and B. Schiele, 'The cityscapes dataset for semantic urban scene understanding,' in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[32] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan and O. Beijbom, 'Nuscenes: A multimodal dataset for autonomous driving,' *arXiv preprint arXiv:1903.11027*, 2019.

[33] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[34] X. Glorot and Y. Bengio, 'Understanding the difficulty of training deep feedforward neural networks,' *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, Jan. 2010.

[35] K. He, X. Zhang, S. Ren and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, 2015. arXiv: 1502.01852 [cs.CV].

[36] D. P. Kingma and J. Ba, 'Adam: A method for stochastic optimization,' in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980.

[37] S. Ioffe and C. Szegedy, 'Batch normalization: Accelerating deep network training by reducing internal covariate shift,' in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, F. R. Bach and D. M. Blei, Eds., ser. JMLR Workshop and Conference Proceedings, vol. 37, JMLR.org, 2015, pp. 448–456. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.html.

[38] S. Santurkar, D. Tsipras, A. Ilyas and A. Madry, 'How does batch normalization help optimization?' In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett, Eds., 2018, pp. 2488–2498. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/905056c1ac1dad141560467e0a99e1cf-Abstract.html.

[39] J. L. Ba, J. R. Kiros and G. E. Hinton, *Layer normalization*, 2016. arXiv: 1607.06450 [stat.ML].

[40] O. Ronneberger, P. Fischer and T. Brox, 'U-net: Convolutional networks for biomedical image segmentation,' in *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference Munich, Germany, October 5 - 9, 2015, Proceedings, Part III*, N. Navab, J. Hornegger, W. M. W. III and A. F. Frangi, Eds., ser. Lecture Notes in Computer Science, vol. 9351, Springer, 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28. [Online]. Available: https://doi.org/10.1007/978-3-319-24574-4%5C_28.

[41] Y. Tay, M. Dehghani, D. Bahri and D. Metzler, *Efficient transformers: A survey*, 2020. arXiv: 2009.06732 [cs.LG].

[42] X. Wang, R. B. Girshick, A. Gupta and K. He, 'Non-local neural networks,' in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, Computer Vision Foundation / IEEE Computer Society, 2018, pp. 7794–7803. DOI: 10.1109/CVPR.2018.00813. [Online]. Available: http://openaccess.thecvf.com/content%5C_cvpr%5C_2018/html/Wang%5C_Non-Local%5C_Neural%5C_Networks%5C_CVPR%5C_2018%5C_paper.html.

[43] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov and S. Zagoruyko, 'End-to-end object detection with transformers,' in *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I*, A. Vedaldi, H. Bischof, T. Brox and J. Frahm, Eds., ser. Lecture Notes in Computer Science, vol. 12346, Springer, 2020, pp. 213–229. DOI: 10.1007/978-3-030-58452-8_13. [Online]. Available: https://doi.org/10.1007/978-3-030-58452-8%5C_13.

[44] N. Parmar, P. Ramachandran, A. Vaswani, I. Bello, A. Levskaya and J. Shlens, 'Stand-alone self-attention in vision models,' in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox and R. Garnett, Eds., 2019, pp. 68–80. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/3416a75f4cea9109507cacd8e2f2aefc-Abstract.html.

[45] H. Wang, Y. Zhu, B. Green, H. Adam, A. L. Yuille and L. Chen, 'Axial-deeplab: Stand-alone axial-attention for panoptic segmentation,' in *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part IV*, A. Vedaldi, H. Bischof, T. Brox and J. Frahm, Eds., ser. Lecture Notes in Computer Science, vol. 12349, Springer, 2020, pp. 108–126. DOI: 10.1007/978-3-030-58548-8_7. [Online]. Available: https://doi.org/10.1007/978-3-030-58548-8%5C_7.

[46] C. Sun, A. Shrivastava, S. Singh and A. Gupta, 'Revisiting unreasonable effectiveness of data in deep learning era,' in *2017 IEEE International Conference on Computer Vision (ICCV)*, Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2017, pp. 843–852. DOI: 10.1109/ICCV.2017.97. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICCV.2017.97.

[47] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, 'ImageNet Large Scale Visual Recognition Challenge,' *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[48] A. Krizhevsky, G. Hinton *et al.*, 'Learning multiple layers of features from tiny images,' 2009.

[49] X. Zhai, J. Puigcerver, A. Kolesnikov, P. Ruyssen, C. Riquelme, M. Lucic, J. Djolonga, A. S. Pinto, M. Neumann, A. Dosovitskiy, L. Beyer, O. Bachem, M. Tschannen, M. Michalski, O. Bousquet, S. Gelly and N. Houlsby, *A large-scale study of representation learning with the visual task adaptation benchmark*, 2020. arXiv: 1910.04867 [cs.CV].

[50] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan and M. Shah, 'Transformers in vision: A survey,' *ACM Comput. Surv.*, Dec. 2021, Just Accepted, ISSN: 0360-0300. DOI: 10.1145/3505244. [Online]. Available: https://doi.org/10.1145/3505244.

[51] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu, Z. Yang, Y. Zhang and D. Tao, 'A survey on vision transformer,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2022. DOI: 10.1109/TPAMI.2022.3152247.

[52] M. Raghu, T. Unterthiner, S. Kornblith, C. Zhang and A. Dosovitskiy, 'Do vision transformers see like convolutional neural networks?' In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang and J. W. Vaughan, Eds., vol. 34, Curran Associates, Inc., 2021, pp. 12 116–12 128. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/652cf38361a209088302ba2b8b7f51e0-Paper.pdf.

[53] R. Ranftl, A. Bochkovskiy and V. Koltun, 'Vision transformers for dense prediction,' in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 12 159–12 168. DOI: 10.1109/ICCV48922.2021.01196.

[54] X. Chu, Z. Tian, Y. Wang, B. Zhang, H. Ren, X. Wei, H. Xia and C. Shen, 'Twins: Revisiting the design of spatial attention in vision transformers,' in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang and J. W. Vaughan, Eds., vol. 34, Curran Associates, Inc., 2021, pp. 9355–9366. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/4e0928de075538c593fbdabb0c5ef2c3-Paper.pdf.

[55] A. Jaegle, F. Gimeno, A. Brock, O. Vinyals, A. Zisserman and J. Carreira, 'Perceiver: General perception with iterative attention,' in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, Jul. 2021, pp. 4651–4664. [Online]. Available: https://proceedings.mlr.press/v139/jaegle21a.html.

[56] J. Ho, N. Kalchbrenner, D. Weissenborn and T. Salimans, *Axial attention in multidimensional transformers*, 2019. arXiv: 1912.12180 [cs.CV].

[57] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso and A. Torralba, 'Scene parsing through ade20k dataset,' in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[58] B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso and A. Torralba, 'Semantic understanding of scenes through the ade20k dataset,' *International Journal of Computer Vision*, vol. 127, no. 3, pp. 302–321, 2019.

[59] R. Mottaghi, X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun and A. Yuille, 'The role of context for object detection and semantic segmentation in the wild,' in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[60] T. Xiao, Y. Liu, B. Zhou, Y. Jiang and J. Sun, 'Unified perceptual parsing for scene understanding,' in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu and Y. Weiss, Eds., Cham: Springer International Publishing, 2018, pp. 432–448, ISBN: 978-3-030-01228-1.

[61] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, F. Wei and B. Guo, *Swin transformer v2: Scaling up capacity and resolution*, 2021. arXiv: 2111.09883 [cs.CV].

[62] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo and L. Shao, 'Pyramid vision transformer: A versatile backbone for dense prediction without convolutions,' in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 548–558. DOI: 10.1109/ICCV48922.2021.00061.

[63] X. Chu, Z. Tian, B. Zhang, X. Wang, X. Wei, H. Xia and C. Shen, *Conditional positional encodings for vision transformers*, 2021. arXiv: 2102.10882 [cs.CV].

[64] L. Jing and Y. Tian, 'Self-supervised visual feature learning with deep neural networks: A survey,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 11, pp. 4037–4058, Nov. 2021, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2020.2992393.

[65] A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee and F. Makedon, 'A survey on contrastive self-supervised learning,' *Technologies*, vol. 9, no. 1, 2021, ISSN: 2227-7080. DOI: 10.3390/technologies9010002. [Online]. Available: https://www.mdpi.com/2227-7080/9/1/2.

[66] T. Chen, S. Kornblith, M. Norouzi and G. Hinton, 'A simple framework for contrastive learning of visual representations,' in *Proceedings of the 37th International Conference on Machine Learning*, H. D. III and A. Singh, Eds., ser. Proceedings of Machine Learning Research, vol. 119, PMLR, Jul. 2020, pp. 1597–1607. [Online]. Available: https://proceedings.mlr.press/v119/chen20j.html.

[67] K. He, H. Fan, Y. Wu, S. Xie and R. Girshick, 'Momentum contrast for unsupervised visual representation learning,' in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 9726–9735. DOI: 10.1109/CVPR42600.2020.00975.

[68] X. Chen, H. Fan, R. Girshick and K. He, *Improved baselines with momentum contrastive learning*, 2020. DOI: 10.48550/ARXIV.2003.04297. [Online]. Available: https://arxiv.org/abs/2003.04297.

[69] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, B. Piot, k. kavukcuoglu koray, R. Munos and M. Valko, 'Bootstrap your own latent - a new approach to self-supervised learning,' in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 21 271–21 284. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/f3ada80d5c4ee70142b17b8192b2958e-Paper.pdf.

[70] X. Chen and K. He, 'Exploring simple siamese representation learning,' in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 15 745–15 753. DOI: 10.1109/CVPR46437.2021.01549.

[71] A. Hatamizadeh, Y. Tang, V. Nath, D. Yang, A. Myronenko, B. Landman, H. R. Roth and D. Xu, 'Unetr: Transformers for 3d medical image segmentation,' in *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2022, pp. 1748–1758. DOI: 10.1109/WACV51458.2022.00181.

[72] H.-Y. Zhou, J. Guo, Y. Zhang, L. Yu, L. Wang and Y. Yu, *Nnformer: Interleaved transformer for volumetric segmentation*, 2021. DOI: 10.48550/ARXIV.2109.03201. [Online]. Available: https://arxiv.org/abs/2109.03201.

[73] Z. Zhou, V. Sodha, M. M. Rahman Siddiquee, R. Feng, N. Tajbakhsh, M. B. Gotway and J. Liang, 'Models genesis: Generic autodidactic models for 3d medical image analysis,' in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*, D. Shen, T. Liu, T. M. Peters, L. H. Staib, C. Essert, S. Zhou, P.-T. Yap and A. Khan, Eds., Cham: Springer International Publishing, 2019, pp. 384–393, ISBN: 978-3-030-32251-9.

[74] F. Haghighi, M. R. H. Taher, Z. Zhou, M. B. Gotway and J. Liang, 'Transferable visual words: Exploiting the semantics of anatomical patterns for self-supervised learning,' *IEEE Transactions on Medical Imaging*, vol. 40, no. 10, pp. 2857–2868, 2021. DOI: 10.1109/TMI.2021.3060634.

[75] A. Hatamizadeh, Z. Xu, D. Yang, W. Li, H. Roth and D. Xu, *Unetformer: A unified vision transformer model and pre-training framework for 3d medical image segmentation*, 2022. DOI: 10.48550/ARXIV.2204.00631. [Online]. Available: https://arxiv.org/abs/2204.00631.

[76] F. Isensee, P. F. Jaeger, S. A. A. Kohl, J. Petersen and K. H. Maier-Hein, 'nnU-net: A self-configuring method for deep learning-based biomedical image segmentation,' *Nature Methods*, vol. 18, no. 2, pp. 203–211, Dec. 2020. DOI: 10.1038/s41592-020-01008-z. [Online]. Available: https://doi.org/10.1038/s41592-020-01008-z.

[77] J. Jain, A. Singh, N. Orlov, Z. Huang, J. Li, S. Walton and H. Shi, *Semask: Semantically masked transformers for semantic segmentation*, 2021. DOI: 10.48550/ARXIV.2112.12782. [Online]. Available: https://arxiv.org/abs/2112.12782.

[78] L. Yuan, Q. Hou, Z. Jiang, J. Feng and S. Yan, *Volo: Vision outlooker for visual recognition*, 2021. DOI: 10.48550/ARXIV.2106.13112. [Online]. Available: https://arxiv.org/abs/2106.13112.

[79] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez and P. Luo, 'Segformer: Simple and efficient design for semantic segmentation with transformers,' in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang and J. W. Vaughan, Eds., vol. 34, Curran Associates, Inc., 2021, pp. 12 077–12 090. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/64f1f27bf1b4ec22924fd0acb550c235-Paper.pdf.

[80] *Anaconda software distribution*, version Vers. 2-2.4.0, 2016. [Online]. Available: https://anaconda.com/.

[81] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, 'Pytorch: An imperative style, high-performance deep learning library,' in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[82] W. Falcon and The PyTorch Lightning team, *PyTorch Lightning*, version 1.4, Mar. 2019. DOI: 10.5281/zenodo.3828935. [Online]. Available: https://github.com/PyTorchLightning/pytorch-lightning.

[83] W. Falcon and K. Cho, 'A framework for contrastive self-supervised learning and designing a new approach,' *arXiv preprint arXiv:2009.00104*, 2020.

[84] M. Consortium, 'Monai: Medical open network for ai,' Feb. 2022. DOI: 10.5281/zenodo.6114127.

[85] M. Själander, M. Jahre, G. Tufte and N. Reissmann, *EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure*, 2019. arXiv: 1912.05848 [cs.DC].

[86] A. B. Yoo, M. A. Jette and M. Grondona, 'Slurm: Simple linux utility for resource management,' in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph and U. Schwiegelshohn, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60, ISBN: 978-3-540-39727-4.

[87] Z. Liu, J. Ning, Y. Cao, Y. Wei, Z. Zhang, S. Lin and H. Hu, *Video swin transformer*, 2021. DOI: 10.48550/ARXIV.2106.13230. [Online]. Available: https://arxiv.org/abs/2106.13230.

[88] K. Clark, B. Vendt, K. Smith, J. Freymann, J. Kirby, P. Koppel, S. Moore, S. Phillips, D. Maffitt, M. Pringle, L. Tarbox and F. Prior, *The cancer imaging archive (tcia): Maintaining and operating a public information repository*, Jul. 2013. DOI: 10.1007/s10278-013-9622-7. [Online]. Available: http://dx.doi.org/10.1007/s10278-013-9622-7.

[89] H. Roth, L. Lu, A. Seff, K. M. Cherry, J. Hoffman, S. Wang, J. Liu, E. Turkbey and R. M. Summers, *A new 2.5 d representation for lymph node detection in ct*, 2015. DOI: 10.7937/K9/TCIA.2015.AQIIDCNM. [Online]. Available: https://wiki.cancerimagingarchive.net/x/0gAtAQ.

[90] C. K. Smith K, *Data from ct_colonography*, 2015. DOI: 10.7937/K9/TCIA.2015.NWTESAY1. [Online]. Available: https://wiki.cancerimagingarchive.net/x/DQE2.

[91] S. Desai, A. Baghal, T. Wongsurawat, P. Jenjaroenpun, T. Powell, S. Al-Shukri, K. Gates, P. Farmer, M. Rutherford, G. Blake, T. Nolan, K. Sexton, W. Bennett, K. Smith, S. Syed and F. Prior, 'Chest imaging representing a COVID-19 positive rural u.s. population,' *Scientific Data*, vol. 7, no. 1, Nov. 2020. DOI: 10.1038/s41597-020-00741-6. [Online]. Available: https://doi.org/10.1038/s41597-020-00741-6.

[92] E. Tsai, S. Simpson, M. P. Lungren, M. Hershman, L. Roshkovan, E. Colak, B. J. Erickson, G. Shih, A. Stein, J. Kalpathy-Cramer, J. Shen, M. A. Hafez, S. John, P. Rajiah, B. P. Pogatchnik, J. T. Mongan, E. Altinmakas, E. Ranschaert, F. C. Kitamura, L. Topff, L. Moy, J. P. Kanne and C. C. Wu, *Medical imaging data resource center - rsna international covid radiology database release 1a - chest ct covid+ (midrc-ricord-1a)*, 2020. DOI: 10.7937/VTW4-X588. [Online]. Available: https://wiki.cancerimagingarchive.net/x/DoDTB.

[93] E. B. Tsai, S. Simpson, M. P. Lungren, M. Hershman, L. Roshkovan, E. Colak, B. J. Erickson, G. Shih, A. Stein, J. Kalpathy-Cramer, J. Shen, M. A. Hafez, S. John, P. Rajiah, B. P. Pogatchnik, J. T. Mongan, E. Altinmakas, E. Ranschaert, F. C. Kitamura, L. Topff, L. Moy, J. P. Kanne and C. Wu, *Medical imaging data resource center (midrc) - rsna international covid open research database (ricord) release 1b - chest ct covid-*, 2021. DOI: 10.7937/31V8-4A40. [Online]. Available: https://wiki.cancerimagingarchive.net/x/K4DTB.

[94] A. A. Yorke, G. C. McDonald, D. Solis and T. Guerrero, *Pelvic reference data [dataset]*, 2019. DOI: 10.7937/TCIA.2019.WOSKQ5OO. [Online]. Available: https://wiki.cancerimagingarchive.net/x/tQGJAw.

[95] T. Tong and M. Li, *Abdominal or pelvic enhanced ct images within 10 days before surgery of 230 patients with stage ii colorectal cancer*, 2022. DOI: 10.7937/P5K5-TG43. [Online]. Available: https://wiki.cancerimagingarchive.net/x/3wL7Bg.

[96] S. G. Armato III, G. McLennan, L. Bidaut, M. F. McNitt-Gray, C. R. Meyer, A. P. Reeves, B. Zhao, D. R. Aberle, C. I. Henschke, E. A. Hoffman, E. A. Kazerooni, H. MacMahon, E. J. Van Beek, D. Yankelevitz, A. M. Biancardi, P. H. Bland, M. S. Brown, R. M. Engelmann, G. E. Laderach, D. Max, R. C. Pais, D. P. Qing, R. Y. Roberts, A. R. Smith, A. Starkey, P. Batra, P. Caligiuri, A. Farooqi, G. W. Gladish, C. M. Jude, R. F. Munden, I. Petkovska, L. E. Quint, L. H. Schwartz, B. Sundaram, L. E. Dodd, C. Fenimore, D. Gur, N. Petrick, J. Freymann, J. Kirby, B. Hughes, A. V. Casteele, S. Gupte, M. Sallam, M. D. Heath, M. H. Kuhn, E. Dharaiya, R. Burns, D. S. Fryd, M. Salganicoff, V. Anand, U. Shreter, S. Vastagh, B. Y. Croft and L. P. Clarke, *Data from lidc-idri*, 2015. DOI: 10.7937/K9/TCIA.2015.LO9QL9SX. [Online]. Available: https://wiki.cancerimagingarchive.net/x/rgAe.

[97] *Facebookresearch/fvcore: Collection of common code that's shared among different research projects in fair computer vision team.* [Online]. Available: https://github.com/facebookresearch/fvcore.

[98] B. Zoph, G. Ghiasi, T.-Y. Lin, Y. Cui, H. Liu, E. D. Cubuk and Q. Le, 'Rethinking pre-training and self-training,' in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 3833–3845. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/27e9661e033a73a6ad8cefcde965c54d-Paper.pdf.

[99] P. An, S. Xu, S. A. Harmon, E. B. Turkbey, T. H. Sanford, A. Amalou, M. Kassin, N. Varble, M. Blain, V. Anderson, F. Patella, G. Carrafiello, B. T. Turkbey and B. J. Wood, *Ct images in covid-19*, 2020. DOI: 10.7937/TCIA.2020.GQRY-NC81. [Online]. Available: https://wiki.cancerimagingarchive.net/x/o5QvB.

Appendix A

MSD qualitative results

The qualitative results from MSD task not selected to be shown in Chapter 4 - Results can be seen in Figure A.1.

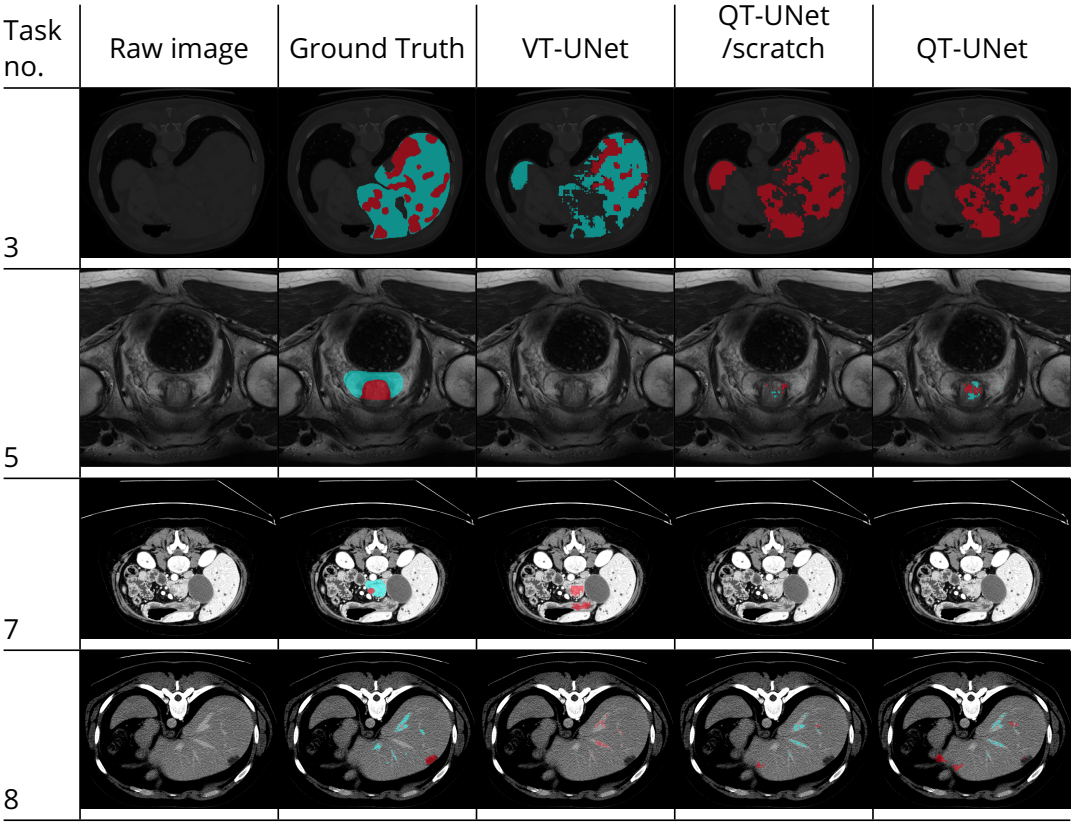


Figure A.1: Qualitative results for select MSD tasks.

Appendix B

CityScapes class mapping

Ten of the 30 classes in the CityScapes dataset are to be ignored when using the dataset. The exact mapping between the original class ID and the mapped class ID and category ID is given in Table B.1, as described in [31]. Table B.2 shows the names of the categories in CityScapes with associated IDs.

Label name	Original class ID	Mapped class ID	Mapped category ID
Unlabeled	0	0	0
Ego vehicle	1	0	0
Rectification border	2	0	0
Out of ROI	3	0	0
Static	4	0	0
Dynamic	5	0	0
Ground	6	0	0
Road	7	1	1
Sidewalk	8	2	1
Parking	9	0	1
Rail track	10	0	1
Building	11	3	2
Wall	12	4	2
Fence	13	5	2
Guard rail	14	0	2
Bridge	15	0	2
Tunnel	16	0	2
Pole	17	6	3
Polegroup	18	0	3
Traffic light	19	7	3
Traffic sign	20	8	3
Vegetation	21	9	4
Terrain	22	10	4
Sky	23	11	5
Person	24	12	6
Rider	25	13	6
Car	26	14	7
Truck	27	15	7
Bus	28	16	7
Caravan	29	0	7
Trailer	30	0	7
Train	31	17	7
Motorcycle	32	18	7
Bicycle	33	19	7
License plate	-1	0	7

Table B.1: The mapping between original class IDs and IDs used for training and evaluation in CityScapes and CityScapesCat.

Category name	Category ID
Void	0
Flat	1
Construction	2
Object	3
Nature	4
Sky	5
Human	6
Vehicle	7

Table B.2: CityScapes category names.

Andreas Hammer Håversen

QT-UNet

NTNU
Norwegian University of
Science and Technology