TU WIEN Informatics

# Few Shot Semantic Segmentation on the Fly

## Using Low-Rank Adaptation in Visual Foundation Models

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Data Science**

eingereicht von

**Martin Miesbauer, BSc**
Matrikelnummer 11901956

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof. Doctor Pedro Hermosilla Casajus

Wien, 5. Mai 2025

| Martin Miesbauer | Pedro Hermosilla Casajus |
|---|---|

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

# Few Shot Semantic Segmentation on the Fly

## Using Low-Rank Adaptation in Visual Foundation Models

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

**Diplom-Ingenieur**

in

**Data Science**

by

**Martin Miesbauer, BSc**
Registration Number 11901956

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Doctor Pedro Hermosilla Casajus

Vienna, May 5, 2025

| Martin Miesbauer | Pedro Hermosilla Casajus |
|---|---|

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

# Erklärung zur Verfassung der Arbeit

Martin Miesbauer, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 5. Mai 2025

Martin Miesbauer

# Danksagung

Ich danke meiner Familie, die mir das Studium ermöglicht haben. Meinen Freunden, die mir die Zeit erträglich gemacht haben, besonders Larissa für das kostenlose Essen.

# Acknowledgements

Calculations were performed using supercomputer resources provided by the Vienna Scientific Cluster (VSC).

# Kurzfassung

Few-Shot Semantic Segmentation (FSS) ist eine Aufgabe in der Computervision, bei der neue Objektklassen mit nur wenigen Beispielen segmentiert werden. Dabei wird jeder Pixel als Vorder- oder Hintergrund markiert. Diese herausfordernde, aber entscheidende Aufgabe ist besonders in Bereichen interessant, in denen keine großen Datensätze zur Verfügung stehen. Diese Diplomarbeit untersucht den Einsatz von Low-Rank Adaptation (LoRA), einer Technik aus Sprachmodellen, um große Visual Foundation Models (VFMs), insbesondere DINOv2, an eine neue Klasse anzupassen.

Wir implementieren eine FSS-Pipeline, in der ein binärer Segmentierer mit den segmentierten Beispielen trainiert wird. Dabei werden Matrizen mit niedrigem Rang in ausgewählte Schichten von DINOv2 injiziert und trainiert, anstatt die bestehenden Parameter von DINOv2 zu verändern. Für die Segmentierung verwenden wir eine einfache lineare Abbildung.

Wir führen Experimente auf drei etablierten FSS Benchmarks, PASCAL-$5^i$, COCO-$20^i$ und FSS-1000, durch und analysieren sowohl quantitative Metriken, wie mIoU und FB-IoU, als auch die qualitative Segmentierungsqualität. Die Ergebnisse zeigen, dass wir viele bestehende Modelle, insbesondere hinsichtlich der Generalisierungsfähigkeit übertreffen konnten. Zwar konnten wir nicht alle State-of-the-Art-Modelle schlagen, aber insbesondere beim Benchmark FSS-1000 sind wir sehr nahe gekommen.

Eine Ablationsstudie zeigt, dass nur wenige Transformationen mit LoRA-Matrizen vom Rang 2 angepasst werden müssen, um die besten Ergebnisse zu erzielen.

Diese Arbeit zeigt, dass LoRA eine effektive Strategie ist, um VFMs an neue Segmentierungsaufgaben anzupassen und somit Few-Shot Learning auch in ressourcenbeschränkten Umgebungen zu ermöglichen.

# Abstract

Few-Shot Semantic Segmentation (FSS) aims to segment novel object classes using only a handful of labeled examples, a challenging yet critical task in domains where large-scale annotated datasets are unavailable. This thesis explores the application of Low-Rank Adaptation (LoRA) to enable efficient FSS using large-scale Visual Foundation Models (VFMs), in particular DINOv2.

We propose an FSS pipeline in which a binary segmenter is trained using the labeled examples. We inject trainable low-rank matrices into selected layers of DINOv2 and train these layers instead of the existing parameters. We use a simple linear pixel-wise classification head.

We perform extensive experiments on three established FSS benchmarks, PASCAL-$5^i$, COCO-$20^i$ and FSS-1000, evaluating the quantitative metrics mIoU and FB-IoU, as well as the qualitative segmentation performance. Our results outperform many existing models, particularly in terms of generalization, although we did not outperform all state-of-the-art models, but came close in the FSS-1000 benchmark.

We present an ablation study which shows that only a few transformations need to be adapted using rank 2 low-rank matrices to achieve the best results.

This work demonstrates that LoRA provides an effective strategy for adapting VFMs to new segmentation tasks, enabling easy few-shot learning in resource-constrained environments.

# Contents

CHAPTER 1

# Introduction

## 1.1 Motivation

Semantic image segmentation (SiS) is a computer vision task that consists of assigning a class label to each individual pixel in an image. It has many different use cases, including detection of road signs in driver assistance systems [MBLAGJ+07] and detection of brain tumors [XZF25]. An example of a semantic segmentation result is shown in Figure 1.1.

Traditionally, SiS models, like all other computer vision tasks, have relied on a separate feature extraction step. This consisted mainly of applying hand-crafted algorithms, such as edge detection, corner detection, and threshold segmentation. The difficulty with this approach is that there are a huge number of feature extraction algorithms to choose from. Selecting the right features must be performed by a human expert. These features can then be used as input in a simple model [OCC+20].


Figure 1.1: Example of a semantically segmented image (Figure from [JYL20])

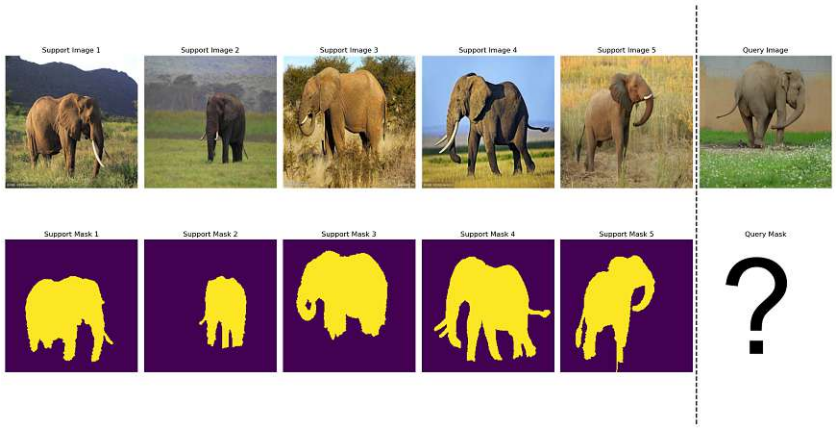



Figure 1.2: An example of an FSS task

In the last decade, almost all state-of-the-art (SOTA) methods for SiS have used deep learning (DL). In this technique, the feature extraction step is incorporated into the model. These models typically have many layers and trainable parameters that are learned by using images and their pixel-wise class labels as training data [HZG20].

DL methods for SiS have their own limitations. They require massive labeled datasets for training. These datasets can be time consuming and expensive to produce. For example, creating the MS COCO dataset [LMB+14] took over 70,000 worker hours. For some specific domains, such as healthcare and agriculture, large-scale labeled data is scarce or unavailable due to privacy concerns, high annotation costs, or domain-specific expertise requirements [CM24]. Humans, on the other hand, can learn new concepts with just a few examples [CLR+23]. This has inspired a related computer vision task Few-Shot Semantic Segmentation (FSS). The main goal here is to develop models that can learn to segment novel object classes from only a handful of labeled examples. This could reduce the need for large labeled datasets for model training. An example of an FSS task is shown in Figure 1.2. Here, the five labeled examples of an elephant with the label form the support set, which is used by the model to learn the segmentation of the query image on the right.

SOTA models for FSS, such as SegGPT [WZC+23], DACM [XLZ22], or GF-SAM [ZGJ+24], still require large amounts of labeled training data. This makes scaling the models quite challenging, as there is always some human factor involved in labeling the images, even if large amounts of images are available for the specific domains. This raises the question: Can we develop a more efficient approach that enables FSS with minimal labeled data?

## 1.2 Research Problem

In many existing solutions for FSS, training is time consuming, while inference remains fast. This is ideal when we have large amounts of data for both training and inference. This thesis addresses the dependency on large-scale labeled datasets by investigating how to adapt pre-trained Visual Foundation Models (VFMs) using Low-Rank Adaptation (LoRA) to effectively segment images with just five labeled support examples.

The central challenge is to determine if LoRA can provide a parameter-efficient fine-tuning strategy that maintains competitive segmentation performance while reducing the computational and storage requirements compared to full fine-tuning. LoRA injects trainable low-rank matrices into the model and only changes their parameters during fine-tuning. This technique has proven beneficial in the domain of large language models (LLMs) [HSW+22].

## 1.3 Research Objectives

The main objective of this thesis is to develop and validate a LoRA-based fine-tuning approach for VFMs to enable efficient and effective FSS using only a few labeled examples.

Specifically, we want to answer the following research questions:

1. What are the highest mean Intersection over Union (mIoU) and foreground-background IoU (FB-IoU) scores we can achieve in established FSS benchmarks using LoRA in VFMs? How does this compare to a baseline and SOTA models?
2. How much memory and computational savings does LoRA in VFMs provide compared to full fine-tuning?
3. What is the optimal configuration for LoRA in VFMs?
4. How does transferring learned knowledge from one benchmark to another affect performance, and how does it compare to SOTA models?

We have the following goals for this thesis:

- Ensure a competitive accuracy, aiming for segmentation results within 1% of SOTA models.
- Demonstrate the benefit of using LoRA, aiming for at least a 1% improvement over the baseline.
- Ensure parameter-efficient fine-tuning, aiming for at least a 50% reduction in memory and computation time compared to full fine-tuning.

By achieving these objectives, we hope to demonstrate that LoRA can significantly reduce the amount of labeled data required for FSS, making it more accessible to domains with limited data availability.

## 1.4 Structure of the Thesis

Chapter 2 provides an overview of the background for this thesis. It includes related work for FSS, including the definition of the task, existing models, benchmarks, and evaluation metrics. It also covers VFMs, and an overview of LoRA. Since we have very limited training data, we also cover techniques for regularization to avoid overfitting and hyper-parameter tuning. Chapter 3 will cover our FSS implementation, and our training and adaptation strategy. Chapter 4 contains the experimental setup and the results of these experiments. In addition, we perform an ablation study on the LoRA configurations and a qualitative analysis of the resulting segmentation masks. We also present the changes in the DINOv2 results caused by LoRA. Chapter 5 discusses the main findings, their strengths and limitations, and directly answers the research questions. In Chapter 6 we conclude the thesis by summarizing its main contributions and providing a direction for further research.

# CHAPTER 2

# Background

## 2.1 Definition of Few-Shot Semantic Segmentation

The task of FSS was first introduced by Shaban et al. in 2017 [SBL+17]. We refine the original formulation for better clarity. Let the support set be defined as $S = \{(I_S^i, M_S^i)\}_{i=1}^k$, where each pair consists of a support image $I_S^i \in [0, 255]^{3 \times h_i \times w_i}$ and its corresponding binary segmentation mask $M_S^i \in \{0, 1\}^{h_i \times w_i}$. Each mask represents the pixels belonging to a common novel class, i.e. all masks in the support set refer to the same semantic category. Given this support set $S$ and a query image $I_q \in [0, 255]^{3 \times h \times w}$, the goal of FSS is to learn a model $f(I_q, S)$ that predicts a binary mask $\hat{M}_q \in \{0, 1\}^{h \times w}$, that segments the same class as annotated in the support set. Typically, $k$ is a small number, (most commonly $k = 1$ and $k = 5$), reflecting the few-shot setting. The tuple $(S, I_q, M_q)$, where $M_q$ is the true segmentation mask of $I_q$, is called an episode.

## 2.2 Benchmarks and Evaluation Metrics

Several benchmarks have been proposed since the inception of FSS. Shaban et al. [SBL+17] introduced Pascal-$5^i$, the first widely used benchmark. It was created using images and annotations from PASCALVOC 2012 [EVGW+] and extended annotations from SDS [HAGM14]. It contains 20 different semantic classes divided into 4 folds.

A more challenging benchmark is COCO-$20^i$ by Nguyen and Todorovic [NT19] based on the MSCOCO [LMB+14] dataset. It contains images that contain one or more of 80 different classes. The classes are again divided into 4 folds to ensure testing on unseen classes.

The FSS-1000 benchmark by Li et al. [LWC+20] contains 1000 different classes, divided into 760 classes for training and 240 classes for testing. They created the dataset from scratch to ensure a diverse set of different classes.

A complete list of all classes in each fold can be found in Appendix 6.2.

All benchmarks typically use the mIoU metric, while some models also report the FB-IoU metric [CLR+23]. The mIoU metric can be calculated as

$$\text{mIoU} = \frac{1}{C}\sum_{c=1}^{C}\text{IoU}_c, \tag{2.1}$$

where $\text{IoU}_c$ is the ratio of true positive classified pixels for class $c$ to the number of false positive classified pixels plus false negative classified pixels. The FB-IoU metric ignores the different classes and calculates the mean IoU of the foreground and background classes. This calculation can be written as

$$\text{FB-IoU} = \frac{1}{2}\sum_{c=1}^{2}\text{IoU}_c. \tag{2.2}$$

## 2.3 Few Shot Semantic Segmentation Models

There have been many different approaches to solving the FSS problem. We will only highlight the major contributions that were either once SOTA or are currently best performing models. These models are roughly ordered by the publication date of the corresponding paper. Most approaches use a backend, which is a learned feature extractor applied directly to the image. This can reduce the resolution and increase the dimensionality of the image. The output of the backend is referred to as the embedding of the image in the feature space. The backend can be trained directly on the data, or it can be pre-trained on some other dataset. In this context, training refers to adjusting the model weights using a set of training episodes, while inference refers to applying the model to testing episodes.

The first FSS model, One-Shot Learning for Semantic Segmentation (OSLSM), was introduced by Shaban et al. in 2017 [SBL+17]. They formulate the task as a two-branch network. The first branch generates classifier parameters from the support image-mask pairs, and the second branch applies these parameters to segment the target object in a query image.

The main idea of Similarity Guidance Network for One-Shot Semantic Segmentation (SG-One) by Zhang et al. [ZWYH20] was to leverage the pixel-wise similarities between the support and the query images. They still use a two-stage design, but incorporate shared weights. The Guidance Branch first uses several convolution layers, then it averages the values of all pixels where the desired object is present. This step is called Masked Average Pooling (MAP). This vector represents the target object in the support images and is multiplied with the query image to compute the cosine similarity. The Query Branch applies $3 \times 3$ convolution layers and multiplies the similarity matrix computed in the first

step with the result. Models that use a vector representing the target class are sometimes called to as prototypical networks.

Class-Agnostic Segmentation Network (CANet) by Zhang et al. [ZLL+19] proposes two new ideas. First, they create a dense comparison between the support and query images. To do this, they apply a ResNet [HZRS16] with shared weights to the query and support images. Then the MAP is computed for the support images, this vector gets then up-sampled and concatenated to the output of the query image. The output is then fed through an iterative optimization module. Here, the predicted masks are refined step by step. Zhang et al. [ZXQ21] showed that CANet and other models can be optimized using Self-Guided and Cross-Guided Learning (SCL). These improve the vector representing the object by using the provided support mask as a guide and comparing it with a predicted support mask.

Prototype Alignment Network (PANet) by Wang et al. [WLZ+19] uses a simple model with an interesting training approach. They implement a basic prototype network with a backbone, MAP and cosine similarity. They then switch the roles of the support and query sets and use the predicted masks of the query image get used as an input to predict the original query masks. This aligns the support and query prototypes.

Feature Weighting and Boosting (FWB) by Nguyen and Todorovic [TZS+22] contributes two novel ideas to the field of FSS. First, the features calculated by the backend are weighted according to their relevance in the support masks. The second contribution consists of multiple vector representations of the object, which are gradually improved during testing. For each of these representations, the accuracy on the support masks is calculated, and a weighted average is used as the final prediction for the query set.

Yang et al. introduced Local Transformation Module (LTM). Here, they use a transformer to obtain weights from the support images and masks. These weights are then spatially applied to the output of the query image.

Zhang et al. proposed Cycle-Consistent Transformer (CyCTR). This method uses transformer blocks to aggregate information from the support images into the query image. They introduce a special kind of attention using cycle-consistency.

Part-aware Prototype Network (PPNet) by Liu et al. [LZZH20] was the first model that tried to decompose the prototypes into several different prototypes. Each of these prototypes can be aware of different parts of the object. In addition, they use unlabeled data to enrich these prototypes.

Prior Guided Feature Enrichment Network (PFENet) by Liu et al. [TZS+22] uses training-free priors by calculating the correlation between the representations of the query image and masked support images. These priors are then used in addition to the original representations of the query image in the final classification.

Region Proportion Regularized Inference (RePRI) by Boudiaf et al. [BKM+21] optimizes a new loss instead of the traditional cross-entropy loss.

Mining Latent Classes (MLC) by Yang et al. [YZQ+21] re-annotates the training images in a first step, by using classes not present in the original masks.

Hypercorrelation Squeeze Network (HSNet) by Min et al. [MKC21] uses the correlation of the features at different levels between each pixel of the query image features and the masked support images features. These 4D tensors of pixel-wise correlations are then iteratively squeezed into segmentation masks. Moon et al. [MSZ+22] improved HSNet and other models by modifying the masked support images. HSNet masks the features of the support image, but there may be more information at the edges. Therefore, Hybrid Masking (HM) takes the features of the masked support image if the pixel is inactive in the masked feature image.

Johnander et al. proposed Dense Gaussian Process Network (DGPNet) [JEF+22], which uses a Gaussian approach. They estimate the probability distribution of the support masks given the support images, derive a function between the support images and the support masks, and apply this mask to the query image.

Hong et al. [HCN+22] approach FSS with Volumetric Aggregation with Transformers (VAT). They reformulate the FSS task as a semantic correspondence task, where the goal is to find corresponding points in semantically similar images.

Dense Cross-query-and-support Attention weighted Mask Aggregation (DCAMA) by Shi et al. [SWZ+22] treats each pixel in the query image as a token and calculates the pixel-wise correlation with all support pixels. The query pixels are then labeled according to the average of the support pixels, weighted with how similar they are.

Xiong et al. [XLZ22] propose Doubly Deformable Aggregation of Covariance Matrices (DCAM). They use Gaussian processes to obtain covariance matrices from the pixels of the query image to the pixels of the support images. These are then passed through a transformer to obtain the segmentation mask.

Zhang et al. [ZSYC22] propose Feature-Proxy Transformer (FPTrans). They revive a classical approach that combines a complicated backend feature extractor with a simple linear classifier. First, they generate class-aware prompts from the support images representing the foreground and several background objects. They then use these prompts, the support images, and the query image as input to a transformer. Proxies are calculated from the output of the support images and used as the weights in a classification head on the output of the query image.

Painter by Wang et al. [WWC+23] is a general-purpose model that can be used in various different computer vision tasks, including FSS. They reformulate all tasks as an image inpainting problem, where each pixel of the image is recolored according to some properties depending on the task. During training, they used training images and the corresponding inpainted image as an input and had the model reproduce masked patches of the inpainted image. During inference, both the support image and the query image are passed through the transformer. The support masks are treated as existing patches of the inpainting and the query mask is treated as missing. The task of FSS is then to

guess the missing patches of the query mask. They further improved upon this concept by proposing **seg**ment everything with a **g**eneralist **P**aint**t**er (SegGPT) a generalist model capable of only segmentation tasks [WZC+23]. They further improved upon Painter by using a random coloring scheme.

Liu et al. proposed Matcher [LZL+24]. This training-free approach uses a powerful image encoder DINOv2 [ODM+23] to encode the masked support images and query image. It then calculates a similarity matrix between the features to identify similar patches in the images. Points with high similarity scores are used as an input for SAM [KMR+23], a foundation model in the field of SiS whose predictions are aggregated for the final prediction.

Prior Guided Mask Assemble Network (PGMA-Net) by Chen et al. [CMZ+24] uses the multi-modal Foundation Model (FM) CLIP [RKH+21] to extract features at different fidelity levels. Then the class specific features are transformed into a class agnostic prior in the form of a probability map. In addition, the pixel to pixel correlation is calculated with an Affinity Extractor. These two results are then assembled into different masks, which are then decoded to obtain the final prediction.

Zhang et al. propose Graph-based Few-shot Segment Anything Semantically (GF-SAM) [ZGJ+24]. They again use the DINOv2 backbone, calculate the pixel-wise correlation between query and support images, they divide the correlation matrix into a part containing the object and a part containing the background based on the support mask. They then algorithmically select prompt points in the query image based on the similarities to the support images. These points are then used to generate masks with SAM [KMR+23]. They then cluster the masks based on which points they contain, and take the union over the masks of each point in the clusters. In a third step, false positives are minimized by first calculating the ratio of positive to negative pixels in the mask obtained from the prompt point. Another way to reduce false positives is to consider self-consistency. The model assesses the similarity between the sample points and the features of the corresponding masks. Finally, the masks of the points that pass both tests are merged to form the final prediction.

The SOTA models have become very complex recently, which makes using them as a stepping stone for new advancements more difficult.

## 2.4 DINOv2

DINOv2 [ODM+23] is a family of large VFMs consisting of up to 1 billion parameters. The term FM was first introduced by Bommasani et al. [BHA+21] and refers to a class of models trained on broad data that can be adapted to a variety of different downstream tasks, by using some or all layers of the FM as a backbone and implementing a custom head for the specific task. During training, the backbone is usually frozen, and only the head is adapted. FMs are widely used for Natural Language Processing (NLP), the most prominent examples are BERT [DCLT19], LLAMA [TLI+23], and GPT-3 [BMR+20].

FMs are typically trained using self-supervised learning, a method that reduces the need for labeled examples. These models benefit from their large size and can have billions to trillions of parameters. For example, the successor to GPT-3, GPT-4 [OAA+24] is rumored to have about 1.8 trillion parameters[AYF+24], although the exact number is unknown.

DINOv2 was implemented using the Vision Transformer (ViT) architecture [DBK+21] and trained from scratch using self-supervised learning. A simple overview of this architecture is shown in Figure 2.1. The image is divided into patches of size $14 \times 14$ (instead of the typical $16 \times 16$ for ViT), these patches as well as an additional learnable class token are then embedded into a larger feature space by a linear transformation. The embeddings are then passed through $n$ transformer blocks represented by the dashed outline in the figure. The most important part of the transformer blocks is the Multi-head attention layer. An overview of this layer is shown in Figure 2.2. The same input is transformed into the query, key, and value matrices via linear transformations, which we will refer to as the Q/K/V-transformations respectively. At the end of the layer, the intermediate vectors (called Context Vectors in Figure 2.2) get concatenated and transformed to the output vector (called Context Vector in Figure 2.2) via a linear transformation. We will refer to this transformation as the O-transformation, to be able to distinguish it from the other linear transformations in the attention layer. The Scaled Dot-Product Attention is a function with no learnable parameters. It is defined as

$$\text{Attention(Q,K,V)} := \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V,$$

where $d$ is the dimension of the feature space.

The largest DINOv2 model ViT-g/14 was trained using a combination of multiple loss functions. The image-level objective consists of gaining a vector representing the whole image ($\text{Rep}_{<\text{class}>}$ in Figure 2.1). This was improved by sampling two different crops from the same image and aligning the outputs. The patch-level objective consists of gaining vectors representing each patch of the image. This was achieved by masking random patches in images and aligning the output with the output to the full image. The model was then frozen, and smaller models were trained using knowledge distillation [HVD15], meaning that the smaller student-model attempts to replicate the output of the larger teacher-model.

## 2.5 Low-Rank Adaptation

LoRA, originally developed for LLMs by Hu et al. [HSW+22], is a technique to reduce the number of trainable parameters in a model for tuning to a specific task, while maintaining accuracy comparable to full fine-tuning. It consists of injecting trainable low-rank matrices of rank $r$ into some or all parameter matrices of the model. Figure 2.3 shows the data flow. With $x \in \mathbb{R}^d$, $W \in \mathbb{R}^{d \times d}$, we have $W_A \in \mathbb{R}^{r \times d}$ and $W_B \in \mathbb{R}^{d \times r}$.

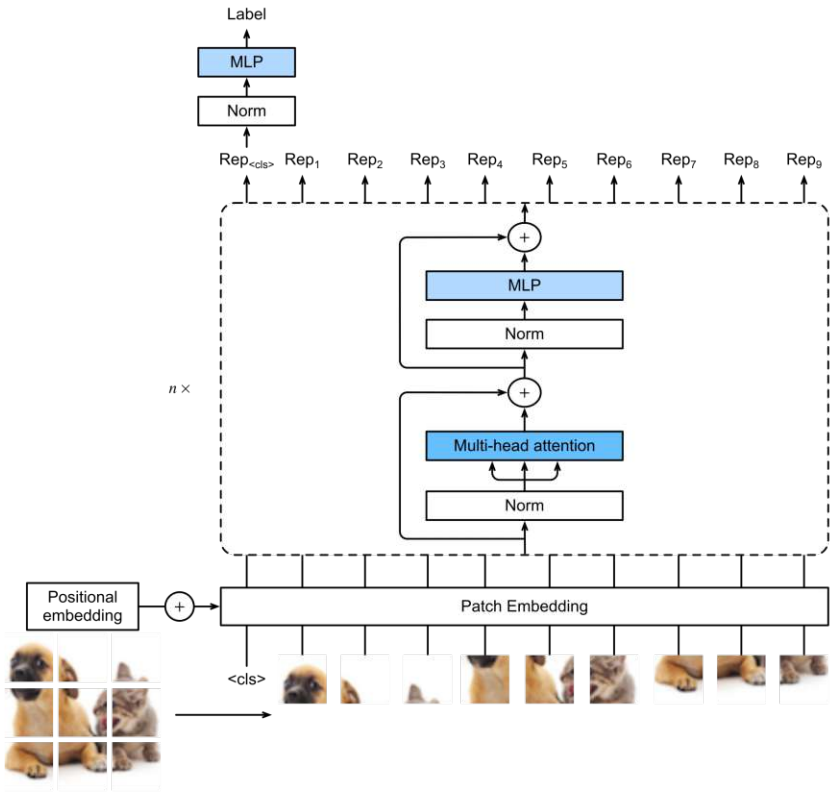



Figure 2.1: Overview on the ViT architecture (Figure from [Com24])

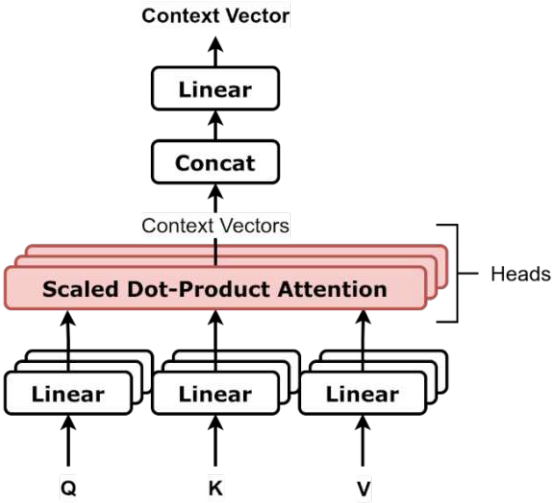



Figure 2.2: Schematic of a typical attention layer used in Transformer models (Figure from [Com25])

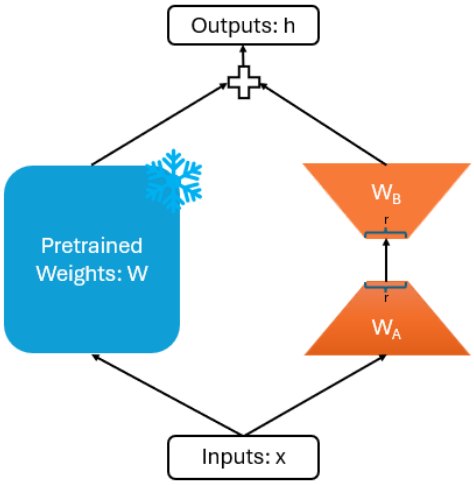



Figure 2.3: Schematic Decomposition in low rank matrices

During training the outputs $h$ are calculated as follows

$$h = W \cdot x + (W_B \cdot W_A) \cdot x,$$

where $W$ is frozen. During inference, we can calculate a new weight as follows

$$W = W + W_B \cdot W_A$$

and use it as usual, which does not add any computation time.

## 2.6 Regularization Methods

Since we want to adapt DINOv2 directly, we have to work with many parameters. Even though the use of LoRA drastically reduces the number of parameters, we still have thousands of trainable parameters (see Table 4.3). Due to the nature of the FSS task, we only have a few support images per episode. This drastically increases the risk of overfitting, i.e. the model performs much better on the training set than on the test set. Therefore, we need regularization techniques to combat this. There are a some conflicting definitions of regularization, we will use the generous definition of Kukačka et al. [KGC17] "Regularization is any supplementary technique that aims at making the model generalize better, i.e. produce better results on the test set".

Weight decay is a way to penalize large parameters in the model. This is done by adding a quadratic cost function to the loss function. D'Angelo et al. [DAAVF24] have shown the effects of weight decay for Deep Learning models.

Another regularization technique is dropout [SHK+14]. During training, the parameters of the hidden layers are randomly set to zero with a fixed probability. Liu et al. have shown that dropout can even reduce underfitting in certain scenarios [LXJ+23].

Another technique, typically used when the data is scarce, is data augmentation [MM22]. There are many different ways to augment the input data. This can be done by manually specifying image transformations or by using automated methods. One such automated method is RandAugment [CZSL20]. This randomly applies one or multiple random augmentations from a pool of different augmentations with an adjustable strength. They have shown comparable improvements to more complicated augmentation strategies with little impact on performance.

Another technique to improve learning ability is a stochastic gradient descent, which adjusts the parameters of the model after a batch of examples. This often finds better solutions than adjusting the parameters after every example or after all examples [KLY18].

## 2.7 Hyper-parameter Tuning

Since we need to fit large models quickly and have few labeled examples, there are certain hyper-parameters that drastically affect the accuracy of our model. These hyper-parameters exist in all DL models, but since we include many different regularization methods, the number of hyper-parameters increases dramatically.

Hyper-parameters are any parameters in a machine learning (ML) model that cannot directly be estimated from the training data by gradient descent. They must be set before the ML model is trained. To increase the performance of the ML model, many different combinations of hyper-parameters must be explored. This is time consuming and computationally expensive [YS20]. The simplest technique is Babysitting. This method is 100% manual tuning and requires a mixture of experience, guesswork, and analysis of previous results. This becomes infeasible with a large number of hyper-parameters. Another simple technique is grid search. It is an exhaustive search of the hyper-parameter space in a given grid. It will find an optimum in the grid, but also becomes infeasible with a large number of hyper-parameters. A more efficient way for hyper-parameter tuning was introduced by Bergstra and Bengio [BB12]. In this technique, a fixed number of hyper-parameter combinations are sampled from the search space and evaluated. They showed that this approach can lead to similar results with a drastically reduced computation time. An improvement to random sampling is to use Bayesian optimization methods. Here, the loss value of previous searches is used to select the next hyper-parameters to try. A commonly used method is the tree-structured Parzen estimator [Wat23]. This method has won several hyperparameter optimization competitions and can be used inside of frameworks like Optuna [ASY+19].

# CHAPTER 3

# Our Few-Shot Segmenter

## 3.1 Segmentation Module

The core of our Few-Shot Segmenter is our segmentation module. An overview of this is shown in Figure 3.1.

The first step in the segmentation module is to upsample the input image. This tries to counteract the aggressive downsampling performed by the DINOv2 model in the next step, by increasing the resolution by a certain factor. This also has the disadvantage of increasing the inference time of DINOv2, and it's memory usage. Therefore, there is a tradeoff between an increased resolution of the embeddings and fast inference. By setting the up-sampling factor to one, this step can be disabled.

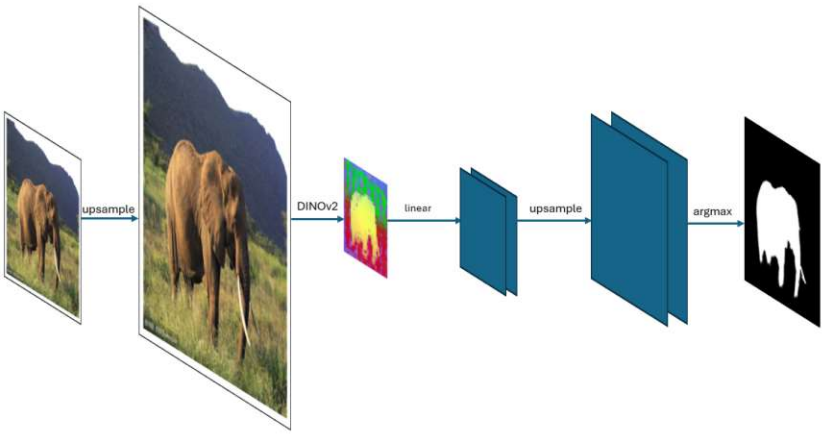



Figure 3.1: Overview of our segmentation module. Note that the first up-sample step is optional.

Next, the upsampled image is passed through the DINOv2 visual foundation model, discussed further in Section 2.4, to obtain an embedding of the input image in the feature space. We then apply a linear transformation to the embeddings with an output dimension of two, which is then gets upsampled back to the original resolution of the input image. We now have a channel dimension of two. We interpret these as values for the foreground and the background of the image. The binary segmentation mask is then calculated as the argmax of these two channels during inference.

## 3.2 Few-Shot Semantic Segmentation Pipeline

The main role of our FSS pipeline is to train a segmentation module for the given FSS episode $(S, I_q)$. We treat the support set $S$ as our training set and train our segmentation module only on these images. First, we freeze the DINOv2 model and train the linear layer for a few epochs. Then, we inject LoRA layers into certain transformations of the attention layers in DINOv2 and train them in addition to the head again for a few epochs. To increase the ability to generalize to the query image, we perform random augmentation of the training set during all epochs and do not use all training samples for each epoch.

## 3.3 Training Strategy

Since all training of the model parameters is done at inference time, we only use the training data to optimize the hyper-parameters of the model using an automated hyper-parameter tuning approach. First, we find the optimal hyper-parameters for the first step of the segmentation pipeline according to the mIoU score on 100 training episodes. To optimize the hyper-parameters for the second stage we iterate over $r$ values of $1, 2, 4, 8$ and layer options $Q$-Transformation, $Q$- and $V$-Transformation and all Transformation layers in the attention layers of DINOv2 for LoRA adaptation of DINOv2. For each of these iterations, we again find the rest of the hyper-parameters using automated hyper-parameter tuning.

During testing, we select the hyper-parameters and LoRA configuration that performed best according to the mIoU score on the training set to use for the inference during testing.

## 3.4 Implementation

Everything was implemented with Python 3.11.3. The model was implemented with PyTorch 2.6. Image augmentation was performed using Kornia 0.8 done with the RandomAugment method. We chose Kornia, because it makes it easy to augment the images and masks simultaniously, ensuring that the same geometric augmentation is applied to both. The first upsampling in the segmentation module was implemented

with a simple nearest-neighbor upsampling method. The upsampling for the foreground-background values was performed using bilinear interpolation to get smoother masks. Automated hyper-parameter tuning was performed using the tree-structured Parzen estimator [Wat23] in Optuna 4.2.1. Our source code, the training and the testing scripts can be accessed at `https://github.com/miesbauerm/FSSLoRA`. As the weights are only trained at inference time, we do not provide them, although the hyper-parameters can be found in our repository.

CHAPTER 4

# Experiments and Results

## 4.1 Dataset Preparation

We perform our experiments for FSS on three established benchmarks to enable easier comparison with other methods and models.

### 4.1.1 PASCAL-$5^i$

We followed the original description by Shaban et al. [SBL+17] to create the PASCAL-$5^i$ dataset. We obtained the images and annotations from PASCALVOC 2012 [EVGW+] and additional annotations from SDS [HAGM14]. We removed all images included in the PASCALVOC 2012 validation set from the SDS training set, as there is some overlap. For each fold $i = 0, \ldots, 3$ we use the classes in the corresponding column of Table 1 in the Appendix as the test label-set $L_{\text{test}}$ and the remaining classes as the training label-set $L_{\text{train}}$. The masks in PASCALVOC 2012 and SDS can contain several different classes. These are stored as categorical values in matrix form.

We create a training set $D^1_{\text{train}}$ by selecting all image-mask pairs from the PASCALVOC 2012 and SDS training sets that contain at least one pixel in the semantic mask from the label-set $L_{\text{train}}$. We remove all pixels in the masks that do not belong to $L_{\text{train}}$ and treat them as background. We create $D^1_{\text{test}}$ in a similar way, but select only image-mask pairs from the PASCALVOC 2012 validation set. To get the final training set $D_{\text{train}}$, we sample 100 episodes in the following way: First, we uniformly sample an image-mask $(I_q, M_q)$ pair from $D^1_{\text{train}}$. Then we uniformly sample a class $l \in L_{\text{train}}$ from the classes present in $M_q$ and use it to create a binary mask $M_q(l)$. We then sample 5 support images from $D^1_{\text{train}} \setminus \{I_q, M_q\}$ with class $l$ present in the mask. We purposefully restrict the number of episodes in our training set to ensure that we can perform the hyper-parameter tuning in reasonable time. To obtain the benchmark $D_{\text{test}}$ we follow the same procedure as

above, selecting only images from the PASCALVOC 2012 validation set, but sample 1000 episodes from $D^1_{\text{test}}$, to make our results comparable to previous models.

### 4.1.2 COCO-$20^i$

The benchmark COCO-$20^i$ by Nguyen et al. [NT19] is created in a very similar way as PASCAL-$5^i$. The base segmentation data used here comes from the MSCOCO [LMB+14] dataset. This contains 80 different classes, divided into 4 different folds, shown in Table 2 in the Appendix. The masks of different objects may overlap in the MSCOCO dataset, but we only look at individual classes. For each fold $i = 0, \ldots, 3$ we again use the classes in the corresponding column of Table 2 as the test label-set $L_{\text{test}}$ and the remaining classes as the training label-set $L_{\text{train}}$.

We create the training set $D^1_{\text{train}}$ by selecting all images from the MSCOCO training set that have a non-empty mask for a class in $L_{\text{train}}$. Masks for classes in $L_{\text{test}}$ are removed. Similarily, the test set $D^1_{\text{test}}$ is formed by selecting all images from the MSCOCO validation set that have at least one non-empty mask for classes in $L_{\text{test}}$. Again, we remove masks for classes in $L_{\text{train}}$. To get $D_{\text{train}}$ and $D_{\text{test}}$ we follow the exact same steps as for the creation of PASCAL-$5^i$.

### 4.1.3 FSS-1000

The FSS-1000 benchmark defined by Li et al. [LWC+20] consists of 1000 different classes. It contains 10 image-mask pairs per class and has to be downloaded from the Google Drive folder of the developers. The train-test split of the classes is fixed and includes 240 classes in the test set. We had to manually add one mask the original creators seem to have overlooked. We did not find any other paper which pointed out this error, but this minor modification to the dataset should not change the interpretation of the results. Figure 4.1 shows our mask. There are also some images which do not have a resolution of $224 \times 224$, but these are rescaled when the benchmark is created. The images show exactly one object and are balanced.

To create the episodes for $D_{\text{train}}$, we uniformly sampled 100 classes from the classes not included in $L_{\text{test}}$. For each class, we sampled 6 images without replacement from the images belonging to the class, and selected one randomly as the query image. The other 5 images form the support set. To create $D_{\text{test}}$, we used each image belonging to a class in $L_{\text{test}}$ once as a query image and sampled 5 support images without replacement from the remaining 9 images belonging to that class.

## 4.2 Experimental Setup

All experiments in this chapter were performed on a half node of the Vienna Scientific Cluster (VSC). The processor used was an AMD 7713 with an NVIDIA A100 GPU and 256 GB of RAM. For the backbone, we chose the smallest model in the DINOv2 family, ViT-S/14.

(a) image (b) mask

Figure 4.1: Peregine falcon with missing mask and our mask

| name | type | min | max | log |
|---|---|---|---|---|
| lr | float | 1e-5 | 1 | true |
| weight_decay | float | 1e-3 | 100 | true |
| dropout | float | 0 | 0.5 | false |
| n_epochs | int | 1 | 10 | false |
| mini_batch_size | int | 1 | 5 | false |
| augment_number | int | 0 | 13 | false |
| augment_strength | int | 1 | 29 | false |

Table 4.1: Hyper-parameter options for the first training stage. “lr” and “weight_decay” are the learning rate and weight decay for the AdamW optimizer used to train our segmenter for each episode. “dropout” is the dropout probability for the image embeddings provided by DINOv2. “n_epochs” represents the number of epochs for the initial training. During each epoch, we select only “mini_batch_size” support images and augment them with “augment_number” of different random augmentations. If the number of augmentations is 0 then no augmentation is performed. “augment_strength” sets the strength of the augmentations.

### 4.2.1 Main tests

We performed the following steps for each fold of PASCAL-$5^i$ and COCO-$20^i$ and for the single FSS-1000 fold: We first used a tree-structured Parzen estimator [Wat23] to select the hyper-parameter combination from the options shown in Table 4.1 that gives the best mIoU performance on $D_{\text{train}}$. We stopped this optimization process after 100 runs.

We then took the hyper-parameters from the best run and used them to pre-train the linear layer in our segmentation module for the second step. We manually tested all r-values of 1,2,4, and 8, as well as and $Q$, $QK$ and $QKVO$ transformations to adapt with

| name | type | min | max | log |
|---|---|---|---|---|
| lr | float | 1e-6 | 1e-3 | true |
| weight_decay | float | 1e-3 | 100 | true |
| dropout | float | 0 | 0.5 | false |
| n_epochs | int | 1 | 10 | false |
| mini_batch_size | int | 1 | 5 | false |
| augment_number | int | 0 | 13 | false |
| augment_strength | int | 1 | 29 | false |

Table 4.2: Hyper-parameter options for the second training stage

| | | transformations | | |
|---|---|---|---|---|
| | | q | qv | qkvo |
| r | 1 | 10,001 | 19,217 | 37,649 |
| | 2 | 19,217 | 37,649 | 74,513 |
| | 3 | 37,649 | 74,513 | 148,241 |
| | 4 | 74,513 | 148,241 | 295,697 |

Table 4.3: Number of learnable parameters depending on LoRA configuration

LoRA. These options were inspired by the original LoRA paper [HSW+22]. We applied LoRA layers based on these settings and again used a tree-structured Parzen estimator to optimize the hyper-parameters seen in Table 4.2 and two other parameters, namely "lora_alpha" a modifier of the learning rate for the LoRA layers only, and "lora_dropout" with float values from 0.1 to 10 and 0 to 0.9 respectively, in 50 runs. The number of trainable parameters is shown in Table 4.3. Note that the linear layer in the segmentation head has 770 trainable parameters and the DINOv2 model ViTS-14 has over 22 million trainable parameters.

We then selected the LoRA configuration with the best mIoU score on $D_{\text{train}}$ and the automatically optimized hyper-parameters and performed the FSS task on the $D_{\text{test}}$ dataset and calculated the mIoU and FB-IoU metrics.

For the PASCAL-$5^i$ and COCO-$20^i$ benchmarks, we could not perform upsampling on the input images in the first step of the segmentation module due to performance restraints. For FSS-1000 we kept the upsampling factor at 1 for training and increased it to 2 and 3 for testing.

### 4.2.2 Baseline Comparison

To see the effect of the LoRA adaptation, we compared it to two baselines. The first baseline freezes the DINOv2 backbone and only adapts the linear layer to the given FSS task. To do this, we take the optimal configurations from step 1 of the training process for each benchmark and fold. We then calculate the mIoU and FB-IoU scores for $D_{\text{test}}$.

The second baseline is the fully fine-tuned adaptation. For this, we again used a tree-

structured Parzen estimator to find the optimal hyper-parameters for fully fine-tuning the entire model using the hyper-parameter options seen in Table 4.2 in 20 runs. This enables us to get a comparison to using LoRA. Again, we did this for all benchmarks and folds and calculated the mIoU and FB-IoU scores on the $D_{\text{test}}$ set.

### 4.2.3 Computational Effort

We also tested the computation time and performance on a consumer machine with an Intel i7-8700k CPU, an NVIDIA RTX 2070 GPU, and 32 GB of RAM. We selected the optimized hyper-parameters and ran 100 FSS episodes for each benchmark and fold. We recorded the maximum amount of VRAM and time taken for each problem and averaged the results.

### 4.2.4 Domain Shift

An important question for FSS is how well the model adapts to novel classes in new domains. To answer this question, we use the hyper-parameters from the $D_{\text{train}}$ sets of the PASCAL-$5^i$ and apply them to the FSS-1000 $D_{\text{test}}$ set. We performed the FSS task on the test set with no and with two times upsampling. Due to computational limitations, we did not try other domain shifts or higher upsampling factors.

Due to the lack of comparison data, we decided to repeat this experiment with other models. The best model for the FSS-1000 for which we had access to the training weights was VAT [HCN+22]. We took their PASCAL-$5^i$ weights and applied them directly to the FSS-1000 test set.

### 4.2.5 Ablation Study

In this part, we want to find out how the LoRA configurations affect the FSS accuracy. For each of the LoRA configurations tried in Section 4.2.1 we took the optimized hyper-parameter configurations and performed the FSS task on the $D_{\text{test}}$ dataset of the FSS-1000 benchmark. We record the mIoU and FB-IoU metrics. We repeat these tests at no image upsampling and two times image upsampling. Due to computational limitations, we did not perform this ablation study on the PASCAL-$5^i$ and COCO-$20^i$ benchmarks, nor did we try higher upsampling factors.

## 4.3 Quantitative Results

### 4.3.1 Main Results

Table 4.4 shows our results on the 5-shot PASCAL-$5^i$ benchmark. In addition, all results of the models described in Section 2.3 are also shown. For each model, we took the best performing version according to the average score. Where available, we have used the values for tests on the original image size, to make the comparison more fair.

| Model | mIoU% | | | | | FB-IoU% | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Fold-1 | Fold-2 | Fold-3 | Fold-4 | Avg. | Fold-1 | Fold-2 | Fold-3 | Fold-4 | Avg. |
| OSLSM | 35.9 | 58.1 | 42.7 | 39.1 | 43.9 | - | - | - | - | - |
| SG-One | 41.9 | 58.9 | 48.6 | 39.4 | 47.1 | - | - | - | - | - |
| CANet | 55.5 | 67.8 | 51.9 | 53.2 | 57.1 | 74.2 | 80.3 | 57.0 | 66.8 | 69.6 |
| SCL (CANet) | 59.5 | 68.5 | 54.9 | 53.7 | 59.2 | - | - | - | - | 70.7 |
| PANet | 51.8 | 64.6 | 59.8 | 46.5 | 55.7 | - | - | - | - | 70.7 |
| FWB | 54.84 | 67.38 | 62.16 | 55.30 | 59.92 | - | - | - | - | - |
| LTM | 57.9 | 69.8 | 56.9 | 57.5 | 60.6 | - | - | - | - | 74.6 |
| CyCTR | 69.3 | 73.5 | 63.8 | 63.5 | 67.5 | - | - | - | - | 75.4 |
| PPNet | 60.25 | 70.00 | 69.41 | 60.72 | 65.10 | - | - | - | - | - |
| PFENet | 63.1 | 70.7 | 55.8 | 57.9 | 61.9 | - | - | - | - | 73.9 |
| RePRI | 64.5 | 70.8 | 71.7 | 60.3 | 66.8 | - | - | - | - | - |
| MLC | 66.2 | 75.4 | 72.0 | 63.4 | 69.3 | - | - | - | - | - |
| HSNet | 71.8 | 74.4 | 67.0 | 68.3 | 70.4 | - | - | - | - | 80.6 |
| HM (HSNet) | 72.2 | 73.3 | 64.0 | 67.9 | 69.3 | - | - | - | - | 79.7 |
| DGPNet | - | - | - | - | 75.5 | - | - | - | - | - |
| VAT | 73.3 | 75.2 | 68.4 | 69.5 | 71.6 | - | - | - | - | 82.0 |
| DCAMA | 75.7 | 77.1 | 72.2 | 74.8 | 74.9 | - | - | - | - | 82.9 |
| DCAM | 72.7 | 75.3 | 68.3 | 69.2 | 71.4 | - | - | - | - | 81.5 |
| FPTrans | 76.7 | 79.0 | **81.0** | 75.1 | 78.0 | - | - | - | - | - |
| SegGPT* | - | - | - | - | **89.8** | - | - | - | - | - |
| PGMA-Net† | **77.7** | **82.7** | 76.9 | **77.0** | 78.6 | - | - | - | - | **86.9** |
| GF-SAM | - | - | - | - | 82.6 | - | - | - | - | - |
| Ours | 67.3 | 75.5 | 80.0 | 76.9 | 74.9 | **84.4** | **86.6** | **89.3** | **86.9** | 86.8 |

Table 4.4: Results on the PASCAL-$5^i$ benchmark, the best model in each column is highlighted in bold. Models where not each fold was reported individually are indicated by a "-". Models marked with $*$ were trained on the test categories. Models marked with a † down-sampled the images and did not use their original size.

We are in the upper third of results, being the second best model for two of the folds for which detailed results were reported. Our average FB-IoU score is just 0.1 percentage points lower than the best model which reported this score. SegGPT included the novel testing classes in their training data, which defeats the purpose of FSS. Excluding this, we are the fourth best model for PASCAL-$5^i$ in our comparison.

Our results for the 5-shot COCO-$20^i$ benchmark are shown in Table 4.5. In addition, all results for the models described in Section 2.3 are also shown. For each of the models, we have taken the best performing model according to the average score. Where available, we have used the values for tests on the original image size.

We did not perform well on this benchmark. There seem to be only two worse models in terms of our mIoU score. The FB-IoU score looks a bit better for us, here our score is just a bit worse than the SOTA models.

Table 4.6 shows our results on the 5-shot FSS-1000 benchmark. It also shows all results of the models described in Section 2.3. For each of the models, we have taken the best

| Model | mIoU% | | | | | FB-IoU% | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Fold-1 | Fold-2 | Fold-3 | Fold-4 | Avg. | Fold-1 | Fold-2 | Fold-3 | Fold-4 | Avg. |
| FWB | 19.13 | 21.46 | 23.93 | 30.08 | 23.65 | - | - | - | - | - |
| CyCTR | 41.1 | 48.9 | 45.2 | 47.0 | 45.6 | - | - | - | - | - |
| PPNet | 48.88 | 31.36 | 36.02 | 30.64 | 36.73 | - | - | - | - | - |
| PFENet | 38.5 | 38.6 | 38.2 | 34.3 | 37.4 | 51.5 | 65.6 | 65.7 | **64.7** | 61.9 |
| RePRI | 38.5 | 46.2 | 40.4 | 43.6 | 42.1 | - | - | - | - | - |
| MLC | 57.8 | 47.1 | 37.8 | 37.6 | 45.1 | - | - | - | - | - |
| HSNet | 45.9 | 53.0 | 51.8 | 47.1 | 49.5 | - | - | - | - | 72.4 |
| HM (HSNet) | 46.5 | 55.2 | 51.8 | 48.9 | 50.6 | - | - | - | - | 72.9 |
| DGPNet | - | - | - | - | 57.9 | - | - | - | - | - |
| VAT | 44.1 | 51.1 | 50.2 | 46.1 | 47.9 | - | - | - | - | 72.4 |
| DCAMA | 55.4 | 60.3 | 59.9 | 57.5 | 58.3 | - | - | - | - | 76.9 |
| DCAM | 44.6 | 52.0 | 49.2 | 46.4 | 48.1 | - | - | - | - | 71.6 |
| FPTrans | 54.2 | 62.5 | 61.3 | 57.6 | 58.9 | - | - | - | - | - |
| SegGPT* | - | - | - | - | **67.9** | - | - | - | - | - |
| Matcher | - | - | - | - | 60.7 | - | - | - | - | - |
| PGMA-Net† | **55.9** | **65.9** | **63.4** | **61.9** | 61.8 | - | - | - | - | **79.4** |
| GF-SAM | - | - | - | - | 66.8 | - | - | - | - | - |
| Ours | 37.7 | 43.9 | 38.7 | 27.2 | 36.9 | **66.2** | **70.4** | **69.4** | 60.7 | 66.7 |

Table 4.5: Results on the 5-shot COCO-$20^i$ benchmark, the best model in each column is highlighted in bold. Models where not each fold was reported individually are indicated by a "-". Models marked with $*$ were trained on the test categories. Models marked with a † downsampled the images and did not use their original size.

| Model | mIoU% | FB-IoU% |
|---|---|---|
| HSNet | 86.5 | 88.5 |
| HM (HSNet) | 88.0 | 88.5 |
| VAT | 90.6 | - |
| DCAMA | 90.4 | **94.1** |
| DCAM | **91.7** | - |
| Painter | 62.3 | - |
| SegGPT | 89.3 | - |
| Matcher | 89.6 | - |
| GF-SAM | 88.9 | - |
| Ours (1x) | 84.5 | 90.5 |
| Ours (2x) | 87.4 | 92.4 |
| Ours (3x) | 88.1 | 92.7 |

Table 4.6: Results of the FSS-1000 benchmark. The best results in bold. The parenthesis for our model indicate image up-sampling factor

| Tuning | mIoU% | | | | | FB-IoU% | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Fold-1 | Fold-2 | Fold-3 | Fold-4 | Avg | Fold-1 | Fold-2 | Fold-3 | Fold-4 | Avg |
| No | 48.6 | 63.1 | 68.7 | 58.4 | 59.7 | 70.6 | 78.6 | 77.8 | 75.8 | 75.7 |
| Full | 48.3 | 67.3 | 70.8 | 60.7 | 61.8 | 71.3 | 82.3 | 78.8 | 77.7 | 77.5 |
| LoRA | **67.3** | **75.5** | **80.0** | **76.9** | **74.9** | **84.4** | **86.6** | **89.3** | **86.9** | **86.8** |

Table 4.7: Comparison of our model to the baselines for PASCAL-$5^i$ benchmark

| Tuning | mIoU% | | | | | FB-IoU% | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Fold-1 | Fold-2 | Fold-3 | Fold-4 | Avg | Fold-1 | Fold-2 | Fold-3 | Fold-4 | Avg |
| No | **40.7** | 41.6 | 36.3 | **36.6** | **38.8** | **66.3** | 66.4 | 68.1 | **68.4** | 67.3 |
| Full | 38.1 | **44.7** | 37.6 | 33.1 | 38.4 | 65.3 | 69.4 | 68.9 | 66.1 | **67.4** |
| LoRA | 37.7 | 43.9 | **38.7** | 27.2 | 36.9 | 66.2 | **70.4** | **69.4** | 60.7 | 66.7 |

Table 4.8: Comparison of our model to the baselines for COCO-$20^i$ benchmark

| Tuning | mIoU% | FB-IoU% |
|---|---|---|
| No | 77.2 | 86.1 |
| Full | 80.6 | 88.2 |
| LoRA | **84.5** | **90.5** |

Table 4.9: Comparison of our model to the baselines for FSS-1000 benchmark. No up-sampling was used for these values

performing variant according to the average score. We have used the values for testing on the original image size, where available. We notice that the models are all quite close to each other in terms of their scores. At first there is a huge three percentage point increase in the score with the image upsampling, this effect decreases when going from 2 times upsampling to 3 times upsampling, but still seems to improve the mIoU and FB-IoU scores. We achieved the second best reported FB-IoU score of the compared models.

### 4.3.2 Baseline Comparison

Table 4.7 shows our model and the comparison with the baselines for the PASCAL-$5^i$ benchmark. We notice a rather large improvement when we adapt the model using LoRA instead of full fine-tuning. This effect can be seen in every fold for both metrics. It is more drastic for the mIoU score.

Table 4.8 shows the comparison of our model with the baselines for the COCO-$20^i$ benchmark. For this benchmark, adapting the model using LoRA seems to actually decrease the performance compared to freezing the backbone. Full fine-tuning resulted in stagnant performance.

Table 4.9 shows the results for the FSS-1000 benchmark. Again, we see an improvement when the backbone is adapted using LoRA and full fine-tuning compared to freezing it. LoRA adaptation led to a higher accuracy compared to the full fine-tuning.

| Tuning | Time per query | | | | | max memory usage | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Fold-1 | Fold-2 | Fold-3 | Fold-4 | Avg | Fold-1 | Fold-2 | Fold-3 | Fold-4 | max |
| No | 0.76s | 0.79s | 0.91s | 1.67s | 1.03s | 160MB | 176MB | 173MB | 202MB | 202MB |
| Full | 1.64s | 2.75s | 4.04s | 4.43s | 3.22s | 745MB | 1.48GB | 2.79GB | 3.40GB | 3.40GB |
| LoRA | 2.16s | 2.12s | 2.05s | 3.22s | 2.39s | 1.22GB | 986MB | 1.19GB | 2.97GB | 2.97GB |

Table 4.10: Computational effort for the PASCAL-$5^i$ benchmark

| Tuning | Time per query | | | | | max memory usage | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Fold-1 | Fold-2 | Fold-3 | Fold-4 | Avg | Fold-1 | Fold-2 | Fold-3 | Fold-4 | max |
| No | 0.79s | 1.93s | 1.54s | 2.50s | 1.69s | 203MB | 234MB | 224MB | 240MB | 240MB |
| Full | 2.57s | 5.50s | 3.54s | 3.38s | 3.75s | 2.30GB | 5.81GB | 2.37GB | 3.30GB | 5.81GB |
| LoRA | 2.75s | 5.72s | 2.61s | 8.48s | 4.89s | 683MB | 2.29GB | 1.54GB | 2.96GB | 2.96GB |

Table 4.11: Computational effort for the COCO-$20^i$ benchmark

| Tuning | Time per query | max memory usage |
|---|---|---|
| No | 1.36 s | 110.69 MB |
| Full | 2.35 s | 671.06 MB |
| LoRA | 1.88 s | 398.61 MB |

Table 4.12: Computational effort for FSS-1000

We see that LoRA led to a good performance on the PASCAL-$5^i$ and FSS-1000 benchmarks. It did not prove useful for the COCO-$20^i$ benchmark.

### 4.3.3 Computational Effort

Table 4.10 shows the computational effort for each episode of the PASCAL-$5^i$ benchmark. The time is averaged, and the memory usage is the maximum of all episodes tested. We can see that the time and memory change depending on the fold. The time for the LoRA adaptation is between 2 and 3.2 seconds, the full fine-tuning takes longer on average with times between 1.6 and 4.4 seconds. The baseline is quite fast, with an average of 1 second per episode. This time includes training the segmentation module for the first time for a novel class. Successive segmentation tasks should be orders of magnitude faster. Memory usage stays below 4 GB, which means we can do this adaptation on normal consumer hardware.

Table 4.11 shows the computational effort for the COCO-$20^i$ benchmark. Again, we averaged the time of 100 episodes and took the maximum amount of memory used by the model. The time per episode is slightly higher than for the PASCAL-$5^i$ benchmark, but still under 10 seconds in every case. Fold-4 seems to be an outlier for LoRA adaptation. Full fine-tuning had the largest memory requirement, but it was still under 6 GB.

Table 4.12 shows the computational effort for the FSS-1000 benchmark. This time the average times are closer together. The maximum memory consumption is less than 1 GB. This could be due to the small image size of $224 \times 224$ pixels per image.

| Model | mIoU% | | | | FB-IoU% | | | |
|---|---|---|---|---|---|---|---|---|
| | Fold-1 | Fold-2 | Fold-3 | Fold-4 | Fold-1 | Fold-2 | Fold-3 | Fold-4 |
| Ours (1x) | 84.9 | 84.0 | 86.9 | 83.0 | 91.2 | 90.6 | 92.3 | 89.9 |
| Ours (2x) | **88.93** | **87.58** | **90.05** | **86.40** | **93.56** | **92.58** | **94.09** | **91.80** |
| VAT (Resnet 101) | 85.0 | 82.6 | 83.2 | 82.1 | 90.1 | 88.4 | 89.0 | 88.0 |

Table 4.13: Results of FSS-1000 benchmark using PASCAL-$5^i$ hyper-parameters

Although we probably cannot compete with other models on inference time, we still achieve usable times and memory usage.

### 4.3.4 Domain Shift

In the last section we showed that the inference time is still usefully fast for most cases. In this subsection we want to analyze if we need to retrain for new domains or if we can reuse the hyper-parameters from another domain. This would mean no additional training time for new domains.

Table 4.13 shows our results on the FSS-1000 benchmark when using the hyper-parameters from the PASCAL-$5^i$ training sets. We improve over VAT in all folds, indicating that our method is more robust to domain shifts. The hyper-parameters from Fold-3 even outperformed the 3 times up-sampled model with hyper-parameters from the FSS-1000 training set. SegGPT did not use the FSS-1000 benchmark for its training, although they used not only the PASCAL-$5^i$ benchmark, but also the COCO-$20^i$ benchmark and additional training data, making the comparison not entirely fair. They achieved a mIoU score of 89.3. We are able to achieve a higher score by using the third fold of the PASCAL-$5^i$ benchmark.

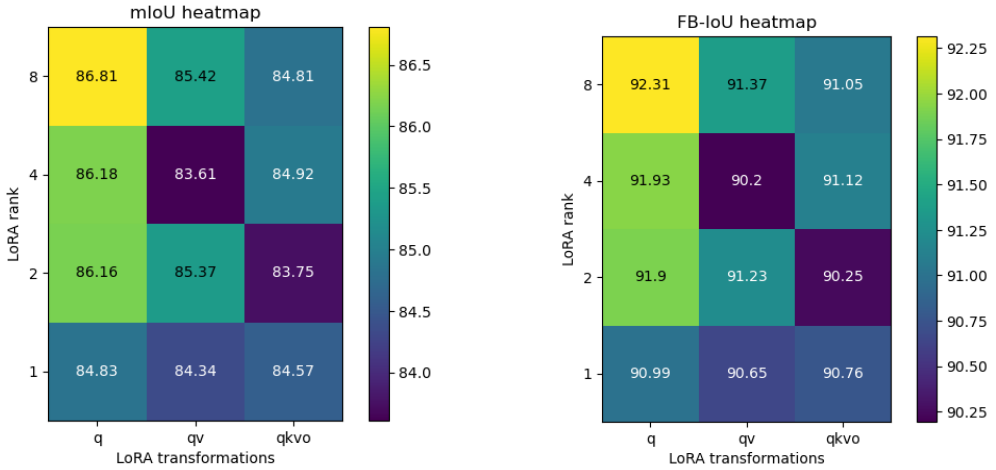
### 4.3.5 Ablation Study

Figure 4.2 shows the effect of different LoRA configurations on the FSS-1000 benchmark results with no upsampling of the input images in the segmentation module. The same can be seen in Figure 4.3 with 2 times upsampling. Both results show that only adjusting the $Q$-transformation using LoRA seems to lead to the best results. It seems that a rank of $r = 1$ is not sufficient, but after that the rank seems to have little effect on the resulting score. We note that we again obtained a higher score than in the main results, when selecting the highest value from the 2 times up-sampled ablation study.

## 4.4 Qualitative Results

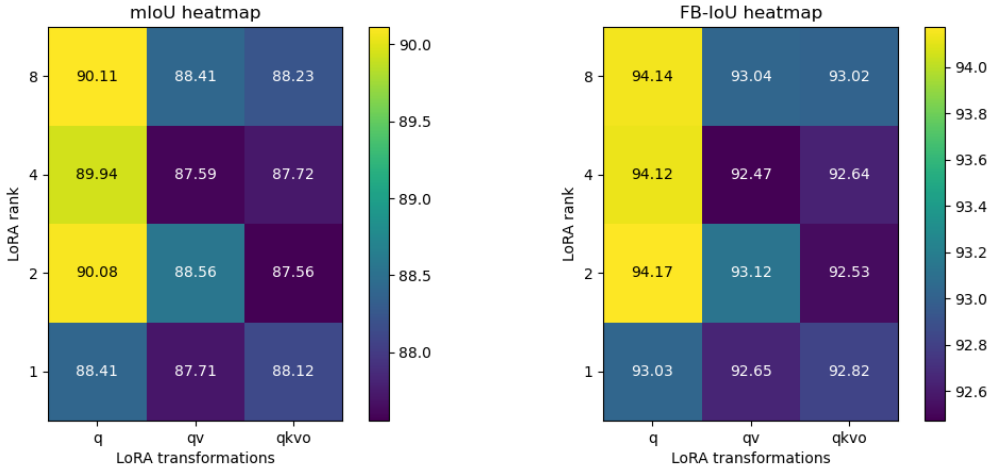
### 4.4.1 Predicted Masks

Figure 4.4 shows some examples of the predicted masks of the PASCAL-$5^i$ benchmark. The five support images are shown on the left, with their masks overlaid in yellow. The




(a) mIoU (b) FB-IoU

Figure 4.2: Ablation study of the effect of LoRA configurations on the FSS-1000 benchmark results using no up-sampling




(a) mIoU (b) FB-IoU

Figure 4.3: Ablation study of the effect of LoRA configurations on the FSS-1000 benchmark results using 2 times up-sampling


(a) Segmentation of Person


(b) Segmentation of Chair


(c) Segmentation of Airplane


(d) Segmentation of Car


(e) Segmentation of Bicycle


(f) Segmentation of Chair

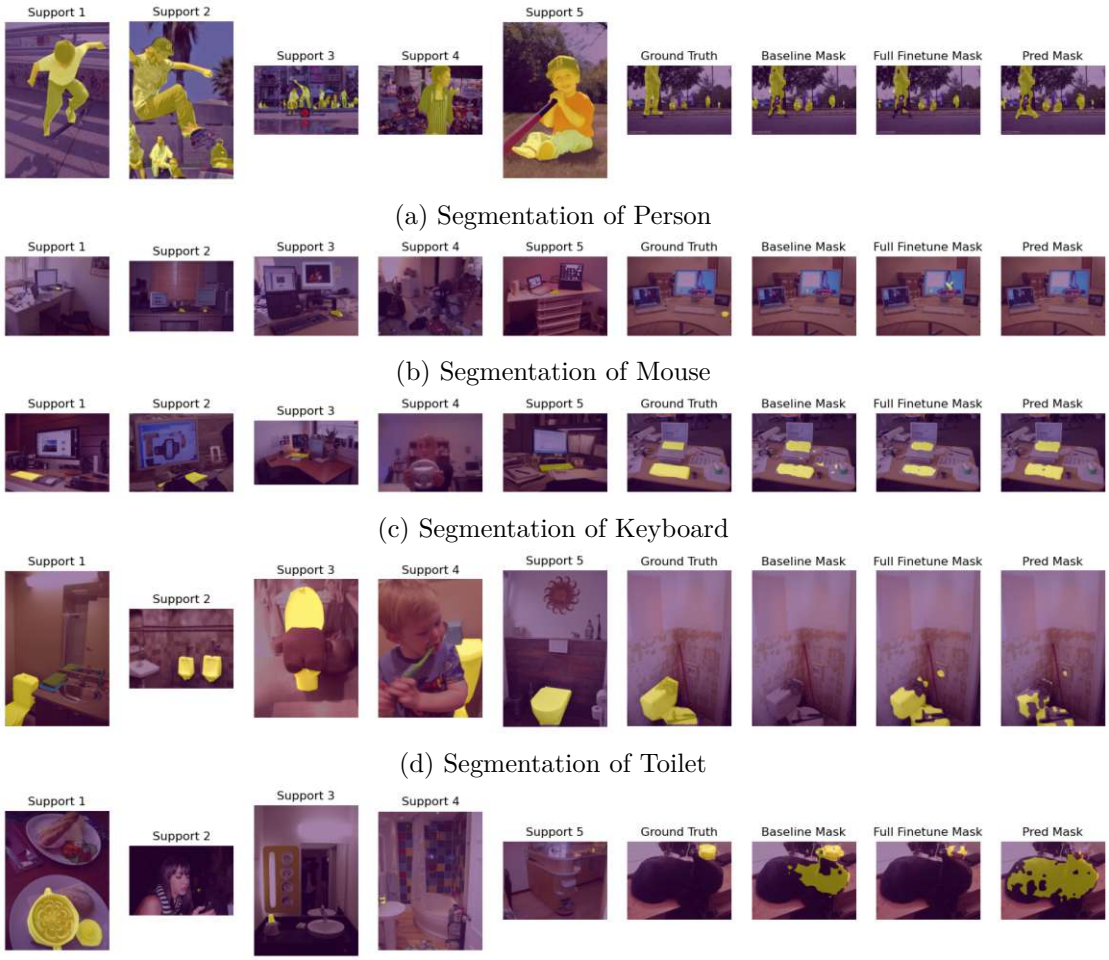
Figure 4.4: Qualitative Examples of PASCAL-$5^i$ Segmentations

right side shows first the ground truth, then the mask of the baseline without any fine-tuning. The second image from the right shows the predicted mask with full fine-tuning of the DINOv2 backbone and the rightmost image the results of our model using LoRA.

The first example shown in Figure 4.4a shows that our model successfully stopped including parts of the chairs in the segmentation mask. The full fine-tune seemed to learn to segment the person better, but also included more parts of the chair. Our approach refined the person segmentation and removed the chair. The second example shown in Figure 4.4b shows the segmentation of Chair. Here we were only able to segment the chairs using LoRA. The baseline and the full fine-tune were not able to detect the chairs at all. The third example shown in Figure 4.4c shows another successful improvement. Here the full fine-tune managed to refine the edges of the airplane, but removed the landing gear. Our approach managed to add the landing gear back in.

The next examples show a few problems with our approach. The segmentation of the car shown in Figure 4.4d shows that our approach struggles with small objects. Neither our model with LoRA nor the baselines were able to detect the small cars in the background. The segmentation of the bicycle shown in Figure 4.4e shows that the full fine-tune successfully managed to exclude the graffiti from the segmentation mask and LoRA was just able to refine the outlines of the bike, but falsely included parts of the graffiti. The last example shown in Figure 4.4f shows the segmentation of a chair. Here LoRA managed to detect the chair in the background, which the baselines did not, but it also falsely includes the dog and large parts of the ground in the segmentation mask.

Figure 4.5 shows some examples of the masks we predicted using our model with LoRA and the baselines of the COCO-$20^i$ benchmark. The first example shown in Figure 4.5a depicts the segmentation results of persons. Our model is able to accurately detect people, but it falsely includes the skateboard and is fuzzy around the edges of the objects. The second example shown in Figure 4.5b depicts the segmentation results of a pc mouse. It shows that our model struggles with small objects, but it reduces false positives compared to the baseline and full fine-tuning. The third example shown in Figure 4.5c depicts the segmentation of a pc keyboard. Our approach using LoRA improved upon the baselines by both reducing the number of false positives compared to the baseline and the number of false negatives compared to full fine-tuning. In Figure 4.5d an example of the segmentation results for a toilet is shown. Our model was not able to improve on the full fine-tuning in this case. The fully fine-tuned model clearly detected the toilet, but falsely excluded the tools and the piece of cloth. Some ceramic fixture on the wall was also included. This might suggest that our model lacks the geometric understanding of the object and does not seem to know that the toilet is behind the tools. It is unclear to us what happened in the example shown in Figure 4.5e. Here the goal was to segment the cup behind the cat. Our model learned to segment the cat instead of the cup. This behavior is noticeable to a small degree in the results of the simple baseline, but here the cup was also part of the segmentation mask. The full fine-tune managed to remove the cat, but also parts of the cup. This episode seems quite difficult to segment, in four of the five support images the cup is very small and in the background. The cups in the




(a) Segmentation of Person

(b) Segmentation of Mouse

(c) Segmentation of Keyboard

(d) Segmentation of Toilet

(e) Segmentation of Cup

Figure 4.5: Qualitative Examples of COCO-$10^i$ Segmentations

support set are opaque and the cup in the query image is translucent.

Figure 4.6 shows examples of masks for the FSS-1000 benchmark without upsampling the input images. It worked very well for the fully fine-tuned and our LoRA model on the Osprey. Only the edges are a bit fuzzy. The toaster was a different type than all the toasters in the support set, but it still seemed to recognize the toaster. The bread was mostly excluded by the LoRA model. We would still rate this as successful, as it is unclear whether the bread should be included or not.

Figure 4.7 shows examples of masks for the FSS-1000 benchmark with two times upsampling of the input images. The first example of the onion segmentation looks really good. The edges look smoother compared to no up-sampling. The chicken example shows that LoRA seems to be able to sharpen the edges a bit compared to the baseline and full


(a) Segmentation of Osprey


(b) Segementation of Toaster

Figure 4.6: Qualitative Examples of FSS-1000 segmentations with no up-sampling


(a) Segmentation of Onion


(b) Segementation of Chicken

Figure 4.7: Qualitative Examples of FSS-1000 segmentations with two times up-sampling

fine-tuning. All approaches were able to achieve good results.

We do well with large objects that are clearly in the foreground. We manage to improve the edges of detected objects in many cases, but still have some issues with them. We do not seem to be able to detect very small objects.

### 4.4.2 Embedding Adaptation

We also looked at the change in the embeddings due to the use of LoRA. Inspired by the DINOv2 paper [ODM+23], we visualized the first three PCA components of the embeddings before and after LoRA. Each component is assigned to a different color channel, and the PCA decomposition was calculated using all images in each figure.

Figure 4.8 shows the changes in the embeddings for Toaster for the FSS-1000 benchmark without upsampling. We can see a little more coherence in the embeddings for Support 4 after the adaptation. For the other embeddings, this effect is less, but still noticeable for Support 2. We do not see any drastic color change, therefore the models parameters have only changed slightly.

Figure 4.9 shows the same for the onion in the FSS-1000 benchmark with two times

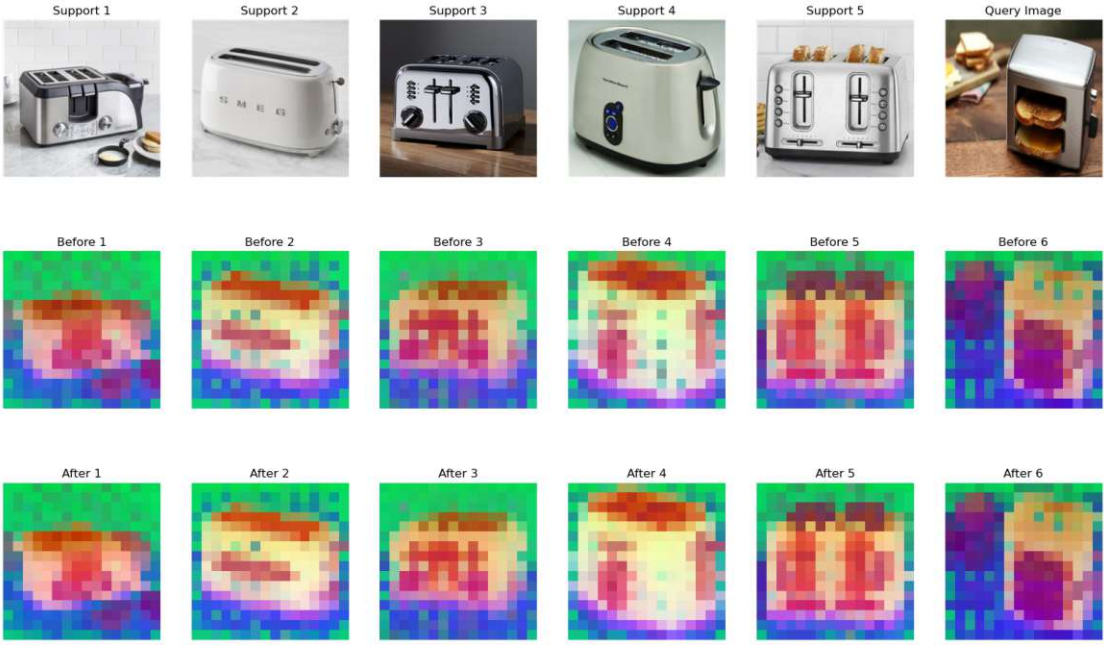



Figure 4.8: Adaption of the Embeddings for Toaster in FSS-1000 without up-sampling

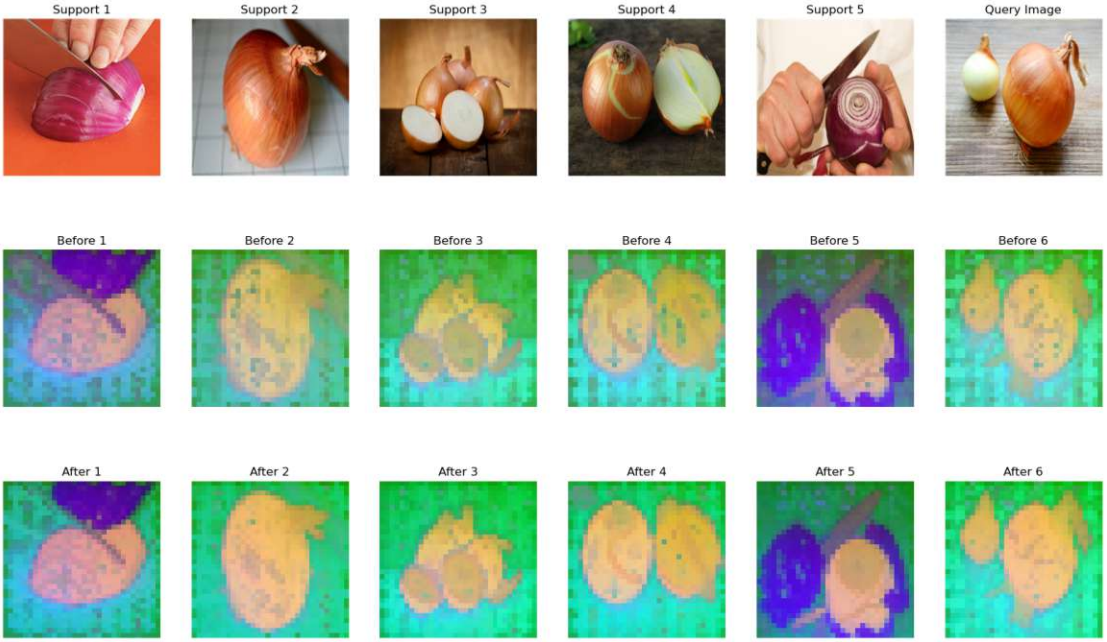



Figure 4.9: Adaption of the Embeddings for Onion in FSS-1000 with two times up-sampling

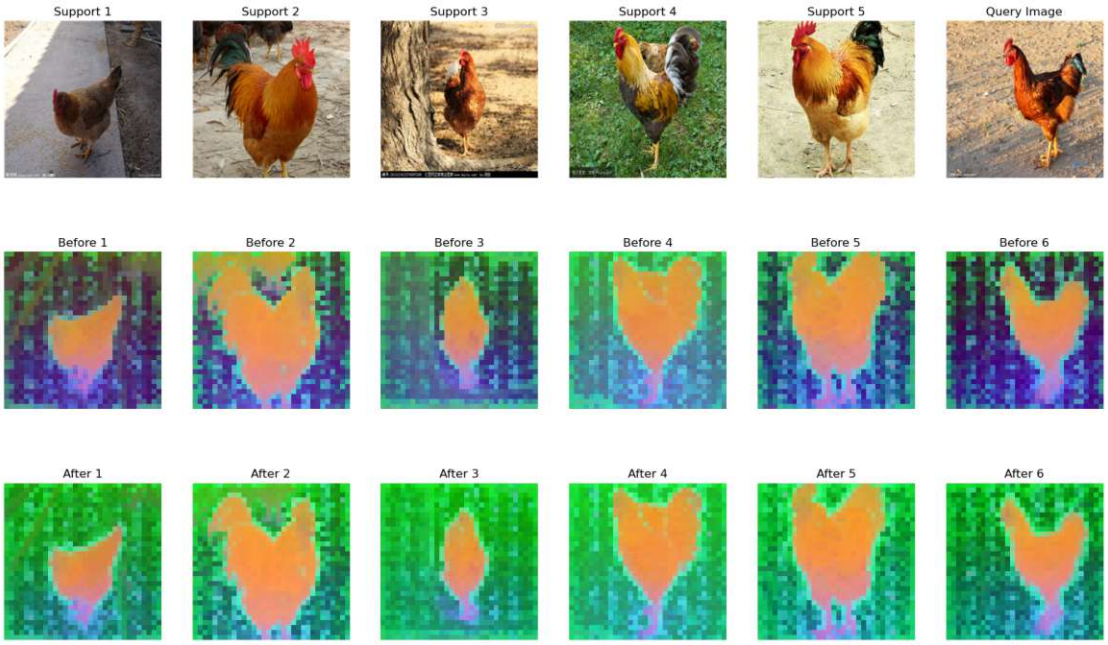



Figure 4.10: Adaption of the Embeddings for Chicken in FSS-1000 with two times up-sampling

upsampling. Again, we see a homogenization of the embeddings. This effect is best seen in Support 2 and Support 4, but also in the Query Image and Support 1.

Figure 4.10 shows the change in the embeddings for the chicken in the FSS-1000 benchmark using two times up-sampling for the input images. We can see that the background changed colors quite drastically, going from blue to green. We can also see that the legs of the chickens have different colors than their bodies. The change in the background may make it easier to distinguish the chicken legs from the background.

In most cases, the changes in the embeddings due to LoRA were small and seemed to mostly increase the internal consistency of the classes. We noticed fewer differently colored pixels in the object masks.

# CHAPTER 5

# Discussion

## 5.1 Main findings

The main finding of this thesis is that LoRA is a viable way to achieve effective FSS using large-scale VFMs, such as DINOv2, with a drastically reduced number of trainable parameters, making it possible to store hundreds to thousands of model variations for faster inference in the future. For two of the three tested benchmarks tested, PASCAL-$5^i$ and FSS-1000, we have shown that our LoRA approach improved upon our baselines and reached scores within a few percentage points for the individual folds in comparison to SOTA models. We have good FB-IoU values, but cannot compare them to most SOTA models, as this metric is not always reported.

We showed that transforming only one layer of the attention module leads to better performance than improving all layers, and a low rank seems to be sufficient to improve the performance.

We showed that FSS performance can be improved for low-resolution images by upsampling them before applying the model.

## 5.2 Strengths and Limitations

### 5.2.1 Strengths

One of the key strengths of our approach is the parameter efficiency. LoRA has reduced the number of trainable parameters by up to 1000 times compared to full fine-tuning, while maintaining or improving performance. This typically translates into faster training, and less overfitting.

Another major strength is the generalizability of our pipeline. We were able to use the hyper-parameter from one domain in another and had no noticeable effect on the segmentation accuracy.

We can share only the LoRA layers and the head using under 1 MB of space, making future segmentation tasks of the same class faster, as the pipeline is very lightweight for the performance we achieve. We can also run this model on mid-range consumer hardware.

### 5.2.2 Limitations

A major limitation of our approach is the patch-based architecture of DINOv2. The backbone embeds a $14 \times 14$ patch of pixels into a single feature vector. This results in an inability to segment small objects and to detect precise object boundaries. This may have caused the relatively poor performance on the COCO-$20^i$ benchmark compared to SOTA models.

We have also noticed that different classes were falsely included in the segmentation masks, when they are not present in the support set. Examples of this are shown in Figures 4.5e and 4.4f.

Another limitation is a high initial inference latency during the first adaptation to a novel class. For a few classes 2 to 4 seconds of inference time is not much, but this could add up in the case of many different novel classes. We suspect that this will not be a major issue in real-world scenarios.

## 5.3 Research Questions

Here we return back to the research questions listed in Section 1.3.

1. We achieved an average mIoU score of 74.9 for PASCAL-$5^i$, 36.9 for COCO-$20^i$, and 88.1 for FSS-1000. The FB-IoU scores were higher with 86.8 for PASCAL-$5^i$, 66.7 for COCO-$20^i$, and 92.7 for FSS-1000. The scores for PASCAL-$5^i$ and FSS-1000 are comparable to traditional SOTA models, but in the case of PASCAL-$5^i$ cannot quite keep up with generalist models like GF-SAM and SegGPT.

2. We did not observe drastic savings in memory usage and computation time when using LoRA compared to full fine-tuning. On average, we required about 15-50% less memory than full fine-tuning. There was no clear trend to see for the inference time.

3. We showed that tuning only the Q-transformation layer in the attention layers was better than tuning more transformations. A rank as low as $r = 2$ was good enough, although there was a small performance gain when using a rank of $r = 8$.

4. We were able to keep the metrics about the same when transferring the knowledge gained on the PASCAL-$5^i$ training sets to the FSS-1000 test set. This was better than doing the same for VAT, a SOTA model for the FSS-1000 benchmark.

CHAPTER 6

# Conclusion and Future Work

## 6.1 Conclusion

In this thesis, we explored the application of Low-Rank Adaptation (LoRA) to Few-Shot Semantic Segmentation (FSS) using the large-scale Visual Foundation Model (VFM) DINOv2. We proposed a segmentation pipeline that uses five labeled support images to train a binary segmenter for a novel class, essentially enabling FSS "on the fly".

The main contributions of this thesis include:

- A novel integration of LoRA into DINOv2 attention layers, allowing fast and parameter-efficient fine-tuning.
- A simple FSS pipeline that can be easily expanded with more complex classification heads.
- Comprehensive benchmarking on PASCAL-$5^i$, COCO-$20^i$ and FSS-1000, with competitive results, especially in terms of FB-IoU and domain shift performance, for PASCAL-$5^i$ and FSS-1000.

Our experiments showed that LoRA can match or exceed full fine-tuning in terms of accuracy, while reducing memory usage slightly. These results demonstrate the potential of LoRA to improve VFMs for few-shot applications.

## 6.2 Future Work

While our results are promising, there is still lots of room for improvement. We were only able to experiment with the smallest member of the DINOv2 family. With more

computation power, we could try to use larger models and see if they are able to increase the performance even further.

Upsampling the input images was a quick way to deal with the patch size of DINOv2. Further research could be done into different ways to improve the feature resolution for DINOv2. Methods like JBU [KCLU07] or frameworks like FeatUp by Fu et al. [FHB+24] and HR-DINOv2 by Docherty et al. [DVC24] should be explored in further research to unlock the full power of VFMs.

We have also noticed that we were able to achieve a higher performance in the ablation study than with the selected LoRA configuration. This suggests that some other methods to choose the LoRA configurations, like choosing additional episodes from the training set as a validation set, should be experimented with.

A fourth point of improvement could be the inclusion of different negative examples that do not depict the class in the support set. These images could be taken randomly or with the help of a classifier from the training set. We expect that this would reduce the number of false positives in the segmentation masks.

With these improvements, we suspect that we could surpass the performance of SOTA models for some FSS benchmarks.

# Overview of Generative AI Tools Used

We used GitHub Copilot as a smart auto-completion tool for implementing our code. ChatGPT was used for help with the structure of the thesis and summarizing the main points. We still corrected every sentence and copied no whole passages from the output.

We used DeepL Write and LanguageTool for proofreading the thesis for grammar mistakes and typos.

# List of Figures

# List of Tables

# Acronyms

**CANet** Class-Agnostic Segmentation Network. 7, 24

**CyCTR** Cycle-Consistent Transformer. 7, 24, 25

**DCAM** Doubly Deformable Aggregation of Covariance Matrices. 8, 24, 25

**DCAMA** Dense Cross-query-and-support Attention weighted Mask Aggregation. 8, 24, 25

**DGPNet** Dense Gaussian Process Network. 8, 24, 25

**DL** deep learning. 2, 13

**FB-IoU** foreground-background IoU. 3, 6, 22–26, 28, 29, 37, 38, 41

**FM** Foundation Model. 9, 10

**FPTrans** Feature-Proxy Transformer. 8, 24, 25

**FSS** Few-Shot Semantic Segmentation. 2–8, 12, 16, 19, 22–24, 37, 41, 42, 45

**FWB** Feature Weighting and Boosting. 7, 24, 25

**GF-SAM** Graph-based Few-shot Segment Anything Semantically. 9, 24, 25, 38

**HM** Hybrid Masking. 8, 24, 25

**HSNet** Hypercorrelation Squeeze Network. 8, 24, 25

**LLM** large language model. 3, 10

**LoRA** Low-Rank Adaptation. 3, 4, 10, 12, 16, 22, 23, 26–29, 31–33, 35, 37, 38, 41, 42, 45, 47

**LTM** Local Transformation Module. 7, 24

**MAP** Masked Average Pooling. 6, 7

**mIoU** mean Intersection over Union. 3, 6, 16, 21–26, 28, 29, 38

**ML** machine learning. 13

**MLC** Mining Latent Classes. 8, 24, 25

**NLP** Natural Language Processing. 9

**OSLSM** One-Shot Learning for Semantic Segmentation. 6, 24

**PANet** Prototype Alignment Network. 7, 24

**PFENet** Prior Guided Feature Enrichment Network. 7, 24, 25

**PGMA-Net** Prior Guided Mask Assemble Network. 9, 24, 25

**PPNet** Part-aware Prototype Network. 7, 24, 25

**RePRI** Region Proportion Regularized Inference. 7, 24, 25

**SCL** Self-Guided and Cross-Guided Learning. 7, 24

**SegGPT** **seg**ment everything with a **g**eneralist **P**ain**t**er. 9, 24, 25, 28, 38

**SG-One** Similarity Guidance Network for One-Shot Semantic Segmentation. 6, 24

**SiS** semantic image segmentation. 1, 2, 9

**SOTA** state-of-the-art. 2, 3, 6, 9, 24, 37–39, 42

**VAT** Volumetric Aggregation with Transformers. 8, 23–25, 28, 39

**VFM** Visual Foundation Model. 3, 4, 9, 37, 41, 42

**ViT** Vision Transformer. 10, 11, 20, 22, 45

**VSC** Vienna Scientific Cluster. 20

# Bibliography


[ASY+19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2623–2631, New York, NY, USA, 2019. Association for Computing Machinery.

[AYF+24] Asma Ben Abacha, Wen-wai Yim, Yujuan Fu, Zhaoyi Sun, Meliha Yetisgen, Fei Xia, and Thomas Lin. MEDEC: A benchmark for medical error detection and correction in clinical notes. *arXiv:2412.19260*, 2024.

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305, February 2012.

[BHA+21] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, S. Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen A. Creel, Jared Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren E. Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas F. Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, O. Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir P. Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Benjamin Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, J. F. Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Robert Reich, Hongyu Ren, Frieda Rong, Yusuf H. Roohani, Camilo

Ruiz, Jack Ryan, Christopher R'e, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishna Parasuram Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei A. Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models. *ArXiv*, 2021.

[BKM+21] Malik Boudiaf, Hoel Kervadec, Ziko Imtiaz Masud, Pablo Piantanida, Ismail Ben Ayed, and Jose Dolz. Few-shot segmentation without meta-learning: A good transductive inference is all you need? In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13974–13983, 2021.

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[CLR+23] Zhaobin Chang, Yonggang Lu, Xingcheng Ran, Xiong Gao, and Xiangwen Wang. Few-shot semantic segmentation: a review on recent approaches. *Neural Computing and Applications*, 35(25):18251–18275, July 2023.

[CM24] Nico Catalano and Matteo Matteucci. Few shot semantic segmentation: a review of methodologies, benchmarks, and open challenges. *arXiv:2304.05832*, 2024.

[CMZ+24] Shuai Chen, Fanman Meng, Runtong Zhang, Heqian Qiu, Hongliang Li, Qingbo Wu, and Linfeng Xu. Visual and textual prior guided mask assemble for few-shot segmentation and beyond. *IEEE Trans. Multim.*, 26:7197–7209, 2024.

[Com24] Wikimedia Commons. File:vision transformer.svg — wikimedia commons, the free media repository, 2024. [Online; accessed 11-April-2025].

[Com25] Wikimedia Commons. File:multiheaded attention, block diagram.png — wikimedia commons, the free media repository, 2025. [Online; accessed 24-March-2025].

[CZSL20] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3008–3017, 2020.

[DAAVF24] Francesco D' Angelo, Maksym Andriushchenko, Aditya Varre, and Nicolas Flammarion. Why do we need weight decay in modern deep learning? In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 23191–23223. Curran Associates, Inc., 2024.

[DBK+21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[DVC24] Ronan Docherty, Antonis Vamvakeros, and Samuel J. Cooper. Upsampling dinov2 features for unsupervised vision tasks and weakly supervised materials segmentation. *CoRR*, abs/2410.19836, 2024.

[EVGW+] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[FHB+24] Stephanie Fu, Mark Hamilton, Laura E. Brandt, Axel Feldmann, Zhoutong Zhang, and William T. Freeman. Featup: A model-agnostic framework for features at any resolution. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[HAGM14] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 297–312, Cham, 2014. Springer International Publishing.

[HCN+22] Sunghwan Hong, Seokju Cho, Jisu Nam, Stephen Lin, and Seungryong Kim. Cost aggregation with 4d convolutional swin transformer for few-shot segmentation. In *European Conference on Computer Vision*, pages 108–126. Springer, 2022.

[HSW+22] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[HVD15] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[HZG20] Shijie Hao, Yuan Zhou, and Yanrong Guo. A brief survey on semantic segmentation with deep learning. *Neurocomputing*, 406:302–321, 2020.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[JEF+22] Joakim Johnander, Johan Edstedt, Michael Felsberg, Fahad Shahbaz Khan, and Martin Danelljan. Dense gaussian processes for few-shot segmentation. In Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *Computer Vision – ECCV 2022*, pages 217–234, Cham, 2022. Springer Nature Switzerland.

[JYL20] Xiao Jiang, Haibin Yu, and Shuaishuai Lv. An image segmentation algorithm based on a local region conditional random field model. *International Journal of Communications, Network and System Sciences*, 13(09):139–159, 2020.

[KCLU07] Johannes Kopf, Michael F. Cohen, Dani Lischinski, and Matt Uyttendaele. Joint bilateral upsampling. *ACM Trans. Graph.*, 26(3):96–es, July 2007.

[KGC17] Jan Kukacka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy. *CoRR*, abs/1710.10686, 2017.

[KLY18] Robert Kleinberg, Yuanzhi Li, and Yang Yuan. An alternative view: When does SGD escape local minima? In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2703–2712. PMLR, 2018.

[KMR+23] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. In

*2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3992–4003, 2023.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing.

[LWC+20] Xiang Li, Tianhan Wei, Yau Pun Chen, Yu-Wing Tai, and Chi-Keung Tang. Fss-1000: A 1000-class dataset for few-shot segmentation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2020.

[LXJ+23] Zhuang Liu, Zhiqiu Xu, Joseph Jin, Zhiqiang Shen, and Trevor Darrell. Dropout reduces underfitting. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 22233–22248. PMLR, 23–29 Jul 2023.

[LZL+24] Yang Liu, Muzhi Zhu, Hengtao Li, Hao Chen, Xinlong Wang, and Chunhua Shen. Matcher: Segment anything with one shot using all-purpose feature matching. In *The Twelfth International Conference on Learning Representations*, 2024.

[LZZH20] Yongfei Liu, Xiangyi Zhang, Songyang Zhang, and Xuming He. Part-aware prototype network for few-shot semantic segmentation. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 142–158, Cham, 2020. Springer International Publishing.

[MBLAGJ+07] Saturnino Maldonado-Bascon, Sergio Lafuente-Arroyo, Pedro Gil-Jimenez, Hilario Gomez-Moreno, and Francisco Lopez-Ferreras. Road-sign detection and recognition based on support vector machines. *IEEE Transactions on Intelligent Transportation Systems*, 8(2):264–278, 2007.

[MKC21] Juhong Min, Dahyun Kang, and Minsu Cho. Hypercorrelation squeeze for few-shot segmenation. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6921–6932, 2021.

[MM22] Alhassan Mumuni and Fuseini Mumuni. Data augmentation: A comprehensive survey of modern approaches. *Array*, 16:100258, 2022.

[MSZ+22] Seonghyeon Moon, Samuel S. Sohn, Honglu Zhou, Sejong Yoon, Vladimir Pavlovic, Muhammad Haris Khan, and Mubbasir Kapadia. Hm: Hybrid masking for few-shot segmentation. In Shai Avidan, Gabriel Brostow,

Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *Computer Vision – ECCV 2022*, pages 506–523, Cham, 2022. Springer Nature Switzerland.

[NT19] Khoi Nguyen and Sinisa Todorovic. Feature weighting and boosting for few-shot segmentation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 622–631, 2019.

[OAA+24] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati,

Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report. *arXiv:2303.08774*, 2024.

[OCC+20] Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep learning vs. traditional computer vision. In Kohei Arai and Supriya Kapoor, editors, *Advances in Computer Vision*, pages 128–144, Cham, 2020. Springer International Publishing.

[ODM+23] Maxime Oquab, Timothée Darcet, Theo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Russell Howes, Po-Yao Huang, Hu Xu, Vasu Sharma, Shang-Wen Li, Wojciech Galuba, Mike Rabbat, Mido Assran, Nicolas Ballas, Gabriel Synnaeve, Ishan Misra, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision. *arXiv:2304.07193*, 2023.

[RKH+21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin,

Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 18–24 Jul 2021.

[SBL+17] Amirreza Shaban, Shray Bansal, Zhen Liu, Irfan Essa, and Byron Boots. One-shot learning for semantic segmentation. In Gabriel Brostow Tae-Kyun Kim, Stefanos Zafeiriou and Krystian Mikolajczyk, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 167.1–167.13. BMVA Press, September 2017.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[SWZ+22] Xinyu Shi, Dong Wei, Yu Zhang, Donghuan Lu, Munan Ning, Jiashun Chen, Kai Ma, and Yefeng Zheng. Dense cross-query-and-support attention weighted mask aggregation for few-shot segmentation. In Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *Computer Vision – ECCV 2022*, pages 151–168, Cham, 2022. Springer Nature Switzerland.

[TLI+23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023.

[TZS+22] Zhuotao Tian, Hengshuang Zhao, Michelle Shu, Zhicheng Yang, Ruiyu Li, and Jiaya Jia. Prior guided feature enrichment network for few-shot segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(2):1050–1065, 2022.

[Wat23] Shuhei Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance. *CoRR*, abs/2304.11127, 2023.

[WLZ+19] Kaixin Wang, Jun Hao Liew, Yingtian Zou, Daquan Zhou, and Jiashi Feng. Panet: Few-shot image semantic segmentation with prototype alignment. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9196–9205, 2019.

[WWC+23] Xinlong Wang, Wen Wang, Yue Cao, Chunhua Shen, and Tiejun Huang. Images speak in images: A generalist painter for in-context visual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6830–6839, June 2023.

[WZC+23] Xinlong Wang, Xiaosong Zhang, Yue Cao, Wen Wang, Chunhua Shen, and Tiejun Huang. Seggpt: Towards segmenting everything in context. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1130–1140, 2023.

[XLZ22] Zhitong Xiong, Haopeng Li, and Xiao Xiang Zhu. Doubly deformable aggregation of covariance matrices for few-shot segmentation. In Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *Computer Vision – ECCV 2022*, pages 133–150, Cham, 2022. Springer Nature Switzerland.

[XZF25] Leyi Xiao, Baoxian Zhou, and Chaodong Fan. Automatic brain MRI tumors segmentation based on deep fusion of weak edge and context features. *Artificial Intelligence Review*, 58(154), 3 2025.

[YS20] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

[YZQ+21] Lihe Yang, Wei Zhuo, Lei Qi, Yinghuan Shi, and Yang Gao. Mining latent classes for few-shot segmentation. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8701–8710, 2021.

[ZGJ+24] Anqi Zhang, Guangyu Gao, Jianbo Jiao, Chi Harold Liu, and Yunchao Wei. Bridge the points: Graph-based few-shot segment anything semantically. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 33232–33261. Curran Associates, Inc., 2024.

[ZLL+19] Chi Zhang, Guosheng Lin, Fayao Liu, Rui Yao, and Chunhua Shen. Canet: Class-agnostic segmentation networks with iterative refinement and attentive few-shot learning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5212–5221, 2019.

[ZSYC22] Jian-Wei Zhang, Yifan Sun, Yi Yang, and Wei Chen. Feature-proxy transformer for few-shot segmentation. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[ZWYH20] Xiaolin Zhang, Yunchao Wei, Yi Yang, and Thomas S. Huang. Sg-one: Similarity guidance network for one-shot semantic segmentation. *IEEE Transactions on Cybernetics*, 50(9):3855–3865, 2020.

[ZXQ21] Bingfeng Zhang, Jimin Xiao, and Terry Qin. Self-guided and cross-guided learning for few-shot segmentation. In *2021 IEEE/CVF Conference on*

*Computer Vision and Pattern Recognition (CVPR)*, pages 8308–8317, 2021.

# Appendix

## Benchmark Classes

The different classes for each fold in the PASCAL-$5^i$ can be seen in Table 1. In Table 2 the different classes for the COCO-$20^i$ benchmark can be seen. The classes used in training for the FSS-1000 benchmark can be seen in Tables 4, 5, 6, and 7 and the classes used for testing in Table 3.

| fold | | | |
|---|---|---|---|
| **PASCAL-$20^0$** | **PASCAL-$20^1$** | **PASCAL-$20^2$** | **PASCAL-$20^3$** |
| aeroplane | bus | diningtable | potted plant |
| bicycle | car | dog | sheep |
| bird | cat | horse | sofa |
| boat | chair | motorbike | train |
| bottle | cow | person | tv/monitor |

Table 1: Classes for PASCAL-$5^i$

| fold | | | |
|---|---|---|---|
| **COCO-20$^0$** | **COCO-20$^1$** | **COCO-20$^2$** | **COCO-20$^3$** |
| Person | Bicycle | Car | Motorcycle |
| Airplane | Bus | Train | Truck |
| Boat | T. light | Fire H. | Stop |
| Park meter | Bench | Bird | Cat |
| Dog | Horse | Sheep | Cow |
| Elephant | Bear | Zebra | Giraffe |
| Backpack | Umbrella | Handbag | Tie |
| Suitcase | Frisbee | Skis | Snowboard |
| Sports ball | Kite | B. bat | B. glove |
| Skateboard | Surfboard | T. racket | Bottle |
| W. glass | Cup | Fork | Knife |
| Spoon | Bowl | Banana | Apple |
| Sandwich | Orange | Broccoli | Carrot |
| Hot dog | Pizza | Donut | Cake |
| Chair | Couch | P. plant | Bed |
| D. table | Toilet | TV | Laptop |
| Mouse | Remote | Keyboard | Cellphone |
| Microwave | Oven | Toaster | Sink |
| Fridge | Book | Clock | Vase |
| Scissors | Teddy | Hairdryer | Toothbrush |

Table 2: Classes for COCO-20$^i$

| | | | | |
|---|---|---|---|---|
| bus | hotel_slipper | burj_al | reflex_camera | abe's_flyingfish |
| oiltank_car | doormat | fish_eagle | barber_shaver | motorbike |
| feather_clothes | wandering_albatross | rice_cooker | delta_wing | fish |
| nintendo_switch | bustard | diver | minicooper | cathedrale_paris |
| big_ben | combination_lock | villa_savoye | american_alligator | gym_ball |
| andean_condor | leggings | pyramid_cube | jet_aircraft | meatloaf |
| reel | swan | osprey | crt_screen | microscope |
| rubber_eraser | arrow | monkey | mitten | spiderman |
| parthenon | bat | chess_king | sulphur_butterfly | quail_egg |
| oriole | iron_man | wooden_boat | anise | steering_wheel |
| groenendael | dwarf_beans | pteropus | chalk_brush | bloodhound |
| moon | english_foxhound | boxing_gloves | peregine_falcon | pyraminx |
| cicada | screw | shower_curtain | tredmill | bulb |
| bell_pepper | lemur_catta | doughnut | twin_tower | astronaut |
| nintendo_3ds | fennel_bulb | indri | captain_america_shield | kunai |
| broom | iphone | earphone1 | flying_squirrel | onion |
| vinyl | sydney_opera_house | oyster | harmonica | egg |
| breast_pump | guitar | potato_chips | tunnel | cuckoo |
| rubick_cube | plastic_bag | phonograph | net_surface_shoes | goldfinch |
| ipad | mite_predator | coffee_mug | golden_plover | f1_racing |
| lapwing | nintendo_gba | pizza | rally_car | drilling_platform |
| cd | fly | magpie_bird | leaf_fan | little_blue_heron |
| carriage | moist_proof_pad | flying_snakes | dart_target | warehouse_tray |
| nintendo_wiiu | chiffon_cake | bath_ball | manatee | cloud |
| marimba | eagle | ruler | soymilk_machine | sled |
| seagull | glider_flyingfish | doublebus | transport_helicopter | window_screen |
| truss_bridge | wasp | snowman | poached_egg | strawberry |
| spinach | earphone2 | downy_pitch | taj_mahal | rocking_chair |
| cablestayed_bridge | sealion | banana_boat | pheasant | stone_lion |
| electronic_stove | fox | iguana | rugby_ball | hang_glider |
| water_buffalo | lotus | paper_plane | missile | flamingo |
| american_chamelon | kart | chinese_knot | cabbage_butterfly | key |
| church | tiltrotor | helicopter | french_fries | water_heater |
| snow_leopard | goblet | fan | snowplow | leafhopper |
| pspgo | black_bear | quail | condor | chandelier |
| hair_razor | white_wolf | toaster | pidan | pyramid |
| chicken_leg | letter_opener | apple_icon | porcupine | chicken |
| stingray | warplane | windmill | bamboo_slip | wig |
| flying_geckos | stonechat | haddock | australian_terrier | hover_board |
| siamang | canton_tower | santa_sledge | arch_bridge | curlew |
| sushi | beet_root | accordion | leaf_egg | stealth_aircraft |
| stork | bucket | hawk | chess_queen | ocarina |
| knife | whippet | cantilever_bridge | may_bug | wagtail |
| leather_shoes | wheelchair | shumai | speedboat | vacuum_cup |
| chess_knight | pumpkin_pie | wooden_spoon | bamboo_dragonfly | ganeva_chair |
| soap | clearwing_flyingfish | pencil_sharpener1 | cricket | photocopier |
| nintendo_sp | samarra_mosque | clam | charge_battery | flying_frog |
| ferrari911 | polo_shirt | echidna | coin | tower_pisa |

Table 3: Test classes for FSS-1000

| | | | | |
|---|---|---|---|---|
| abacus | ab_wheel | acorn | ac_ground | ac_wall |
| adhensive_tape | adidas_logo1 | adidas_logo2 | afghan_hound | african_crocodile |
| african_elephant | african_grey | agama | aircraft_carrier | airedale |
| airliner | airship | air_strip | albatross | almond |
| ambulance | american_staffordshire | anemone_fish | angora | apple |
| apron | arabian_camel | arctic_fox | armadillo | armour |
| artichoke | ashtray | assult_rifle | aubergine | avocado |
| baboon | baby | backpack | badger | bagel |
| balance_beam | balance_weight | bald_eagle | balloon | ballpoint |
| banana | band-aid | banded_gecko | banjo | barbell |
| barometer | baseball | baseball_bat | baseball_player | basketball |
| basset | bassoon | bathtub | battery | beacon |
| beagle | beaker | beam_bridge | bear | beaver |
| bedlington_terrier | bee | beer_bottle | beer_glass | bee_eater |
| bee_house | bell | besom | bighorn_sheep | birdhouse |
| bison | bittern | black_grouse | black_stork | black_swan |
| blenheim_spaniel | blossom_card | bluetick | boa_constrictor | bolete |
| bolotie | bomb | border_terrier | boston_bull | bottle_cap |
| bouzouki | bowtie | box_turtle | bra | bracelet |
| bradypod | brain_coral | brambling | brasscica | briard |
| brick | brick_card | brick_tea | briefcase | broccoli |
| brown_bear | brush_pen | buckingham_palace | buckler | bulbul_bird |
| bullet_train | bushtit | butterfly | cabbage | cableways |
| cactus | cactus_ball | cairn | calculator | camel |
| camomile | candle | cannon | canoe | can_opener |
| capuchin | carambola | carbonara | cardoon | carousel |
| carp | carrot | carton | car_mirror | car_wheel |
| cassette | cauliflower | ceiling_fan | celery | cello |
| chainsaw | chalk | cheese | cheese_burger | cheetah |
| cherry | chess_bishop | chest | chickadee_bird | chicken_wings |
| chicory | chihuahua | children_slide | chimpanzee | chinese_date |
| chopsticks | christmas_stocking | cigar | cigarette | cleaver |
| cn_tower | cocacola | cocktail_shaker | coconut | coffeepot |
| coffin | coho | collar | colubus | comb |
| common_newt | computer_mouse | conch | consomme | conversion_plug |
| convertible | conveyor | corn | cornet | cornmeal |
| cosmetic_brush | cottontail | coucal | cougar | cowboy_hat |
| coyote | cpu | crab | cradle | crane |
| crash_helmet | crayon | cream | crepe | cricketball |
| cristo_redentor | crocodile | croissant | croquet_ball | cucumber |
| cumquat | cup | cushion | daisy | dandie_dinmont |
| dart | dhole | diamond | diaper | digital_clock |
| digital_watch | dingo | dinosaur | dishwasher | donkey |
| dough | dowitcher | dragonfly | drake | drum |
| drumstick | dugong | dumbbell | dutch_oven | earplug |
| eel | eft_newt | eggnog | egg_tart | egret |
| egyptian_cat | electric_fan | electronic_toothbrush | eletrical_switch | english_setter |
| envelope | equestrian_helmet | esport_chair | espresso | espresso_maker |
| excavator | face_powder | feeder | ferret | fig |
| file_cabinet | fire_balloon | fire_engine | fire_hydrant | fire_screen |
| flat-coated_retriever | flatworm | flowerpot | flute | flying_disc |
| folding_chair | fork | forklift | fountain | fox_squirrel |
| french_ball | frog | frying_pan | fur_coat | garbage_can |
| garbage_truck | garfish | garlic | gas_pump | gas_tank |
| gazelle | gecko | german_pointer | giant_panda | giant_schnauzer |
| gibbon | ginger | gliding_lizard | globe | golden_retriever |
| goldfish | golfcart | golf_ball | goose | gorilla |

Table 4: Train classes for FSS-1000 (1/4)

| | | | | |
|---|---|---|---|---|
| gourd | grasshopper | great_wall | green_mamba | grey_fox |
| grey_whale | guacamole | guinea_pig | gypsy_moth | gyromitra |
| hair_drier | hami_melon | hammer | hammerhead_shark | hamster |
| handcuff | handkerchief | handshower | hard_disk | hare |
| harp | hartebeest | harvester | har_gow | hatchet |
| hawthorn | head_cabbage | hen_of_the_woods | hippo | hock |
| hook | hornbill | hornet | hotdog | housefinch |
| howler_monkey | hummingbird | hyena | ibex | iceberg |
| icecream | ice_lolly | igloo | impala | indian_cobra |
| indian_elephant | ipod | ironing_board | jacamar | jackfruit |
| jacko_lantern | jay_bird | jellyfish | jinrikisha | jordan_logo |
| joystick | kangaroo | kappa_logo | kazoo | keyboard |
| killer_whale | kinguin | kitchen_knife | kite | kit_fox |
| koala | kobe_logo | kremlin | kwanyin | lacewing |
| ladder | ladle | ladybug | ladyfinger | lady_slipper |
| lampshade | langur | laptop | lark | lawn_mower |
| leatherback_turtle | leeks | lemon | leopard | lesser_panda |
| lettuce | lhasa_apso | lifeboat | light_tube | lion |
| lionfish | lipstick | litchi | llama | loafer |
| lobster | loggerhead_turtle | loguat | lorikeet | louvre_pyramid |
| lycaenid_butterfly | lynx | macaque | macaw | mailbox |
| mango | manx | maotai_bottle | maraca | mario |
| marmot | marshmallow | mashed_potato | matchstick | mcdonald_sign |
| mcdonald_uncle | measuring_cup | medical_kit | meerkat | melon_seed |
| memory_stick | microphone | microsd | microwave | military_vest |
| miniskirt | mink | modem | monarch_butterfly | mongoose |
| monitor | monocycle | mooli | mooncake | mortar |
| motarboard | motor_scooter | mountain_tent | mount_fuji | mouse |
| mouthpiece | mud_turtle | mule | muscle_car | mushroom |
| nagoya_castle | nail_scissor | narcissus | necklace | neck_brace |
| nematode | night_snake | nike_logo | obelisk | ocicat |
| oil_filter | okra | olive | one-armed_bandit | orang |
| orange | oscilloscope | ostrich | otter | owl |
| ox | paddle | paint_brush | panda | panpipe |
| panther | papaya | paper_crane | paper_towel | parachute |
| parakeet | parasol | parking_meter | parrots | parsnip |
| partridge | pastry | pate | pavilion | pea |
| peach | peanut | pear | peashooter | peacock |
| pedestal_fan | pelican | pen_holder | pencil | penguin |
| persimmon | phalanger | pharaoh_hound | philippine_eagle | phoenicopterus |
| phone_case | photo_album | pickelhaube | pig | pigeon |
| pikachu | pillow | pineapple | pine_nut | ping_pong |
| pink_dolphin | pinwheel | pipe | pizza | pliers |
| plum | pocket_watch | polecat | pole_house | police_van |
| police_vest | pomeranian | popsicle | porcupine | porgy |
| portrait | portuguese_man_of_war | potted_plant | pouch | pound_cake |
| powder_cake | power_bank | power_strip | prawn | pretzel |
| printer | prison_uniform | pterodactyl | ptarmigan | pudding |
| pug | pumpkin | punching_bag | purse | quail |
| quilt | rabbit | raccoon | racer_snake | racket |
| radar | radiated_tortoise | radio | radish | raft |
| ragdoll | rail_fence | railway | ram | ramen |
| raspberry | rat | rattlesnake | razor_clam | red_cliff |
| red_panda | red_slipper | redstart | refrigerator | remote_control |

Table 5: Train classes for FSS-1000 (2/4)

| | | | | |
|---|---|---|---|---|
| rhinoceros | rice | rice_ball | rice_cooker | rifle |
| ring | ringneck_snake | river_otter | road_roller | robe |
| robin | robot | rock_beauty | rocking_chair | roller |
| rolling_pin | roman_helmet | rose | roti | rough_collie |
| rowboat | rubber_eraser | rubber_tree | rugby_ball | ruler |
| safety_belt | safety_hat | sailboat | salad | salamander |
| salmon | salt_shaker | sandal | sandwich | sardine |
| sari | saw_blade | saxophone | scallion | scarf |
| school_bus | scissors | scorpion | scottish_deerhound | scottish_fold |
| screwdriver | scroll | sculpture | seagull | seal |
| seaplane | seashell | sea_turtle | sea_urchin | sealyham_terrier |
| seastar | selfie_stick | semang_statue | serpent | serving_tray |
| sewing_machine | shashlik | shampoo | shark | sheep |
| sheet_music | shelf | shellfish | shiitake | shimeji |
| shining_fish | ship | shirt | shopping_bag | shopping_cart |
| shorts | shovel | shrimp | siamese | sickle |
| sidecar | sidewinder | signboard | silkworm | silver_fish |
| sink | siren | skate | skewer | ski |
| ski_goggles | skillet | skull_cap | skunk | skyscraper |
| sled | sleeping_bag | slipper | sloth | slug |
| smoking_pipe | snail | snake | sneaker | sneaker_sandal |
| sneezing_monkey | snow_leopard | snowball | snowboard | snowman |
| soap | sock | soda | sofa | softshell_turtle |
| solar_panel | soldier_crab | sombrero | soup_spoon | space_heater |
| space_shuttle | sparrow | spatula | speaker | spectacles |
| sphinx | spider | spider_monkey | spinach | sponge |
| spoon | sports_car | squid | squirrel_monkey | stage_curtain |
| stainless_cup | stamp | stapler | star_fruit | starfish |
| steak | steam_locomotive | steering_wheel | stethoscope | stingray |
| stinkbug | stir_fried_noodle | stirrup | stockpot | stone_lion |
| stool | stop_sign | stork | strawberry | striped_hyena |
| stupa | submarine | suitcase | sundae | sunflower |
| sunglasses | sunhat | sunrise | supermario_hat | sushi |
| swab | swan | sweatshirt | sweet_pepper | sweet_potato |
| sword | swordfish | syringe | table_lamp | table_tennis_bat |
| tachometer | taco | tadpole | tambourine | tank |
| tape_dispenser | tape_measurer | tapir | tarantula | tea_egg |
| tea_kettle | teacup | teapot | teddy_bear | telephone |
| telephone_booth | television | temple_of_heaven | tench | tennis_ball |
| tent | teriyaki | tern | test_tube | thimble |
| thorntail | thread | throne | tibetan_mastiff | tick |
| tie | tiger | tiger_cat | tiger_shark | tights |
| toad | toast | toaster | toilet | toilet_paper |
| tomato | toolbox | tophat | torii | toro |
| tortoise | totem_pole | toucan | towel | toy |
| tractor | traffic_cone | traffic_light | train | train_station |
| tram | trash_bin | travel_bag | tree_frog | tripod |
| trombone | trout | truck | trumpet | trunk |
| tuba | tube_coral | tumeric | tuna | turban |
| turkey | turnip | turtle | tusk | tweezers |
| twin_tower | umbrella | umpire | unicorn | vacuum_cleaner |

Table 6: Train classes for FSS-1000 (3/4)

| | | | | |
|---|---|---|---|---|
| valve | van | vase | ventilation_fan | video_projector |
| violin | volcano | volleyball | vulture | waffle |
| waffle_iron | walking_stick | wallaby | walrus | wardrobe |
| wasabi | wasp | water_bottle | waterbuck | waterfall |
| watermelon | weasel | weighing_scale | welsh_corgi | west_highland |
| whale | wheat | wheelchair | whiskey_bottle | white_radish |
| wild_boar | wild_dog | wildebeest | wind_bell | windmill |
| window_screen | wine_glass | wine_rack | winter_melon | wok |
| wolf | wolverine | wombat | wood_apple | woodpecker |
| woodwind | wool | wrecker | wrench | yak |
| yam | yellow_hammer | yellow_pepper | yellowtail | yoga_mat |
| yorkshire_terrier | yoyo | yucca | zucchini | zongzi |

Table 7: Train classes for FSS-1000 (4/4)