TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

Few-shot object detection using self-supervised learning

Authors	Daniel Reisenbüchler, Daniel Sens, Onat Sahin, Rahul Parthasarathy Srikanth
Mentors	M.Sc. M. Sundholm (PreciBake)
	M.Sc. H. Belhassan (PreciBake)
Co-Mentor	M.Sc. Michael Rauchensteiner (Department of Mathematics)
Project Lead	Dr. Ricardo Acevedo Cabra (Department of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Department of Mathematics)

Jul 2021

Contents

1 Introduction

Nowadays, machine learning is being used in more and more companies to facilitate workflows or even to automate them entirely. PreciBake is a company developing AI solutions for the food-tech and baking industry. Their machine learning team is continuously working on developing and improving their ML algorithms for tasks such as image classification, object detection (OD) and many others. A typical use case is, for instance, the virtual baker [47], specifically designed for bakeries and restaurants. This oven is equipped with a camera facing its baking surface (for example as in Figure 2c) and an object detection algorithm used to automatically detect the class of food which needs to be heated. Therefore, users no longer have to worry about which program to set for the particular baking product. One of the biggest difficulties in developing OD models is the data collection and annotation process, as well as training process of the ML model. The procedure of annotating many images and subsequent training of the model is very time-consuming and cumbersome. Although the OD algorithms work very well, they are unfortunately sensitive to different task environments. The goal of our project is to adapt a generic object detection ML model which is invariant to non-stationary environments, for example, varying camera angles (e.g. Figure 2a and 2b), lighting conditions or a stove on which food is prepared instead of an oven. Another requirement the desired algorithm should satisfy is that it should be able to detect objects of new classes, while trained with only very few examples from these new classes. For instance, detecting a pizza when the algorithm was initially designed to just detect pastries such as breads or croissants. For now, the typical workflow of designing and deploying an OD algorithm is:

1. Install a camera facing the task environment and collect training data
2. Label hundreds of training and test images
3. Train an object detection model
4. Test and deploy the model

This process has to be repeated for every new task environment and therefore has to be re-iterated for new object classes to be detected.
Moreover, from the point of view of development of ML models, most of the existing object detection algorithms consist of two stages: a feature extractor (backbone) and a detection head. The backbone (e.g. a CNN) is the first stage of the whole pipeline, which processes an input image into a feature representation, from which various object shapes can be visualized. This feature representation of the specific image is forwarded to the detection head to regress bounding boxes (localization) and classify the objects contained in the regressed bounding boxes. The training process of the whole model (visualized in Figure 1) consists of:

(i) Using a pre-trained backbone (typically trained on large public datasets such as ImageNet [36]),

(ii) build a detection head on top of the feature extractor and train the whole model on a task agnostic object detection dataset.

(iii) In the last step, the whole model is finetuned by training with a dataset obtained from the target environment.

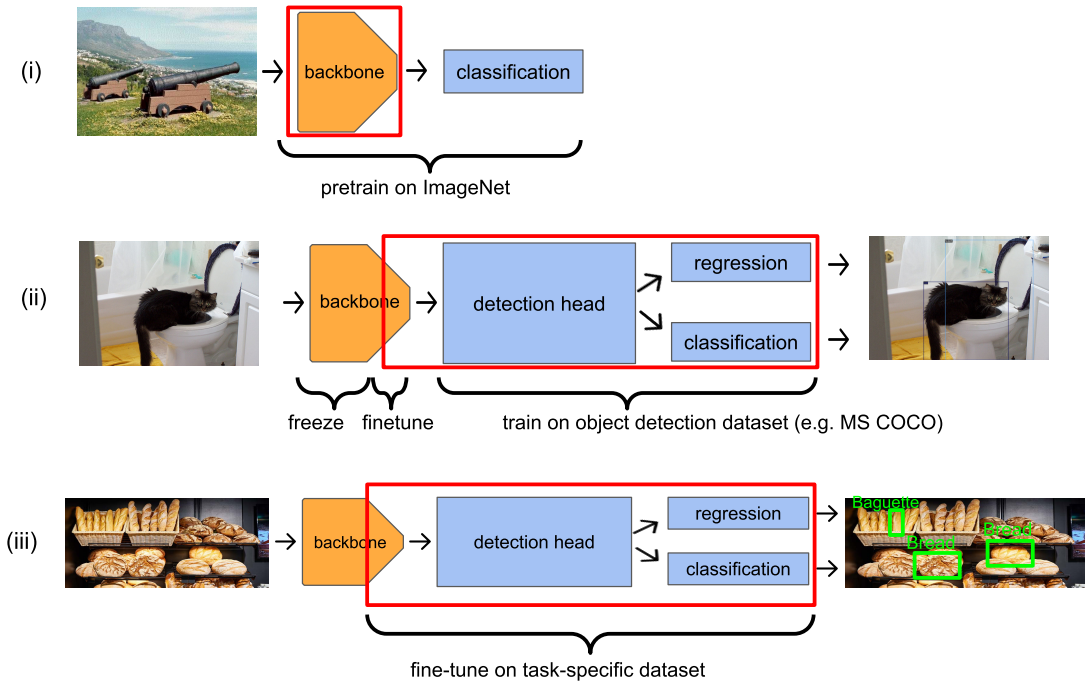
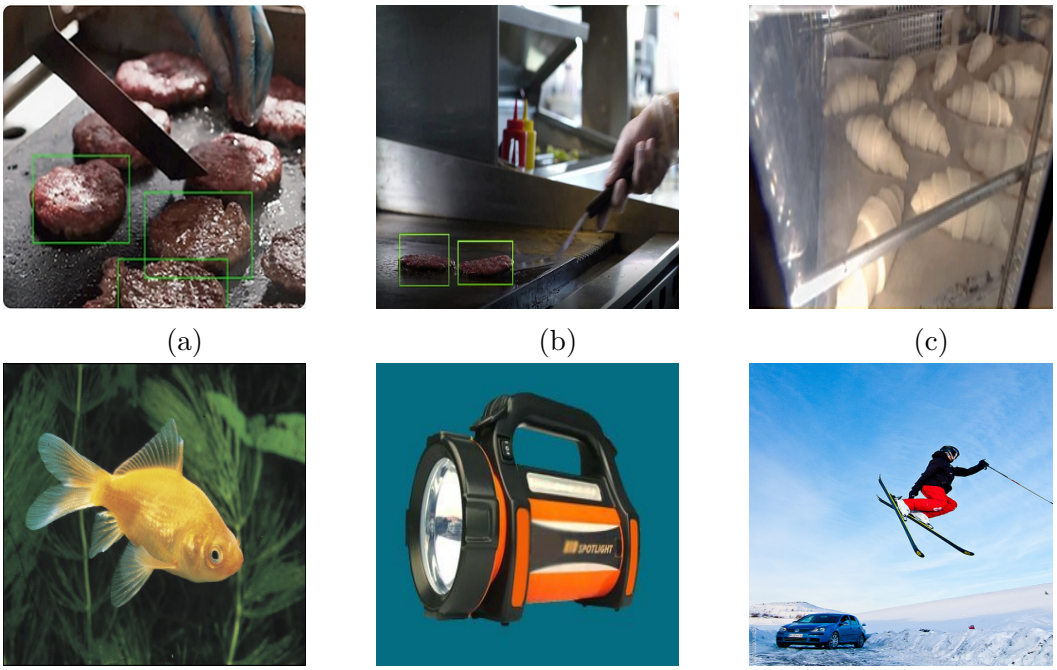


Figure 1: Visualization of the training process (i) - (ii) - (iii). The image in the first row is from ImageNet dataset, while the one in the second row is from MS COCO dataset and the one in the third row is taken and adapted from [38].

Problems in step (i):
Despite training with a finetuning strategy in (iii), the first layers of the backbone are frozen and therefore trained on a different dataset. Moreover, if the detection head should adequately work with just a few labeled target images, the backbone will never be trained on target data (training in step (iii) will be omitted in the whole pipeline later) and thus suffer from a domain shift. The first row of Figure 2 illustrates possible images from target datasets and the second row shows images from ImageNet. When contextualizing both datasets, one can recognize that the objects in the images of the target dataset have very similar shapes and colors, while on the other hand the objects in the images from ImageNet are very different in terms of shapes and colors. This may lead to the fact that the classifier of the detection head is not able to discriminate various similar objects. To overcome this issue, the feature extractor could be trained in supervised fashion on the target dataset, which would require a large number of data annotations. In addition, if the backbone needs to be adapted to a new target environment (e.g. when collecting data from a new customer), even more annotating and an elaborating data maintaining process would be necessary.

Solutions for step (i):
To overcome these issues, the feature extractor can be trained in a self-supervised manner, which requires no labels. Furthermore, as self-supervised training incorporates various data augmentation techniques, such as random cropping and blurring, the backbone

becomes more robust to domain shifts. In this project we investigated different self-supervised learning strategies and the ability to transfer pre-trained weights of the feature extractor to the downstream task of object detection. In the case of an insufficient amount of target data, this approach also brings the flexibility of using more related datasets without worrying about things like whether the dataset is annotated or not, amount of images in this dataset that require labeling, the annotation formats (e.g. json/csv), etc. We also studied and implemented another state-of-the-art backbone architecture, Swin Transformer [26], which is different from the standard practices that commonly use CNNs.



(a) (b) (c)

(d) (e) (f)

Figure 2: Data investigation: The first row visualizes images from target datasets of PreciBake [38] and the second row illustrates images from ImageNet.

Problems in step (ii) and (iii):
Steps (i) and (ii) are done once in total, only step (iii) will be re-iterated for every new customer. Even when just finetuning the whole model, this process requires hundreds of labeled images to yield high accuracy (according to the PreciBake-Team). The naïve approach of finetuning the whole model with just a few images instead of hundreds was a priori tried by PreciBake. It turned out that the model overfitted the training data and did not lead to satisfactory results.

Solution for step (ii) and (iii):
To circumvent these problems, we investigated in ML models specifically designed for object detection with the constraint of a sparse amount of training data. This setting is dubbed as *Few-shot object detection* and is an emerging area of research. The current approaches extend existing object detection algorithms to this particular setup. In general, most methods compare the objects contained in the few annotated examples with the

objects detected in a new image. The model needs to be trained once as done in step (ii), while step (iii) is not necessary anymore and can be omitted. The only thing that needs to be done for every new customer is to label few images that contain objects from the target classes and provide them to the model for comparison. Therefore, such a model could potentially be trained once for any object detection task without a huge labeling effort and then configured to a specific task environment with only a few descriptive examples.

In summary the process (i) - (ii) - (iii) would change to:

(I) Pre-train the feature extractor on a target/similar dataset or an union of both, utilizing self-supervised learning

(II) Train the detection head on a task agnostic dataset (e.g. MS COCO) only once

(III) For every task new environment: provide a few annotated examples

It is our belief that this approach could overcome the domain shift between the task environments, significantly reduce the labeling procedure and quickly adapt to the new task environment.
For an overview and introduction of involved ML topics, we describe self-supervised learning, object detection and few-shot object detection in general in chapter 2. In chapter 3, we further describe the specific algorithms and model architectures used in this project. The Experiments and Results chapter 4 describes how we utilized self-supervised learning to pre-train a feature extractor. We implemented a full object detection pipeline and assessed the ability of transferring backbones trained with self-supervised learning to object detection. We also describe the challenges we faced and compare our results with models trained with supervised learning. This first line of experiments are related to (I). Afterwards, we extend the object detection pipeline to the few-shot object detection setting by using a different detection head. The few-shot object detection experiments are related to (II) and (III). Unfortunately, we could not test this approach using data from PreciBake since current self-supervised methods need much more computational resources than available to us. We provide results for backbones pre-trained with self-supervised methods on ImageNet dataset and assess object detection as well as few-shot object detection using the MS COCO dataset.

2 Related Work

Object detection is a sub-field of computer vision, which aims to localize and classify objects in images. Despite utilizing various architectures and training techniques, most object detection algorithms [34, 17, 59, 44] are based on the same pattern - a backbone for feature extraction and subsequently a detection head for localization and classification of objects. The detection head operates on the feature representations obtained through the backbone. Therefore the selection of a suitable feature extractor is crucial. In modern object detectors, most backbone architectures are based on CNNs, for instance EfficientNet [40, 41] in EfficientDet [42], and ResNet [16], which is the standard architecture in benchmarks such as MS COCO [25] and PASCAL [10]. Recent advances [26, 55] suggest

a modeling shift from CNNs to Vision Transformers [9] on tasks such as object detection and semantic segmentation. Swin-T [26] and Pyramid-T [48] have proven to be suitable Transformer based candidates. Even though it is common practice to use ResNets in object detection, the best performing object detection model [7] on MS COCO benchmark is built on top of a Vision Transformer, suggesting to not underestimate the power of Transformers in object detection [29]. Moreover, backbones can be extended by methods such as feature pyramid networks (FPN) [23] and U-Nets [35], which leverage hierarchical feature representations obtained from different stages in backbone architectures and consequently facilitate the object detection task.
Modern detection heads can be divided into two general approaches: one-stage detectors and two-stage detectors. The first line of work [33, 32, 2, 60] investigates proposal-free detectors which does not explicitly generate proposal boxes but directly predict the class confidence scores and bounding box coordinates over a dense grid. On the other hand, two-stage detectors [34, 3, 22, 8] extract class-agnostic region proposals of potential objects from a given feature representation. These proposed boxes are then further refined and classified into different categories. However, these detectors are trained in a supervised fashion and they still enormously rely on large amounts of annotated training data. For that reason, most of these algorithms suffer from huge performance drops [15] when just a few labeled images are available as in the few-shot object detection tasks.

Few-shot object detection refers to the task of recognizing objects of novel classes in an image, where only a few labelled instances are available for each class. The model is trained on a base dataset with abundant labelled instances and is expected to recognize novel objects at test time, of which only a small support dataset is available. The key point in this setup is that the support set only contains objects that were not present in the base dataset, so that the model cannot obtain prior knowledge about the support set from the base dataset. If there are a total of K labelled support images for each of the N novel classes, the problem is called N-way K-shot detection.
Recently more effort have been put into the study of few-shot object detection (FSOD). Prior to this, most work tended to focus on few shot classification where the localization of the object is not important. This makes FSOD inherently more complex. Most works attempting to address this task can be formulated in two paradigms: Transfer-learning-based and Meta-learning-based [50] approaches. Methods that use transfer learning include PNPDet[58], TFA [49] and MPSR [51], where novel concepts are learned via finetuning. In contrast, methods that use meta-learning extract meta-level knowledge that can efficiently adapt to new categories by constructing and learning various auxiliary tasks, in which target categories are dynamically conditioned on support images. These include Meta-YOLO [19] and ONCE [31], which are based on single-stage detectors, and Meta R-CNN [56], which is built on Faster R-CNN [34]. FSOD with Attention-RPN and Multi-Relation Detector [11] aims to learn good priors for novel objects with an Attention-RPN. However, there is still a major challenge. Most of the current benchmark datasets contain classes which are very different in shape and color, which eases the task of distinguishing between multiple instances in an image. In most real-world applications, however, the task is to distinguish between objects that look very similar, as we described in the problems of step (i). Following [43] which shows that learning good feature representations is the key to achieve high performance in few-shot image classification, we searched for an

efficient way to obtain a good embedding without relying on labels.

Self-supervised learning (SSL) is a form of unsupervised learning. In this work, we use the term SSL, in the sense of *not supervised by human-annotated labels*. SSL strategies generally incorporate two facets: pretext tasks and loss functions. The phrase *pretext* entails that the task being solved is not of particular interest, but is solved only for the true intention of learning a good feature representation. Recent studies [18, 54, 55] focus on designing various pretext tasks, where one of the most promising trends among them is contrastive learning [30]. The general idea of contrastive learning is to transform one image into multiple views using data augmentation techniques, minimize the distance between views of the same image while maximizing the distance between views from different images in the training set. This idea can be thought of as building dictionaries. The keys in the dictionary are sampled from data (multiple views of an image) and represented by an encoder architecture. The encoders are trained in a self-supervised fashion to perform dictionary look-up. An encoded image should be similar to its matching key and dissimilar to other keys. A main purpose of SSL is to learn visual representations and features, that can be transferred to downstream tasks such as image classification and object detection by finetuning.

3 Methods

This chapter describes the Transformer architectures we used as feature extractors, the architecture of a feature pyramid network as backbone extension, followed by an overview of three self-supervised training strategies, we used to train the backbones. Since self-supervised training is intractable with our computational resources, we used the weights of self-supervised pre-trained models to initialise the backbone models. This process is related to step (I). To overcome the problems in step (ii) and (iii), we further outline a standard detection head *Faster R-CNN* designed for object detection. The architecture of the Faster R-CNN architecture serves as a base for the *Attention RPN and Multi-Relation Detector*, this detection head is specifically designed for the FSOD setup.

3.1 Backbone architectures

The following three backbone architectures were selected with respect to their performance in self-supervised training. In [4] we first discovered that Vision Transformer yields promising results on classification tasks when trained self-supervised. However, as explained later in chapter 4, these results did not translate well to object detection. We then sought another transformer architecture that performs better as a backbone for object detection and discovered the Swin Transformer. We also used the Swin Transformer in combination with an FPN which leverages the hierarchical feature maps of the Swin Transformer. Experiments were also conducted with self-supervised ResNet50 backbones, using the self-supervised learning techniques of DINO and DetCo. For more details about ResNets we refer to [16].

3.1.1 Vision Transformer

The Vision Transformer (ViT) [9] is a standard Transformer Encoder [46] applied to images. It expects a 1-dimensional sequence of token embeddings as input. To enable processing of 2-dimensional images, an image is first divided into patches of a fixed size. Each patch is then transformed into a 1-dimensional vector using a trainable linear projection. In this way, image patches can be treated in the same manner as tokens (words) in natural language processing (NLP). Similar to NLP, where the tokens represent word embeddings, in this setting a vectorised patch is called a patch embedding. A vector of zeros (of same size) is appended to the front of the sequence of all patch embeddings. This [class] token serves as a learnable embedding and whose state after the last attention layer is used to perform classification by attaching a classification head to it. Positional embeddings are added to the patch embeddings to preserve the position information, and the resulting sequence of embedding vectors is used as input for the Transformer encoder.

3.1.2 Swin-Transformer

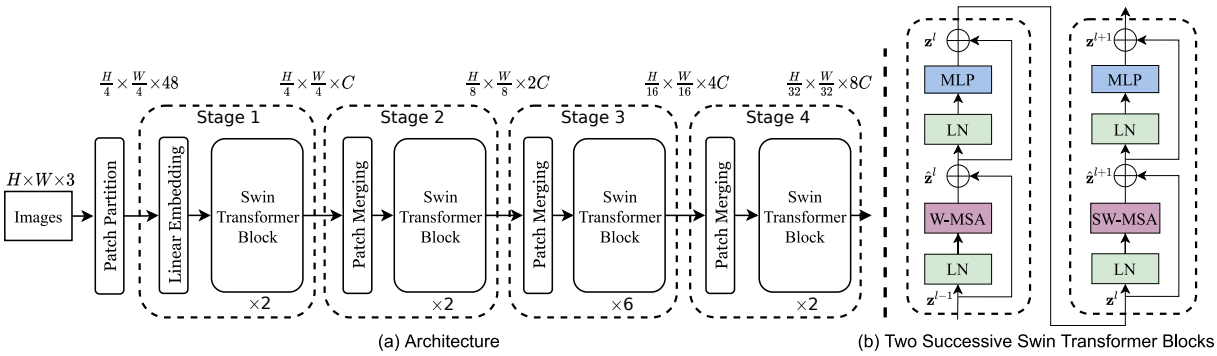


Figure 3: **(a)** The architecture of the tiny version of the Swin Tranformer. **(b)** Schematic structure of a Swin Transformer Block. The first attention layer is a windowed multihead self-attention (W-MSA) and the second one is a Shifted W-MSA (SW-MSA). Figure taken from [26].

The Swin Transformer [26] (illustrated in figure 3) first divides an input image into non-overlapping patches of equal size. The raw pixel values of each patch are then concatenated to create a 1-dimensional feature vector. A linear embedding layer is then applied to this feature vector to project it into a latent state with vector size C. The series of latent patch embeddings serves as input for the first Swin Transformer block.
Instead of the global self-attention, which calculates the relationships between all other patches, here the self-attention is only calculated within fixed-size windows that divide the image evenly and without overlap. In order to enable the flow of information across neighboring windows, an additional attention layer is applied. In this layer, the window configuration of the previous layer is shifted, resulting in a new window partitioning of the image. The self-attention calculation for the new partitioning crosses the boundaries of the previous layer, creating connections between them.
To create hierarchical feature maps, a patch merging layer is applied before each subsequent Swin Transformer block. It reduces the number of patch embeddings, which lowers

the resolution of the feature map at this stage of the network. In this layer, features of neighbouring patches are concatenated into a $4C$-dimensional vector and a linear layer with an output size $2C$ is applied. This samples down the resolution of the feature map in height and width by 2 each, and doubles the length of the remaining vectors. After each patch merging layer, another Swin transformer block is applied and together they form a stage. The output feature maps of each stage together form the output of the entire network.

3.1.3 Feature Pyramid Network

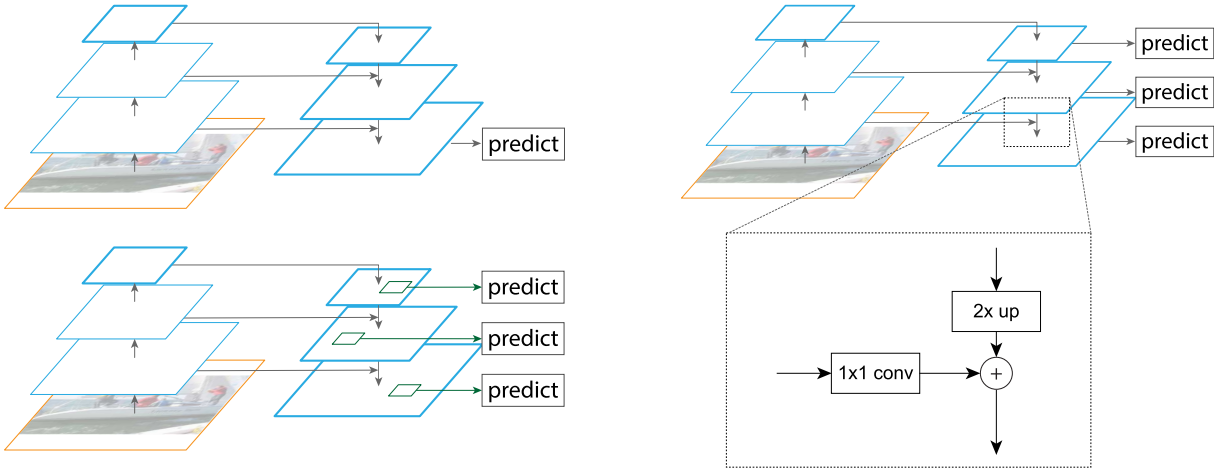


Figure 4: **Left:** Top figure illustrates the common top-down architecture where predictions are made at the fines level. Bottom figure illustrates the *feature pyramid* architecture where predictions can be made at all levels. **Right:** Building block showing the lateral connection and top-down pathway, merged by addition. Figure taken from [23].

Feature pyramid networks (FPN) [23] has become a important component in recognition systems for detecting objects at different scales. As an intermediate module between the backbone and detection head, FPNs leverage the pyramidal shape of a convolutional neural network's (the backbone) feature layers to create a feature pyramid that has strong semantics at all scales. To achieve this goal, the architecture combines low-resolution, semantically strong features with high-resolution, semantically weak features via a top-down pathway and lateral connections (Figure 4, right side). Improving the representation learning across different levels of the backbone (e.g. CNNs or Swin-T) can benefit from scale-awareness for object detection.

3.2 Self-supervised learning strategies for the backbone components

3.2.1 DINO - SSL for ResNet and Vision Transformer

DINO [4] is a self-supervised training technique primarily proposed for pre-training Vision Transformer for downstream tasks such as classifications and is also extendable to other

architectures like ResNet. The method gathers the idea from knowledge distillation networks, where the student network learns to produce the output embedding that is closer to the output from the teacher network and hence derives its name Self-**Di**stillation with **No** labels. From a given input image, different views of the image are constructed using a multi-crop strategy [5] to cover the larger global views of the image, x_1^g and x_2^g (covering more than 50 percent of the image area) as well as the local views of the image (covering less than 50 percent of the image area). The teacher only receives the global views of the image as the input, however the student receives both the local as well as the global view of the image.

The intuition behind this trainings strategy is that the student must learn to output an embedding from local views with lesser context of the image to match that of teacher networks which gets as its input a much larger context of the image. Another aspect of the DINO training is that the teacher network does not have gradients propagated from the loss functions but rather uses an exponential moving average of the student's weights, for instance, by using a momentum encoder network [18].

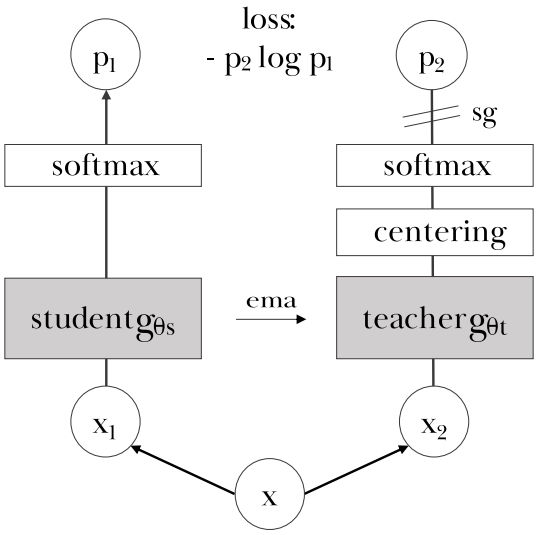


Figure 5: The student and teacher network have the same architecture. The augmentations x_1 and x_2 represent the different views of the training image which are passed as inputs to the networks. The centering used in the teacher network helps in preventing collapse to trivial embeddings. In addition, a softmax function helps the network to define and learn unsupervised class definitions. Figure taken from [4].

3.2.2 DetCo - SSL for ResNet

DetCo - Unsupervised Contrastive Learning for Object Detection [54] proposes a training framework for feature extractors, that is designed for transferring well to the downstream task of object detection, in contrast to other SSL methods such as DINO or MoCo [18]. These have primarily produced state of the art results in self-supervised image classification. It inherits most of the components from MoCo v2 and extends by introducing a multi-stage contrastive loss. In addition, it also takes advantage of further locals views

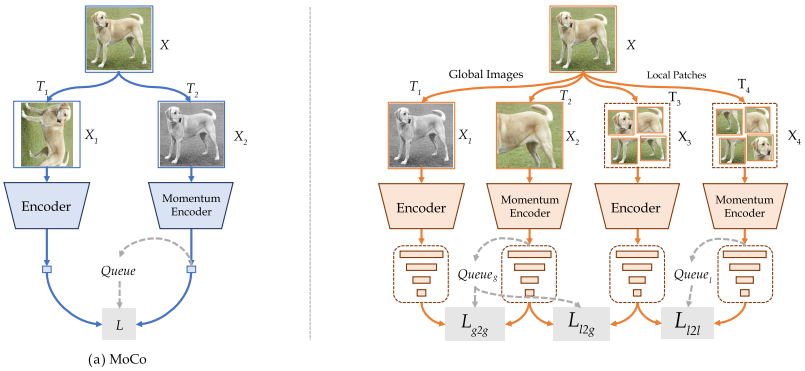


Figure 6: **Left:** The MoCo self-supervision framework which serves as the baseline for DetCo **Right:** Illustrates the DetCo framework. It improves on MoCo by using the cross global-local contrastiveness and intermediate hierarchical losses. Figure taken from [54].

as the input to the network. Each stage of the network aims to minimize the local-local, local-global as well as the global-global losses. Thus the loss function for the i-th stage of the network is:

$$L(I_q, I_k, P_q, P_k) = \sum_{1}^{4} w_i (L^i_{g\leftrightarrow g} + L^i_{g\leftrightarrow l} + L^i_{g\leftrightarrow l})$$

where I represents the global image patches and P the local patch sets.

The intermediate contrastive loss helps the network to learn the higher and lower level semantic features of the image. This is crucial for object detection, since most SOTA object detection networks use features from multiple scale. For instance, Feature Pyramid Networks in Faster R-CNN based networks or single stage detectors such as RetinaNet [24]. The cross global and local contrastive loss increases the representational ability of the network, since it is forced to learn the global image features with less contextual information present in the local patches.

3.2.3 MoBY - SSL for Swin Transformer

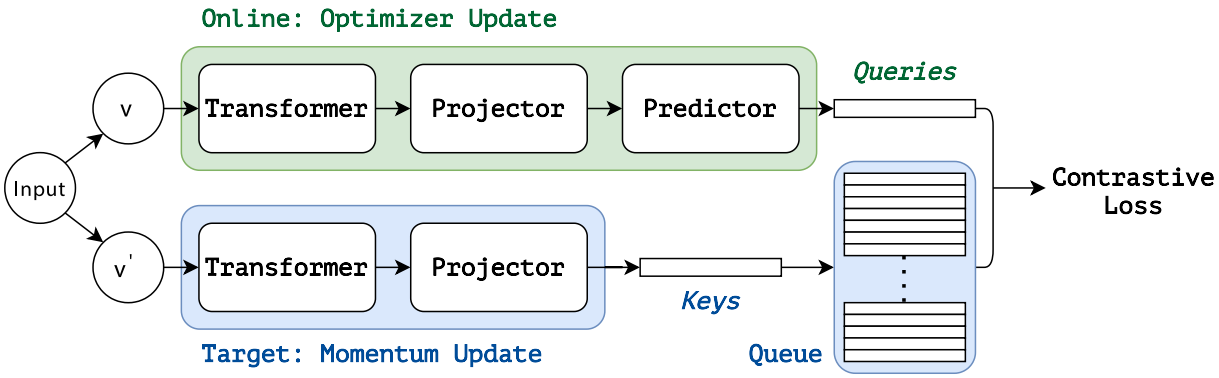


Figure 7: Visualisation of the MoBY training procedure. Figure taken from [55].

MoBY [55] is a self-supervised learning approach especially suited for self-supervised training of Swin-Transformers. By combining techniques from **Mo**Co v2 [6] and **BY**OL [14], it inherits the momentum design, the key queue, and the contrastive loss used in MoCo v2, and inherits the asymmetric encoders, asymmetric data augmentations and the momentum scheduler in BYOL. (ref to figure) The MoBY approach incorporates two encoders, an online encoder and a target encoder. Both encoders consist of a backbone (Swin-T) and a projector head (2-layer MLP). The online encoder is equipped with an additional prediction head (2-layer MLP), which makes the encoders asymmetric. Both networks receive a different data-augmented view of an image and the target network updates a dictionary containing previous feature representations. A contrastive loss is used to learn the representations. For the feature representation of the online encoder q, its contrastive loss is computed as

$$L_q = -log\frac{exp(q \cdot k_+/\tau)}{\sum_{i=0}^{K} exp(q \cdot k_i/\tau)}$$

where k_+ is the target feature for the other view of the same image, k_i is a target feature in the key queue (dictionary), K is the size of the key queue and τ is a hyperparameter (temperature term). While the online encoder is learned by gradient propagation from losses, the target encoder is updated by using the moving average of the online encoder weights with momentum mechanism.
The resulting trained weights are well suited for transferring to downstream tasks of image classification and object detection. Therefore this training strategy is of particular interest for our work.

3.3 Faster R-CNN

The Faster R-CNN [34] belongs in the category of two-stage detectors. It can be decomposed into two networks, one to generate class-agnostic region proposals and one to refine and classify them into different classes. The former is the Region Proposal Network (RPN) and the latter is the Fast R-CNN detector [12]. The RPN, which is the main contribution of the Faster R-CNN, first takes an image of arbitrary size as input and feeds it into the backbone. The output of the backbone can be either a single feature map or a set of hierarchical feature representations with the same number of channels. To generate region proposals, a small network slides over the output feature map(s) of the backbone. This small network takes as default input an 3 x 3 spatial window at its current position of the input and maps each window to a low-dimensional feature. This feature is then further processed in parallel by two fully connected layers, one applying a box-regression loss and the other a classification loss. The classification layer estimates the probability that the generated proposal contains an object, which is called the objectness score. The regression layer outputs a 4-dimensional vector parameterised (following [13]) to yield 4 coordinates belonging to a reference box called an anchor.
An anchor is centred at the current position of the sliding window and is associated with a scale and aspect ratio. At each location of the sliding-window, the RPN simultaneously predicts multiple region proposals for a predefined set of anchors. By default, all proposals with an Intersection over Union (IoU) score greater than 0.7 are recognized as positive proposals and the top 2000 proposals with the highest objectness score are then passed to the Fast R-CNN detector for box refinement and classification.

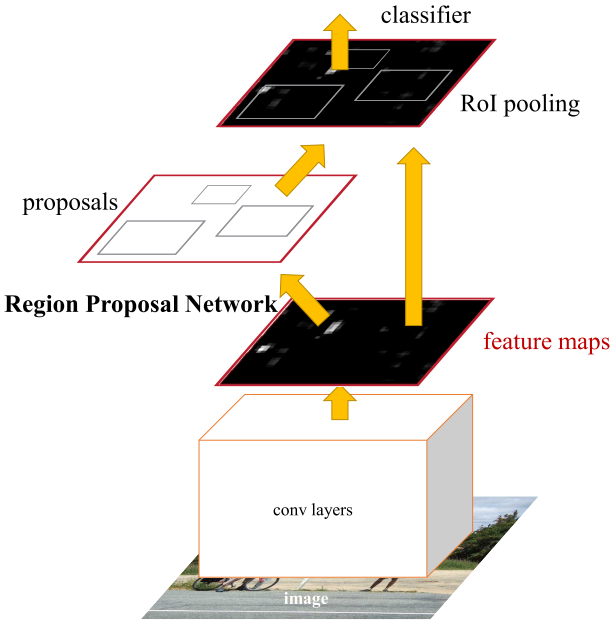


Figure 8: Illustration of the Faster R-CNN components. The conv layers can be replaced by any backbone architecture. The RPN computes class-agnostic bounding boxes and the corresponding features of the input feature map are then passed to the classification head, which is a Fast R-CNN. Figure taken from [34].

The architecture described here was proposed in the original paper and is also known as Faster R-CNN C4, which refers to the fact that features from the final convolutional layer of ResNet50 backbone's 4th stage is used as the input to the RPN. Another variant of this architecture, presented later in [23], is the Faster R-CNN FPN. It uses a feature pyramid network as an extension to the backbone to compute hierarchical feature maps that serve as input to the RPN instead.

3.4 Attention-RPN and Multi-Relation Detector

Extending a Faster R-CNN to the FSOD setting is not an easy task, as the performance of the RPN would suffer greatly from the lack of labelled instances. The objectness score of potential positive proposals would be very low and thus novel objects would be overlooked or many false positive proposals would be generated. Motivated by the matching network [20], [11] tries to overcome this problem by learning an attention RPN. For each class in the support set, it adds a so-called support branch to the query branch, into which the input image is fed.
During training, the support images and the query images are passed through a weight shared backbone that produces feature maps from the support images and the query image. For each support class, the average feature map over all support images is used as support feature map. A separate similarity feature map is computed for each support feature map using a non-learnable function that takes the support feature map and the query feature map as input. These similarity maps form the input to the attention RPN. The main difference is that each similarity map contains support information of the respective class, which allows the Attention RPN to search for proposals for exactly that

class. The proposals are therefore no longer class-agnostic and thus no longer need to be classified, but only verified. These similarity feature maps form the input to the RPN.

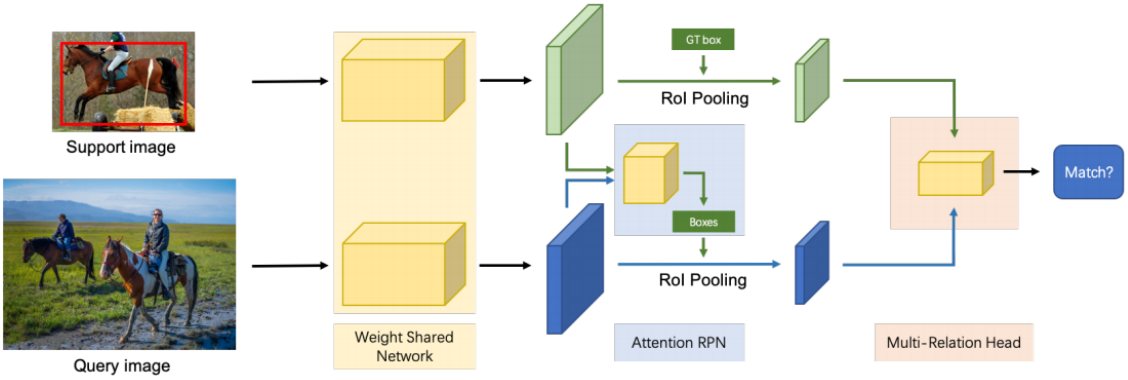


Figure 9: Schematic illustration of the Attention RPN and Multi-Relation Detector Model. Figure taken from [11].

The Attention RPN is followed by the multi-relations detector which consists of a local, a global and a patch relation module. Each module learns to relate support features to the proposals from the attention RPN, which are extracted from the query feature map. Every module first computes a separate feature map with its separate non-learnable function using the support feature and query features of the respective proposal. The calculated feature maps are then used as input for the respective head and the sum of the output matching scores of the individual heads is then used as the final matching score. A separate box regression layer is added to the patch relation head to perform bounding box refinement. The network is trained with a two-way contrastive training strategy to help distinguish between different classes and to avoid the relation detector learning to output high match values for all inputs, which is a common phenomenon called collapse. For more details about the training strategy, we refer to [11].

4 Experiments and Results

In the first part of the experiments, we examine different self-supervision techniques for pre-training the backbones using unlabelled data. We initialised a *ResNet-50* (CNN) and a *Swin-T* (Transformer) as backbones with weights from models trained using self supervision techniques as described in section 3.2. We also further extend the backbones with a *FPN* to improve detection at different object scales. To evaluate the performance of these self-supervised pre-trained backbones in object detection, we conducted experiments using *Faster R-CNN* as detection head. This first line of experiments therefore address the steps (I) and (II), described in the introduction.
In the second line of experiments, we examine the performance of self-supervised backbones for few shot object detection (chapter 4.2). We use *Attention-RPN and Multi-Relation Detector* as detection head and study the transferability of the best performing self-supervised backbone from the previous experiment. Attention-RPN and Multi-Relation Detector architecture is an extension of Faster R-CNN. It uses a Faster R-CNN

Schedule	Batch Size	Base LR	LR Steps	Num. of Iter.	Num. of Epochs
1x	16	0.02	(60000, 80000)	90000	∼12
Ours	4	0.002	(60000, 120000)	150000	∼5

Table 1: Detectron2's default 1x training schedule and our own shorter schedule, used in ResNet50 + Faster R-CNN experiments.

as a query branch with additional support branches, and replaces the RPN with the proposed Attention-RPN. Because the architectures are implemented in a similar way, it allow us to train both of them in a similar setting, while also allowing us to use the same backbones used with Faster R-CNN directly. This experiment is in accordance to step (III) of our proposed approach.
To build our models, we used the Detectron2 framework [52]. This library, published by Facebook AI Research, includes many state of the art object detection and segmentation methods. All modules from Faster R-CNN are pre-implemented in Detectron2 and an implementation of the detection head Attention-RPN and Multi-Relation Detector is available in the authors github repository. The results of our experiments are reported using variants of average precision (AP), which is the common used metric in the MS COCO benchmark. We further describe the AP metrics in the Appendix, section 6.

4.1 OD experiments (Faster R-CNN)

Few-shot object detection is more demanding than object detection. Therefore, we decided to assess the transferability of self-supervised pre-trained backbones to downstream tasks as well as building the whole machine learning pipeline for the easier object detection task. Among various detection heads we decided to use Faster R-CNN, as it is the base architecture of the detection head Attention-RPN and Multi-Relation Detector later used in FSOD and thus very similar. We focused on two different backbone variants, a CNN architecture (ResNet50) and a Transformer (Swin-T) architecture. Both are comparable in terms of the amount of parameter, inference time and complexity. At the end of this chapter we compare the results on object detection, obtained by holding the detection head fixed while using different backbone architectures and initialisation, through results on a 12 epoch training on MS COCO dataset using AP metrics. We used the best performing backbone equipped with self-supervised weights for our next studies in FSOD.

4.1.1 Experiments with ResNet-50 Backbone

Initial experiments in this section are done with the Faster R-CNN C4 architecture from Detectron2's model zoo. The GPU used is NVIDIA TITAN V with 12 GB VRAM. Due to time constraints, it was not feasible to train models with this setup using Detectron2's default training schedule, which is called "1x" and runs for 12 epochs. This schedule will be mentioned as 1x for the rest of this section. Because of this infeasibility, it was necessary to create a shorter training schedule. The proposed training schedule (∼5 epochs), along with the default 1x schedule can be seen in Table 1. With this setup, training for 1 MS COCO epoch takes ∼11 hours.

Backbone	**Initialization of backbone**	AP	AP_{50}	AP_{75}
ResNet50[53]	supervised	28.316	49.198	28.704
ResNet50	self-supervised (DINO)	11.270	24.328	8.965
ResNet50	self-supervised (DetCo)	22.393	39.260	22.455

Table 2: Results of ResNet50 backbone + Faster R-CNN experiments using the MS COCO dataset. Our own training schedule, described in Table 1 is used for these experiments.

Initially, ResNet50 backbone trained with supervised training (Detectron2's default pre-trained ResNet50 weights) and with DINO self-supervised training (weights taken from DINO's Github repository) are used as backbone to train with our Detectron2 training schedule. Validation set performance of these experiments can be seen in Table 2. AP values of the model, initialized with the DINO pre-trained backbone, are significantly lower than the one initialized with supervised backbone.
We also observed that the work proposing DINO self-supervision method reports image classification results with ResNet50 as well as Vision Transformer backbone, while reporting results on other downstream tasks such as image retrieval and segmentation only using Vision transformer backbones. This also helps us hypothesize that the performance obtained on classification using DINO trained ResNet50 backbones do not transfer to the object detection. The reason for this is that DINO (and many other similar contrastive self-supervised methods) depend on contrastive losses between global embeddings of two different random crops and augmentations of the original image, created with different sub-networks (student and teacher sub-networks, in DINO's case).
In order to further ascertain this claim, the features maps of the last 3 layers (out of the 5 layers) of the ResNet50 backbone, that is trained supervised and using the DINO technique are visualised in Figure 10. It is apparent that self-supervised DINO trained backbone does not preserve the locality of the possible objects in the images for generating the embedding for downstream tasks, since it is not crucial for image classification. The feature maps obtained from the supervised trained backbones show that the network preserves the relative spatial locality of the objects. This also suggests that the backbones trained with self-supervision for classification do not provide the same performance for object recognition.
The findings in the DetCo study state that, using local crops of the original image in this way causes the loss of contextual information of the image, while improving the lower bound of mutual information, which results in better contrastive learning. In other words, forcing the network to produce a similar embedding for different local crops, while complementing classification performance, removes spatial context in the image, which is crucial for object detection. For these reasons, the next experiment is done using a backbone pre-trained with DetCo, which claims to be a self-supervised approach specialized in the object detection downstream task. Moreover, it aims to preserve the spatial context of images by using local patches and global images together, along with embeddings from both final and earlier layers of the sub-networks. From the results in Table 2, it is obvious that DetCo is a huge step-up compared to DINO, and also the best self-supervised training approach for ResNet50 that we have experimented on.
The same three experiments are repeated on the PASCAL VOC 2007 dataset, to further

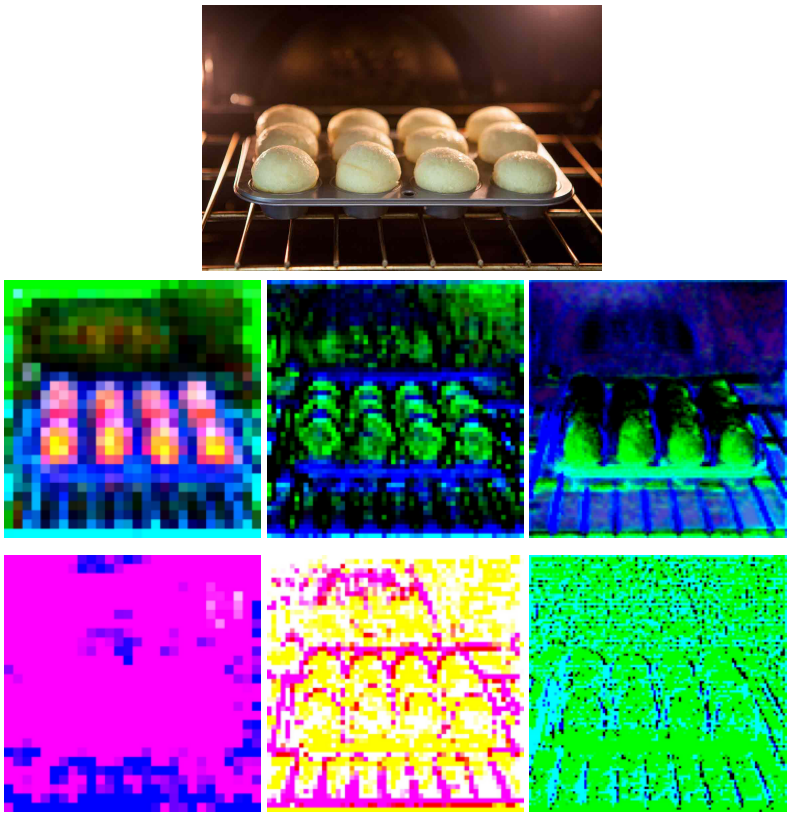


Figure 10: **Top:** The input image fed into the ResNet50 backbone network **Middle:** Visualisation of the feature maps obtained from supervised backbone **Bottom:** Visualisation of the feature maps obtained from DINO trained self-supervised backbone

validate the results obtained on the MS COCO dataset, and results are comparable in terms of performance gaps between DINO and DetCO. Figure 11 shows graphs for both MS COCO and PASCAL VOC 2007 for comparison. The main reason for doing the PASCAL VOC 2007 experiments, other than further validating the results obtained with the MS COCO dataset. Moreover, it is possible to try new ideas and experiments quickly with brief training times, since PASCAL VOC 2007 is a very small dataset with only 5511 training images, compared to MS COCO with 118287 training images. In this analysis, we again observe that the network using a backbone initialised with weights obtained through DetCo performs better than the DINO based backbone.
After validating the observations obtained using the MS COCO and PASCAL VOC 2007 dataset, the next experiment was done by extending the DetCo pre-trained backbone with FPN (referred as Faster R-CNN FPN), instead of using Faster R-CNN C4. The idea behind this experiment is that adding a FPN to the architecture could achieve even better results than the previous results acquired with DetCo approach by providing the detection head strong semantics from all scales of the backbone. However, the results obtained with PASCAL VOC 2007 dataset, reported in Figure 12, show that the performance is slightly lower than previous PASCAL VOC 2007 results. The main suspicion is that just training with 5511 images might not be enough to observe the effects of FPN with a self-supervised pre-trained backbone. This intuition is also supported by the fact that DetCo paper does not report any Faster R-CNN FPN results on PASCAL VOC 2007, even though there

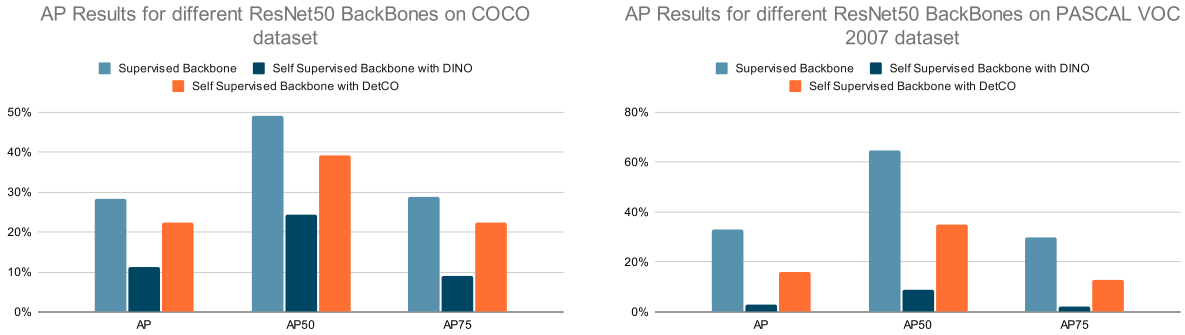


Figure 11: Results of the experiments done with backbones using supervised, DINO, and DetCo methods. Our training schedule from Table 1 is used for all of these experiments. **Left:** Results on MS COCO dataset. **Right:** Results on PASCAL VOC 2007 dataset.

Backbone	**Initialization of backbone**	AP	AP_{50}	AP_{75}
ResNet50 + FPN[53]	supervised	37.931	58.839	41.05
ResNet50 + FPN	self-supervised (DetCo)	30.181	50.268	31.898

Table 3: Comparison between Detectron2's baseline results with their pre-trained ResNet50 + FPN backbone and our experiment with the same backbone initialized with DetCo pre-trained weights on Faster R-CNN. The AP values shown are validation results after a full 1x training schedule, detailed in Table 1. The only difference for DetCo's training is that Base LR is reduced to 0.002 to prevent diverging loss.

are results reported on Faster R-CNN C4. We also think this could be also justified by the bias-variance trade-off, since the FPN introduces more complexity into the whole network. To better evaluate the effect of FPN, the same experiment is repeated with the MS COCO dataset. Due to time constraints, running a full experiments with this setup was not possible (∼55 hours for 5 epochs), but there was a clear advantage from the very beginning: Faster R-CNN FPN is able to provide close performance to Faster R-CNN C4 in terms of AP, with a much shorter training time. Training Faster R-CNN FPN for one MS COCO epoch takes ∼4 hours, compared to ∼11 hour MS COCO epoch training time of Faster R-CNN C4. Even if it does not reach the Faster R-CNN C4's performance in the same number of epochs like seen in the PASCAL VOC 2007 experiments, it's possible to train for more than double the number of epochs in the same time. In fact, with this training speed, it is feasible to train for a full 1x schedule.
Therefore, we conclude that Faster R-CNN FPN with a ResNet50 backbone pre-trained with the DetCO approach is our best self-supervised model with a ResNet50 backbone, and trained that model for a full 1x training schedule. Our results of this setup is shown in Table 3, along with Detectron2's baseline results from the same model (with a ResNet50 backbone pre-trained with supervised training). Performance of our model comes quite close to Detectron2's baseline performance.

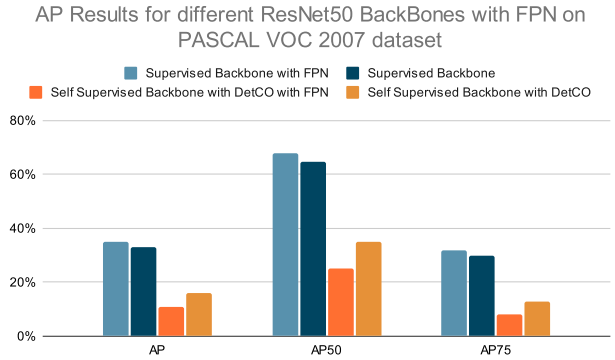


Figure 12: Experiments done with FPN on PASCAL VOC 2007, using our own training schedule from Table 1

4.1.2 Experiments with Transformer Backbone

Backbone architecture

The idea of using a Vision Transformer (ViT) as backbone came to us during the reading phase about self-supervised learning, while many studies about ViT's used as feature extractor were published during the time of this work. After the reading phase we started investigating in the outputs of the last layer of a self-supervised trained ViT using DINO. When feeding in an input image with the shape $[3, 796, 1200]$ into a pre-trained ViT (we used DeiT-8 variant [45]), the output looked very promising at the first glance (figure 13). For the visualization we extract the $[\,\text{class}\,]$ embedding from the output of the last attention layer of the ViT, which is of shape $[6, 14851, 14851]$. We reshape the $[\,\text{class}\,]$ embedding to a tensor of shape $[6, 49, 75]$ and apply a max operator on the first dimension. We then use interpolation to resize the height and width of the tensor to match those of the input image.

Figure 13: Left: An image feed into ViT. Right: Visualisation of the ViT output. The processed image is from [39].

However, the subsequent RPN in the detection head (Faster R-CNN) assumes an input-size with much lower height and width, but bigger channel size to come up with meaningful

region proposals. To circumvent this, additional deconvolutional layers or direct upsampling could be applied after the last self-attention module of the ViT. This approach is also investigated in [1], but with relatively low performance. We therefore decided to use Swin Transformer, which is originally designed as general purpose backbone for various computer vision tasks. To make a fair comparison with ResNet50 we used the tiny version (Swin-T) which has approximately the same amount of parameters and also 4 stages. In contrast to common CNNs, there is no pre-implementation of any variant of Vision Transformer in the object detection library detectron2, which we used within this project. Therefore we implementented a new backbone class from scratch by defining Swin-Transformer and also a FPN.
In more detail, given an image with tensor-shape $[C, H, W]$ processed through 4 self-attention blocks of the Swin Transformer results in the following output-shapes:

$$F_1^{in} = \left[96, \frac{H}{4}, \frac{W}{4}\right], F_2^{in} = \left[192, \frac{H}{8}, \frac{H}{8}\right], F_3^{in} = \left[384, \frac{H}{16}, \frac{H}{16}\right], F_4^{in} = \left[768, \frac{H}{32}, \frac{H}{32}\right]$$

The set of hierarchical feature representations on different semantic levels serves as input of a feature pyramid network $F_{FPN}^{in} = \{F_i^{in}\}_{i=1}^4$. The output-tensor

$$F_{FPN}^{out} = \left[\left[256, \frac{H}{4}, \frac{W}{4}\right], \left[256, \frac{H}{8}, \frac{W}{8}\right], \left[256, \frac{H}{16}, \frac{W}{16}\right], \left[256, \frac{H}{32}, \frac{W}{32}\right]\right]$$

of the FPN is obtained as described in section 3.1.3. F_{FPN}^{out} is the concatenation of all 4 output stages of the FPN. Thus, the output of the overall backbone contains the feature representation as the base for the detection head and therefore the input for the region proposal network. Compared to the output-shape of the ViT, F_{FPN}^{out} contains 4 tensors with higher channel size and lower height and width. Figure 14 illustrates the complete feature extractor and visualizes the outputs of the different stages and shapes for a particular processed image.
The visualization of stage 1 in figure 14 illustrates that contained objects are clearly separated from the background. This serves as a good basis for the subsequent detection of objects. Interestingly, the reflections in the background are clearly visible in the outputs of all stages, which could mislead the detection head to detect objects which are not really contained. This is an important point to consider in industrial applications, e.g. the described virtual baker in chapter 1. For further analysis of the output shapes, especially visualizations of different stages of Swin Transformer and the last stage of a ViT, we provide a colab notebook.

Faster R-CNN with Swin-T + FPN backbone on MS COCO

The parts of the Faster R-CNN detection head are already implemented in detectron2 along with architecture specific default hyperparameter, suggested by the authors of Faster R-CNN. For particular training setups and hyperparameter related to training, we follow a large scale study [15] about pre-training of backbones and object detection on MS COCO. For a quick research, the 1x training schedule (12 epochs on MS COCO) is recommended when using a ResNet50 backbone. Since Swin-T has a similar structured architecture,

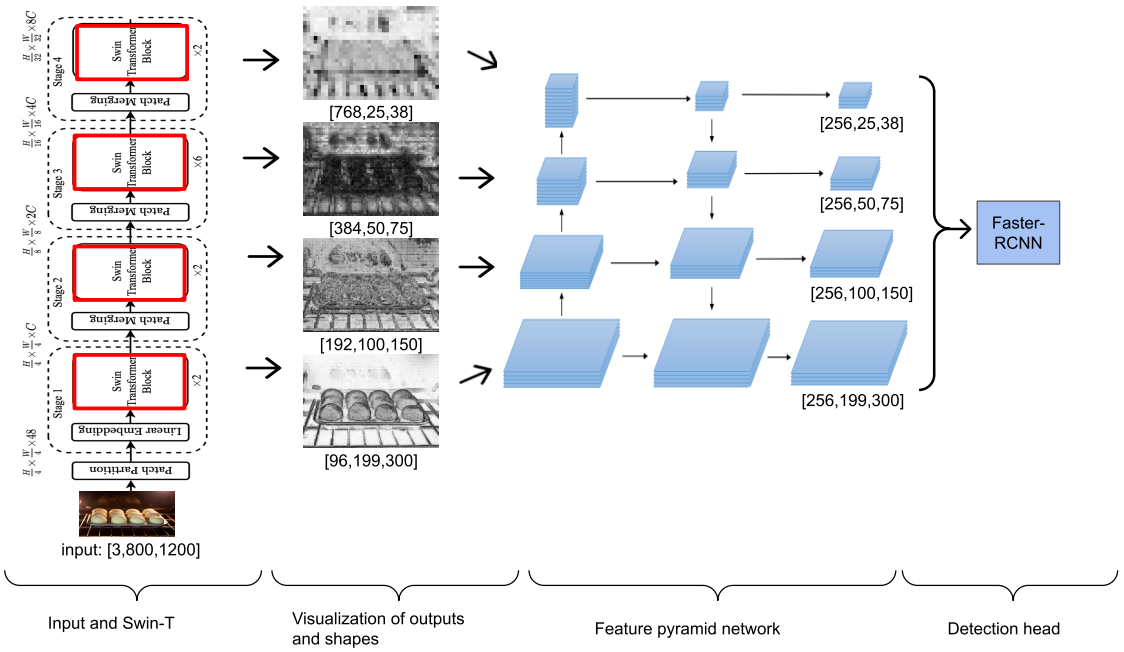


Figure 14: Visualization of the hierarchical structure of Swin-blocks and FPN.

in terms of 4 stages and the amount of parameters, we closely follow this setup with slight modifications. All trainings setups are done on google colab with a Tesla P100 (16GB VRAM) for 12 epochs and a batch size of 8. A smaller batch size (compared to default settings in 1x training schedule) leads to noisier gradients. Therefore, we slightly decreased the initial learning rate to 0.00005 and further reduce it by a factor of 0.1 after 120000 (∼8 epochs) and 160000 iterations (∼11 epochs). As suggested in [26] for training Swin Transformer, we use AdamW [27] as optimizer for training of detection heads and finetuning of the backbone. We initialized Swin-T with pre-trained supervised weights and self-supervised weights (obtained by MoBY) and the other components by default. Self-supervised training using the MoBY strategy needs 300 epochs on ImageNet and a batch size of 512, which is not possible for us due to limited computational resources. We therefore decided to take pre-trained weights provided in the github repository of MoBY [37]. To compare the performance between a supervised and self-supervised pre-trained Swin-T, we started 2 setups as described in table 4. In both cases we train the detection head and FPN from scratch, while finetuning the last 2 stages of Swin-T and freezed the first 2 stages. We have not found any studies on finetuning of Transformer backbones and thus decided to finetune the last 2 stages to make a comparison with ResNet50 possible. We use the results of a training (provided by detectron2) with comparable settings using a supervised pre-trained ResNet50 backbone as a baseline for first comparison during the experiments. Results are evaluated on the variants of the AP metric, as this metric is commonly used in object detection.

Among all tested setups, it turns out that the backbone consisting of Swin-T + FPN with initialization of self-supervised pre-trained weights and finetuning technique performs best in terms of all metrics. We therefore adapt this setup for the FSOD experiments. Interestingly, initialization with supervised pre-trained weights performs comparably well.

Backbone	Initialization of Backbone	AP	AP_{50}	AP_{75}
ResNet50 + FPN[53]	supervised	37.931	58.839	41.05
Swin-T + FPN	supervised	38.893	57.898	42.348
Swin-T + FPN	self-supervised (MoBY)	38.917	57.647	42.270

Table 4: Training of a Faster R-CNN using different initializations within a backbone consisting of Swin-T + FPN.

Finetuning turns out to lead to good results and thus should also be used with Swin Transformer. Overall, all results are better than a standard supervised pre-trained ResNet-50. Furthermore, we did no additional hyperparameter tuning, this validates the great potential of Transformer backbones even with default settings. However, the loss is still decreasing at the end of the training, which shows that the training did not converge and there is still a lot of potential for even better results.

Overall results

Table 5 shows an overall comparison including the best performing backbones trained on 12 epochs. All backbone architectures are finetuned in the last two stages. Supervised pre-trained ResNet50 backbones with and without FPN (which are commonly used when comparing object detection methods) serve as baselines.

Backbone	Initialization of Backbone	AP	AP_{50}	AP_{75}
ResNet50	supervised	35.679	56.096	38.025
ResNet50 + FPN	supervised	37.931	58.839	41.05
ResNet50 + FPN	self-supervised (DetCo)	30.181	50.268	31.898
Swin-T + FPN	self-supervised (MoBY)	38.917	57.647	42.270

Table 5: Final comparison of different backbones used in Faster R-CNN. The results from the first row are taken from [53].

Interestingly, the pre-trained self-supervised ResNet50 backbone with subsequent FPN comes relatively close to the performances of standard supervised trained ones. Moreover, our pre-trained self-supervised Swin-T + FPN backbone outperforms the supervised ResNet50 backbones and also the self-supervised ResNet50 + FPN by large margins. We belief that the difference between self-supervised trained CNNs and Transformer arise through the specific architectures and self-supervised training strategy.

4.2 FSOD experiments (Attention-RPN and Multi-Relation Detector)

In this line of experiments we keep our full machine learning pipelines and adjust the detection head from Faster R-CNN to Attention-RPN and Multi-Relation Detector to extend the object detection setting to few-shot object detection and further adapt some

hyperparameter for this setup.

Data splits for FSOD setup
In order to train the network in the few-shot setting, we divide the training set of MS COCO into novel and base classes. The network should learn to predict the novel classes in the test image when trained primarily on the base classes. We choose the 20 classes that are also part of PASCAL VOC dataset as the novel classes and the remaining 60 classes of the MS COCO dataset as the base classes. The few-shot performance is evaluated as the AP metric for the 20 novel classes of the MS COCO validation split.

FSOD setup with ResNet-50 backbone
In this FSOD network, we use the self-supervised pre-trained (DetCO) ResNet50 as backbone in Attention RPN and Multi-Relation Detector. The first two stages of the backbone are frozen while training the few-shot object detector. The detector is trained for 5 epochs using SGD optimizer on the base classes with a batch size of 6 and the learning rate of 0.001. It is then finetuned on the novel classes using 2 support ways and 9 support samples and evaluated for few-shot performance. While this two-step training approach is taken directly from the official implementation of Attention RPN and Multi-Relation Detector, number of epochs had to be reduced to get some results in time.

FSOD setup with Swin-T + FPN backbone
This FSOD model consists of the extended backbone Swin-T and FPN (as described in figure 14) and on top of that the Attention-RPN and Multi-Relation Detector detection head. We train the whole model except the first two stages of Swin-T, and therefore finetune the last 2 stages. Since the new detection head needs much more VRAM of the GPU (Tesla P100, 16GB VRAM, on google colab), we use a batch size of 2 and thus adjust the learning rate accordingly to 0.0000125. We train for 3 epochs using AdamW optimizer, as training for 12 epochs needs more than 14 days training time.

Results

Unfortunately, we were not able to get meaningful results from the FSOD experiments in both setups. In both cases the AP evaluation metric yielded values $\ll 1$ after the training, while in [11] a value of 11.1 is reported. We assume this arises through the fact that we might need additional hyperparameter tuning as well as that we froze the first 2 stages in each backbone architecture. Since the feature extractors of both models are pre-trained on ImageNet, the first two layers contain low level features from the categories of this dataset. One problem could to be that the domain shift from ImageNet to MS COCO is too large. One reason supporting this thought is that objects in ImageNet images often take up most of the space (e.g. in the second row of Figure 1). On the other hand, objects in MS COCO vary greatly in size, where objects of the same categories can have different low level features depending on their scale in the image. In a standard object detection setting, the Swin Transformer might be able to circumvent this problem due to its higher modeling power, but the ResNet50 seems to already suffer from a drop in performance due to the freezing of the first two layers, as seen in the object detection experiments. If we also take into account that we need to detect objects of novel classes in the FSOD

setting, it becomes more clear that we need low level features that are suited for detecting objects on various scales and thus generalize better to unseen classes. Also, compared to the supervised pretraining, various different methods are used in self-supervised training strategies (mirrored in the pre-trained weights), which could hamper the harder task of few-shot detection and therefore explain the performance gap. This could be a related problem, as we faced when using DINO pre-trained weights in the object detection setup.

5 Conclusions

In this project, we experimented with different self-supervised training approaches in a standard object detection setting as well as in a few-shot object detection setting. More specifically, we investigated whether a faster R-CNN can benefit from a self-supervised trained backbone. For our studies, we focused on the two backbone architectures ResNet50 and Swin Transformer. We also tried to assess whether the performance gain, also transfers to the few-Shot setting.

Both self-supervised learning and few shot object detection are recent and very active research topics with new publications improving upon the previous ones rapidly. The main value of this project is that we have documented the basic theory of self-supervised learning and few shot object detection. These two promising approaches could save a considerable amount of effort in maintaining computer vision systems and also significantly reduce the time they currently take to adapt to new environments.

Experiments with backbones trained with self-supervised methods have given promising results. With ResNet50 initialized with DetCo pre-trained weights, we were able to get close to Detectron2's supervised baseline results with regular, completely supervised Faster R-CNNs. And with Swin-T, we were able to surpass the supervised baselines. These results on a general dataset such as MS COCO prove that it is possible to train useful feature extractors for object detection without the need for labeling data. In industrial applications, for instance the virtual baker from PreciBake, where the models need to adapt quickly to new target environments, training backbones on target images without the need for labeling could save a lot of effort and time. These experiments have also shown that keeping the spatial context of the images, as done in DetCo, is key to success. Our observation is that classification-focused approaches like DINO give poor results because the spatial context is removed, as the focus is only on creating a good feature representation for accurate classification.

Experiments on few shot object detection, on Attention-RPN and Multi Relation Detector architecture, have not been as successful. The goal was to use the best backbones pre-trained with self-supervised methods as observed from the experiments with Faster R-CNN, and use them with Attention-RPN and Multi Relation Detector. This did not yield good results, since more computational resources and additional hyperparameter tuning are needed. The reported results of this architecture on MS COCO dataset also do not compare to non-few shot object detectors, so we believe that research on few shot object detection is still in its early stages and for now, the performance is not enough to combine with self-supervised backbones and use in practical industrial applications. We believe it can still be used to improve secondary tasks instead of being the main object detector. One idea is that it could be used as an "automatic labeling machine", which facilitates

and speeds up the labeling process needed to build a standard object detector.

6 Further Work

In this section we discuss various ways this research could be expanded in the future. First of all, it is necessary to continue with the FSOD experiments using Attention-RPN and Multi Relation Detector, as we didn't have the time to fully explore the architecture and run the trainings procedure long enough. One possible extension is to experiment on other few-shot object detectors, such as MetaDETR [57]. This architecture also utilizes support images, but adapt Transformer architectures as detection head. Another idea, arised in general object detection, is to use a composite backbone network architecture [21]. This work uses two backbones in a sequential fashion and two FPNs in a parallel fashion. Thus, the detection head benefits from a better feature embedding. Our self-supervised trained ResNet50 and Swin-T could therefore be used jointly. As research in few shot object detection and self-supervised learning continues, there is no doubt that new and better approaches will arise.
Another, more application based idea is to apply the architectures and ideas we have obtained during our experiments to PreciBake's own data. As discussed in the previous section, even though the performance obtained with backbones trained with self-supervised methods and few-shot detection architectures do not compete with fully supervised baselines, they can still make a huge difference in the industrial applications where it can save tons of time and effort required to obtain and label data. A possible experiment could be to train backbones using unlabeled PreciBake data from scratch using self-supervised training, instead of taking ones trained on ImageNet, which could then be used with Faster R-CNNs or Attention RPN with Multi Relation Detectors, to finetune for a specific customers case, using some labeled images from the said customer. Since the feature extractor here is specifically pre-trained on Precibake's own data, it is possible that it gives better results than backbones trained on general datasets such as ImageNet, MS COCO etc.

References

[1] Josh Beal et al. *Toward Transformer-Based Object Detection*. 2020. arXiv: 2012.09958 [cs.CV].

[2] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV].

[3] Zhaowei Cai and Nuno Vasconcelos. *Cascade R-CNN: Delving into High Quality Object Detection*. 2017. arXiv: 1712.00726 [cs.CV].

[4] Mathilde Caron et al. *Emerging Properties in Self-Supervised Vision Transformers*. 2021. arXiv: 2104.14294 [cs.CV].

[5] Mathilde Caron et al. *Unsupervised Learning of Visual Features by Contrasting Cluster Assignments*. 2020. arXiv: 2006.09882. URL: https://arxiv.org/abs/2006.09882.

[6] Xinlei Chen et al. *Improved Baselines with Momentum Contrastive Learning*. 2020. arXiv: 2003.04297 [cs.CV].

[7] Xiyang Dai et al. *Dynamic Head: Unifying Object Detection Heads with Attentions*. 2021. arXiv: 2106.08322 [cs.CV].

[8] Jian Ding et al. *Learning RoI Transformer for Detecting Oriented Objects in Aerial Images*. 2018. arXiv: 1812.00155 [cs.CV].

[9] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].

[10] M. Everingham et al. "The Pascal Visual Object Classes Challenge: A Retrospective". In: *International Journal of Computer Vision* 111.1 (Jan. 2015), pp. 98–136.

[11] Qi Fan et al. *Few-Shot Object Detection with Attention-RPN and Multi-Relation Detector*. 2020. arXiv: 1908.01998 [cs.CV].

[12] Ross Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV].

[13] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. arXiv: 1311.2524 [cs.CV].

[14] Jean-Bastien Grill et al. *Bootstrap your own latent: A new approach to self-supervised Learning*. 2020. arXiv: 2006.07733 [cs.LG].

[15] Kaiming He, Ross Girshick, and Piotr DollÃ¡r. *Rethinking ImageNet Pre-training*. 2018. arXiv: 1811.08883 [cs.CV].

[16] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[17] Kaiming He et al. *Mask R-CNN*. 2018. arXiv: 1703.06870 [cs.CV].

[18] Kaiming He et al. *Momentum Contrast for Unsupervised Visual Representation Learning*. 2020. arXiv: 1911.05722 [cs.CV].

[19] Bingyi Kang et al. *Few-shot Object Detection via Feature Reweighting*. 2019. arXiv: 1812.01866 [cs.CV].

[20] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. "Siamese Neural Networks for One-shot Image Recognition". In: 2015.

[21] Tingting Liang et al. *CBNetV2: A Composite Backbone Network Architecture for Object Detection*. 2021. arXiv: 2107.00420 [cs.CV].

[22] Minghui Liao et al. *Mask TextSpotter: An End-to-End Trainable Neural Network for Spotting Text with Arbitrary Shapes*. 2019. arXiv: 1908.08207 [cs.CV].

[23] Tsung-Yi Lin et al. *Feature Pyramid Networks for Object Detection*. 2017. arXiv: 1612.03144 [cs.CV].

[24] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV].

[25] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].

[26] Ze Liu et al. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. arXiv: 2103.14030 [cs.CV].

[27] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG].

[28] Fahad Parvez Mahdi, Kota Motoki, and Syoji Kobashi. "Optimization technique combined with deep learning method for teeth recognition in dental panoramic radiographs". In: *Scientific Reports* 10.1 (2020). DOI: 10.1038/s41598-020-75887-9.

[29] *Object Detection on COCO test-dev*. https://paperswithcode.com/sota/object-detection-on-coco. Accessed: 2021-06-28.

[30] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. *Representation Learning with Contrastive Predictive Coding*. 2019. arXiv: 1807.03748 [cs.LG].

[31] Juan-Manuel Perez-Rua et al. *Incremental Few-Shot Object Detection*. 2020. arXiv: 2003.04668 [cs.CV].

[32] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: 1612.08242 [cs.CV].

[33] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[34] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV].

[35] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[36] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[37] *Self-Supervised Learning with Vision Transforme This repo is the official implementation of "Self-Supervised Learning with Swin Transformers"*. https://github.com/SwinTransformer/Transformer-SSL. Accessed: 2021-06-30.

[38] *Source of image used in the third row in visualization of the trainings process in the introduction*. https://images-ext-1.discordapp.net/external/POlQtrtshACor1TjHAkcICEZw0r_90ETcYI1QsGW7M8/https/content3.jdmagicbox.com/comp/def_content/bakeries/default-bakeries-14.jpg. Accessed: 2021-06-30.

[39] *Source of processed image in Swin blocks visualization*. https://www.simplyrecipes.com/thmb/WTKtVPpIVO5F3sVQgUpyoiUwZAc=/1200x796/filters:no_upscale():max_bytes(150000):strip_icc()/__opt__aboutcom__coeus__resources__content_migration__simply_recipes__uploads__2010__03__bake-brazilian-cheese-bread-in-oven-.jpg. Accessed: 2021-07-01.

[40] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: 1905.11946 [cs.LG].

[41] Mingxing Tan and Quoc V. Le. *EfficientNetV2: Smaller Models and Faster Training*. 2021. arXiv: 2104.00298 [cs.CV].

[42] Mingxing Tan, Ruoming Pang, and Quoc V. Le. *EfficientDet: Scalable and Efficient Object Detection*. 2020. arXiv: 1911.09070 [cs.CV].

[43] Yonglong Tian et al. *Rethinking Few-Shot Image Classification: a Good Embedding Is All You Need?* 2020. arXiv: 2003.11539 [cs.CV].

[44] Zhi Tian et al. *FCOS: Fully Convolutional One-Stage Object Detection*. 2019. arXiv: 1904.01355 [cs.CV].

[45] Hugo Touvron et al. *Training data-efficient image transformers distillation through attention*. 2021. arXiv: 2012.12877 [cs.CV].

[46] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[47] *Virtual baker Example about the application of object detection at Precitaste*. https://www.youtube.com/watch?v=wA-CUcnXkjs. Accessed: 2021-06-30.

[48] Wenhai Wang et al. *Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions*. 2021. arXiv: 2102.12122 [cs.CV].

[49] Xin Wang et al. *Frustratingly Simple Few-Shot Object Detection*. 2020. arXiv: 2003.06957 [cs.CV].

[50] Lilian Weng. "Meta-Learning: Learning to Learn Fast". In: *lilianweng.github.io/lil-log* (2018). URL: http://lilianweng.github.io/lil-log/2018/11/29/meta-learning.html.

[51] Jiaxi Wu et al. *Multi-Scale Positive Sample Refinement for Few-Shot Object Detection*. 2020. arXiv: 2007.09384 [cs.CV].

[52] Yuxin Wu et al. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[53] Yuxin Wu et al. *Detectron2 Model Zoo*. https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md#faster-r-cnn. Accessed: 2021-07-16. 2019.

[54] Enze Xie et al. *DetCo: Unsupervised Contrastive Learning for Object Detection*. 2021. arXiv: 2102.04803 [cs.CV].

[55] Zhenda Xie et al. *Self-Supervised Learning with Swin Transformers*. 2021. arXiv: 2105.04553 [cs.CV].

[56] Xiaopeng Yan et al. *Meta R-CNN : Towards General Solver for Instance-level Few-shot Learning*. 2020. arXiv: 1909.13032 [cs.CV].

[57] Gongjie Zhang et al. *Meta-DETR: Few-Shot Object Detection via Unified Image-Level Meta-Learning*. 2021. arXiv: 2103.11731 [cs.CV].

[58] Gongjie Zhang et al. *PNPDet : efficient few-shot detection without forgetting via Plug-and-Play sub-networks*. 2021.

[59] Shifeng Zhang et al. *Bridging the Gap Between Anchor-based and Anchor-free Detection via Adaptive Training Sample Selection*. 2020. arXiv: 1912.02424 [cs.CV].

[60] Shifeng Zhang et al. *Single-Shot Refinement Neural Network for Object Detection*. 2018. arXiv: 1711.06897 [cs.CV].

Appendix

AP Metrics

Mean Average Precision (mAP) is the standard metric used for object detection tasks. The value is calculated by using a mean value of precision averaged over all the different classes for varying threshold of Intersection over Union (IoU). The threshold value for IoU determines if the predicted object bounding box from the network is considered as correct. For all the predicted bounding boxes where the intersection over union with the ground truth value exceeds the threshold with the same predicted class as ground truth, it is considered as correctly detected.

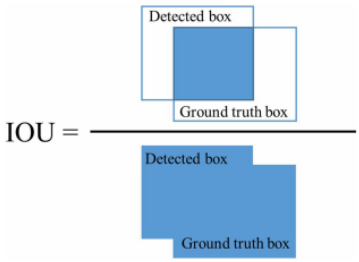


Figure 15: Illustration of the Intersection over Union. Figure taken from [28].

In general, the mAP metric uses varying IoU thresholds between 0.5 and 0.95 with step size of 0.05, and averages the precision over each of the threshold steps and across all the classes to get the AP metric value for evaluation. We also report the values at $AP50$ and $AP75$ which are the average precision values at the IoU thresholds of 0.5 and 0.75 respectively.