University of Padova

Department of Physics and Astronomy "Galileo Galilei"

Master Thesis in Physics of Data

Extending SSL patches spatial relations in Vision Transformers for object detection and instance segmentation tasks

Supervisor
Lamberto Ballan
University of Padova

Co-supervisor
Elena Izzo
University of Padova

Master Candidate
Filippo Ziliotto

Academic Year
2022-2023

"Everything will be fine in the end. If it's not fine, it's not the end"
— Anonymous

Abstract

Vision Transformer (ViT) architecture has become a de-facto standard in computer vision, achieving state-of-the-art performances in various tasks. This popularity is given by a remarkable computational efficiency and its global processing self-attention mechanism [1]. However, in contrast with convolutional neural networks (CNNs), ViTs require large amounts of data to improve their generalization ability. In particular, for small datasets, their lack of inductive bias (i.e. translational equivariance, locality) can lead to poor results. To overcome the issue, SSL techniques based on the understanding of spatial relations among image patches without human annotations (e.g. positions, angles and euclidean distances) are extremely useful and easy to integrate in ViTs architecture. The correspondent model, dubbed RelViT, showed to improve overall image classification accuracy, optimizing tokens encoding and providing new visual representation of the data [2]. This work proves the effectiveness of SSL strategies also for object detection and instance segmentation tasks. RelViT outperforms standard ViT architecture on multiple datasets in the majority of the related benchmarking metrics. In particular, training on a small subset of COCO, results showed a gain of +2.70%, +2.20% in mAP for instance segmentation and object detection respectively.

Contents

Listing of figures

Listing of tables

Listing of acronyms

ViT . . . . . Vision Transformer

CNN . . . . . Convolutional Neural Network

MSA . . . . . Multi-Head Self-Attention

MLP . . . . . Multi Layer Perceptron

SL . . . . . Supervised Learning

SSL . . . . . Self-Supervised Learning

RelViT . . . . . Relational Vision Transformer

SOTA . . . . . State-of-the-art

mAP . . . . . Mean average precision

LN . . . . . Normalization Layer

NLP . . . . . Natural Language Processing

R-CNN . . . . . Region Based CNN

RPN . . . . . Region Prediction Network

FPN . . . . . Feature Pyramid Network

Swin-ViT . . . . . Hierarchical Vision Transformer using Shifted Windows

T2T-ViT . . . . . Tokens-To-Token Vision Transformer

1 Introduction

The Vision Transformer is a fairly recent model designed to address computer vision tasks. It is based on the Transformer model, which is state-of-art in natural language processing, and adapts it for image processing [1]-[3]. The key innovation of Vision Transformer is to treat patches of the image in a similar way as words are treated in sentences. Unlike other models, relying only on convolutional layers, Vision Transformer is based on the self-attention mechanism, allowing any of the elements of a sequence to interact with both itself and all the others. These elements, called tokens, are the vectors obtained from the non-overlapping patches of the input image, plus an extra one called classification token. During the training process, all the tokens in the sequence are passed and processed through the layers of the Vision Transformer, however for classification tasks only the correspondent token is passed to the final supervision for prediction, i.e. the head. Although ViT architectures have shown to achieve results comparable to those of convolutional neural networks in various vision tasks, they requires a significant amount of labeled data to reach SOTA performances. Yet, the process of creating and labeling such large datasets is not practical as it requires a significant amount of time and resources, which could may led in unwanted biases. To overcome these limitations, it is crucial to explore new ways to improve the generalization abilities of the Vision Transformer across domains. As an alternative, one potential solution could be to use self-supervised learning techniques, similar to the ones used in the BERT model [4]. Self-supervised learning is an approach that leverages the advantages of both supervised and unsupervised learning techniques. With this method, models are trained using labeled data, where labels are generated from the data itself

as unsupervised signals. In models like BERT, self-supervision is used in a pre-training phase to acquire a solid background knowledge, where the model gain a broad understanding of the data, which is then fine-tuned using standard supervised learning. Currently, the exploration of self-supervised techniques integrated with Vision Transformer is limited, but the studies and experiments that have been conducted show a promising direction to pursue [5]-[6]. A recent study demonstrated that by incorporating self-supervised learning techniques in image classification tasks through relational heads, the overall accuracy is improved across multiple datasets, e.g. CIFAR10 and CIFAR100 [2]-[7]-[8]. This new model, dubbed RelViT, worked especially well in small-sized datasets, showing its ability of improved generalization. In light of this, the current thesis work aims to extend the same analysis also for object detection and instance segmentation tasks. To further prove the effectiveness of these techniques, experiments were carried out using also different ViT-based architectures, such as Swin-ViT and T2T-ViT [9]-[10]. The choice of these two models arises naturally from the fact that they share similar characteristics with the plain Vision Transformer, but being more accurate in detection tasks. The self-supervision adopted in this work, can be easily integrated into the Vision Transformer architecture, its inspiration arises from the Jigsaw Puzzle task. This task has been previously studied as a way to enhance a model's ability to understand visual representations [11]-[12]. The core idea is to use patches, cropped from an input image, to try to learn the spatial relations between each other, thanks to self-attention mechanism. This solution could be beneficial for many reasons:

- The self-supervised tasks generated can be solved using the same backbone network as the ViT model, making transfer learning straightforward. This also extends easily to backbones for detection tasks.
- The proposed approach also enables the ViT model to learn the relationships between patches by considering the tokens obtained from them. As a result, all tokens, not just the classification token, are optimized during the training process, which offer the model new visual representations of the data.
- Learning effective self-supervised relations, could also be beneficial by reducing the total training time at the fine-tuning stage of a given task.

The thesis is organized as follows:

- Chapter 2 describes the general architecture and elements for the Vision Transformer model, presenting also other two ViT-based architectures (i.e. Swin-ViT, T2T-ViT) used in order to support the results obtained with the plain ViT.

- Chapter 3 describes detection problems such as object detection and instance segmentation, providing at the same time the description of the main frameworks used in SOTA research. It introduces the metrics used in this work, as well as explaining the architectural integration of Vision Transformer as a backbone for detection models.
- Chapter 4 explains self-supervision and Jigsaw Puzzle as a solving tasks. Moreover, focuses on the created self-supervised tasks based on spatial relations between patches and their detailed integration into ViT architecture, i.e. Relvit model.
- Chapter 5 presents the experiments and results carried out during these thesis work either for object detection and instance segmentation. It also discuss the datasets used and their correspondent properties.
- Chapter 6 reports the concluding remarks of the work.

2

Vision Transformer

Vision Transformer (ViT) architecture is a recent development in the field of computer vision which offers an efficient and powerful alternative to convolutional neural networks (CNNs) [1]. This architecture builds up from the Transformer model, which is SOTA for natural language processing (NLP). In terms of computational efficiency, the Vision Transformer is significantly more efficient than CNNs, however, it requires a large amount of data to achieve high levels of generalization. This chapter aims to provide a detailed description of the Vision Transformer architecture, with a focus on the self-attention mechanism, which is the basis of the model itself. Additionally, it will present two other ViT-based architectures such as Hierarchical Vision Transformer using Shifted Windows (Swin-ViT) and Tokens-To-Token Vision Transformer (T2T-ViT), important for supporting the results of this work achieved with the plain ViT [9]-[10].

The chapter is specifically divided into four sections:

- Section 2.1 provides a detailed description of the Vision Transformer model architecture, with a focus on the self-attention mechanism.
- Section 2.2 discusses the other ViT-based architectures cited before. It is worth noticing that, since their mechanisms is similar to the Vision Transformer, some architectural details will be skipped. Still, it will provide the necessary understanding of their functioning.
- Section 2.3 explores some results of ViT models in SOTA research, both for classification and detection tasks.

- Section 2.4 will briefly discuss about the integration of self-supervised techniques in the Vision Transformer to improve the overall generalization capability of the model. Yet, a more detailed analysis will be presented in Chapter 4.

2.1 Model Architecture

ViTs architecture is shown in Figure 2.1, it follows the original Transformer architecture as closely as possible, adapting it for vision tasks. It can be divided into four main elements:

- Converting 2D patches to 1D vectors: the input image is divided into fixed-size patches. The patches are obtained by dividing the original image into non-overlapping squares, whose number is determined either by the size of the image and the chosen patch size (which both are hyperparameters of the model).
- Classification token and positional embeddings integration: a classification token is added to the sequence of patches, to indicate the beginning and end of the sequence. Positional embeddings are added, instead, to indicate the spatial position of each patch in the original image.
- A Transformer Encoder: the sequence of patches is fed into the Transformer Encoder, which uses self-attention mechanism to learn the relationships among the patches and produce a representation of the image. This is repeated all along the L encoder blocks of the model.
- Final prediction layers: additional layers are added on top of the Transformer Encoder, to produce the final prediction. These layers are intended as simple linear layers for classifications tasks, while for object detection and segmentation tasks they are integrated as large framework heads. This will be examined in depth in Chapter 3.

Let's now unfold each step of the list above in a more precise way, as well as in mathematical terms. Vision Transformer model essentially starts by flattening each $2D$ patch into a $1D$ vector, i.e. by linearly projecting the patch matrix to a d_{model} dimensions vector which can vary in length depending on the type of ViT. The d_{model} in fact, is a constant sized latent vector defined throughout the whole Transformer Encoder layers, and each layer provides a $(d_{model}, N_{patches})$ vector as output. The linear projections of the flattened patches (called embedded patches) are arranged in a sequence following their natural order. This is necessary because the plain ViT itself does not have a sense of the patch order, and this may affect the results when trained.

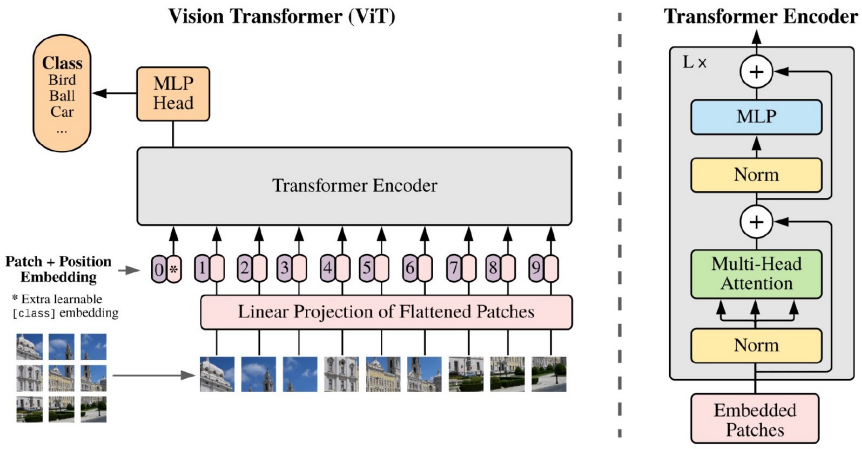


Figure 2.1: Vision Transformer model overview. The algorithm divides an input image into fixed-size patches, linearly embeds each of them, integrates a classification token to the sequence, adds positional embedding and feeds the resulting sequence of vectors to a standard Transformer encoder. The illustrative picture is taken from [1].

The second step is to introduce the classification token and positional embeddings. The classification token is a learnable random vector of size d_{model}, which is inserted at the beginning of the sequence of embedded patches. It plays a crucial role in the final prediction. Essentially, for classifcation tasks, only the first element of the sequence output of the Transformer Encoder is passed to the final head layer.

On the other hand, the positional embeddings are a sequence of 1D vectors, which is summed up to the sequence of the classification token plus embedded patches. They are learned during the training phase and aim to acquire positional information about each patch of the original image.

The third step starts when, the projected patches, combined with the classification token and encoded positions, are passed on to the Encoder stage. The Transformer Encoder is constructed with an L number of encoder blocks. Each block is composed of layers that alternate between Multi-Head Self-Attention (MSA) and Multi-Layer Perceptrons (MLP). Before every layer a Normalization module is applied, as well as residual connections after each layer. This is shown in Figure 2.1. Basically, throughout every block of the Transformer Encoder the input has the same shape of the output.

At the end of the Transformer computation, in the case of image classification, the first output of the last encoder block (i.e. the classification token) is then passed through an MLP layer called the MLP head, responsible for making the final predictions.

In mathematical terms, given an input image $x \in \mathbb{R}^{H\times W\times C}$, where (H,W) is the resolution of the image and C is the number of channels, a sequence of flattened $2D$ patches is extracted as $x_p \in \mathbb{R}^{N\times(P^2\times C)}$. In this case (P,P) is the resolution of each image patch and $N = \frac{HW}{P^2}$ is the resulting number of patches, which also acts as the effective input sequence length for the Transformer. The set of flattened patches $x_0 = [x_p^1, x_p^2, ..., x_p^N]$ is linearly projected to z_{LP} such that:

$$z_{LP} = [x_p^1 E, x_p^2 E, ..., x_p^N E], \quad E \in \mathbb{R}^{(P^2\cdot C)\times d_{model}} \tag{2.1}$$

The classification token x_{clf} is added at the beginning of each z_{LP} tensor, resulting in:

$$z'_{LP} = [x_{clf}, x_p^1 E, x_p^2 E, ..., x_p^N E], \quad x_{clf} \in \mathbb{R}^{1\times d_{model}} \tag{2.2}$$

The positional embedding is then added to z'_{LP} in a way such that:

$$z_0 = z'_{LP} + E_{pos} \quad E_{pos} \in \mathbb{R}^{(N+1)\times d_{model}} \tag{2.3}$$

and the resulting z_0 is passed on to the Transformer Encoder. At this point, each encoder block computes the output z_l as follows:

$$\begin{cases} z'_l = MSA(LN(z_{l-1})) + z_{l-1} & l = 1, ..., L \\ z_l = MSA(LN(z'_{l-1})) + z'_{l-1} & l = 1, ..., L \end{cases} \tag{2.4}$$

Finally, in the case of classification, the first output of the Transformer encoder blocks, is passed on to the MLP head:

$$logits = MLP_{head}\left(z_L^0\right) \tag{2.5}$$

and the final prediction is done using the obtained *logits*.

2.1.1 Self-Attention Mechanism

The self-attention mechanism is a core component of transformer-based models, it allows each element of the input sequence to interact with both itself and all the remaining $n-1$ elements. It then learns which elements to pay more attention to, and where. This is accomplished by constructing all the elements of the input sequence into three vectors called query, key, and value. These vectors are then mapped to the corresponding element in the output sequence. In particular, the output element is calculated by performing a weighted average of the values,

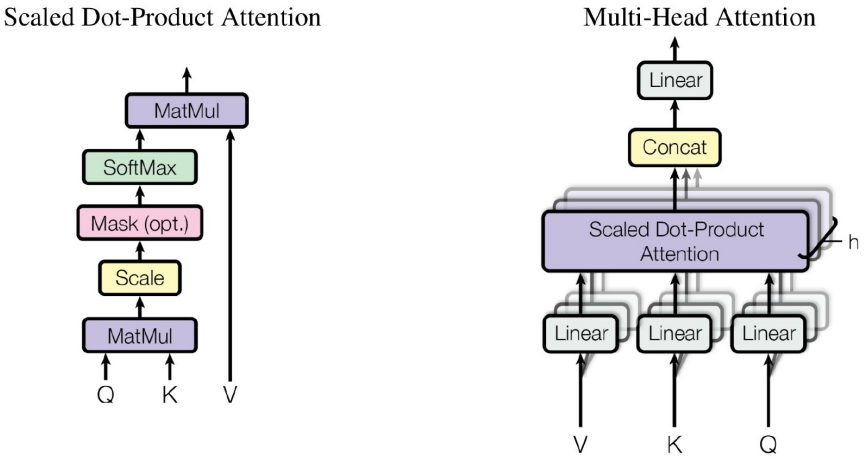


Figure 2.2: On the left, Scaled Dot-Product Attention. On the right, Multi-Head Attention consists of several attention layers running in parallel. The illustrative picture is taken from [3].

with the weights determined by a function of the query and key vectors, called the "score". A visual example of this process is shown in Figure 2.2.

In natural language processing, the self-attention mechanism allows the model to selectively attend to different parts of the input, in order to extract important features. For example, in machine translation, the model needs to understand the meaning of each word in the input sentence and how it relates to the other words to be able to translate the sentence. The self-attention mechanism allows the model to do this by computing a weighted sum of the hidden states of all the words in the input sentence, where the weights are determined by the relationship between the words. This allows the model to focus on different words in the input sentence based on their relevance, which is fundamental for the task of translation.

The self-attention mechanism used in Transformer architecture is based on the so-called Scaled Dot-Product Attention, chosen to optimize speed and for being space-efficient. Mathematically speaking, taking a sequence of n vectors as input such that:

$$x = [x^1, x^2, ..., x^n], \qquad x \in \mathbb{R}^{n \cdot d_{model}} \tag{2.6}$$

The idea is to first create three sequences of n tensors: Query, Key and Value (V,Q,K). These matrices are computed multiplying the input x for three parameters, respectively W^Q, W^K and

W^V, learned during the training process.

$$\begin{cases} Q = x \cdot W^Q & \text{where} \quad W^Q \in \mathbb{R}^{d_{model} \times d_k}, Q \in \mathbb{R}^{N \times d_k} \\ K = x \cdot W^K & \text{where} \quad W^K \in \mathbb{R}^{d_{model} \times d_k}, K \in \mathbb{R}^{N \times d_k} \\ V = x \cdot W^V & \text{where} \quad W^V \in \mathbb{R}^{d_{model} \times d_v}, V \in \mathbb{R}^{N \times d_k} \end{cases} \tag{2.7}$$

It is important to note that Q and K vectors must belong to the same space $\mathbb{R}^{N \times d_k}$, where d_k can be smaller, equal or larger than d_{model}. Meanwhile, the V vector can belong to a different space $\mathbb{R}^{N \times d_v}$, where d_v can also be smaller, equal or larger than d_k. The second step is to calculate the score S, a sequence of n vectors, which provides information about how much each element of x is important in relation to a fixed input x_i. This allows the model to identify the most important relationships between an input vector x_i and all elements of x. The score S is calculated by multiplying the query Q by the transpose of the key K, using the following equation:

$$S = Q \cdot K^T, \quad S \in \mathbb{R}^{nxn} \tag{2.8}$$

Finally the model has to compute the weights in order to get the weighted sums. To achieve more stable gradients, the score matrix S is divided by $\sqrt{d_k}$, and a *softmax* function is applied over the rows to estimate the weights. The resulting weights matrix is obtained through scalar multiplication with the value matrix V to compute the final output $z = [z^1; z^2; ...; z^n]$. In mathematical form:

$$z = Attention(Q, K, V) = softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V, \quad z \in \mathbb{R}^{n \times d_k} \tag{2.9}$$

2.1.2 MultiHead Self-Attention

The Multi-Head self-attention mechanism improves upon the standard self-attention by introducing multiple attention matrices, referred to as heads, allowing the model to focus on various combinations of elements in the input sequence. Additionally, it allows the model to gather information from different representation subspaces. Each head is calculated using a unique set of weights matrices $\left(W_Q, W_K, W_V\right)$ initialized randomly. The final output is obtained by concatenating the heads together and multiplying by a matrix, U_{msa}, whose weights are learned through training. All this, results in capturing more complex relationships between different

parts of the input data. This is a simple, yet very powerful mechanism at the core of the achievements of Transformer based Models.

$$z = [h^1; h^2; ....; h^b] \cdot U_{MSA}, \quad U_{MSA} \in \mathbb{R}^{b \cdot d_v \times d_{model}} \tag{2.10}$$

2.1.3 Positional Embedding

Tipically, Convolutional neural nets (CNNs) process inputs as a grid of pixels and learn to recognize patterns and features using convolutional filters. In contrast, Vision Transformers process inputs as a sequence of image patches and identify patterns and features using self-attention mechanisms. This enables the network to weigh the importance of different regions of the image when making predictions. To better understand the overall structure of the input data and how different patches are related to each other, positional embeddings are added into the model. This provides information about the position of each patch in the input sequence. There are several ways to encode positional information, such as sinusoidal positional embeddings and learnable positional embeddings. Sinusoidal embeddings are pre-computed and added to the input data, while learnable embeddings are learned by the network during training. The work in question uses learnable positional embeddings where a vector of size d_{model} is assigned to each patch as previously explained. A graphical representation of Transformers positional embedding is shown in Figure 2.3.

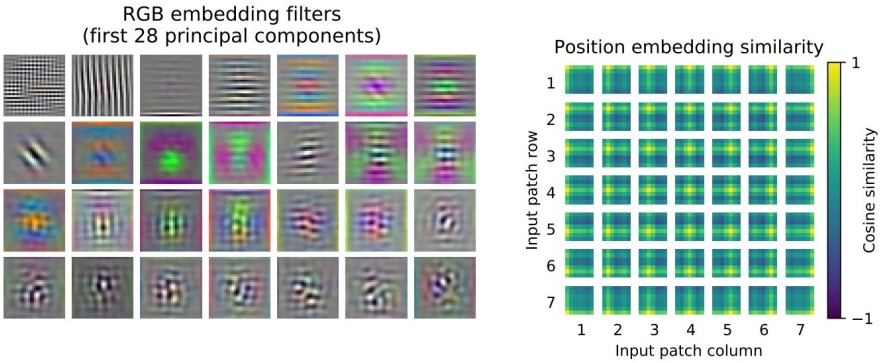


Figure 2.3: Left: Filters of the initial linear embedding of RGB values of ViT-L/32. Right: Similarity of position embeddings of ViT-L/32. Tiles show the cosine similarity between the position embedding of the patch with the indicated row and column and the position embeddings of all other patches. The image is taken from [1].

2.1.4 Model Variants

There are several variants of the Vision Transformer (ViT) model, based on the number of parameter used [1]-[?]. Some of these variations are listed in Table 2.1. The variables involved are:

- Blocks: the L number of Transformer encoder blocks, which extend throughout the entire architecture.
- Hidden size d_{model}: the constant sized latent vector through all Transformer Encoder's layers.
- Heads: The number of heads in the Multi-Head Self-Attention (MSA) layers within each block.

Model	Blocks	Hidden size d_{model}	Heads	Params
ViT-Tiny	12	192	6	5.8M
ViT-Small	12	384	6	21.5M
ViT-Base	12	512	12	86M
ViT-Large	24	1024	16	307M
ViT-Huge	32	1024	16	632M

Table 2.1: Details of Vision Transformer model variants. The last two bigger models are deprecated for this thesis work, given the limited computational resources.

The number of learnable parameters of the architecture increases with the increase of the values above. The model names "Tiny", "Small", "Base", "Large" and "Huge" are directly related to the model complexity as shown in the last column of Table 2.1. A brief notation is generally used to indicate the model size and the input patch size, which is a short model name followed by the patch size. For example, ViT-L/8 stands for the "Large" model variant (ViT-Large in Table 2.1) with 8×8 input patch size. This notation will be used throughout this work. It's worth noting that the Transformer's sequence length is inversely proportional to the square of the patch size, thus models with smaller patch sizes are computationally more expensive. The reason for this is that the number of patches, fed as input to the model, scales as $N = \frac{HW}{p^2}$.
The main ViT variant used in the thesis is ViT-S/16, i.e. ViT-small with patch size equal to 16 and a input image of size 224×224. This choice was due to the limited computational resources available during the experiments, which prohibited the training of ViT-B and bigger models.

2.2 ViT based Architectures

In this thesis work, experiments were performed using also different Transformer based architectures. We in fact provide some study on detection tasks using Swin-ViT and T2T-ViT to prove the effectiveness of the self-supervised learning techniques presented in Chapter 4. These two models proved to achieve state-of-the-art results, improving upon the plain ViT on different tasks [9]-[10]. This section describes their functioning and the reasons behind their improvement upon the plain Vision Transformer. However, the explanation will not be fully detailed since they were used only as a support for the results of the Vision Transformer (which is the core of the work).

2.2.1 Swin-ViT

"Hierarchical Vision Transformer using Shifted Windows" is a successor to the popular Vision Transformer architecture, able to achieve SOTA results in tasks such as object detection and semantic segmentation [9]. The key idea behind Swin Transformer is constructing a hierarchical representation by starting from small-sized patches and gradually merging neighboring patches in deeper Transformer layers. As shown in Figure 2.4, with these hierarchical feature maps, Swin Transformer model can conveniently leverage dense prediction techniques such as feature pyramid networks (FPN) achieving great results on detection problems. Another key design element of Swin Transformer is the introduction of the shift in the window partition between consecutive self-attention layers. This shifted strategy bridge the windows of the preceding layer, providing connections among them that significantly enhance modeling. Computation wise, the number of patches in each window is fixed, thus the complexity becomes linear to image size, making it very efficient. It is important to note that regardless of its great scalability, Swin-ViT requires a little bit more training steps than the plain ViT. Still, this architecture could be also beneficial to find a bridge between computer vision and natural language processing, facilitating the integration of the fields and modeling knowledge from both domains [9].

Summarizing, in contrast with the plain ViT having only a single-scale feature map, Swin-ViT is composed of hierarchical feature maps that enables efficient processing of high-resolution images as well as state for the art results in different tasks. Hence, the choice of this architecture for detection tasks arises naturally. Table 2.2 illustrates the different model variants of the Swin-ViT architecture. The variants Swin-T, Swin-S and Swin-L, are versions of about 0.25×,

Model	C	N° layers	Params
Swin-Tiny	96	{2,2,6,2}	28M
Swin-Small	96	{2,2,18,2}	50M
Swin-Base	128	{2,2,18,2}	88M
Swin-X	192	{2,2,18,2}	235M

Table 2.2: Model variants regarding Swin Transformer model. C is the channel number of the hidden layers in the first stage.*Note that the numbers of parameteres could vary a bit depending on the implenetation used. In this work the implementation used was the one of the original paper [9].

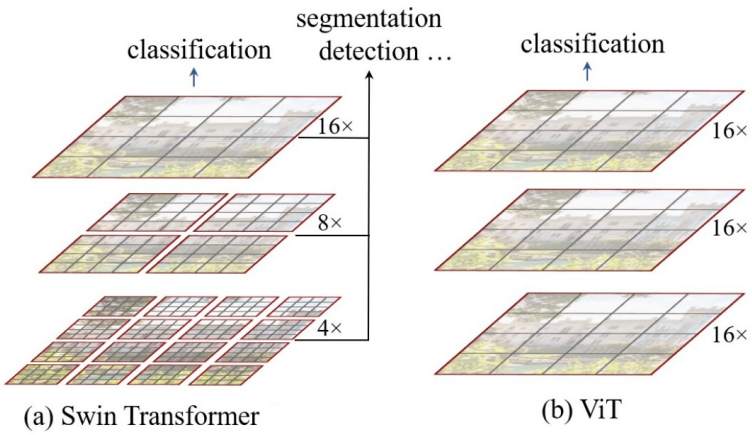


Figure 2.4: Example of the difference between the two Transformers models. Swin Transformer is able to build hierarchical feature maps by merging image patches (shown in gray) in deeper layers and has linear computation complexity to input image size due to computation of self-attention only within each local window (shown in red). It can thus serve as a general-purpose backbone for both image classification and dense recognition tasks. (b) In contrast, plain ViTs produce a single low resolution feature map and have quadratic computation complexity to input image size due to computation of self-attention globally. The image is taken from [9]

.

0.5× and 2× in size and computational complexity with respect to the base model Swin-B. Moreover, the complexity of Swin-T and Swin-S is similar to those of ResNet-50 and ResNet-101, respectively.

2.2.2 T2T-ViT

To overcome the limitations of simple tokenization and the redundant attention backbone design of ViTs, researches proposed an alternative architecture called "Tokens-to-Token Vision Transformer" (T2T-ViT) which can progressively tokenize the image to tokens improving results of vanilla ViT on mid-sized datasets. In particular, it is able to achieve more than a 3.0%

improvement when trained from scratch on ImageNet with respect to the plain VIT. It also outperforms ResNets and achieves comparable performance with MobileNets by directly training on ImageNet [10]. Morover, T2T-ViT reduces the parameter count of a vanilla ViT by half as shown in Figure 2.5.
The model consists of two main components:

- A layer-wise "Tokens-to-Token module" (T2T module) able to undestand the local structure information of the fed image, while at the same time reducing the lenght of the tokens progressively.
- An efficient "T2T-ViT backbone" able to draw the global attention relation on tokens from the previous model, this choice was motivated by looking at CNN architecture design after some empirical study as discussed in [10]. This leads to an optimized memory usage, allowing the model to be trained and deployed on devices with limited memory resources.

Since it is a ViT based architecture, T2T-ViT uses the same multi-head self-attention to enable the model to attend to different parts of the input image. This allows the model to capture long-range dependencies and learn represent Overall, this deep-narrow structure for the backbone is adopted to reduce redundancy and improve the feature extraction richness. This approach has shown to be effective for image classification tasks, achieving SOTA performance on the ImageNet benchmark.

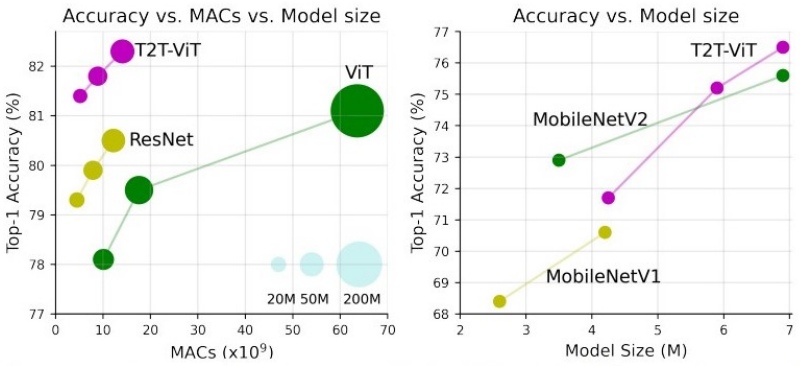


Figure 2.5: Comparison between T2T-ViT with ViT, ResNets and MobileNets when trained from scratch on ImageNet. Left: performance curve of MACs vs. top-1 accuracy. Right: performance curve of model size vs. top-1 accuracy. The image is taken from [10]

.

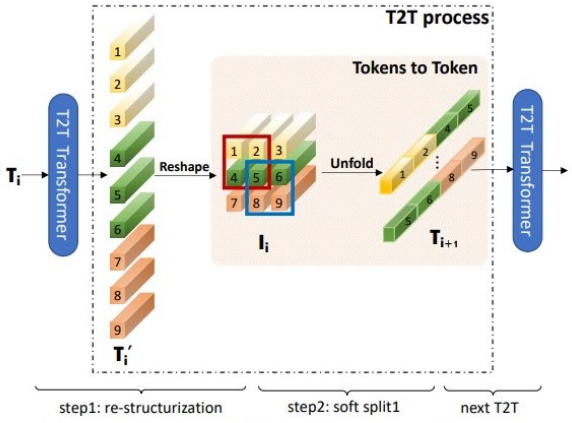


Figure 2.6: Illustration of T2T process. The T2T transformer can be a normal Transformer layer or other efficient transformers like Performer layer at limited GPU memory. The image is taken from [10]

.

2.3 Vision Transformer Benchmarks

This section presents the performance results of transformer-based architectures for various computer vision tasks, specifically focusing on ViT and Swin-ViT architectures. The objective is to showcase how these models can be used effectively for different tasks in computer vision while achieving SOTA performance.

To showcase the effectiveness of ViT-based architectures, the section provides results for classification and detection tasks. Table 2.3 shows results for classification tasks, which reach cutting-edge accuracy in standard image classification datasets, such as ImageNet and CIFAR-100 [1].

On the other hand, for detection tasks; the section presents the results for Swin-ViT, a variant of ViT that has shown promising performance in object detection. Specifically, Tables 2.5 and 2.4 show the mean average precision (mAP) results on the COCO dataset, improving on the previous CNNs based architectures [9].

2.3.1 Classification

Transformers are now widely used for classification purposes, given their remarkable performances on widely used datasets, as Table 2.3 confirms.

Research results suggest that the Vision Transformer (ViT) model outperforms the ResNet model on all of the datasets tested, requiring also significantly fewer computational resources

Dataset	ViT-H/14 JFT	ViT-L/16 JFT	ViT-L/14 I21K	ResNet152x4
ImageNet	**88.55**	87.76	85.30	87.54
ImageNet Real	**90.72**	90.54	88.62	90.54
CIFAR-10	**99.50**	99.42	99.15	99.37
CIFAR-100	**94.55**	93.90	93.25	93.51
Oxford-IIIT Pets	**97.56**	97.32	94.67	96.62
Oxford Flowers-102	99.68	**99.74**	99.61	99.63
VTAB (19 tasks)	**77.63**	76.28	72.72	76.29
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k

Table 2.3: Comparison between Vision Transformer and ResNet on popular image classification benchmarks. The results are obtained pre-training the model with bigger dataset. Vision Transformer models pre-train on the JFT-300M dataset outperform ResNet-based baselines on all datasets, while taking substancially less computational resources to pre-train. On the other hand, ViT pre-train on the smaller public ImageNet-21k dataset performs well but does not outperform ResNet.

while training [1]. This is evident from the last row in Table 2.3. Furthermore, when considering the relationship between transfer performance and total pre-training compute, ViTs excels over ResNets in performance-compute trade-off, using 2-4 times less compute to achieve the same level of performance. However, a significant limitation is their necessity to have high data volumes to perform well. They in fact achieve excellent results when pre-trained on large scale datasets, such as JFT-300M or ImageNet-21k, and fine-tuned on smaller datasets (e.g. CIFAR10, CIFAR100). However, if trained directly from scratch on small-sized datasets, performance may decrease [2]. In this context, test accuracy in ViT models is tightly connected to the creation of bigger and bigger labeled datasets.

2.3.2 Object Detection & Instance Segmentation

It is challenging to adapt Transformer models from natural language processing to computer vision due to the inherent differences between the two domains, including the vast range of scales of visual elements and the significantly higher pixel density of images compared to words in text. Still, Transformers based architecture are now widely used also for instance segmentation and object detection SOTA research purposes, as a good backbone for feature extraction. This works by feeding the backbone output information to the detection head which outputs the bounding boxes and masks output for each object detected in the image, with a correspondent confidence score.

There are various methods to merge the transformer outputs with the correspondent detection head. This adaptation can be just a simple reshape of the outputs of specific encoder blocks, as well as further integrating them with CNNs (i.e. the model's "neck") [13]. The latter turns out to be better and is referred as the main architecture used for the analysis in Chapter 5. ViT-based models regarding Tables 2.5 & 2.4 use this same data flow process: backbone $\longrightarrow$ neck $\longrightarrow$ head.

Various frameworks							
Method	Backbone	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	Params	FLOPs	FPS
Cascade MASK R-CNN	R-50	46.3	64.3	50.5	82M	739G	18.0
	Swin-T	**50.5**	**69.3**	**54.9**	86M	745G	15.3
ATSS	R-50	43.5	61.9	47.0	32M	205G	28.3
	Swin-T	**47.2**	**66.5**	**51.3**	36M	215G	22.3
RepPointsV2	R-50	46.5	64.6	50.3	42M	274G	13.6
	Swin-T	**50.0**	**68.5**	**54.2**	45M	283G	12.0
Sparse R-CNN	R-50	44.5	63.4	48.2	106M	166G	21.0
	Swin-T	**47.9**	**67.3**	**52.3**	110M	172G	18.4

Table 2.4: Results on COCO object detection using different methods as head of the model. R-50 stands for Resnet50, while Swin-T stands for Swin-Tiny. The last two columns corresponds to the computational complexity of the model and the frame-rate per second during evaluation stage, respectively. The table is taken from [9].

Various Backbones & Cascade Mask R-CNN									
	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}	Params	FLOPs	FPS
DeiT-S	48.0	67.2	51.7	41.4	64.2	44.3	80M	889G	10.4
R50	46.3	64.3	50.5	40.1	61.7	43.4	82M	739G	18.0
Swin-T	**50.5**	**69.3**	**54.9**	**43.7**	**66.6**	**47.1**	86M	745G	15.3
X101-32	48.1	66.5	52.4	41.6	63.9	45.2	101M	819G	12.8
Swin-S	**51.8**	**70.4**	**56.3**	**44.7**	**67.9**	**48.5**	107M	838G	12.03
X101-64	48.3	66.4	52.3	41.7	64.0	45.1	140M	972G	10.4
Swin-B	**51.9**	**70.9**	**56.5**	**45.0**	**68.4**	**48.7**	145M	982G	11.6

Table 2.5: Results on COCO object detection and instance segmentation, using Cascade Mask R-CNN as head and only comparing different backbones model. The last two columns corresponds to the computational complexity of the model and the frame-rate per second during evaluation stage, respectively. The table is taken from [9].

2.4 The Importance of SSL for Vision Transformers

Transformers have demonstrated outstanding performance on natural language processing (NLP) tasks. The reason for their exceptional improvements over recurrent neural networks (RNNs) is not only due to their scalable properties, but also from the incorporation of self-supervised pre-training (SSL) methods (as discussed in further detail in Chapter 4). BERT, for example, a transformer-based model for NLP, introduced the concept of dividing the task into two stages: pre-training on a self-supervised upstream task with a large amount of data, and then fine-tuning on a supervised downstream task with less data. Using this approach, BERT achieved new state-of-the-art results on 11 NLP tasks [4]. Currently, applications of self-supervised techniques in vision transformers for computer vision tasks are quickly increasing. Some experiments have been conducted using techniques such as masked patch prediction or knowledge distillation. In the case of Dino, this model pre-trains a Vision Transformer through a process called self-distillation. It uses two networks with the same architecture: a student and a teacher, working in parallel. The teacher receives an image as input, while the student receives a patch of the same image. This allows the student to learn how to predict global features from local patches, with the constraint that the student's output distribution should be the same as the teacher's. This approach shows that self-supervised learning could be a key factor to develop a BERT-like model based on vision transformers. As such, investigating new unsupervised signals extracted from images for pre-training vision transformer models is an important and necessary step in overcoming the limitations of the architecture and making vision transformers the new state of the art in the computer vision field [2]. This thesis builds upon previous work, done for classifications tasks, trying to investigate the use of SSL techniques also for object detection and instance segmentation.

3

Frameworks for Object Detection and Instance Segmentation

Instance segmentation and object detection are a core problem in the computer vision with numerous applications in various industrial fields such as autonomous driving, surveillance, and robotics. The ability to accurately and efficiently detect objects (eventually masking them) in images and videos is essential for enabling machines to understand and interact with the world around them. In recent years, detection tasks have been revolutionized by deep learning-based approaches, which have achieved state-of-the-art performance on various benchmarks. However, they remain quite challenging, especially when dealing with complex scenes and occlusions. This chapter aims to introduce how these tasks are accomplished, illustrating the architectures and frameworks which are most commonly used for research and industry standards, as well as the main metrics used to evaluate the model results.
The chapter will be divided into four main sections:

- Section 3.1 illustrates what are these detection problem and how they differ between each other.
- Section 3.2 describes the most popular detection frameworks used in computer vision nowadays, providing an in depth discussion of the architectures details of the ones used for this thesis work.
- Section 3.3 explains how these models are evaluated given such tasks.

- Section 3.4 will give an example of how ViTs can be implemented for such detection tasks. Describing how to integrate them as "good" feature extraction backbones for these frameworks.

3.1 Problem Statement

Instance segmentation, as well as object detection became very popular after the great success that CNNs had in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [14]. In past research, they were treated as two different tasks. However, with the recent advances in the field, researchers found that both tasks can be performed simultaneously at the cost of a small increase in the amount of computation required. Compared with the simpler object detection, the result of instance segmentation brings more practical significance to the final prediction. In other words, instance segmentation not only detects the presence of objects but also provides a detailed understanding of their spatial extent within the image, by further refining their borders, separating the foreground and background at a pixel level.
Another core tasks of computer vision is semantic segmentation, referring to the process of partitioning an image into multiple regions or segments, each of which corresponds to a different object or region of interest. The *"semantics"* arises from the fact that the final goal is labeling each pixel in an image with a category label, without distinguishing between different instances of the same category. This provides a more fine-grained understanding of the underlying structure of an image where also the background is annotated. For example, in a street scene image, semantic segmentation would label all pixels corresponding to cars as "car" without differentiating between individual car objects. Figure 3.1 gives a great visual example of the difference between these detection tasks.

Generally speaking, the steps in order to accomplish the detection tasks described above follows this path:

- Location: solves the problem related to determining the position of the object. The goal is to locate the actual position of each object.
- Classification: solves the problem of understanding the class of each object in the scene.
- Score: assigns a confidence level to the predicted object class. In real world application, only scores that are above 0.5 are addressed as usable predictions.

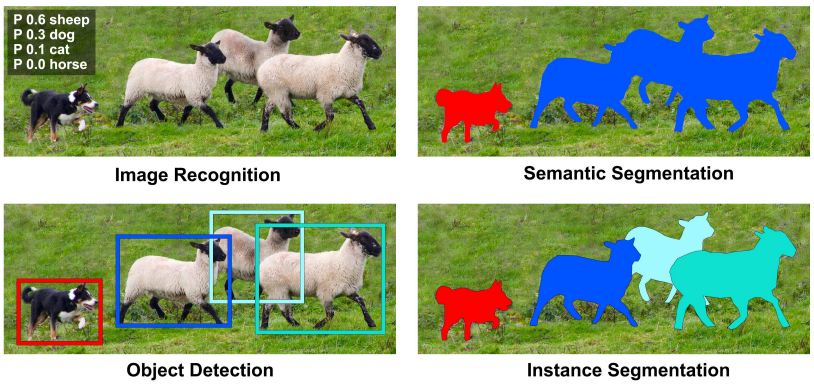


Figure 3.1: Comparison example between Object Detection and Instance Segmentation. The image is taken from COCO dataset.

- Segmentation: (for segmentation tasks) divided into instance level and scene level segmentation, trying to mask the interior of the borders of the predicted object.

3.2 Detection Frameworks

Over the years, multiple architectures have been developed to tackle detection problems. The main problem to solve, is finding a good way to extract the relevant features of the image and objects inside it. Without entering the details of all the architectures since not all of them are used in this thesis work, here are some general approaches and algorithms that are used:

- Convolutional Neural Networks: some simple CNNs stacked on top of each other creates an architecture particularly well-suited for image classification and object detection tasks, because of their ability to generalize well and extract relevant feature information. The following frameworks are all CNN-based architectures.

- Region-based CNNs (R-CNNs): are a family of algorithms that use CNNs to detect and classify objects in images. They use a two-stage approach, where object proposals are generated in the first stage and classification and localization are performed in the second stage. This is the base for Fast R-CNN, as well as Faster R-CNN which builds on top of the previous [15]-[16].

- You Only Look Once (YOLO): a fast object detection algorithm that uses a single convolutional network to predict object bounding boxes and class probabilities directly from

full images, without the need of the division in sub-regions. This architecture is commonly used in real world application because of its processing efficiency despite not having the best accuracy results. It is worth noticing that its results can still somewhat compare with SOTA architectures [17].

- Mask R-CNN: is an extension of the R-CNN family of algorithms that adds the capability of instance segmentation, which involves identifying and segmenting out individual objects within an image. It can improve the shortcomings of Faster R-CNN in small object detection, adding an overall small computational cost to the model[18].

- RetinaNet: a one-stage object detection architecture that uses a focal loss function to address the class imbalance problem in object detection. This loss assigns higher weights to hard examples, i.e. examples that are misclassified with high confidence, helping the model to focus on learning the hard examples and improves the detection performance of rare objects [19].

This is a brief explanation of the some examples of frameworks that could be use for detection tasks. This chapter will focus only on two of these frameworks, Faster R-CNN and Mask R-CNN which are the one used in the thesis work and serve as heads for the Vision Transformer backbone. This choice is due to the fact that they are the most commonly used in the field and in practical applications. They divide into two main components: the backbone which is related to the feature extraction and information encoding; the head which takes the backbone outputs, and performs the detection tasks. As discussed in Chapter 4 this frameworks heads are jointly integrated with the SSL heads of RelViT architecture. The following sections will discuss in detail how this R-CNN head works.

3.2.1 Faster R-CNN

In 2016, researches form Microsoft proposed a novel framework called Faster R-CNN, for object detection tasks. It achieved state-of-the-art results on COCO dataset and differed from standard region based methods which typically relied on inexpensive features extraction and economical inference schemes [15]-[20]. The main achievement, consisted in feeding the input image into a novel module, called *Region Proposal Network* (RPN), as well as in the detection branch simultaneously. This resulted in a higher computational efficiency compared to the SOTA methods used before (e.g. Fast R-CNN). As shown in Figure 3.2, the network structure of Faster R-CNN is mainly composed of three modules: feature extraction, RPN,

and classification regression. Mathematically speaking, during the training stage the total loss is expressed as the sum of the losses regarding the different detection modules:

$$\mathcal{L} = \mathcal{L}_{obj} + \mathcal{L}_{cls} + \mathcal{L}_{RPN} + \mathcal{L}_{box} \tag{3.1}$$

Region Proposal Network

The Region Proposal Networks (RPN) is a key component of the Faster RCNN object detection model. It is responsible for generating a set of region proposals, or regions of interest (RoIs), that are likely to contain objects. The RPN is trained to generate these regions by using a sliding window approach, where it slides a small network over the entire image, and for each location, generates a set of rectangular object proposals, each of which is represented by a bounding box and a corresponding objectness score. These scores are used to filter out unlikely regions, leaving a smaller set of high-confidence regions that are passed on to the next stage of the model for further processing. The RPN is designed to be computationally efficient and able to process images in real-time, making it a valuable component for object detection in industrial applications.

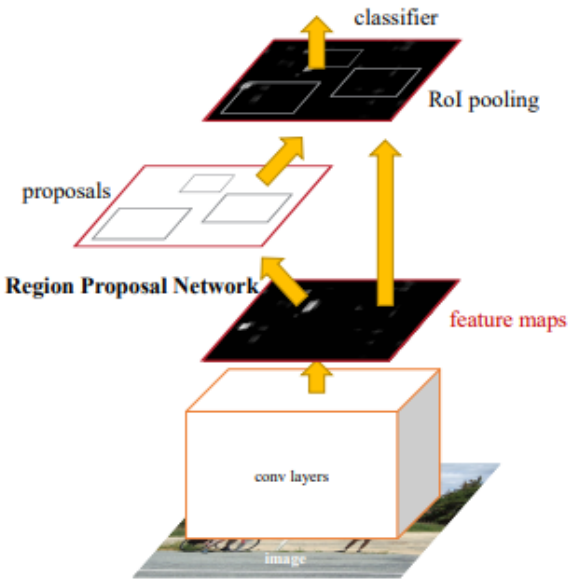


Figure 3.2: Faster R-CNN head architecture overview. Illustration of how the RPN regions generates the proposal bounding boxes for each relevant feature extracted by the backbone. The backbone is assumed to be a CNN-based architecture. The image is taken from [15].

RoI Pooling

RoI (Region of Interest) pooling is a technique used to extract a fixed-size feature map from each of the regions proposed by the RPN. The RoI pooling layer is responsible for resizing the region proposals, which may have different scales and aspect ratios, to a fixed size, typically a square feature map. This is accomplished by dividing the region proposal into a fixed grid of sub-regions, and for each sub-region, the maximum value of the activations is chosen as the output feature. This operation is designed to retain the most important information from each region proposal, while also making the feature maps of each region proposal have equal sizes. RoI pooling is another key element for the efficiency of the whole framework, allowing the model to process multiple regions at once and reducing the spatial dimension of the features, i.e. reducing the computation required by the next layers.

Generally we can summarize the flow of data of the entire model as follows:

- A backbone is used to get the feature maps relative to the input image. Typically, the backbone is a CNN-based architecture (e.g. ResNet50), but also Swin-ViTs proved to be a good SOTA competitor [9]-[21].
- Then, the RPN is used to tentatively classify the foreground and background and generate proposed regions and scores for the bounding boxes of each object.
- Like in Fast R-CNN model, here the ROI pooling layer is then used to produce a fixed-size feature map, taking both the RPN and feature maps outputs.
- Finally, the classification confidence score of the object is obtained in the classification branch, and the positioning of the coordinate of the object is carried out in the regression branch. For practical use, the output should return a dictionary, where each key is a tensor containing: boxes, labels, scores of each image in the batch. Labels with scores < 0.5 are usually deprecated.

3.2.2 Mask R-CNN

An even more general model for detection problems is Mask R-CNN, an instance segmentation framework able to detect objects while also generating the corresponding segmentation masks. It builds on its predecessors Faster R-CNN, taking the same exact architecture but adding a novel *RoiAling* block which is a simple quantization-free layer that faithfully preserves

exact spatial locations a mask branch in the final stages [18]. Moreover, Mask R-CNN is easy to generalize to other tasks still utilizing the same framework, e.g. allowing to estimate human poses keypoints. The data flow follows the same steps as its predecessors with a little addition in the end. In fact, Mask R-CNN also includes a separate branch for predicting object masks, which takes the feature maps from the RoI pooling layer as input and generates a binary mask for each object in the image. The mask branch uses a similar architecture as the classifier and bounding box regressor, but with a different output. The output of the mask branch is combined with the output of the classifier and bounding box regressor to generate the final output of the model, which includes: class labels, bounding box coordinates, and object masks for each detected object. Hence, the loss in the training phase extends naturally from the Faster R-CNN one by adding only a new terms correspondent to the masks output:

$$\mathcal{L} = \mathcal{L}_{obj} + \mathcal{L}_{cls} + \mathcal{L}_{RPN} + \mathcal{L}_{box} + \mathcal{L}_{mask} = \mathcal{L}_{FasterR-CNN} + \mathcal{L}_{mask} \tag{3.2}$$

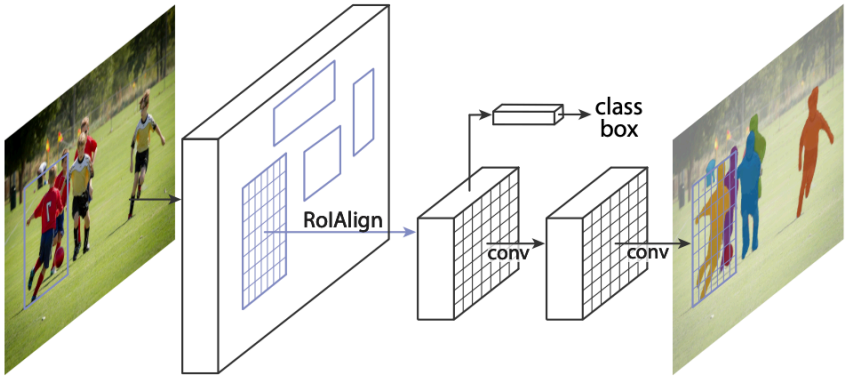


Figure 3.3: The Mask R-CNN framework for instance segmentation [18]. The output is composed of bounding boxes, as well as mask instances for targets in the image (in this case a group of people). The mask prediction is done in parallel with all the object detection tasks. The image is taken from [18].

ROI Align

RoIAlign is an improvement over the traditional RoI pooling used in Faster R-CNN. In fact, RoI pooling works by dividing the RoI image into a fixed grid of sub-regions, taking the maximum activation value for each sub-region as the output feature. This unfortunately can cause a loss of spatial accuracy, especially for small objects or objects with complex shapes. RoIAlign instead, addresses this problem by using bilinear interpolation to align the RoI with the feature

map, allowing the model to retain more precise spatial information and generating more accurate object masks. Morover, RoIAlign uses a higher resolution feature map than RoI pooling, which helps to retain more detailed information about the objects. Overall, this tecnique improves the accuracy of instance segmentation by providing more precise spatial information to the model, which allows it to generalize better than its predecessor.

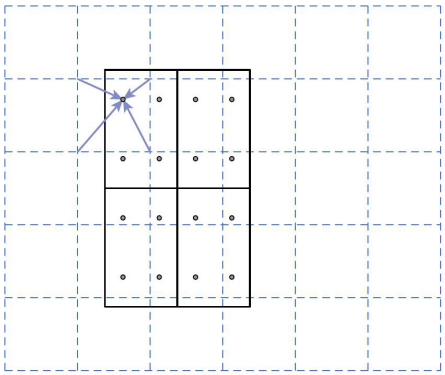


Figure 3.4: Example of how RoI Align works. The dashed grid represents a feature map, the solid lines an RoI (with 2×2 bins in this example), and the dots the 4 sampling points in each bin. RoIAlign computes the value of each sampling point by bilinear interpolation from the nearby grid points on the feature map. No quantization is performed on any coordinates involved in the RoI, its bins, or the sampling points. The picture is taken from [18].

3.3 Model Evaluation

In computer vision, several indexes are used to evaluate the model performance in detection tasks. The research standard metric is the *mean-average precision* (mAP), which have to be calculated both for the predicted objects box (mAP_{box}) and mask (mAP_{mask}) in the case of instance segmentation. Many object detection algorithms, such as Faster R-CNN, MobileNet SSD, and YOLO, use mAP to evaluate their models for publishing their research. mAP in fact,is particularly well-suited for evaluating predictions in scenarios where there are multiple objects of interest and where detecting all the objects in the scene is important. Another reason of the usefulness of this metric is the ability to incorporate the trade-off between precision and recall considering both false positives (FP) and false negatives (FN) for the final prediction accuracy. This section aims to explain how the mAP is computed giving detailed insights about the evaluation of the model used in the thesis work.

3.3.1 Mean Average Precision

The mean of average precision (mAP) values are calculated over recall and precision scores, ranging from 0 to 1. The following explanation of the computation of the metric is given taking into account only object detection tasks, e.g. bounding boxes, but an analogous reasoning could be also applied for segmentation problems. The main steps for the calculations are:

- IoU (Intersetion over Union): the first step to classify the predictions accuracy, is by estimating the intersection between the predicted and ground truth area for each object in the image. A predicted bounding box is considered a true positive if this ratio is above a certain threshold (e.g. 50, 75, 90) which defines the mAP. This is done for each predicted box which is correctly labeled, given the ground truth correspondent annotation. Eventually, the predicted bounding boxes are sorted by their confidence score in descending order and all the scores below a certain threshold aren't considered in the final computation. An example is shown in Figure 3.5.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} \tag{3.3}$$

- Recall & Precision: the second step consists in computing precision and recall values for each class.

$$\begin{cases} \text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \\ \text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \end{cases} \tag{3.4}$$

  where True Positives (TP) are detections that match a ground truth box with an IoU above the threshold, False Positives (FP) are detections that do not match any ground truth box, and False Negatives (FN) are ground truth boxes that do not have a corresponding detection.

- ROC curve: we then plot precision-recall curves for each threshold value, calculating also the area of these curves for each class in the dataset, giving the different average precisions (APs) values. This is usually done by using the simple trapezoidal rule:

$$\text{AP}^{IoU} = \int_0^1 p(r)dr \tag{3.5}$$

  where p(r) is the interpolated precision at recall r.

- AP: we have to average the APs of all the N classes. This results in a single number for each IoU threshold chosen. This threshold ranges from 0.1 to 0.95 with 0.05 intervals,

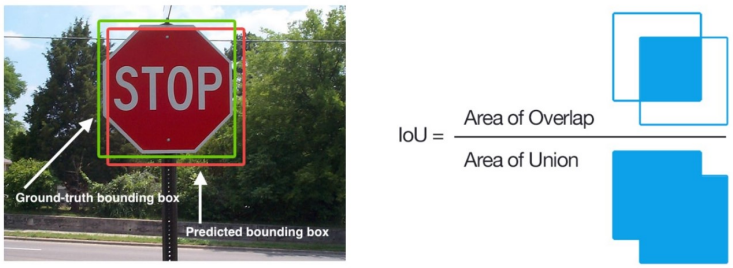


Figure 3.5: Visual example of the IoU coefficient used to evaluate a bounding box prediction.

giving 17 equal intervals.

$$\text{mAP}^{IoU} = \frac{1}{N}\sum_{i=1}^{N} \text{AP}_i^{IoU}, \qquad \text{where} \quad IoU \in [0.1, 0.95] \tag{3.6}$$

- mAP is the simple mean of all the mAP^{IoU}, giving the overall performance of the model.

Depending on the threshold of the IoU set a priori, AP splits into different variants: AP^{50} and AP^{75}. The first one is the AP value when the IoU threshold is at least 0.5, while the latter is the AP value when the IoU threshold is at least 0.75 which is a more strict metric, as it requires a higher degree of overlap between predicted and ground truth bounding boxes to count as a correct detection.
For a practical usage all these calculations are not very efficient, this is why for the evaluation of each model validation set we calculated the AP coefficients only at certain stages of the training process (i.e. every n epochs).

3.4 ViTs as backbone for Detection tasks

As explained before, frameworks used in detection problems typically rely on a backbone able to extract relevant information about the input image. In practice, the most easy way to do this is to attach a simple ResNet architecture, pretrained on Imagenet-22k, to the (Mask)Faster R-CNN head. This gives the model a very powerful set of CNNs to encode the different features of the objects. As shown in recent years, also the use of Vision Transformer based architectures can improve detection results in the standard COCO dataset over CNN-based models [9]-[10]. Hence, this section describes how to modify a plain ViT architecture, in order to use it as a good

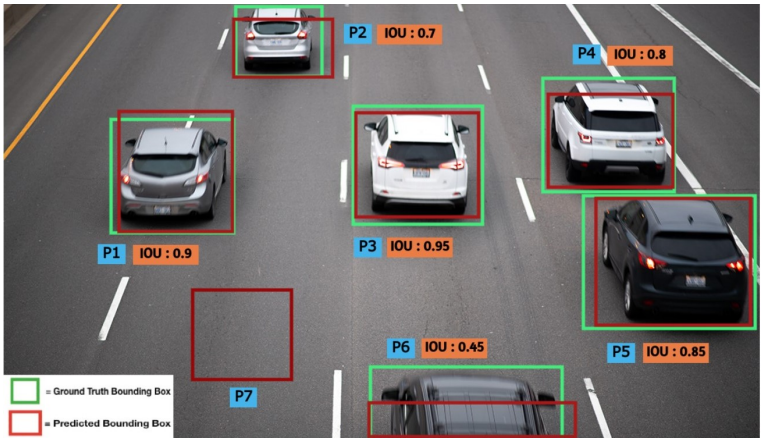
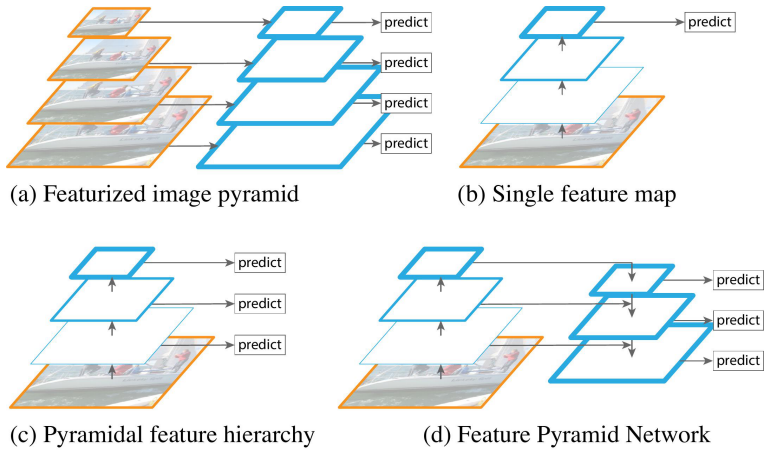


Figure 3.6: An example of the metric precision, given the IoU threshold for object detection. For example if IoU threshold = 0.8 then the precision in detecting all the cars is 66.67%, while if the IoU threshold = 0.2 then precision results in 100% accuracy. If we take the standard mAP^{50} then we would have 83.33% precision.

backbone feature extractor. This process is carried out implementing, between the backbone and the head, a so called *Feature Pyramid Network*.

3.4.1 Feature Pyramid Networks

Feature Pyramid Network (FPN) is a neural network architecture that is commonly used in object detection and semantic segmentation tasks. FPN is designed to address the problem of scale variation in images, which can make it difficult to detect objects at different sizes [22]. It works by creating a pyramid of feature maps with different spatial resolutions and semantic levels, simultaneously. The lowest level of the pyramid contains high-resolution feature maps that capture fine details, while higher levels of the pyramid contain lower-resolution feature maps that capture the global context of the image. FPNs also use a top-down pathway and lateral connections to create the pyramid of feature maps. The top-down pathway involves downsampling the feature maps from higher levels of the pyramid and upsampling them to match the resolution of lower levels feature maps. The lateral connections then merge the features from different levels, giving the model the ability to capture both fine details and global context. In addition, this architecture is computationally efficient. Overall, the FPN architecture has become an important tool in the field of computer vision, enabling a range of applications that require robust and accurate object detection and segmentation. A visual example is given by Figure 3.7.



(a) Featurized image pyramid (b) Single feature map

(c) Pyramidal feature hierarchy (d) Feature Pyramid Network

Figure 3.7: (a) Using an image pyramid to build a feature pyramid. Features are computed on each of the image scales independently, which is slow. (b) Recent detection systems have opted to use only single scale features for faster detection. (c) An alternative is to reuse the pyramidal feature hierarchy computed by a ConvNet as if it were a featurized image pyramid. (d) the Feature Pyramid Network (FPN) is fast like (b) and (c), but more accurate and is the one used in the thesis work. In this figure, feature maps are indicate by blue outlines and thicker outlines denote semantically stronger features.The image is taken from [22].

3.4.2 ViT Backbone for Detection

Vision Transformers can achieve SOTA results in image classification and became a standard over the years for that particular tasks. Unfortunately, on the contrary of Swin-ViTs, plain ViTs are very inefficient when used as a feature extractor for detection tasks. This is due to the fact that the output of all the transformer encoders has the same shape at each $i-th$ block and the final output produces a single feature map, which is not convenient for our problem. Since we are working mainly with Vision Transformer, in this thesis we modified their output in order to extract multiple features, to better improve the final results. Inspired by [13]-[23], this is addressed taking the output of the $i-th$ encoder, every four transformer blocks. The final output will then be a dictionary with 4 different feature maps of the same shape.

For example, given an image of shape $(B, 3, 224, 224)$ the output of the $i-th$ ViT block would have a shape of (B, E_{dim}, N_P) (where B is the batch size, E_{dim} is the embedding dimension of the ViT and N_P is the number of patches). This output is then reshaped by taking the fraction between the image size and patch size to create a 4-dimensional vector which becomes a feature map of shape $(B, -1, \frac{224}{P}, \frac{224}{P})$ (where P is the patch size, and -1 is given by the reshape applied to automat). A visual example is shown in Figure 3.8. The same process was used also when working with different ViT-based architectures such as T2T-ViT and Swin-ViT, where the only difference was the shape of the feature maps produced.

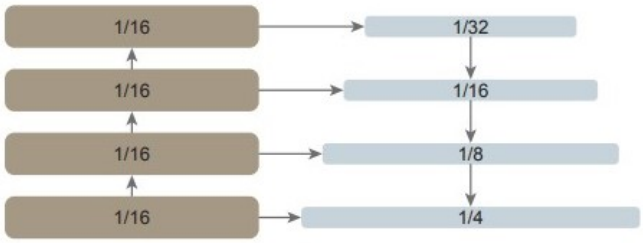


Figure 3.8: Visual example of how the $i-th$ output of the transformer blocks (left) is extracted to form multiple stages then processed by the FPN (right). In this case the chosen patch size is 16. The blocks regard only the Vision Transformer model, even if the same idea can be extended also for Swin-ViT and T2T-ViT. The image is taken from [23].

4

Self-Supervised Patches Spatial Relations for Vision Transformer

Chapter 2 introduced Vision Transformer architecture, focusing on its dependency on self-attention mechanism. This process makes the model able to relate the elements of a sequence both to themselves and all the others. The last section of the same chapter also emphasized the potential benefits of researching new self-supervised tasks in order to improve ViTs performances. In the recent years in fact, self-supervised learning has emerged as a groundbreaking technique for learning new representations from unlabeled data, providing a powerful tool to advance the SOTA in various domains of artificial intelligence.

In this context, the aim of this thesis is to explore the potential of self-supervised learning for Vision Transformers in the domains of object detection and instance segmentation. This necessity arises from the poor performances regarding ViTs on small-sized datasets, when trained entirely from scratch, i.e. without relying on a "good" pretrain on a large scale dataset such as Imagenet.

This work builds on top of the results achieved in [2], where self-supervised learning was utilized in order to learn spatial relations between image patches in classification contexts. Hence, this chapter aims to illustrate the correspondent model, named RelViT, giving a detail description of its implementation as well as explaining its simple integration into ViTs architecture.

Before digging into RelViT, the following section will introduce the main paradigms of machine learning and their characteristics.

The chapter is divided into four main sections:

- Section 4.1 describes the concept of learning paradigm, explaining both supervised and unsupervised ways models learn. Eventually, introducing self-supervised tasks.
- Section 4.2 gives a general introduction on self-supervised techniques and how they can improve deep learning models.
- Section 4.3 explains which are the self-supervised labels used in this thesis work.
- Section 4.4 introduces RelViT architecture and all the correspondent relational heads, as well as their integration in ViT models for detection tasks.

4.1 Understanding How Machines Learn

A machine learning algorithm follows a specific learning paradigm to acquire knowledge from data. Different learning paradigms are designed to suit the characteristics of the available data, and to extract valuable information to solve the task at hand. Among the various learning paradigms, two of the most common ones are supervised learning and unsupervised learning. While supervised learning involves learning from labeled data with known outputs, unsupervised learning deals with unlabeled data and seeks to identify underlying patterns and structures. The next sections enters the details of these two approaches, introducing also a different paradigm known as self-supervised learning. This last one is the core foundation of these thesis work and plays a crucial for RelViT model.

4.1.1 Supervised and Unsupervised Learning

Supervised and unsupervised learning are two widely used techniques in the fields of machine learning and AI. In supervised learning, algorithms are trained using data that consists of input-output pairs (x, y). The algorithm objective is to learn a function that describes the relationship between the input x and the output y, and to be able to generalize to new, unseen data. The term "supervised" refers to the fact that a human has to observe the input data and establish its corresponding output y, also known as the target or label, which is not inherently present in the data. In computer vision, supervised learning is commonly employed for tasks like classification, object detection, and segmentation. However, supervised learning has its limitations,

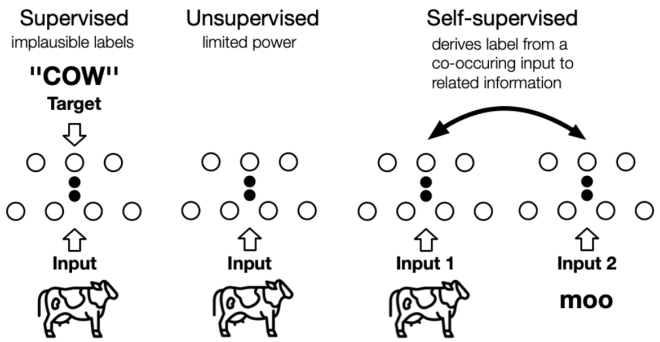


Figure 4.1: Picturesque example on the difference between various learning paradigms.

such as the need to create larger and larger labeled datasets in order to have better test results, as well as the difficulty to avoid possible biases due to the large amounts of data. The latter has also some ethical implications that are challenging to address, due to the limited interpretability that neural networks models have.

On the contrary, unsupervised learning uses unlabeled data that only provides the input x. The objective is to identify patterns or regularities in the data, which can be achieved using large amounts of unlabeled data without any inductive biases. Nonetheless, many unsupervised techniques may not be suitable in order to provide background knowledge for conventional supervised tasks. Thus, new methods are required, such as self-supervised learning, that can leverage unlabeled data to learn visual representations without human supervision.

4.1.2 Self-Supervised Learning

Self-supervised learning, in the recent years, has emerged as a promising approach in the field of machine learning and AI models. Its increasing popularity is due to its ability to combine the advantages of unsupervised and supervised learning. In this technique, labeled data is utilized, but the target is obtained through an unsupervised signal within the data itself, thereby eliminating labeling biases and providing a robust background knowledge for standard supervision. This method has significantly impacted natural language processing (NLP) through a transfer learning approach that uses a pre-training and fine-tuning scheme. During pre-training, the algorithm learns useful representations of the data using self-supervision. This knowledge is then transferred to the fine-tuning phase, which is trained using labeled data for the supervised task. In computer vision, this technique is increasingly being used to acquire visual representations from vast amounts of unlabeled images or videos. Various self-supervised tasks have been

proposed and investigated, such as predicting the RGB channels of an image given the grayscale version of the same colored image, or predicting the correct orientation of a rotated image [24]. Some works propose contrastive methods using a siamese architecture [25]. Finally, the core work of these thesis is based on self-supervised approaches which try to learn spatial relations between image patches [2]. The next sections will describe some of these self-supervised tasks that utilize patches of the input image to solve a sort of jigsaw puzzles.

4.2 Self-supervised patches spatial relations

Jigsaw puzzles are a popular game among children that is known to help develop various skills, including problem-solving and concentration. Additionally, the game requires a high spatial knowledge of the object in the picture in order to be reassembled, which helps children develop sophisticated visual generalization abilities. These abilities are also critical for solving tasks in computer vision. Consequently, the jigsaw puzzle task has been introduced in computer vision to provide context to the algorithm and enhance its generalization abilities across different domains. The primary advantage of using the jigsaw puzzle task is that it can be employed as a self-supervised technique by leveraging an unsupervised signal within the input image.

As in the original game, in this context the jigsaw puzzle is applied simply by dividing the input image into non-overlapping patches (as ViTs inherently do). With the known positions of the patches, several self-supervised tasks can be created, which can label data by exploring their spatial properties. For instance, in Figure 4.2, the input image is divided into 9 non-overlapping patches, that may be randomly shuffled into one of P possible permutations. Both the original image and the shuffled one can be passed to a machine learning algorithm that aims to learn simultaneously a specific vision task and the applied permutation during the training process. Another way to apply self-supervision is to use two different stages: a pre-training phase, where the model learns the spatial patches properties, followed by fine-tuning on a specific task. These two approaches are called "downstream-only" and "Upstream + Fine-Tune", respectively. The latter will be simply referred as Upstream throughout this thesis. They both demonstrated to improve classification results, with respect to the ones obtained by training ViTs from scratch [2]. The present thesis aims to provide evidence that SSL can also improve on detection tasks, such as object detection and instance segmentation.

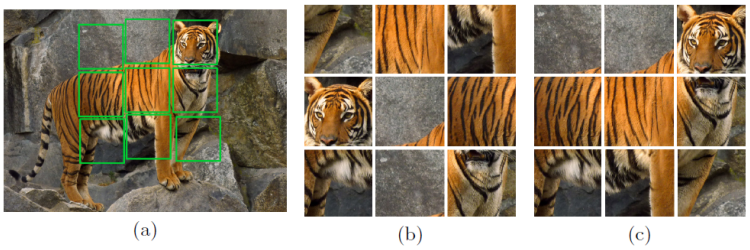


Figure 4.2: Jigsaw puzzle example for computer vision. The image is decomposed in patches *(a)*. This patches can be shuffled *(b)* and then sent to the model which has to learn to encode the spatial information of the input and reorder it *(c)*.

4.2.1 The Relations Label

This thesis work investigates a self-supervised method that is able to understand the underlying spatial relations between each couple of patches in the original image. To obtain this result a label has to be defined, prior to the training process, for each input image. This is done by creating a so called, $2D$ "relation" matrix. The idea, step by step, is as follows:

- First of all we have to choose what kind of relation to implement, defining a rule for relating pairs of patches. Based on this rule, a set of predetermined relations $R = \{R_1, R_2, ...\}$ among the patches can be established.
- To encode the type of relationship between each pair of patches, a $2D$ matrix is generated. This matrix is constructed such that its rows and columns represent the patches, and each element in the matrix captures information about the relation between the two corresponding patches from a set of possibilities R. Since the order in which the patches are related matters, the $2D$ matrix has dimensions of $N \times N$, where N is the number of patches. It is important to note that the question of *"What is the relation of patch* $\boldsymbol{x}$ *with respect to patch* $\boldsymbol{y}$*?"* is distinct from *"What is the relation of patch* $\boldsymbol{y}$ *with respect to patch* $\boldsymbol{x}$*?"* due to the order of the patches. Hence, the relation $A \rightarrow B$ may differ from the one $A \leftarrow B$. An example is shown in Figure 4.3.

4.3 Self-Supervised Labels in This Study

The last section illustrated the so-called matrix label, a label created so that each image could work as a self-supervised signal input . This matrix label contains information about the kind of relations between couples of image patches, following a set of rules chosen a priori. This section presents which are the four patches spatial relations presented in [2] and studied also

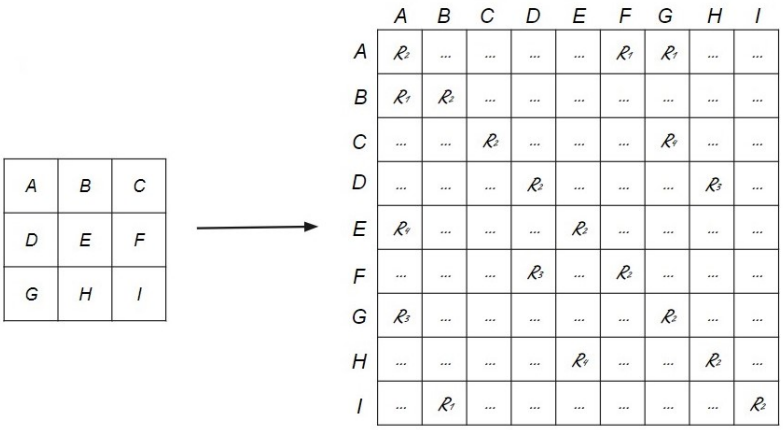


Figure 4.3: Example of Relations Matrix (on the right) starting from an image divided into 9 patches (on the left). It is a 9×9 matrix with a row and a column for each patches to take into account the relations between each ordered couple.

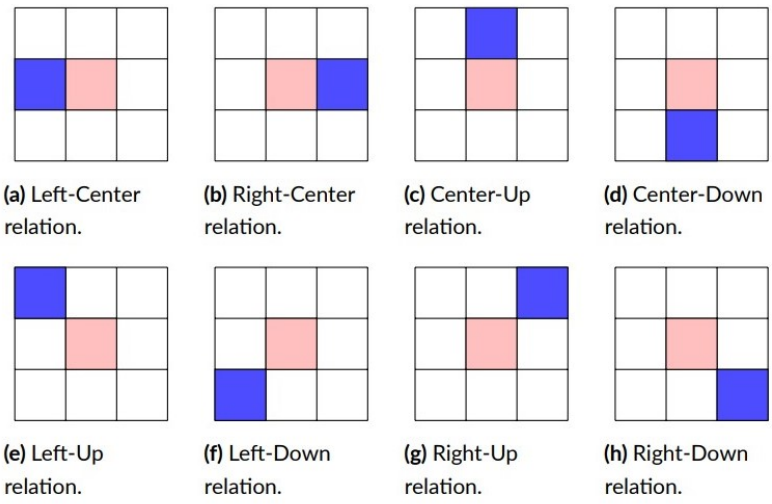
in this work. They are called: relative positions matrix, distances matrix, angle matrix and absolute position matrix. It is noteworthy that previous findings indicated that only relative and absolute positions relations yield the best combination, hence the remaining ones were not used. In Chapter 5 we provide a little ablation experiment to search for the best combination of these four labels.

4.3.1 Relative Position Matrix

Relative Positions Matrix is a label created with the aim to learn relative positions between patches, i.e. to determinate where a patch x with respect to a patch y is. This type of relation has been defined using a set of 9 fixed classes to fill the matrix label, as follows:

- *CC: Center-Center*, meaning that patch x is exactly in the same position of patch y in the no shuffle image. In other words, x and y are the same patch.
- *LC: Left-Center*, meaning that the position of patch x is on the same horizontal line and on the left with respect to the position of patch y.
- *RC: Right-Center*, meaning that the position of patch x is on the same horizontal line and on the right in relation to the position of patch y.
- *CU: Center-Up*, meaning that the position of patch x is on the same vertical line and in the upper horizontal line with respect to the position of patch y.
- *CD: Center-Down*, meaning that the position of patch x is on the same vertical line and in lower horizontal line in relation to the position of patch y.

- *LU: Left-Up*, meaning that the position of patch x is on the left and upper line with respect to the position of patch y.
- *LD: Left-Down*, meaning that the position of patch x is on left and lower line in relation to the position of patch y.
- *RU: Right-Up*, meaning that the position of patch x is on the right and upper line with respect to the position of patch y
- *RD: Right-Down*, meaning that the position of patch x is on the right and lower line in relation to the position of patch y.



(a) Left-Center relation. **(b)** Right-Center relation. **(c)** Center-Up relation. **(d)** Center-Down relation.

(e) Left-Up relation. **(f)** Left-Down relation. **(g)** Right-Up relation. **(h)** Right-Down relation.

Figure 4.4: Each figure illustrates 8 of the 9 possible classes of relations to fill the Relative Positions Matrix. Each image describes the relation between the blue-colored patch and the pink-colored one. The remaining one is the "trivial" class, where x and y are the same element.

4.3.2 Distance Matrix

The Distance Matrix is a label designed to facilitate the learning of the spatial relationships between pairs of patches, with the focus on determining whether a patch x is closer to a patch y or a patch z. The distance between two points is computed by taking the center of every patch. It is noteworthy that this matrix is symmetric, reflecting the fact that distance is defined as the Euclidean distance between the centers of patches (scaled to range from -1 to 1). Further details and clarification can be found in Figure 4.5, which illustrates the whole rule definition procedure.

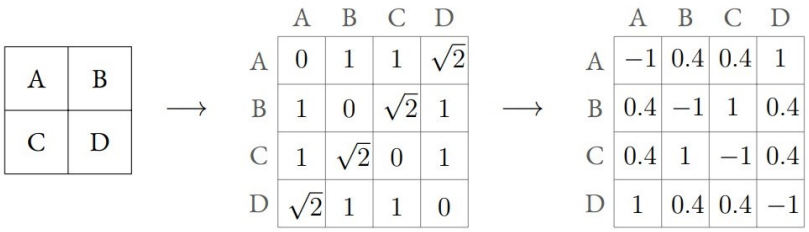


Figure 4.5: On the left, the original input image divided into 4 non-overlapping patches. In the middle, the distance matrix. On the right, the scaled version.

4.3.3 Angles Matrix

An Angles Matrix is generated as a label to determine the angle between two patches, x and y. The resulting labels are defined taking the angle between two vectors that originate from the origin and end at the centers of the patches, as illustrated in Figure 4.6. Mathematically speaking, the vector $\vec{u}_x$ connects the origin O to the center of patch x, and the vector $\vec{v}_x$ connects the origin O to the center of patch y. Hence, the correspondent angle θ can be calculated as:

$$\theta = arccos\left(\frac{\vec{u}_x \cdot \vec{v}_x}{\|\vec{u}_x\|\|\vec{u}_x\| + \varepsilon}\right) \tag{4.1}$$

where $\|\cdot\|$ is the euclidean norm and ε is a small positive constant used to avoid numerical problems during the training stage. It is worth noticing that even in this case the matrix is eventually scaled between -1 and 1.

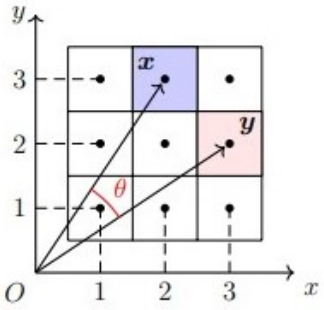


Figure 4.6: Frame of reference (F.O.R.) for the Angle matrix. As we can see the vectors are defined starting from the origin O and connecting to the centre of each patch. θ is the angle between the two vectors that connects the origin to the centers of x and y patches.

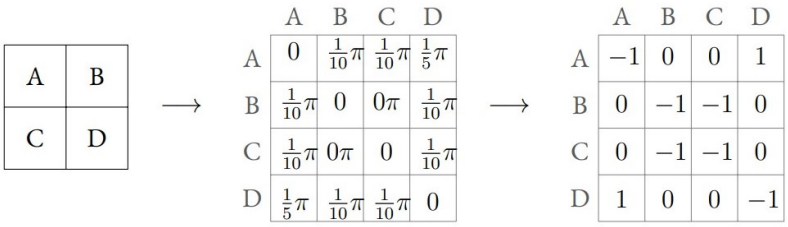


Figure 4.7: The distance matrix in the center and on the right the scaled version. The original image is divided into four non-overlapping patches.

4.3.4 Absolute Position

The final type of relation utilized in this study regards the absolute position matrix. As illustrated in Figure 4.3, a sequential alphabetical order is assigned to each patch, which can be simply transformed in a numerical ordering ranging from 1 to the total number of patches N. Unlike the previous relational matrices which are of size $N \times N$, this matrix is represented by a vector containing the position of each patch in the original image. For instance, if the image is divided into four patches, the corresponding positions of patches $[A, B, D, C]$ are $[1, 2, 3, 4]$, respectively. This label, ideally, enables the model to locate each object in the image, making it more robust for vision tasks. In general, it also provides the ability to encode spatial context, which is essential for the overall understanding of images structure.

$$\text{Abs}_{pos} = \Big[1, \quad 2, \quad \ldots, \quad N\Big] \tag{4.2}$$

4.4 RelViT Architecture

As previously said, the four relational labels, aim to understand the spatial relations between image patches. This improves Vision Transformer ability to generalize much better than the same model trained from scratch [2]. This section gives a detailed explanation of how these relations are integrated in ViTs architecture.

The integration of self-supervised patches spatial relations tasks in Vision Transformer model is achieved with the addition of different attention based heads, each one predicting a given spatial feature. Figure 4.8 illustrates the overall architecture, depicting how the Vision Transformer encoder output, i.e. the $n - 1$ processed flattened patches, are fed to the heads. It is

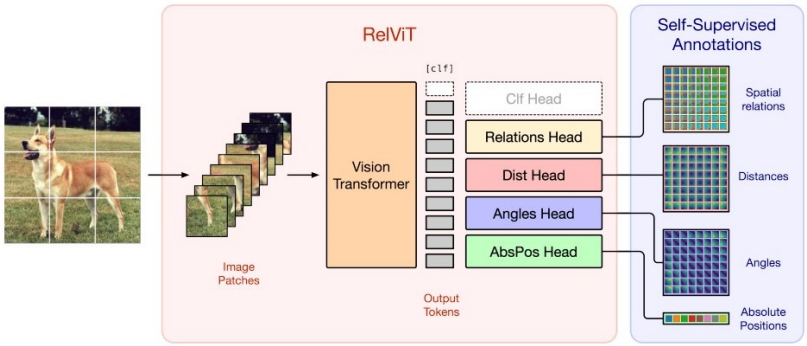


Figure 4.8: A simple visualization of RelViT during pre-training. The picture is taken from [2].

worth noticing that, in the case of object detection and segmentation the classification token isn't considered as it doesn't bring value to the final results.
In simple terms, RelViT model is a composition of a backbone, i.e. the plain ViT, plus four different heads. During the learning process, all the heads run in parallel. Hence, the total loss of the entire process is computed by summing up the losses of each head involved in the process, in order to update the models weights. Mathematically speaking:

$$\mathcal{L}_{tot} = \alpha\mathcal{L}_{Rel-Pos} + \beta\mathcal{L}_{Abs-Pos} + \gamma\mathcal{L}_{Angle} + \delta\mathcal{L}_{Dist} \tag{4.3}$$

where $\mathcal{L}_{Rel-Pos}$, $\mathcal{L}_{Abs-Pos}$, $\mathcal{L}_{Dist}$ and $\mathcal{L}_{Angle}$ are the loss related, respectively, to the relative positions matrix task, absolute position matrix task, distances matrix task and angles matrix task. Since for the majority of the experiments we are not taking into account the last two heads:

$$\begin{cases} \alpha, \beta = 1 & \text{for} \quad \mathcal{L}_{Rel-Pos}, \mathcal{L}_{Abs-Pos} \\ \gamma, \delta = 0 & \text{for} \quad \mathcal{L}_{Dist}, \mathcal{L}_{Angle} \end{cases}$$

4.4.1 Training Phases

Before delving into the description of the heads architecture it is worth describing RelViT training process. This small section illustrates how the learning is carried out during experiments. There two approaches are called "*Upstream + Fine-tune*" and "*Downstream-only*". The first one divides the learning in two phases: pretraining, where the goal is to learn only the spatial relations depicted above and minimize the related objectives; fine-tuning, where the model is trained only on a specific tasks (in this case object detection and instance segmentation), start-

ing from the weights obtained by the previous phase. This is accomplished by simply loading the pretrained weights directly into the ViT backbone used for the task. The latter approach is used when pretraining and finetuning are done in parallel, starting from scratch.

4.4.2 Heads Composition and Loss

Head for relative positions

The 2-dimensional relative positions matrix provides information about the positions of patches in relation to each other. The corresponding output tensor is 3-dimensional and contains class-related values. To predict each class, a logit vector is computed for each entry in the matrix. The correspondent $\text{Head}_{rel-pos}$ is constructed as a Multi-Head self-attention with a number of heads equal to the number of classes used to define the relative positions. This study uses the 9 classes introduced in the previous section. The process then ends after the computation of the set of scores $S = \{S^1, S^2, \ldots, S^{c=9}\}$. Eventually, these scores are scaled and stacked to form the final output. For instance, given an input sequence $x \in \mathbb{R}^{N \times dmodel}$ and c classes, c scaled scores matrices are calculated as follows:

$$S^k = \frac{\left(x \cdot W_Q^k\right) \cdot \left(x \cdot W_K^k\right)^T}{\sqrt{d_k}}, \qquad where \quad \begin{cases} k \in 1, 2, \ldots, c \quad x \in \mathbb{R}^{N \times d_{model}} \\ W_Q^k, W_K^k \in \mathbb{R}^{d_{model} \times d_k} \end{cases} \tag{4.4}$$

The mean cross-entropy is used as the loss function to assess the final predictions, which are estimated for each entry in the output tensor corresponding to the 9 classes. In practice, given the true matrix M with entries m_{ij} and the predicted tensor T with elements defined as T_{ijk}, a softmax function is initially applied over the classes to transform the tensor T as follows:

$$T'_{ijk} = \frac{e^{T_{ijk}}}{\sum_{k=1}^{c} T_{ijk}}, \quad for \quad \begin{cases} k \in 1, 2, \ldots, c = 9 \\ i, j \in 1, 2, \ldots, N \end{cases} \tag{4.5}$$

Hence, the final loss is defined as:

$$\mathcal{L}_{rel-pos} = -\frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{c=9} p_{ij}(c = k) log_2(T'_{ijk}) \tag{4.6}$$

where N is the number of patches while $p_{ij}\,(c = k)$ is the probability that the class in the

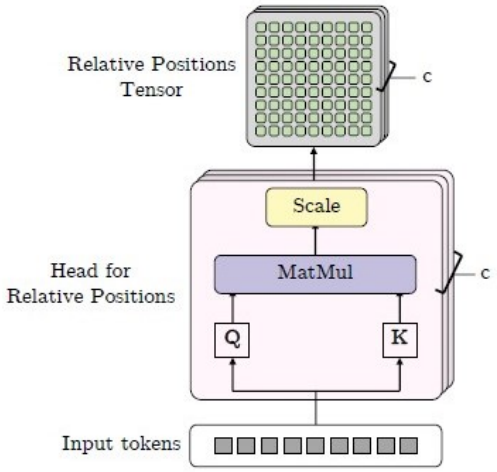


Figure 4.9: Relative Positions Head overview. The head takes as inputs the tokens processed, creating the relative positions Tensor as output and eventually calculating the correspondent loss value.

entry m_{ij} is the class k.

Head For Distances

The distances matrix is a 2-dimensional matrix that contains information about the distances between each pair of patches. In contrast to the relative position head, the output objective for the Head_{dist} block is a 2-dimensional tensor. This is accomplished by building the correspondent block as a scaled dot-product self-attention layer. Specifically, the input sequence x, which consists of N vectors with dimension d_{model}, is linearly transformed to produce two matrices, Q and K, both $\in \mathbb{R}^{N \times d_k}$. Subsequently, Q is multiplied by the transpose of K, and the resulting matrix is divided by $\sqrt{d_k}$. Thus:

$$HEAD_{dist}(x) = \frac{\left(x \cdot W_Q^k\right) \cdot \left(x \cdot W_K^k\right)^T}{\sqrt{d_k}}, \quad \text{where} \quad \begin{cases} x \in \mathbb{R}^{N \times d_{model}} \\ W_Q^k, W_K^k \in \mathbb{R}^{d_{model} \times d_k} \end{cases} \tag{4.7}$$

Figure 4.10 illustrates the whole mechanism. In this context, the loss used to evaluate the predictions is the MSE-loss, computed as the mean of the distances between each true matrix entry and predicted one. Mathematically speaking, given the true distances matrix D with entries d_{ij}

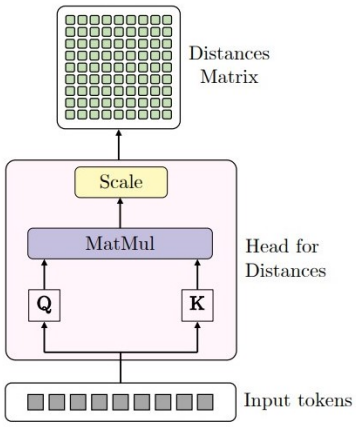


Figure 4.10: Distances Head overview. The head takes as inputs the tokens processed, creating the distance Tensor as output and eventually calculating the correspondent loss value.

and the predicted one D with entries d_{ij}, the distances loss is estimated as:

$$\mathcal{L}_{dist} = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} \left(d_{ij} - \hat{d}_{ij} \right)^2 \tag{4.8}$$

Head For Angles

The angle matrix is a 2-dimensional matrix that contains information about the angles between each couple of patches. The core technical implementation is the same as of the distance head. Hence, we could refer to Figure 4.10 for a visual explanation. In the same way:

$$HEAD_{angle}(x) = \frac{\left(x \cdot W_Q^k \right) \cdot \left(x \cdot W_K^k \right)^T}{\sqrt{d_k}}, \quad \text{where} \quad \begin{cases} x \in \mathbb{R}^{N \times d_{model}} \\ W_Q^k, W_K^k \in \mathbb{R}^{d_{model} \times d_k} \end{cases} \tag{4.9}$$

The loss function used to evaluate the predictions is the MSE-loss, which is calculated as the average of the angles between each true matrix entry and the corresponding predicted entry. In mathematical terms, let A be the true angles matrix with entries a_{ij}, and let $\hat{A}$ be the predicted angles matrix with entries $\hat{a}_{ij}$. The angles loss is computed as follows:

$$\mathcal{L}_{angle} = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} \left(a_{ij} - \hat{a}_{ij} \right)^2 \tag{4.10}$$

where N represents the number of patches in the original image.

4.4.3 Head for Absolute Position

The final head is the absolute position head, which assigns an absolute position target to each token z_i (i.e. each patch) by simply taking its corresponding position in the patch embedding of the image. The model then performs a classification task over the predicted absolute positions $\{1, 2, \ldots, N\}$. For this head, the loss function is a standard cross-entropy loss, same to the one used for the relative position head:

$$\mathcal{L}_{Abs-Pos} = -\frac{1}{N}\sum_{i=1}^{N} CE\left[\varphi_{abs-pos}(z_i), i\right] \tag{4.11}$$

where CE stands for Cross-entropy, $\varphi(z)$ is the relation used and each class i corresponds to the ordered value of the patches in the original image as explained in Section 4.3.4:

$$\varphi(z) = zW + b \quad \text{where} \quad i \in 1, ..., N$$

4.5 RelViT backbone for Detection tasks

In Chapter 3 we described how Vision Transformer can be integrated in a detection framework by using it as a backbone for feature extraction. The same idea can be applied analogously to RelViT: we take the ViT backbone and attach to it the four relational heads, plus an extra head which is the detection one. This last head has to be intended as a large CNN based model, e.g. Mask R-CNN, able to further process the extracted information and output the bounding boxes and/or binary masks. A visual example of the entire architecture is depicted in Figure 4.11.
The implementation is fairly modular, hence *Upstream* and *Downstream-only* tasks can be performed in a relatively simple way. The goal is to prove that the inclusion of the four relational heads, in the final model, allow the capture of additional spatial information between patches, which can be useful for detection tasks in small-sized datasets.

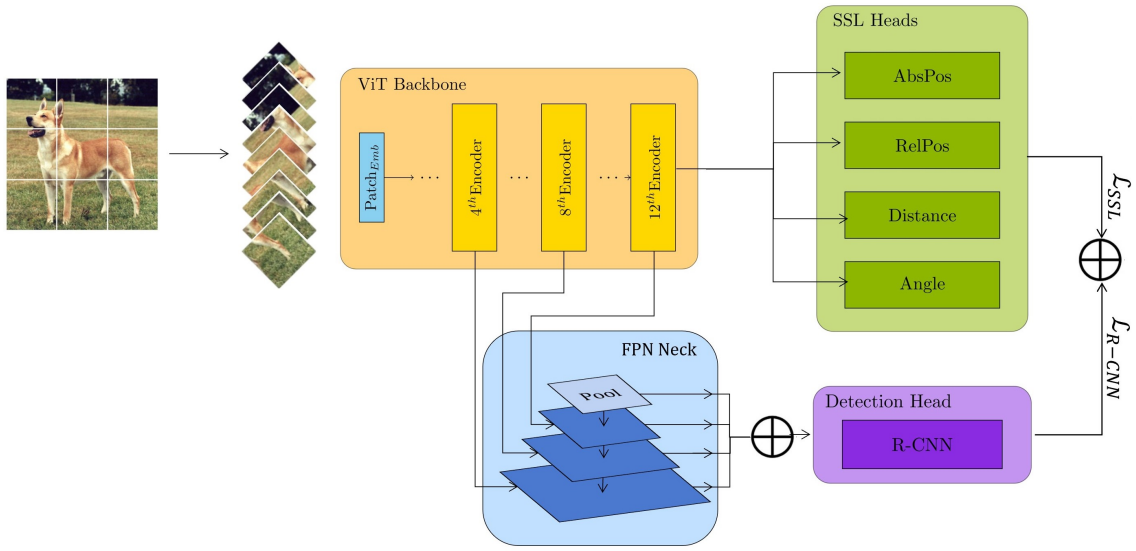


Figure 4.11: Illustration of the RelViT archtiecture integrated with the detection head. The ViT backbone (in orange) outputs are fed in parallel to the relational heads (in green) and to the FPN module (in blue). The FPN adds a pooling layer to the processed information. Finally, the outputs follows from the detection head (in purple).

5

RelViT Experiments on Object Detection and Instance Segmentation

Previously, we introduced RelViT model, describing in detail the architecture and objective of the heads. This chapter instead is focused on presenting all the experimental results conducted in this thesis work. To ensure a fair comparison the upstream approach was used, i.e. a self-supervised pretraining plus a fine-tuning to learn detection tasks. In this context, the outcomes will be defined as "pretrain" results, while those obtained from the standard ViT trained from scratch only on detection tasks will be referred to as the "baseline". Moreover, from now on when stating "RelViT" achievements we will refer to the model given by the backbone, the relational heads as well as the detection head.
Ablation studies were carried out using a downstream-only approach, instead.
The goal of this thesis work is to enhance the performance of the Vision Transformer model through the utilization of a set of self-supervised tasks. To evaluate the efficacy of the relational heads, experiments were conducted using standard vision benchmarks for object detection and instance segmentation. Hence, the metrics used are mAP, mAP_{50} and mAP_{75}.
This chapter is divided into three sections:

- Section 5.1 provides a detailed overview of the datasets used in the experiments.
- Section 5.2 describes the hyperparameter configurations used in both the upstream and fine-tuning stages, respectively.

- Section 5.3 reports the final experimental results, comparing the performance of RelViT with that of the standard ViT taken as the baseline. In addition, it presents the outcomes of ablation studies that were conducted.

5.1 Datasets and Configurations

To assess the performance of RelViT in detection tasks, we first utilized some of the most widely used datasets in the field, such as COCO and PASCAL-VOC. Object detection was performed on all datasets, while instance segmentation was only performed on two of them, namely COCO and Oxford-Pet. Each used dataset and further details are reported below:

5.1.1 Datasets

- **COCO**: the MS COCO dataset is widely used to benchmark object detection, image segmentation, and captioning models. It contains nearly 330,000 training images across 91 categories, as well as 5,000 validation images [20]. The dataset is highly challenging and is considered the standard benchmark for detection tasks in SOTA research. Figure 5.1 shows an example of image-annotation pairs in COCO. It is worth noting that it is highly unbalanced in favor of the "person" class, which is present in over 60,000 images. This poses a challenge for models to learn to detect other objects accurately. In order to test RelViT in relatively small-sized datasets, for this work we are taking a subset of COCO containing 50,000 training images and 3,000 validation images. From now on, it will be referred as COCO50k.

- **PASCAL-VOC**: the PASCAL Visual Object Classes 2007 is a well-known dataset that includes 20 object categories such as vehicles, household items, animals, and more [26]. Each image contains bounding box annotations and object class annotations, with a total of about 11,000 images. Before the rise in popularity of COCO this dataset was the standard benchmark for detection model evaluation, due to its high-quality annotations.

- **KITTI**: the KITTI dataset is popular in the autonomous driving platforms field, offering various vision tasks such as stereo, optical flow, and visual odometry [27]. It provides standard object detection annotations, including monocular images and bounding boxes. In this context, we utilized the annotations of 7480 training images to extract bounding boxes for 8 different object classes including car, pedestrian, truck, and so on. However, similar to COCO, it is important to note that this dataset is significantly unbalanced towards "car" and "pedestrian" classes.

- **SVHN**: the Street View House Numbers dataset is popular for classification tasks. It pose a hard real-world problem of recognizing digits and numbers in natural scene images, providing at the same time a large amount of labeled data [28]. It is divided into two data formats: the first containing MNIST-like 32-by-32 images centered around a single character, to perform classification; the second having about 30,000 training samples with character-level bounding boxes. Experiment-wise, in order to perform object detection the second format was chosen.

- **Oxford-Pet**: Oxford-Pet is very simple dataset widely used for classification tasks, containing 37 category pet dataset with roughly 200 images for each class [29]. Since all the images have an associated ground truth annotation of breed, head ROI, and pixel level trimap segmentation, it is possibile to perform either object detection and instance segmentation. The first one for the heads of the pets, while the latter for the entire body. In the experiments the tasks will be referred to the datasets: *Oxford-pet* and *Oxford-Pet Mask*, respectively.

Notably, all the datasets described above are relatively small, ranging from 3,000 to 50,000 training images. This choice is due for two critical factors: the limited computational available; the investigation of the extent to which self-supervised techniques enhance ViT's capabilities in handling small datasets for detection tasks. This research direction has the potential to yield valuable insights into the behavior and performance of transformer models in this context, thereby contributing to the advancement of the field.

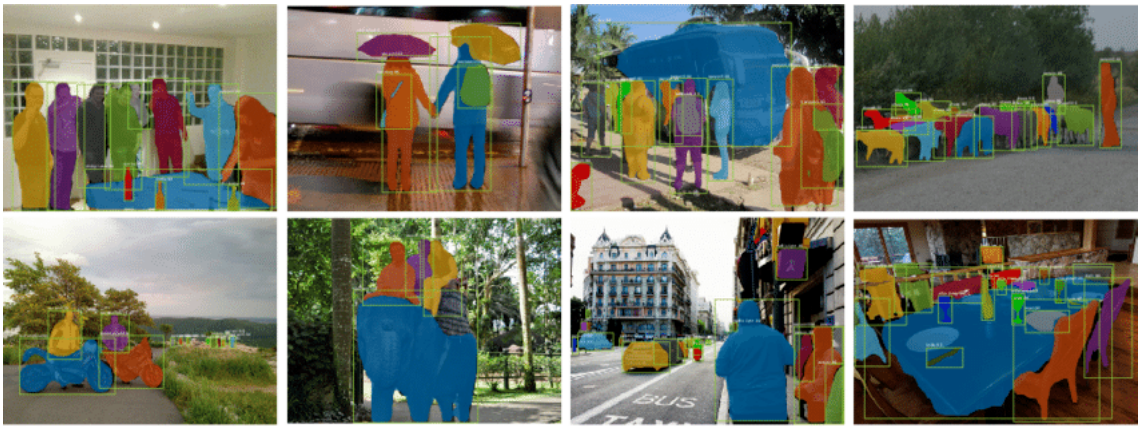


Figure 5.1: Example images taken from COCO dataset. Each object in the image has its own bounding box as its own segmentation mask. The image is taken from [9].

5.2 Configurations

5.2.1 Upstream Stage

Various backbone models were employed in the experiments conducted in this work. Nevertheless, the configurations of the upstream phase with RelViT remained as consistent as possible throughout the whole analysis. To ensure a fair comparison between the models, all the backbones yield a similar number of parameters. In particular the chosen architectures were: ViT-S/16, Swin-Tiny and T2T-14 as listed in Table 5.1.

Model	Params
ViT-S	22.3M
Swin-T	28M
T2T-14	21.5M

Table 5.1: Backbone used for detection tasks. All the models yield similar number of parameters.

Hyperparameters

The upstream phase is carried out using the same set of default hyperparameters defined in [2], with a few exceptions. The setup used for all training tasks is described:

- **Initialization**: all the weights are random initialized using a seed equal to 2023. Both classification token and positional embeddings, if used, are initialized as zero vectors.
- **N-epochs**: the number of training epochs defines the amount of time spent to learn the self-supervised tasks. The default value was 100 epochs, but for some datasets, i.e. Oxford-Pet and PASCAL-VOC, the value has to be respectively increased to 500 and 200 in order to learn effectively the given tasks.
- **Image augmentation**: the data augmentation applied to each image is composed of a random horizontal flip. In addition, each image is normalized to have mean equal to 0 and standard deviation equal to 1 over channels in the training set.
- **Patch augmentation**: all the patches, obtained from the augmented image, are randomly resized-cropped and both a color shift with color-jittering and a gray-scale are randomly applied to avoid chromatic aberration issue, with probability 0.8, 0.2 respectively [25].

- **Learning Rate**: the default Pytorch implementation of the AdamW optimizer was used in this work. The weight decay was set to 0, and the learning rate had a linear warm-up starting from 0, reaching a value of $5 \cdot 10^{-4}$ after 10 epochs and slowly decaying back to 0 at epoch 100. The choice of this learning rate was based on previous experiments, demonstrating that both ViT and RelViT models significantly improved their accuracy when switching from SGD to AdamW [1]-[2].
- **Patches Shuffling**: in order to prevent the model from learning trivial solutions and facilitate the acquisition of meaningful features, the input image patches were shuffled randomly during the experiments. This approach was necessary for models other than the plain Vision Transformers. Hence, for the latter patch shuffling was not utilized.

5.2.2 Fine-tuning Stage

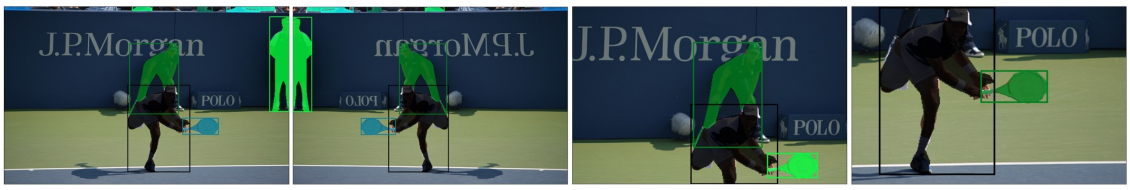
The fine-tuning stage, which uses Faster R-CNN or Mask R-CNN models, typically requires more training time than the upstream phase due to the integration of the backbone with the detection heads. This integration results in a substantial increase in the number of model parameters, as shown in Table 5.2. Moreover, the model's "neck" component for feature extraction, consisting in a feature pyramid network (FPN) with multiple convolutional layers, also contributes to the total parameter count.

Object Detection		
Backbone	**Head**	**Params**
ViT-S	Faster R-CNN	59.4M
Swin-T	Faster R-CNN	66.7M
T2T-14	Faster R-CNN	60.6M
Instance Segmentation		
ViT-S	Mask R-CNN	63.2M

Table 5.2: Models parameter size for the fine-tuning stage, for both object detection and instance segmentation. Instance segmentation was performed using only a single architecture, i.e. ViT-S/16.

Hyperparameters

The configurations settings of this phase mainly regards the final detection heads. All the models were implemented using the same hyperparameters described in the corresponding papers



(a) (b) (c) (d)

Figure 5.2: Visual example of the data augmentation used for this work, during the fine tuning stage. The images is taken from COCO and displays the object masks together with their bounding boxes. (a) the original image, (b) is the flipped version. (c) & (d) are two example of IoU crop augmentation. The final augmentation is a combination of both.

[9]-[10]-[23]. The setup of the detection heads for both the baseline and the pretrain are the same. Additionally since the frameworks for object detection and instance segmentation share very similar characteristics, the head's hyperparameters didn't change for the two models. A detailed list is described as follows:

- **Number of Epochs**: the number of training epochs was set to 100 and to ensure fair comparison, it remained constant throughout all the experiments.
- **Image Size**: image size was set to be $(224, 224)$ for all datasets. This was kept constant also regardless of the backbone used in the experiments.
- **Learning Rate**: the learning rate used was AdamW, which proved to achieve better results than SGD with momentum for ViT based architectures. The starting value was set to be 10^{-4} with a stepsize decay of 0.1 after 80 epochs.
- **Weights decay**: weights decay was initialized to 10^{-4} for the plain ViT and T2T-ViT models, while for Swin-ViT the choice was 0.05. These choices follow the setup of the original papers.
- **Augmentation**: the augmentation used was chosen to be "ssdlite", which is popular for detection tasks. It is a composition of a random horizontal flipping and a random IoU crop, both with a probability of 0.5. Figure 5.2 gives a visual example. No augmentation was used for the validation set.
- **Batch size**: the batch size was set to be 16 and 8 for the training and validation sets, respectively.
- **Head parameters**: other R-CNN models parameters were set to the ones of the default implementation of Pytorch, both for object detection and instance segmentation frameworks.

5.3 Results

In this section, we will analyze the results of the experiments on object detection and instance segmentation. According to Table 5.3, the results show that RelViT consistently outperformed the baseline on all datasets, which is evidence of its effectiveness in detection tasks. The experiments were conducted using the same backbone, ViT-S/16. Figure 5.6 illustrates some qualitative results obtained on PASCAL-VOC.

Impressively, RelViT achieved a substantial improvement of 9.39% in mean average precision (mAP) for object detection in the Oxford-Pet dataset, representing the most significant enhancement observed in the study. This can be attributed to two factors: firstly, the high number of training epochs during the upstream phase, which were in total 500; secondly, the small size of the dataset, a domain in which RelViT performs well.

Regarding also instance segmentation, RelViT achieved a noteworthy improvement of 2.20% and 2.70% in mean average precision (mAP_{box} and mAP_{mask}) compared to the baseline in the COCO50k dataset. This dataset has a pronounced bias towards the "person" class, prompting the calculation of metrics specifically for this class. In this regard, RelViT achieved improvements of 2.85% and 2.21% in the mAP^{person}_{box} and mAP^{person}_{mask} benchmarks, respectively.

Furthermore, in order to establish the general efficacy of self-supervised techniques, a comparative analysis was conducted using different backbones. Table 5.4 presents the outcomes

Object Detection

Dataset	↑ $\mathbf{mAP^{box}}$			↑ $\mathbf{mAP^{box}_{50}}$			↑ $\mathbf{mAP^{box}_{75}}$		
	Baseline	Pretrain	**Improv.**	Baseline	Pretrain	**Improv.**	Baseline	Pretrain	**Improv.**
COCO50k	14.93	17.63	+2.70	24.95	27.70	+2.75	14.00	17.64	+3.64
VOC-2007	17.91	19.24	+1.33	35.24	37.14	+1.90	15,70	17.39	+1.69
KITTI	26.81	29.39	+2.58	51.67	57.39	+5.72	25.04	25.93	+0.89
SVHN	24.20	25.25	+1.05	56.01	58.51	+2.50	16.11	16.51	+0.40
O-Pet*	32.77	42.16	+9.39	58.29	71.15	+12.86	33.74	44.53	+10.79
O-Pet*^{Mask}	29.99	34.75	+4.76	49.49	56.95	+7.46	30.65	37.18	+6.53

Instance Segmentation

	↑ $\mathbf{mAP^{mask}}$			↑ $\mathbf{mAP^{mask}_{50}}$			↑ $\mathbf{mAP^{mask}_{75}}$		
COCO50k	13.53	15.73	+2.20	28.02	30.85	+2.83	13.09	15.56	+2.47
O-Pet*$_{Mask}$	33.86	42.22	+8.36	50.26	58.73	+8.47	38.84	51.31	+12.47

Table 5.3: Detection benchmarks comparison between different datasets using Upstream + Fine-tune schedule. The metrics used are mAP, mAP_{50}, mAP_{75}. In the case of instance segmentation tasks, the mAP^{mask} is also added. In all cases RelViT successfully achieved better results than the baseline trained from scratch. All results were evaluated using the same model, i.e. using ViT-S/16 backbone, which yields $\sim$ 22Mln parameters. *O-Pet stands for Oxford-Pet.

of this analysis for the KITTI dataset, indicating that the baseline was outperformed across all measures. In particular, RelViT approach results in an improvement of $+2.58\%$, $+5.72\%$ for the mean average precisions (mAP, mAP_{50}) using the plain ViT backbone. These suggests that utilizing the simple Vision Transformer as the underlying architecture for object detection, as opposed to utilizing diverse ViT based architectures, yields the most significant and notable enhancement in performance metrics, when comparing RelViT against its corresponding baseline. However, additional experiments on different datasets are needed to confirm this final statement.

Object Detection

Backbone	↑ $\mathbf{mAP}^{box}$			↑ $\mathbf{mAP}^{box}_{50}$			↑ $\mathbf{mAP}^{box}_{75}$		
	Baseline	Pretrain	**Improv.**	Baseline	Pretrain	**Improv.**	Baseline	Pretrain	**Improv.**
ViT-S/16	26.81	29.39	+2.58	51.67	57.39	+5.72	25.04	25.93	+0.89
Swin-T	34.13	34.94	+0.81	64.33	64.91	+0.58	33.09	33.67	+0.58
T2T-14	32.14	33.75	+1.61	59.74	61.99	+2.25	28.59	31.74	+3.15

Table 5.4: Object detection benchmarks comparison between different ViT based architectures on KITTI using Upstream + Fine-tune schedule. The metrics used are mAP, mAP_{50}, mAP_{75}. In all cases RelViT successfully achieved better results than the baseline trained from scratch. All models consist of a backbone plus a Faster-RCNN head, yielding similar number of parameters for comparison, i.e. $\sim$ 60Mln.

5.3.1 Ablation Studies

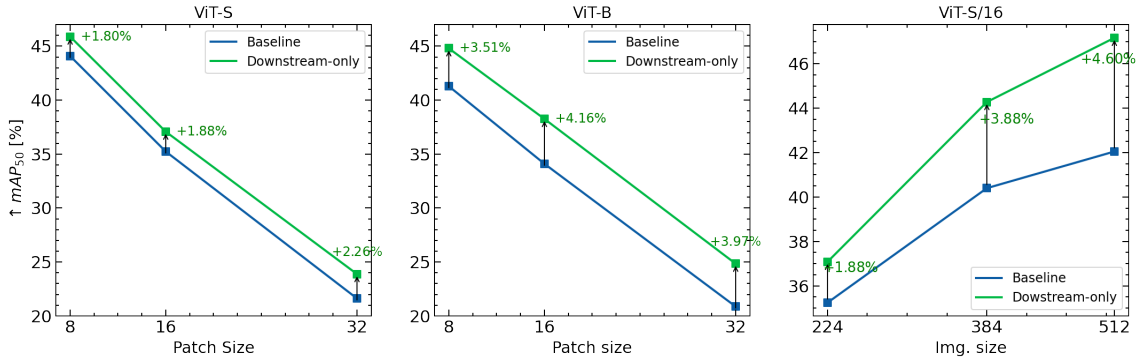


Figure 5.3: mAP_{50} benchmark comparison, using Downstream-only approach and experimenting with different patch sizes for ViT-S (left) and ViT-B (center). mAP_{50} benchmark comparison, using Downstream-only approach and experimenting with different image sizes (right). RelViT always outperforms the baseline trained from scratch.

The performance of RelViT model, for object detection tasks, varying the patch size has been studied. In this case, RelViT was trained following a downstream-only task scheme. The

main metric is the mAP_{50}, since the dataset chosen is PASCAL-VOC. The specific selection of this dataset was due to its popularity and moderate size, which provided an advantageous opportunity to assess the efficacy of the models across a range of complexity levels.
Each image have been uniformly resized to dimensions of $(224, 224)$ in accordance with prior experimental methodologies. This image size gives the possibility to study three different patch sizes, i.e. 8, 16 & 32. As explained in chapter 2, the lower the patch size the higher will be the model complexity as well as the computation required.
Figure 5.3 compares the final accuracy levels obtained using RelViT downstream-only (green curve) with the baseline trained from scratch (blue line). The comparison is done both for ViT-S and ViT-B models. The two colored curves share the same trend for both ViT models when varying the patch sizes. In fact, throughout the plot the improvement remains overall constant, which is in contrast with results found for classifications tasks [2]. Moreover the improvement of RelViT for the base model is more than double with respect to the small one (from 1.9% to 4.2% for patch size $= 16$). This is probably due to the number of parameters that are optimized through RelViT self-supervised learning. In fact the ViT-B model contributes with more than half of the total model parameters (ViT plus Faster R-CNN), while ViT-S contributes for only a third of the total parameter count. Hence suggesting, that for detection tasks, the larger the Vision Transformer the higher will be the accuracy improvement with RelViT. It is also interesting to see how the baseline accuracy of ViT-B is worse than the one of ViT-S baseline.

Furthermore, the performance of RelViT using downstream-only approach were evaluated

(a) (b)

Figure 5.4: ViTs backbone positional embeddings comparison between the baseline (a) and the downstream-only approach (b). The image size is set to 224 and patch size to 16. RelViT helps with the correct learning of the embeddings by providing some spatial information of the input patches to the model.

Method	SpRel	AbsPos	Ang	Dist	↑ mAP_{50}
Baseline	✗	✗	✗	✗	35.24
Upstream	✓	✓	✗	✗	37.14
Downstream	✓	✓	✗	✗	37.08
Downstream	✓	✓	✓	✗	38.43
Downstream	✓	✓	✗	✓	**38.83**
Downstream	✓	✓	✓	✓	38.58

Table 5.5: Ablation experiments varying the Self-supervised tasks on PASCAL-VOC dataset using ViT-S/16. Upstream results in the table refers to the fine-tuning stage. All tasks combination improve the baseline, with the best being the use of absolute, relative positions and distance heads.

using different image sizes. As for the previous ablation study, the model utilized for the experiments was ViT-S/16. Before being fed to the Vision Transformer, the input image is resized to three different values, i.e. 224, 384 and 512. Then, the detection head output is subsequently resized to the original image size in order to evaluate the mAP metrics.
Figure 5.3 (right) illustrates the outcomes of the analyses, indicating that a higher image size has a positive influence on RelViT. ViT-S/16, with an image resized to 512, demonstrates the highest improvement of $+4.60\%$. This is most likely because, as shown in Figure 5.4, RelViT enhances the learning of positional encoding, which is significantly optimized for larger images.

An ablation study on VOC dataset was conducted to test the effectiveness of different self-supervised relations defined in chapter 4. The experiments utilized a ViT-S/16 as the backbone for the model. Previous research has demonstrated that the combination of absolute and relative positions is the most effective out of the four relational heads in classification tasks [2]. Therefore, these two were chosen as the default for all the experiments reported.
According to the results displayed in Table 5.5, the inclusion of the distance relation in object detection tasks results in an enhancement of $+1.75\%$ in the outcomes. Conversely, incorporating also the angle attention head, to the three previous ones, does not provide any benefit to RelViT and leads to a decrease in overall performance.

Baseline	RelViT
Success cases	
	
Failure cases	
	

Table 5.6: Qualitative comparison between the baseline and RelViT results on PASCAL-VOC dataset, using ViT-S/16. On the upper part the success cases in which RelViT improves over the baseline, on the lower part the failure cases of both models. The images display only predictions with the correspondent confidence score above 0.5.

6
Conclusion

This thesis work focuses on the investigation of self-supervised techniques, based on spatial relations among patches of the input image. The goal is to improve Vision Transformer results specifically for object detection and instance segmentation. This necessity arises form the poor performing benchmarks of ViTs when trained form scratch on small-sized datasets. The underlying model used is RelViT, which consists in four relational heads that are attached to the plain ViT backbone. Only two heads, named relative position and absolute position, are used to learn the self-supervised input signals through a MSA layer. The first head objective is to learn where a patch x with respect to a patch y. The second learns the numerical order of the patches in the original image. RelViT has been shown to outperform the plain ViT baseline trained from scratch in classification tasks [2].

In order to extend the same analysis for detection tasks, specific frameworks must be added, i.e. Faster R-CNN or Mask R-CNN, in a way such that the ViT serves as the backbone for feature extraction. The detection head, then learns to identify regions of interest within the image, assigning objectness, scores to those regions, and performing bounding box regression to precisely localize each object.

The experiments conducted in this study highlight the advantages of using self-supervised tasks during a pre-training phase, followed by transferring the acquired knowledge to a supervised fine-tuning task. This approach can ultimately lead to improved performance and reduced reliance on large labeled datasets.

Evaluating RelViT model on multiple detection benchmarks, brings significant improvement

across all the datasets considered. Specifically, on a subset of the COCO dataset, RelViT exhibited an increase of $+2.70\%$ and 2.20% in terms of mean average precision (mAP_{box}) and mean average precision for mask prediction (mAP_{mask}), respectively.
The performance of RelViT was also evaluated using different ViT based architectures, such as Swin-ViT and T2T-ViT. Results again demonstrated a consistent improvement with respect to the baseline, thereby underscoring the efficacy of the RelViT strategy in diverse contexts.
Ablation studies were conducted to investigate the impact of different patch sizes for detection tasks, showing that the overall improvement achieved by RelViT remains constant for both ViT-S and ViT-B models across multiple patch size values. Additionally, the improvement in performance for the ViT-B model is observed to be twice that of the ViT-S model, suggesting that the efficacy of RelViT increases with the size of the Vision Transformer backbone.
Finally, experiments were conducted to determine the optimal combination of relational heads to employ. The analysis revealed that the best overall performance is attained by utilizing the distance head in conjunction with the two primary positional heads.
In conclusion, although RelViT outperforms the baseline in all detection tasks, it does not quite match state-of-the-art results when using an already pretrained model as the initial starting point. Thus, further investigation is necessary to enhance RelViT's performance and competitiveness with the current state-of-the-art models. Furthermore, it may be worthwhile extending the same analysis to video benchmarks by transforming spatial relation tasks into temporal relation tasks. This would provide an avenue for exploring RelViT in video-related applications and make a meaningful contribution to the current knowledge of the effectiveness of self-supervised techniques.

References

[1] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[2] X. Ma, W. Nie, Z. Yu, H. Jiang, C. Xiao, Y. Zhu, S.-C. Zhu, and A. Anandkumar, "Relvit: Concept-guided vision transformer for visual relational reasoning," 2022. [Online]. Available: https://arxiv.org/abs/2204.11167

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[5] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," 2021. [Online]. Available: https://arxiv.org/abs/2104.14294

[6] S. A. A. Ahmed, M. Awais, and J. Kittler, "Sit: Self-supervised vision transformer," *CoRR*, vol. abs/2104.03602, 2021. [Online]. Available: https://arxiv.org/abs/2104.03602

[7] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[8] ——, "Cifar-100." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[9] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," 2021. [Online]. Available: https://arxiv.org/abs/2103.14030

[10] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z. Jiang, F. E. Tay, J. Feng, and S. Yan, "Tokens-to-token vit: Training vision transformers from scratch on imagenet," 2021. [Online]. Available: https://arxiv.org/abs/2101.11986

[11] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," 2015. [Online]. Available: https://arxiv.org/abs/1505.05192

[12] S. Bucci, A. Dx27;Innocente, Y. Liao, F. M. Carlucci, B. Caputo, and T. Tommasi, "Self-supervised learning across domains," *IEEE Transactions on Pattern Analysis amp; Machine Intelligence*, vol. 44, no. 09, pp. 5516–5528, sep 2022.

[13] S. Lao, Y. Gong, S. Shi, S. Yang, T. Wu, J. Wang, W. Xia, and Y. Yang, "Attentions help cnns see better: Attention-based hybrid image quality assessment network," 2022. [Online]. Available: https://arxiv.org/abs/2204.10485

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[15] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2015. [Online]. Available: https://arxiv.org/abs/1506.01497

[16] R. Girshick, "Fast r-cnn," 2015. [Online]. Available: https://arxiv.org/abs/1504.08083

[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2015. [Online]. Available: https://arxiv.org/abs/1506.02640

[18] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," 2017. [Online]. Available: https://arxiv.org/abs/1703.06870

[19] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," 2017. [Online]. Available: https://arxiv.org/abs/1708.02002

[20] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2014, cite arxiv:1405.0312Comment: 1) updated annotation pipeline description and figures; 2) added new section describing datasets splits; 3) updated author list. [Online]. Available: http://arxiv.org/abs/1405.0312

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[22] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," 2016. [Online]. Available: https://arxiv.org/abs/1612.03144

[23] Y. Li, H. Mao, R. Girshick, and K. He, "Exploring plain vision transformer backbones for object detection," 2022. [Online]. Available: https://arxiv.org/abs/2203.16527

[24] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," 2018. [Online]. Available: https://arxiv.org/abs/1803.07728

[25] X. Chen and K. He, "Exploring simple siamese representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 15 750–15 758.

[26] 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, 2012.

[27] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, Aug. 2013. [Online]. Available: https://doi.org/10.1177%2F0278364913491297

[28] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng, "Reading digits in natural images with unsupervised feature learning," *NIPS*, 01 2011.

[29] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar, "Cats and dogs," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

Acknowledgments

Questi anni di università mi hanno permesso di capire molte cose su me stesso. Durante gli alti e bassi, voi tutti ci siete sempre stati.

Non posso che iniziare ringraziando la mia famiglia per avermi sostenuto durante tutto il percorso universitario, non senza dificoltà.

Grazie Mamma.

Grazie Papà.

Grazie Greta.

Un grazie va alla grande famiglia allargata dei parenti tutti. Grazie Nonni.

Un grazie va al Professor Lamberto Ballan per lo splendido corso che tiene, l'affascinante lavoro che sta perseguendo con tutto il gruppo di VIMP, e per l'opportunità datami nonostante le tempistiche strette.

Un grande grazie va sicuramente ad Elena, per avermi aiutato e seguito durante tutto il lavoro della tesi.

Un grazie va ai miei amici e compagni tutti, che sono decisamente troppi da mettere in questi ringraziamenti. Vi porterò sempre nel cuore.

Il più grande grazie però va a te Giorgia, non so come ma riesci sempre a sopportarmi e al contempo rendermi felice. Grazie.