On the decomposition of visual sets using Transformers

by

Andrea Alfieri

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday July 12, 2021 at 2:00 PM.

Student number:	5128315	
Project duration:	November 1, 2020 – July 12, 2021	
Thesis committee:	Dr. J.C. van Gemert,	TU Delft, supervisor
	Dr. S. Pintea,	TU Delft
	Dr. L. Chen,	TU Delft

This thesis is confidential and cannot be made public until December 31, 2021.

Contents

Preface

The main contribution of this project has been presented in the form of an academic article. The remaining chapters of this report will serve as contextual information that can help the reader understand the full picture of the work carried out over the period of my thesis project in collaboration with Dr. J.C. van Gemert and Ph.D. Yancong Lin. The following lines provide the abstract of our research.

Transformers can generate predictions auto-regressively by conditioning each sequence element on the previous ones, or can produce output sequences in parallel. While research has mostly explored upon this difference on tasks that are sequential in nature, we study this contrast on visual set prediction tasks, to analyze the core behaviour of the Transformer model. Multi-label classification, object detection and polygonal shape prediction are all visual set prediction tasks. Precisely predicting polygons in images is an important set prediction problem because polygons are representative of numerous types of objects, such as buildings, people, or obstacles for aerial vehicles. Set prediction is a difficult challenge for deep learning architectures as sets can have different cardinalities and are permutation invariant. We provide evidence on the importance of natural orders for Transformers, analyze the strengths and weaknesses of different solutions that can solve the set prediction task directly, and show the benefit of decomposing complex polygons into sets of ordered points in an auto-regressive manner.

As Transformers are revolutionizing the Computer Vision research, I have tried with this project to bring my small contribution to the research field that I believe will influence the next decade the most. My time in Delft and these past months have been strenuous, fun, weird, exciting, challenging and full of accomplishments and failures. I have learned much more than I could have ever hoped for. I would like to thank Yancong for his help as my daily supervisor and for his kind support within and without the scope of this project. As we are both approaching the end of an important segment of our lives, I want to wish you the best of luck for your next steps. I would like to thank Dr. van Gemert for his support throughout these months, for believing in my capabilities from the beginning of the project, and for pushing me above my own boundaries. Good mentorship holds an immeasurable value for me.
I would like to thank Dr. Pintea and Dr. Chen for their interest in my thesis, evaluation of my work, and help throughout my Master's degree and my thesis project.

Andrea Alfieri
Delft, July 2021

1

Academic article

On the decomposition of visual sets using Transformers

Andrea Alfieri
Delft University of Technology

Abstract

Transformers can generate predictions auto-regressively by conditioning each sequence element on the previous ones, or can produce output sequences in parallel. While research has mostly explored upon this difference on tasks that are sequential in nature, we study this contrast on visual set prediction tasks, to analyze the core behaviour of the Transformer model. Multi-label classification, object detection and polygonal shape prediction are all visual set prediction tasks. Precisely predicting polygons in images is an important set prediction problem because polygons are representative of numerous types of objects, such as buildings, people, or obstacles for aerial vehicles. Set prediction is a difficult challenge for deep learning architectures as sets can have different cardinalities and are permutation invariant. We provide evidence on the importance of natural orders for Transformers, analyze the strengths and weaknesses of different solutions that can solve the set prediction task directly, and show the benefit of decomposing complex polygons into sets of ordered points in an auto-regressive manner.

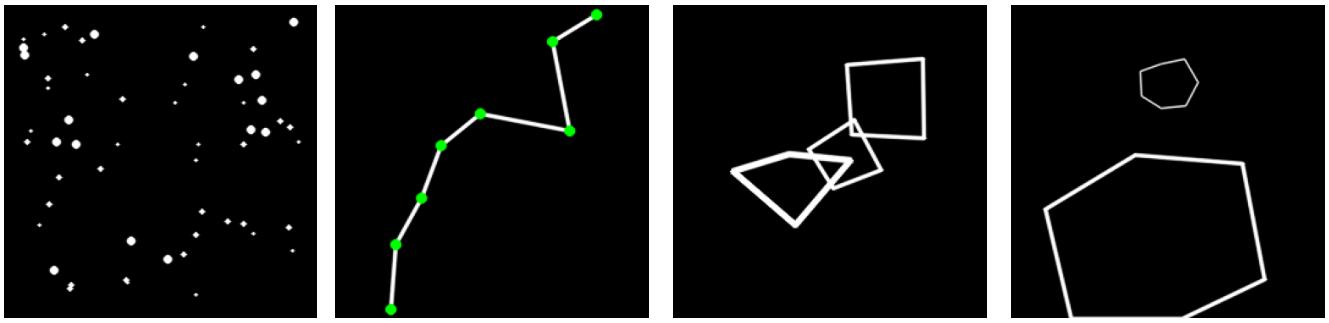
1. Introduction

Predicting polygonal shapes in images is a high-level task that can be representative of different problems that the literature has tried solving with ad-hoc neural network solutions. The main example of this is about the automatic extraction of buildings from high-resolutions satellite images [11, 20], which can create opportunities for urban planning and world population monitoring [15]. Similarly, polygon detection can be used by the vision-based flight control system of unmanned aerial vehicles (UAVs) to recognize complex shapes in the environment with the use of a single camera. The performance of the detection module is important as it directly affects the precision of the UAV's navigation. This is the challenge that motivated the creation of autonomous drone racing events, where UAVs navigate through obstacles without any help from pilots [16].

In this work, we look at the polygonal shape prediction task as a *set prediction* task: many kinds of data can be naturally represented using sets and many machine learning tasks can be viewed as a set prediction problem, such as predicting the set of points of a polygon [41], detecting objects in an image [4, 40], estimating the pose of humans by detecting a set of key-points [3, 33], or predicting multiple labels for the same sample [29]. The difficulty in such problems arises for two main reasons: because the *cardinality* (i.e. the number of elements) of the sets is unknown and can vary among different samples, and because sets are naturally *permutation-invariant*, since the order of the elements does not matter. Both of these are properties that many canonical deep learning models are not able to work with.

Transformers [34] have achieved interesting results on the set prediction task [4, 19], because of their ability to perform the global computation required by this problem through the attention module [1], a solution capable of aggregating information from the entire set while also satisfying the permutation invariance property.

Despite being first presented as an auto-regressive sequence-to-sequence model [34] that could generate output tokens one by one, different works [5, 10, 13] have tried to introduce Transformer models capable of producing the output tokens in parallel, to reduce latency during inference. In fact, thanks to the use of *masked* self-attention [34], the auto-regressive approach can be parallelized at training time, when the target samples are known, but this cannot happen at test time as the model needs to actually generate a token to condition the following one on it. However, the vast majority of these works focused on tasks that are sequential in nature such as machine translation [10, 13] or speech recognition [5], and could not outperform their auto-regressive state-of-the-art counterpart. On the other hand, parallel Transformers have successfully been used to solve the set prediction task directly [4, 23], but required additional computationally intensive solutions such as oversampling or the use of heavy matching algorithms. In this work, we study upon the difference between these models on set prediction tasks, which are not sequential in nature, to highlight their strengths and weaknesses.


(a) Toy datasets. From the left to right: points, line, gates and polygons

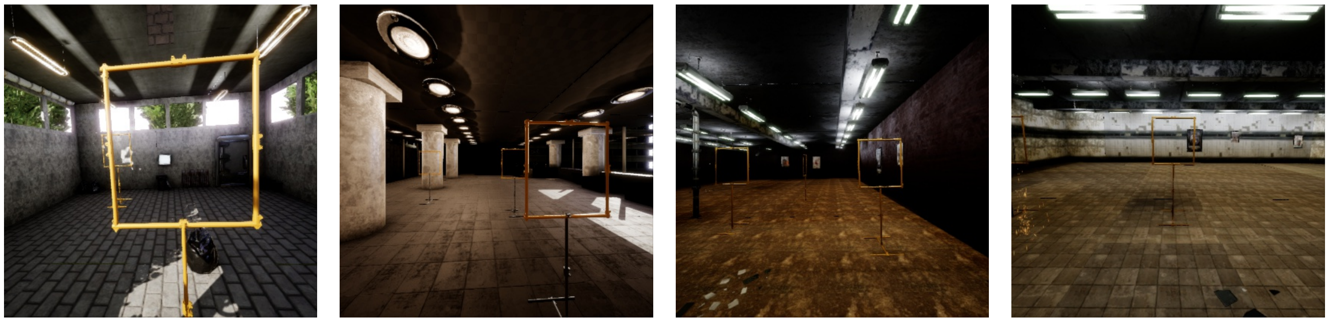

(b) Real gate dataset. The two leftmost images are examples of the two scenes from the training set where light sources are internal and external. The others are from the test set where light only comes from sources inside the room.

Figure 1: **Examples from all five datasets.**

The difference between parallel and auto-regressive Transformers is fundamental. Let's consider an image containing three objects which we are trying to detect and three random variables A, B and C representing the positions of the objects in the image. When asking a parallel Transformer to learn this task, we are actually asking the model to learn the joint probability of these three variables, conditioned on the model parameters Θ, namely:

$$P(A, B, C \mid \Theta) \tag{1}$$

In contrast, when the auto-regressive approach is presented with the same task, what it needs to learn is the chain of conditional probability distributions defined by:

$$P(A \mid \Theta) \cdot P(B \mid A, \Theta) \cdot P(C \mid A, B, \Theta) \tag{2}$$

In the second case, the Transformer is asked to produce one element of the set at a time, by exploiting information about the previous predictions. This is similar to machine translation, in which the Transformer outputs one word, conditioned on the previously generated sentence. Equations 1 and 2 are equal by definition of the general product rule of probability, and Transformers are the first general architecture that is capable of modelling both. However, using equation 2 with Transformers imposes an order even when there is no natural one in the task we are facing, both on the predictions

$$P(A) \cdot P(B \mid A) \quad \text{vs} \quad P(B) \cdot P(A \mid B) \tag{3}$$

and the conditions

$$P(A \mid B, C) \quad \text{vs} \quad P(A \mid C, B) \tag{4}$$

We show that the conditional decomposition of the set can be beneficial for Transformers when there is an explicit order in the prediction of the elements of the set, such as when predicting the set of points in a line. Moreover, we try to provide more insights regarding the importance of the prediction orderings and the order of the conditional variables.

We implement different auto-regressive and parallel models and test them on four set prediction datasets and one sequential dataset to provide the reader with a full picture of the advantages and disadvantages of both approaches in this novel scenario.

2. Related work

Similarly to how generating random numbers is complicated for deterministic computing machines, generating sets of arbitrary order and cardinality can be arduous for neural networks working with vectors of pre-determined shape. A

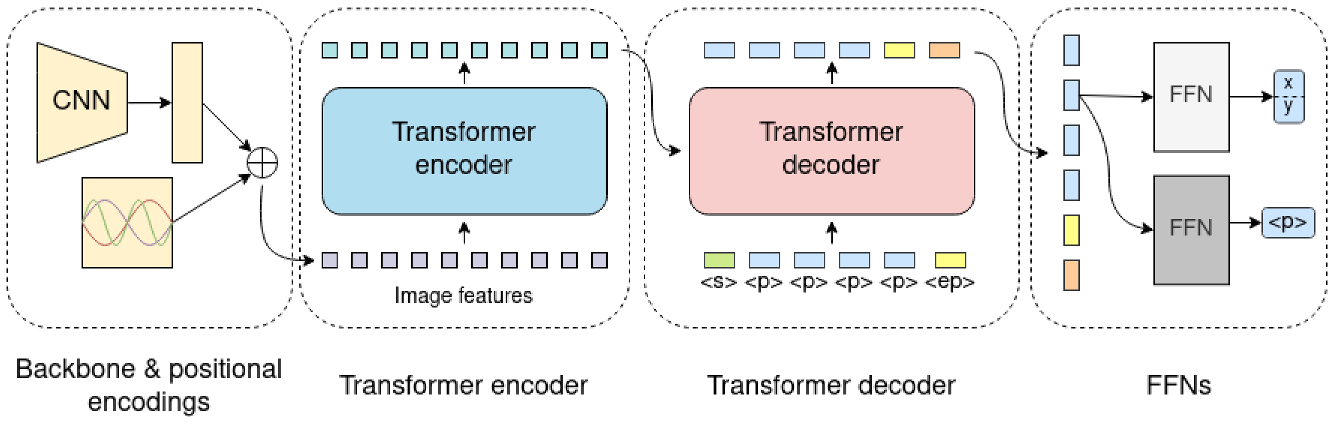


Figure 2: **The auto-regressive model for predicting polygons.** This model exploits the *sentences of tokens* idea [34] which generates tokens auto-regressively. There are only four possible tokens in its vocabulary: *start*, *point*, *end-of-polygon* and *end*.

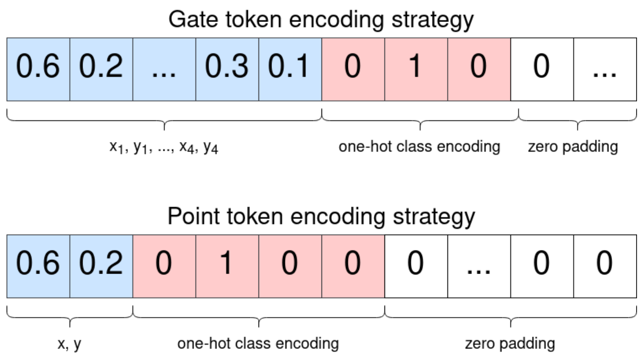


Figure 3: **Embeddings for the point and gate classes**. The first $2n$ elements (in blue) of the vector represent the (x, y) coordinates of the n vertices/points, normalized by the image width and height. The following elements (in red) represent the class label in one-hot format. The vector is padded with zeros to reach the required dimensionality of 256.

ploy commonly used to solve this problem is to predict more elements than necessary and then discard the excess. This is the case for common object detectors [21, 27] that implement postprocessing strategies like non-maximal suppression on a great number of *anchor boxes*, but also for novel Transformer-based architectures like DETR [4], which predicts n so-called *object queries* and then requires the model to classify most of them into the *no-object* class, therefore setting an inductive bias towards the maximal cardinality n of the output set. In this case, having such an inductive bias directly affects the model speed since the complexity of the attention layer is $O(n^2)$ with respect to the sequence length. Moreover, the target labels are matched optimally to the DETR output using the Hungarian algorithm [18], which is also $O(n^3)$ in the worst case scenario. Auto-regressive Transformers are missing both of these components and can be trained in much less time [17].

The majority of set prediction architectures has built upon works that studied the mathematical properties of deep learning architectures capable of working with input and output sets [19, 35, 36, 39]. Zaheer *et al*. "Deep Sets" model [39] has proven that transforming the elements of the input set into some latent representation and then combining them through a permutation invariant function is a universal approximator of any set function. With some limitations, the attention layer of Transformers can also be viewed as a generalization of the sum operation of Deep Sets and is therefore also a universal approximator of set functions [36]. Moreover, Lee *et al*. [19] demonstrated that some information about the interaction of the set elements needs to be discarded during the set pooling operation, which does not happen when using the self-attention mechanism to aggregate features, and showed how using attention can prevent from under-fitting for problems with stronger interactions among the set elements.

Transformers were originally introduced by [34] as novel auto-regressive, sequence-to-sequence models, and gained popularity thanks to their ability to dispense entirely with recurrence and support parallel processing of sequences. Their stunning results on machine translation and other language tasks [2, 7, 22, 26] have recently shed a light on these architectures from the computer vision community, which has successfully designed Transformer-based architectures for image recognition [8], object detection [4], segmentation [38] and other visual tasks [6, 12, 32, 37].

Parallel decoding for Transformers is a solution that was previously applied also in the machine translation domain, but required strong inductive biases such as an iterative refinement of the output [10] or the use of *fertilities* [13] to reach competitive results. However, auto-regressive solutions still remain the state-of-the-art in this field. Strong baselines of this kind have also been hard to outperform in the domain of speech recognition by parallel models [5]. In our work, we show once and for all that conditioning each prediction on the previous ones serves as a strong inductive bias for tasks that present a natural order in their sequences such as machine translation and speech recognition.

3. Method

This section describes the datasets and the models used to study auto-regressive and parallel Transformers on set prediction tasks of diverse complexities.

3.1. Datasets

Examples from all of the datasets used for our experiments are shown in figure 1. We use 4 toy setting datasets and 1 dataset of realistic synthetic images. The toy setting images are manually generated during training and if not specified otherwise, we generate 3 million images for training and 10.000 images for testing.

The **points toy setting** dataset contains images with n white points randomly distributed over the image space. Each point's size is uniformly sampled from three possible sizes. The task is to predict the x, y coordinates of all points in any order. This dataset is an instance of a pure set prediction problem, with points representing the set elements.

The **line toy setting** dataset contains images picturing a single white line composed by 7 segments. A green point of fixed size is placed on top of each end of a segment. The line is generated by first drawing a straight line going from the bottom left corner of the image to the top right corner, and then randomly shifting 8 equally distributed points perpendicularly to the line direction by $r \in [-15\%, 15\%]$ with respect to the image size. The task is to predict the x, y coordinates of the 8 points following the line order, starting from the end on the bottom left of the image. The set elements are represented in this case by the green points, and this dataset is an example of a set prediction task where there exist an explicit order in which we need to predict its elements.

The **gates toy setting** dataset contains images with n polygons of 4 corners, called *gates*. Each gate is generated by defining 4 equally spaced points on a circumference of random radius $r \in [5\%, 40\%]$ with respect to the image size. Each point is then shifted randomly in the direction of the radius and the four points are finally connected to define the gate. All four edges of the gates are of the same thickness, uniformly sampled out of 3 possible choices.

The **real gates** dataset [9] contains realistic synthetic images generated with a graphical engine, which simulate the flight of a drone in different environments containing empty wire frame objects (EWFOs or *gates*). This dataset is a simulation of the images that a UAV would face in the IROS 2018 Autonomous Drone Race [16]. The train set contains 26.000 images from two different scenes, with light coming from outside and inside the rooms. The test set contains 3.000 images from two additional scenes with light coming only from artificial sources placed inside the rooms. For all scenes, different walls and pavement textures are used, as well as different artificial shapes and light intensities for internal lamps. All images contain 1 to 4 gates. For this dataset and for the gate dataset, the task is to predict the position in the image of the four corners of each gate. These two datasets represent a set prediction scenario closer to real-life problems, where the complexity of the single set element prediction (the gate) is greater than a simple point prediction. The real gates dataset stresses this complexity even further by picturing gates with different backgrounds and different lighting situations and allows us to investigate upon the models' behaviours on limited amounts of data.

Finally, the **polygons toy setting** dataset contains images with n polygons of $m \in [3, 7]$ corners, generated using the same technique as the gates toy setting dataset. As multiple non-convex polygons can be represented by the same set of points, the task is to predict the m corners of a polygon in clockwise order. Any starting point is accepted, and distinct polygons can be predicted in any order. This dataset represents the hardest set prediction task in which we require our models to predict a *set of ordered sets*.

For all datasets, n is an adjustable parameter that can be customized for different experimental settings. All toy setting samples consist of images of dimension 256x256 with 3 color channels, while images from the real gates dataset have shape of 400x400. The positional coordinates of the target labels are represented by vectors with values $\in [0, 1]$ as height/width relative to the image size.

3.2. Models

All of the models that we implemented and tested are derived from DETR [4], an end-to-end Transformer which achieves competitive results against Faster R-CNN [28]. It takes advantage of a CNN backbone and a parallel encoding-decoding strategy to solve object detection as a set prediction task. In short, DETR is composed of four modules: a CNN backbone, a Transformer Encoder, a Transformer Decoder, and a feed-forward network (FFN). An example of one of our models is shown in figure 2. It pictures the auto-regressive version used for the polygon toy setting dataset.

DETR takes as input an RGB image and extracts a high-level feature map of shape $d \times (H \times W)$, where d is the size of number of channels, and H and W are the spatial dimensions. The feature map is supplemented with fixed positional embeddings before the Transformer encoder. The Transformer encoder is a stack of 6 self-attention mechanisms, each of which consists of a standard self-attention layer followed by a feed forward network, a residual connection and layer normalization [4]. All of the models we tested are identical up to this stage of the architecture, but differ in the subsequent modules.

3.2.1 Parallel models

The design of the Transformer decoder of parallel models is similar to the one in DETR, which is a stack of 6 multi-head self-attention mechanisms [4]. The self-attention layer takes the output of the previous module as queries, and the output of the Transformer encoder as keys and values. The decoder takes as input *object queries*, which are N learned vectors of dimension 256. The N *object queries* are converted *in parallel* into N output embeddings by the decoder, which are subsequently fed into a simple feed forward network (FFN) for classification and position regression. Each of the N output embeddings is a prediction and N is generally much larger than the actual number of objects. Therefore, a *no-object* class label is required to identify slots that are not responsible for any prediction. During training, the N predictions are matched to the target labels by the Hungarian algorithm [31]. The cost matrix for matching is the weighted sum of the classification loss and the bounding box regression loss.

Depending on the task, we modify the dimensionality of the FFN for position regression accordingly. For example, on the points and the line datasets, the output is a vector of size 2 representing the (x, y) coordinates of a point normalized by the image width and height, while the output on the gate dataset is an 8-element vector indicating the four vertices of a gate in clock-wise order. The class labels for all datasets are the same, namely the *object* class and the *no-object* class.

On the polygon dataset, the FFN is replaced by a simple multi-layer Elman RNN [30], since the output dimension is also a variable. The RNN model takes as input the embeddings from the *object* class and generates a sequence of points one by one. The RNN is the smallest possible modification we could make to the architecture to work with polygons and have the smallest impact on the model's properties and our experiments.

3.2.2 Auto-regressive models

The auto-regressive models diverge from the parallel ones on all set prediction tasks, because they work with *sentences of tokens* that are generated one by one, by conditioning the next prediction on the previous ones. In particular, each token is a vector of dimension 256 that represents an element of the set, or acts as a special representation. Special tokens can be:

- *start* or **S**: This token is at the beginning of all sentences and defines the starts of the computation.
- *end* or **E**: This token is at the end of all sentences and terminates the computation.
- *end-of-polygon* or **EOP**: This token separates polygons from polygons inside the same image. It acts similarly to the *period* token used in machine translation to separate words belonging to different sentences.

Depending on the task, object tokens can be:

- *point* or **P**: On the point dataset and the line dataset, each point is a 2-element vector representing the position of the (x, y) coordinate normalized by the image width and height.
- *gate* or **G**: On both toy and real gate datasets, a gate is represented as a vector of size 8 $p \in [0, 1]^8$, which defines the normalized coordinates (x, y) of all 4 vertices.

To batch sequences of different lengths together, we pad sentences with the *end* token up to the same length. For example, given a batched sequences where the maximal length is 8, we pad the shorter sequence with 4 visible points only with two extra **E**:

$$\mathbf{S},\ \mathbf{P},\ \mathbf{P},\ \mathbf{P},\ \mathbf{P},\ \mathbf{E},\ \mathbf{E},\ \mathbf{E}$$

We use the same Transformer decoder as the parallel models, but adopt the *masking* technique for parallel training, which prevents each token from attending to subsequent positions [34]. The token embedding is a fixed length vector of size 256 as shown in figure 3.

The auto-regressive approach predicts all polygons in a certain order. If not specified otherwise, we always predict objects going from left to right in the image, splitting ties with top-to-bottom order. We sort polygons and gates accordingly by their centers, computed as the average of their vertices. Particularly, for the polygon dataset, we also impose an order on **points** such that the polygon is clockwisely defined by the points, otherwise the same set of the **points** may result in several polygons of different shapes.

4. Experiments

First, we validate on the toy line and point datasets that the auto-regressive approach is preferable when there is an explicit natural order in the predictions and that the parallel

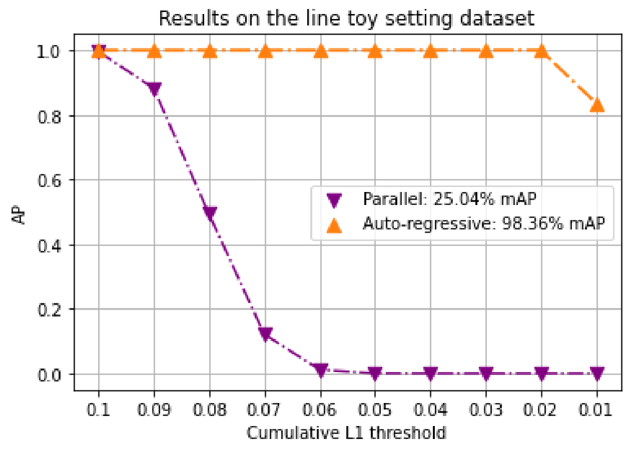


Figure 4: **Results on the line dataset.** Both networks are able to predict the green points in the correct order within a threshold of 0.10, but the auto-regressive one is much more precise and achieves a perfect AP up to a threshold of 0.02.

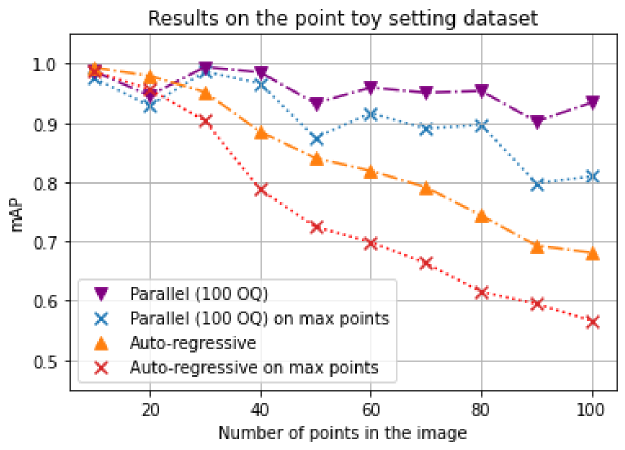


Figure 5: **Results on the points toy setting dataset.** The auto-regressive model shows marginally better performance than the parallel one on sets of limited cardinality, but its performance is lacking when the set cardinality grows. *OQ* stands for *object queries*. The models with *max points* represent results on images with exactly n points while the ones without *max points* indicates that the number of points in an image varies from 1 to n.

solution is better for pure set prediction tasks. We then compare the two strategies on more challenging gate datasets where the set element is no longer a single point and where data is scarce. Finally, we explore deeper into the auto-regressive solution to study upon the importance of the prediction order and the order of the conditional variables.

On toy datasets, we train all models with 3.000.000 images. On the real gate datasets with 26.000 images, we train all models for 300 epochs. We apply multiple data augmentation techniques, including horizontal flip, vertical flip, hue shift, Gaussian noising.

We train all models on a single NVIDIA GeForce RTX 2080 Ti GPU with AdamW [24], and set the Transformer's learning rate to 10^{-4}, the backbone's learning rate to 10^{-5}, and the weight decay to 10^{-4}. The learning rate is dropped by a factor of 10 after 200 epochs, or after 2.000.000 images for the toy settings. Mask R-CNN [14] is also trained for 300 epochs with learning rate 0.005, which is decreased by 10 after 200 epochs.

4.1. Evaluation

The evaluation metric is mean Average Precision, averaged over different IoU thresholds ($[0.50, 0.55, \ldots, 0.95]$) for gates and polygons, and over different point-to-point L_1 distance thresholds ($[0.10, 0.09, \ldots, 0.01]$) for points, computed directly on the relative coordinates.

4.2. Line and point datasets

With this experiment we show that the conditional decomposition of sets by auto-regressive Transformers is beneficial in the presence of a natural order in the elements of the set. Results on the lines toy setting dataset are shown in

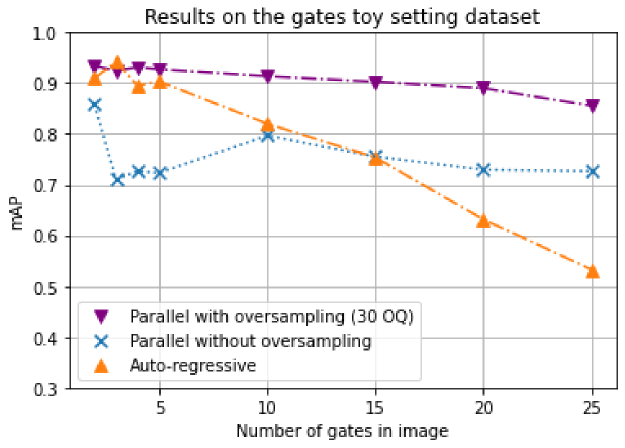


Figure 6: **Results on the gates toy setting dataset.** Each point of the plot is an experiment in which the model is trained and tested on images with 1 to n gates. They show how oversampling is necessary for the Transformer and how the conditional model is not strong against larger sets. *OQ* stands for *objects queries*.

figure 4. For this task, a line prediction is considered a false positive if the sum of all L_1 point distances is greater than the given threshold. The experiment shows that in this setting the auto-regressive solution is much more precise than the parallel counterpart as it is able to achieve perfect AP up

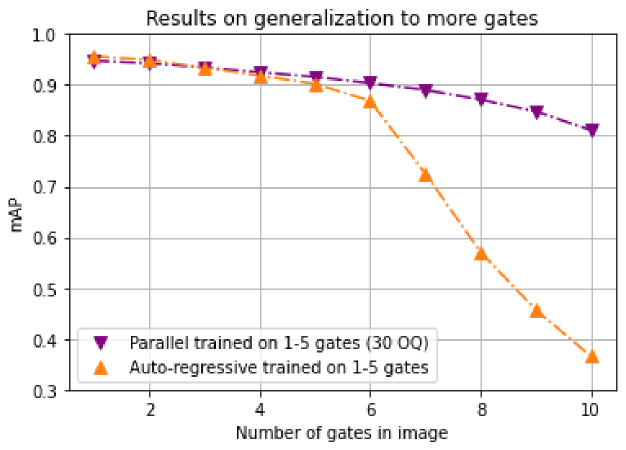


Figure 7: **Generalization capability of the different models.** The parallel one is able to detect many more gates than the auto-regressive one when diverging from the training distribution. We believe oversampling to serve as a strong inductive bias in this sense as each object query is assigned to an object at least once during training.

to a threshold of 0.02.

On the other hand, the experiments on the points toy setting study the behaviour of parallel and auto-regressive models on a pure set prediction task. The models are now expected to predict the n points in any order, but the auto-regressive one is trained by always showing point in left-to-right order. Results on this dataset are shown in figure 5. In this experiment, all models are trained on images with 1 to n points. We present test results on images with 1 to n points, as well as results on images with exactly n points. Results show that on set prediction tasks, the auto-regressive approach is effective when the cardinality is low, but quickly deteriorates as the cardinality of the set increases.

The two experiments on lines and points prove that the presence of a natural ordering in the task faced is indeed an important discriminative factor towards the performance of the parallel and auto-regressive models in set prediction. Moreover, we show the advantage of the auto-regressive approach in low-cardinality set prediction tasks. The parallel ones show significant advantage in predicting high-cardinality sets but at the cost of using redundant object queries.

4.3. Gates toy setting and real gates

The following experiments verify whether the observations on the line and point datasets generalize in complicated scenarios where the set element is a gate of four vertices. First, we evaluate the performances of auto-regressive approach and its parallel counterpart on the toy gate dataset, as shown in figure 6 where removing oversampling indicates that we use as many object queries as the maximal number of objects in the images. The parallel model with oversampling outperforms the one without oversampling substantially, validating the need for the parallel models to oversample the set cardinality. Moreover, we show once more the auto-regressive approach performs comparably to the parallel one on low-cardinality sets, but suffers from the growing cardinality.

In figure 7 we provide our findings on the generalization capabilities of these models, where the number of gates in test images is up to twice more than the number during training. The auto-regressive model struggles at detecting more gates, while the parallel one only shows minor performance decrease and achieves an mAP of 0.8 even on images with twice the amount of gates. We believe this behavior to be directly caused by the oversampling strategy of the parallel models: each of the 30 object queries is assigned to at least one gate during training, which implicitly tells the model that there could be more objects than the ones it sees in each image.

Finally, we numerically compare the general performances of all models on both toy and real gate datasets with images containing 1 to 6 gates. We choose Mask R-CNN as the baseline. Table 1 summarizes our findings and shows parallel Transformer outperforms Mask R-CNN by 5% mAP on the toy setting dataset and by 4% mAP on the real gate dataset. The auto-regressive approach is not able to achieve comparable results as there is no natural order in this task. However, the auto-regressive model requires less than 2 minutes of training time per 10,000 images as it does not require Hungarian algorithm or the oversampling technique.

4.4. Polygons toy setting

Results on the polygon toy setting dataset are shown in table 2. This experiment explores an hybrid scenario in which the set elements we are trying to predict are also sets with an imposed order, because a set of points can represent multiple polygons if the order of its points is not given. The auto-regressive approach is the special one described at the end of section 3.2.2, which predicts polygons as a sequence of points followed by the *end-of-polygon* token, and it outperforms the parallel model substantially by over 20% absolute mAP. We show that the auto-regressive solution can become a viable one when the problem we are facing can be split into smaller problems. We speculate that this could be a direct consequence of the conditional decomposition of the joint probability, as imposing the conditional order on these models can serve as a strong inductive bias by reducing the search space of the model.

Models	Gates toy setting	Training time [min/10k images]	Real gates	Training time [min/10k images]
Mask R-CNN	90.67%	23.6	61.30%	13.85
Parallel	**95.45%**	2.2	**65.00%**	3.85
Auto-regressive	87.67%	**1.5**	59.03%	**1.85**

Table 1: **Numerical results on gate datasets**. Scores are represented as mAP averaged over 10 IoU thresholds [0.50, 0.55, . . . , 0.95]. The parallel model outperforms Mask R-CNN even with limited amounts of data. The auto-regressive model is much faster to train but does not achieve the same performance as there is no natural order in this task.

Models	Polygons toy setting	Training time [min/10k images]
Parallel	53.55%	5.1
Auto-regressive	**76.41%**	**2.15**

Table 2: **Numerical results on polygon dataset.** Scores are represented as mAP averaged over 10 IoU thresholds [0.50, 0.55, . . . , 0.95]. The auto-regressive approach outperforms the parallel one by over 20% absolute mAP. We speculate that this could be a direct consequence of conditional decomposition of the set.

	Using positional encodings	Not using positional encodings
Order based on polygon position in the image	$-1.90\% \pm 1.07\%$	–
Order based on polygon area	$-7.68\% \pm 4.53\%$	$-2.95\% \pm 1.30\%$

Table 3: **Numerical results on the importance of the position encodings and condition orders.** This tables shows mAP difference of adding positional encodings and imposing orders. The top performing model is spatially ordered (left-to-right and top-to-bottom) and has no positional encoding. Our observations are that: (1) changing the imposed order on auto-regressive Transformers has a great impact on the performance as shown in the first column; (2) adding position encodings decreases the performance .

This behaviour might not show up on the previous experiments because gates, points and lines are simple set elements which do not enlarge the search space enough. As polygon detection is representative of many common computer vision tasks, we leave further exploration of this approach as future work.

4.5. Orders

We conclude our experiments by providing results on the importance of orders that are imposed by the conditional decomposition of a set by the auto-regressive Transformer. In particular, we study the influence of different artificial orderings in permutation-invariant tasks. Moreover, we study if making the auto-regressive Transformer invariant to the order of the *previous* predictions can be beneficial for the next token prediction. We run all of experiments on the toy gate dataset, real gates and the polygon datasets multiple times, by using two different object orderings and by adding or removing the positional encodings of the attention layer. Without the positional encodings, the Transformer decoder is unaware of the index in the sequence of the previous predictions. When predicting a new point, it only knows which points were predicted before, but not their orders. The two orderings are the left-to-right, top-to-bottom order and the small-to-large order in terms of the area covered by the objects. Experiment results are shown in table 3, where using the left-to-right, top-to-bottom order and removing the positional encodings lead to the best performance in all experiments. We show the mean absolute difference of other models compared to best one. The result shows that the artificial order *matters* for auto-regressive models. Moreover, we show that fixed positional encodings are not beneficial in this setting, in contrast to the original Transformer architectures. This is expected as different orders of the same words in NLP can have different meaning, while this does not matter in our setting, as knowing the location of a point can already prevent our model to predict the same point twice.

5. Conclusion

In this work, we have investigated upon the two most important variants of the Transformer architecture: the parallel one and the auto-regressive one. We have applied them on a fundamental task such as the set prediction task and we have developed different controlled environments to study their most relevant properties. We showed how the presence or absence of a natural order in the object predictions is a strong factor towards the performance of these models and highlighted some additional properties of the auto-regressive model, such as its strength on sets of small cardinality or on set prediction problems that can be easily split into smaller tasks. Finally, we have investigated upon the different orderings that we are required to fix on the target labels to train the auto-regressive Transformer.

One limitation of this research is that most experiments are conducted on toy datasets only, and it is unclear whether the observations and conclusions on toy datasets generalize on challenging real-world datasets. A task that would be suitable to expand our research could be that of predicting buildings from satellite images. The crowdAI mapping challenge [25] is an example of a dataset where this could be done, as it represents a scenario close to our polygons dataset.

As future work, we find it important to further explore on the polygon detection task as it is a fundamental problem that is representative of different computer vision ones. Testing additional fixed or learned token embeddings and providing evidence upon their effect on the model performance would also be of interest to the community.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 1

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 3

[3] Z. Cao, R. Wang, X. Wang, Z. Liu, and X. Zhu. Improving human pose estimation with self-attention generative adversarial networks. In *2019 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pages 567–572. IEEE, 2019. 1

[4] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020. 1, 3, 4, 5

[5] W. Chan, C. Saharia, G. Hinton, M. Norouzi, and N. Jaitly. Imputer: Sequence modelling via imputation and dynamic programming. In *International Conference on Machine Learning*, pages 1403–1413. PMLR, 2020. 1, 4

[6] H. Chen, Y. Wang, T. Guo, C. Xu, Y. Deng, Z. Liu, S. Ma, C. Xu, C. Xu, and W. Gao. Pre-trained image processing transformer. *arXiv preprint arXiv:2012.00364*, 2020. 3

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 3

[8] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 3

[9] P. Dürnay. Detecting empty wireframe objects on micro-air vehicles: Applied for gate detection in autonomous drone racing. 2018. 4

[10] M. Ghazvininejad, O. Levy, Y. Liu, and L. Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. *arXiv preprint arXiv:1904.09324*, 2019. 1, 4

[11] N. Girard and Y. Tarabalka. End-to-end learning of polygons for remote sensing image classification. In *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*, pages 2083–2086. IEEE, 2018. 1

[12] R. Girdhar, J. Carreira, C. Doersch, and A. Zisserman. Video action transformer network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 244–253, 2019. 3

[13] J. Gu, J. Bradbury, C. Xiong, V. O. Li, and R. Socher. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*, 2017. 1, 4

[14] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 6

[15] V. Iglovikov, S. Seferbekov, A. Buslaev, and A. Shvets. Ternausnetv2: Fully convolutional network for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 233–237, 2018. 1

[16] S. Jung, S. Cho, D. Lee, H. Lee, and D. H. Shim. A direct visual servoing-based framework for the 2016 iros autonomous drone racing challenge. *Journal of Field Robotics*, 35(1):146–166, 2018. 1, 4

[17] L. Kaiser. Tensor2tensor transformers new deep models for nlp, 2017. 3

[18] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. 3

[19] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753. PMLR, 2019. 1, 3

[20] Z. Li, J. D. Wegner, and A. Lucchi. Topological map extraction from overhead images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1715–1724, 2019. 1

[21] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector.

In *European conference on computer vision*, pages 21–37. Springer, 2016. 3

[22] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019. 3

[23] F. Locatello, D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf. Object-centric learning with slot attention. *arXiv preprint arXiv:2006.15055*, 2020. 1

[24] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 6

[25] S. P. Mohanty, J. Czakon, K. A. Kaczmarek, A. Pyskir, P. Tarasiewicz, S. Kunwar, J. Rohrbach, D. Luo, M. Prasad, S. Fleer, et al. Deep learning for understanding satellite imagery: An experimental survey. *Frontiers in Artificial Intelligence*, 3, 2020. 9

[26] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. 2018. 3

[27] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 3

[28] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015. 4

[29] S. H. Rezatofighi, V. K. BG, A. Milan, E. Abbasnejad, A. Dick, and I. Reid. Deepsetnet: Predicting sets with deep neural networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5257–5266. IEEE, 2017. 1

[30] A. Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020. 5

[31] R. Stewart, M. Andriluka, and A. Y. Ng. End-to-end people detection in crowded scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2325–2333, 2016. 5

[32] C. Sun, A. Myers, C. Vondrick, K. Murphy, and C. Schmid. Videobert: A joint model for video and language representation learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7464–7473, 2019. 3

[33] W. Tang and Y. Wu. Does learning specific features for related parts help human pose estimation? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1107–1116, 2019. 1

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017. 1, 3, 5

[35] O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015. 3

[36] E. Wagstaff, F. Fuchs, M. Engelcke, I. Posner, and M. A. Osborne. On the limitations of representing functions on sets. In *International Conference on Machine Learning*, pages 6487–6494. PMLR, 2019. 3

[37] F. Yang, H. Yang, J. Fu, H. Lu, and B. Guo. Learning texture transformer network for image super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5791–5800, 2020. 3

[38] L. Ye, M. Rochan, Z. Liu, and Y. Wang. Cross-modal self-attention network for referring image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10502–10511, 2019. 3

[39] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola. Deep sets. *arXiv preprint arXiv:1703.06114*, 2017. 3

[40] Y. Zhang, J. Hare, and A. Prügel-Bennett. Deep set prediction networks. *arXiv preprint arXiv:1906.06565*, 2019. 1

[41] Y. Zhang, J. Hare, and A. Prügel-Bennett. Fspool: Learning set representations with featurewise sort pooling. *arXiv preprint arXiv:1906.02795*, 2019. 1

2 Attention

This chapter will provide an overview on how the attention mechanism works. After reading this, the reader should be able to comprehend why this mechanism is achieving extraordinary results in different fields of Deep Learning, ranging from natural language processing to computer vision, and how the key components of our Transformer architecture work.

2.1. The dot product as a measure of similarity

In general, the **dot product** is a mathematical operation on vectors. It can be defined as a function $f : (\mathbf{a}, \mathbf{b}) \rightarrow c$ which takes two vectors of equal size $\mathbf{a} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^n$ and returns a single scalar value $c \in \mathbb{R}$. It is defined as:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i \cdot b_i \tag{2.1}$$

but it is in its geometrical representation that it becomes interesting for us:

$$\mathbf{a} \cdot \mathbf{b} = ||a|| \cdot ||b|| \cdot cos(\alpha) \tag{2.2}$$

where α is the angle between the two vectors. Since α defines how the two vectors are oriented with respect to each other, the dot product takes its maximal value when the two vectors point in the same direction and reaches its minimum in case of vectors pointing in opposite directions. This is highlighted in figure 2.1. As modern deep learning architectures work with high-dimensional feature vectors, the dot product can be used to measure their pair-wise similarity. The most common application for this is in Natural Language Processing. If we were to represent words using high-dimensional vectors, we would want words holding a similar meaning to also be similar in their representation. This is the idea that drove the attention mechanism first, and its use in the Transformer architecture then.

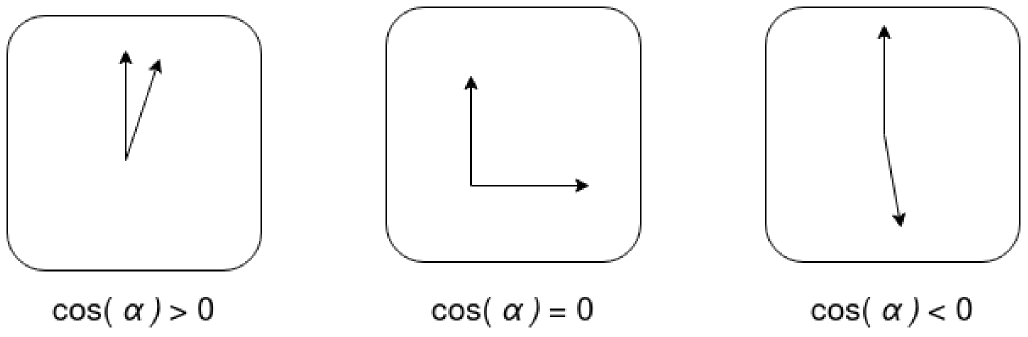


Figure 2.1: As the direction of the two vectors diverge, the dot product decreases. This property can be used as a similarity measure of the two vectors by the attention mechanism.

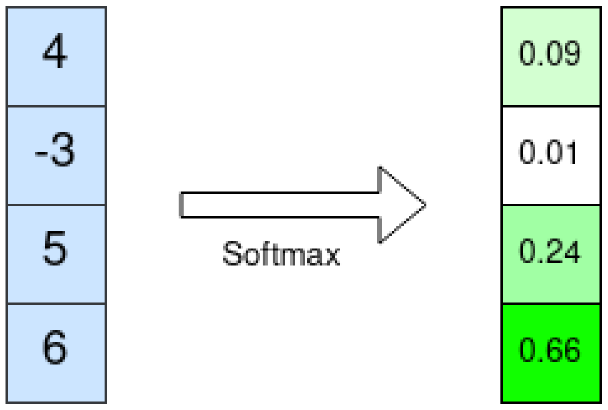


Figure 2.2: The softmax function takes any vector v and transforms it into a vector of the same size whose elements are $v_i \in [0,1]$ and they all sum to one: $\sum_i v_i = 1$

2.2. The Softmax function

The softmax function is a function $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ which takes the elements of a vector **v** and transforms them into a probability distribution. It is defined as:

$$\sigma(\mathbf{v})_i = \frac{e^{v_i}}{\sum_j e^{v_j}} \tag{2.3}$$

The reason why the softmax transforms the vector elements into a probability distribution is because after its application all elements of the output vector are bounded $v_i \in [0,1]$ and sum to one $\sum_i v_i = 1$. Therefore, it is often used as the final layer of a neural network classifier, such that cross entropy loss can be applied directly on its outputs. This is the same setting that we use in the classification branch of the FFNs that follow the Transformer decoder. A visualization of the softmax function can be see in figure 2.2.

2.3. Self-attention

The self-attention module is a simple application of the dot product and the softmax function. Let's consider a set of $n = 4$ feature vectors x_i of dimension d. The self-attention layer scans through the list of feature vectors and considers them one by one. This example is shown in figure 2.3, taken from this blogpost[1], which pictures the step of the x_2 feature vector.

[1]`http://peterbloem.nl/blog/transformers`

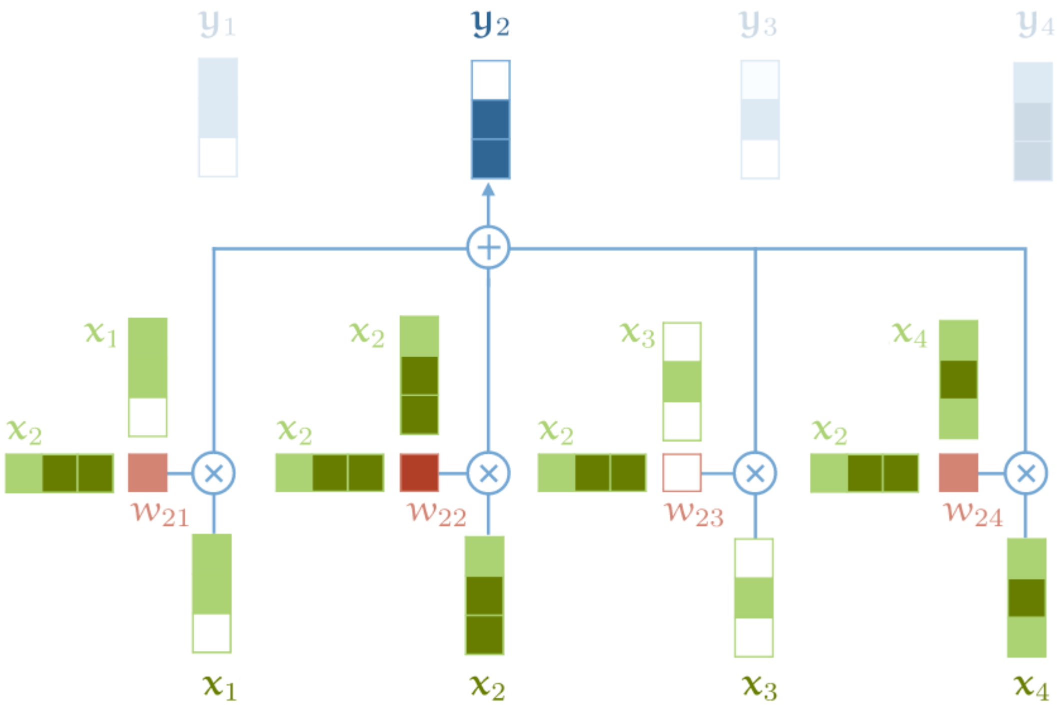


Figure 2.3: The self-attention layer. Please notice how each feature vector is used as *key*, *query* and *value*.

The feature vector x_2 is compared using the dot product to all other vectors $x_1, ..., x_4$ and the results are stored in a vector $\mathbf{o} = [o_1, ..., o_4]$. The softmax function is then applied on this vector to transform it into a set of weights $w_1, ..., w_4$. The weights are pictured in the image ranging from white to red as they are closer to 0 or 1. Please notice how the weight w_{ii} will always be one of the most active ones as x_i is identical to itself. After computing the weights, each feature vector x_i is multiplied at this step by its corresponding weights w_{2i} and the resulting vectors are summed together to finally output the sequence element y_2. In short, the output vector y_2 is computed by taking into consideration only the most similar vectors to x_2, with intensities based on the strength of this similarity. This process is repeated for all input vectors until the sequence is complete.

Each vector is used in three different roles throughout the entire computation. These roles take the name of *key*, *query* and *value*. A vector is a value when multiplied against its weight. It is a query when it is its turn to be compared against all other vectors, and it is a key when being compared against a query.

Naturally, this entire process can be expressed in terms of matrix multiplication to allow for a more efficient GPU computation, but the module still keeps its $O(n^2)$ complexity as each vector needs to be compared against all others.

2.4. Masked self-attention

The attention layer is a sequence-to-sequence layer which gained popularity thanks to its global computation, meaning that each element of the output sequence is computed by using direct information from all input elements. This property turns out to be extremely useful to model long-range interactions among input elements, which is often trivial in natural language processing tasks. Let's take the following sentence as example:

The ***Mona Lisa*** is a half length portrait painting by Italian artist Leonardo da Vinci. Considered an archetypal masterpiece of the Italian Renaissance, ***it*** has been described as "the best known, the most visited, the most written about, the most sung about, the most parodied work of art in the world"

In this sentence, the word "**it**" refers to the "**Mona Lisa**", although these word are very far apart in the sentence. Recurrent models such as GRUs [3] or LSTMs [6] would have problems modeling this type of interaction, which is why attention based architectures such as BERT [4] and GPT-3 [1] have been achieving incredible results on these tasks.

To work with sequences of words, we would like to keep the parallel computation property of the attention layer while also avoiding output tokens to be generated using information from "the future". For example, we can consider the task of synthetic word generation, in which we would like to feed our model with an incomplete sentence and ask it to generate new words. This is the scenario in which the **masked self-attention** technique can be useful.

Let's consider again the beginning of the previous example sentence. We want to feed our model the entire sequence $\mathbf{x} = \{x_1, ..., x_n\}$ = "The Mona Lisa is a half length portrait painting" and expect as output the same sequence shifted by one to the left $\mathbf{y} = \{y_1, ..., y_n\}$ = "Mona Lisa is a half-length portrait painting by". Each word should be generated only by looking at the previous ones.

If we were to solve this task using the self attention layer as described in the previous section, the token y_7 = "portrait" could just look at the input token x_8 = "portrait" and copy-paste the input to the output. To avoid tokens from attending to subsequent positions, we can *mask-out* them by manually setting the dot product to $-\infty$. After running the softmax, this weight will then be calculated as zero and information from these vectors would not flow to the output. In the example of image 2.3, this would mean to manually set the products of $x_2 \cdot x_3$ and $x_2 \cdot x_4$ to $-\infty$. The softmax will then compute the weights and finally output $w_{23} = w_{24} = 0$.

2.5. Keys, queries, values and multiple heads

To be precise, each feature vector is not used as it arrives at the input by the attention layer, but is first transformed into queries, keys and values by using three learned matrices **Q**, **K** and **V** which allow the model to move each vector to a different latent representation before these operations.

Also, novel Transformer models often use a technique called **multi-head attention**. It is identical to self-attention, but splits each input vector into m vectors of dimension $\frac{d}{h}$ with h being the number of heads. The self-attention mechanism is then applied on all heads independently and by using different learned matrices. The outputs are finally concatenated back into a single sequence of vectors of dimension d. This technique expands the module ability to focus on different positions and gives the attention layer multiple representation subspaces to work with.

Another important detail concerning the dot product is the **scaling** technique. If we consider two feature vectors $\mathbf{v}$ and $\mathbf{u}$ of dimension $d_k = 256$ whose components are independent random variables with 0 mean and a variance of 1, computing their dot product $\mathbf{v} \cdot \mathbf{u} = \sum_{i=1}^{d_k} v_i u_i$ would give us a vector whose components have zero mean and variance d_k. For stability during training, we want these values to have a variance of one. Therefore, we divide by $\sqrt{d_k}$ before applying the softmax. The final attention layer is expressed in terms of matrix multiplications by:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.4}$$

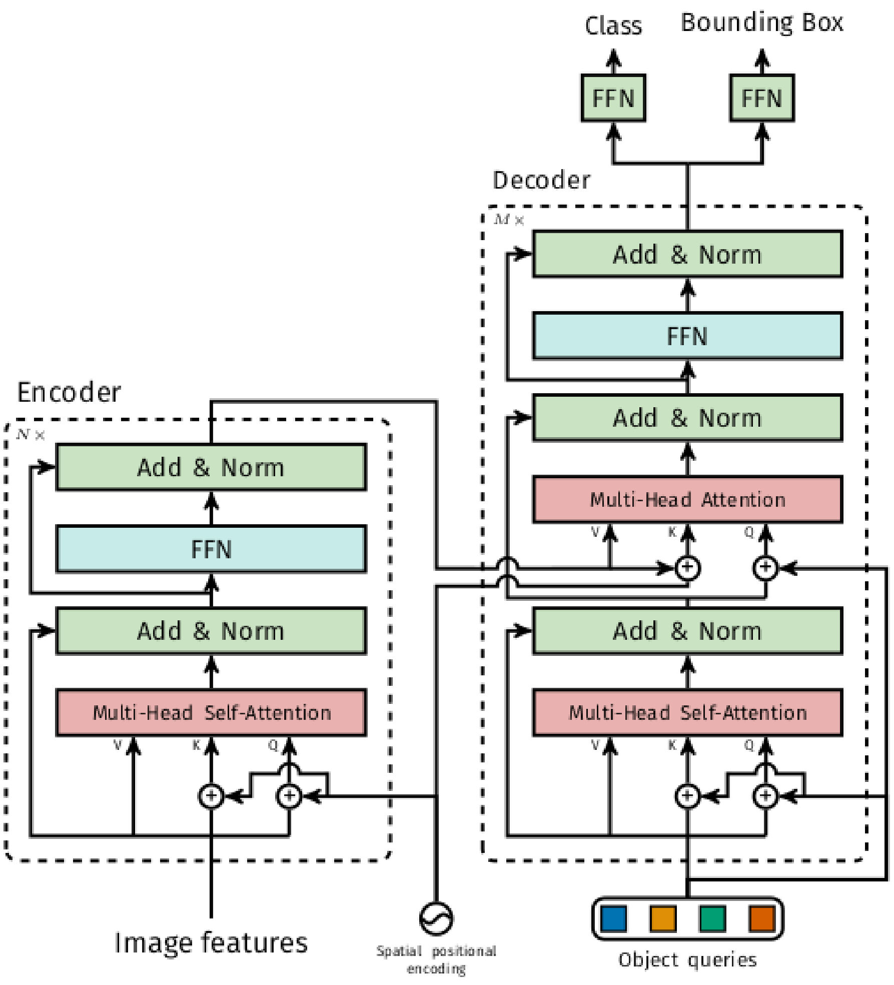


Figure 2.4: The DETR Transformer architecture. Image taken from [2].

Finally, the last detail which is necessary to understand the attention mechanism used by the Transformer architecture is the difference between *attention* and *self-attention*. Attention simply consist in using multiple different sentences as input to the attention layer. All sentences must contain tokens of identical size to allow for the dot product computation, but the sequence length can vary. For example, the attention layer of the Transformer decoder of our architecture takes its queries from the output sequence of the previous self-attention layer, but takes keys and values from the output sequence of the Transformer encoder. The rest of the computation is identical to the one describe above.

2.6. The Transformer architecture

A visualization of the Transformer architecture can be seen in figure 2.4. It pictures the version defined by [2], called DETR, from which we derived all of our models. It is divided into a stack of $N = 6$ encoders and $M = 6$ decoders. The encoders contain a multi-head self-attention module which works exactly as it was described above, followed by a fully connected feed-forward network. The decoders also contain a multi-head self-attention layer, a multi-head attention layer with keys and values from the output of the encoder and queries from the output of the previous layer, and a feed-forward network. The feed-forward networks of the encoder and the decoder consist of two linear transformations with

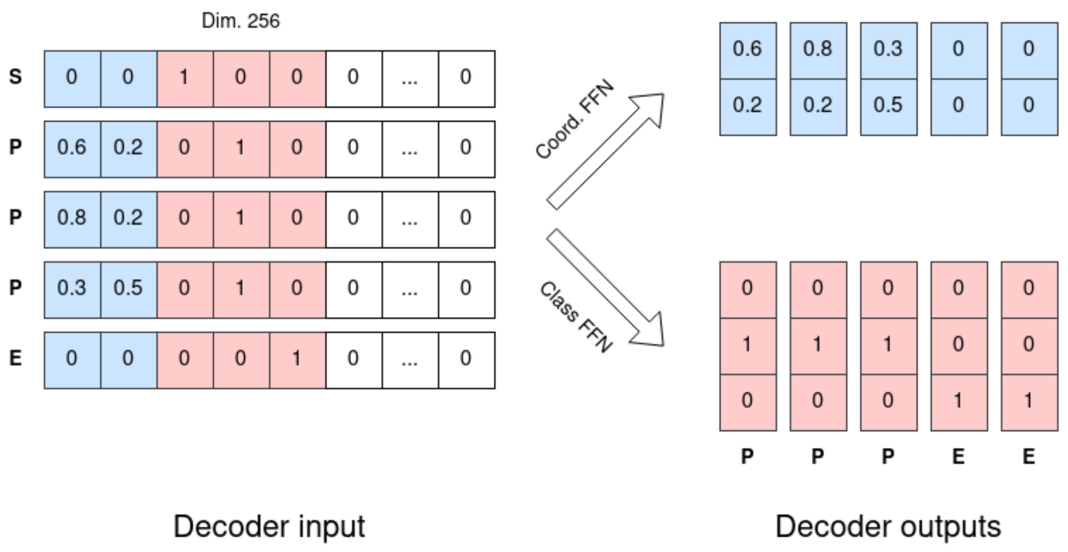


Figure 2.5: The Transformer decoder of our architectures takes a sequence of embeddings as input and outputs the same sequence shifted by one to the left from two separate branches. One classifies each token into a class, and the other one outputs the coordinates for the relevant tokens.

a ReLU activation in between:

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2.5}$$

In short, they input and output vectors of dimension d, with $d = 256$ in our case, but upscale them to $d_{ff} = 2048$ in the hidden layer. The Transformer also makes use of dropout, residual connections and layer normalization as described by [15].

In our auto-regressive version of the Transformer, the object queries are swapped with the token embeddings and the multi-head self-attention layer of the decoder is modified to work with an attention mask that prevents each token to attend to subsequent positions. In practice, the task we require the decoder to accomplish is to output the input sentence shifted by one to the left. Please notice how the input sentence is provided in its embedded version while two sentences are expected as output: one is provided by the classification branch which tells us about each token class, while the second sentence comes from the coordinates branch which outputs the point or gate coordinates for the relative tokens, and 0 for the rest. An example taken from the points toy setting dataset can be seen in figure 2.5.

2.7. Spatial positional encodings

The attention layer as described above is invariant to permutations of the input and does not use information about the position in the sequence of each token to generate its output. This can be a problem for scenarios like NLP, in which the order of the words in the sentence matters, but can be solved using *positional encodings*: vectors of the same size as the feature vectors, which contain information about the position in the sequence of the tokens. They can be seen as a representation of the indices $i \in [1, d_k]$ of the tokens. To use them, we can simply sum them to the token embedding before the attention layer.

The positional encodings are defined in the original Transformer paper [15] by a cosine and a sine function:

$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{2i/d_{model}}}) \tag{2.6}$$

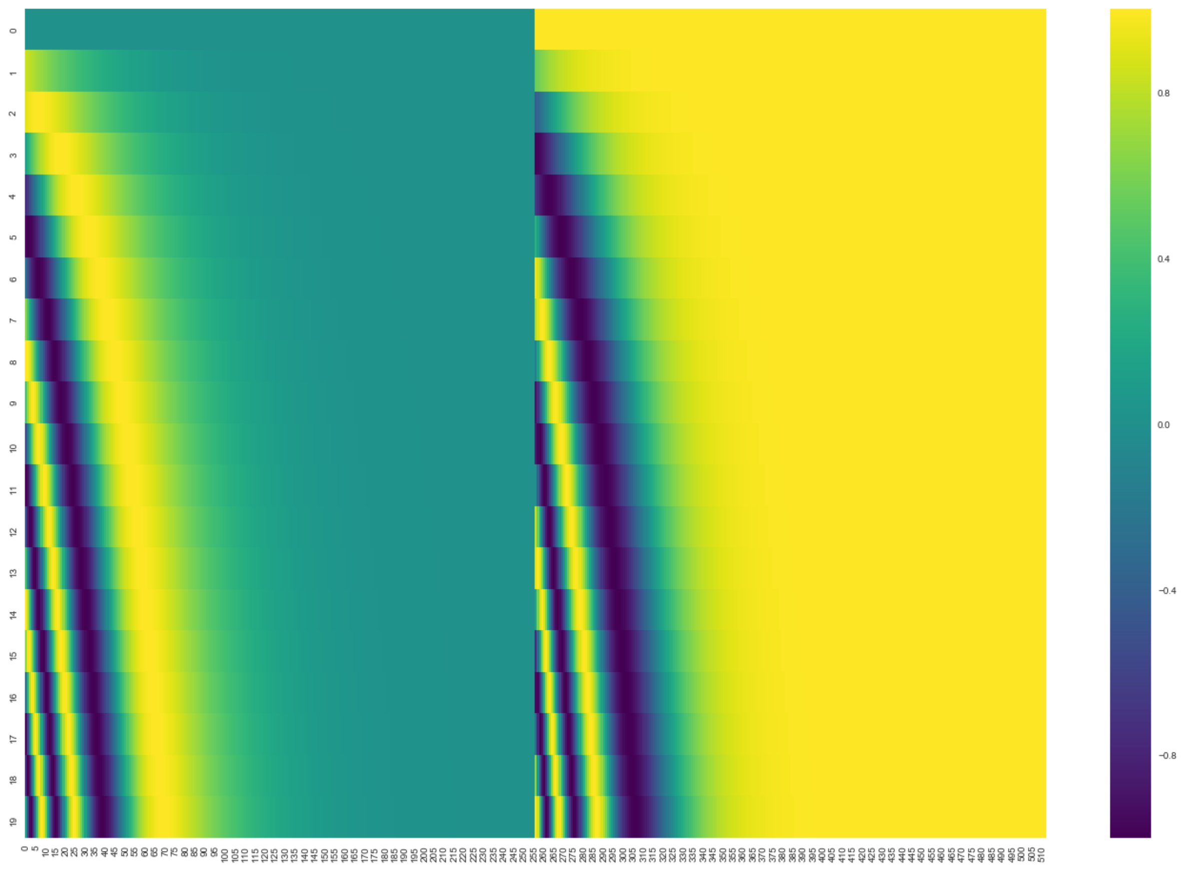


Figure 2.6: A visualization of the positional encodings used by us, if vectors were of dimension 512. Each index works with a sine and a cosine function with frequencies based on the position in the sequence. These functions provide values for each element of the token embedding, such that they can just be added to it before the attention layer.

$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{2i/d_{model}}}) \tag{2.7}$$

In our work we use two slightly different equations and consider i the absolute index of the position, while [15] works with relative indices.

$$PE_{(pos,i)} = sin(\frac{pos}{10000^{2i/d_{model}}}) \qquad i \in [0, 127] \tag{2.8}$$

$$PE_{(pos,i)} = cos(\frac{pos}{10000^{2i/d_{model}}}) \qquad i \in [128, 255] \tag{2.9}$$

Figure 2.6 shows a visual representation of the positional encodings with an image taken from this blogpost[2].

2.8. Word embeddings

This section slightly diverges from the discussion on attention but is important to understand how the original Transformer model worked and why we decided to use fixed token embeddings in our work. It is based on [14]. To solve the machine translation task and allow for parallel training, the original Transformer architecture [15] used *word embeddings*. In short, word embeddings are the representation of a word into a vector of fixed dimension, generally ~ 300. The previous traditional method used for word representations was the *bag of word* model [13], in which a huge one-hot encoding vector

[2]`https://jalammar.github.io/illustrated-transformer/`

was used to represent each word, quickly scaling the vector dimension to an intractable space when using large vocabularies. The most interesting property of word embeddings is that they are built using unsupervised training techniques and by exploiting context information. They can be trained simply by looking at a very large corpus of text and they allow for "intuitive" results on common operations, such that combining them in the vectorial space and then looking at the nearest neighbor of the resulting vector produces results like:

$$v_{\text{KING}} - v_{\text{MAN}} + v_{\text{WOMAN}} = v_{\text{QUEEN}}$$

$$v_{\text{PARIS}} - v_{\text{FRANCE}} + v_{\text{ITALY}} = v_{\text{ROME}}$$

$$v_{\text{BIGGER}} - v_{\text{BIG}} + v_{\text{COLD}} = v_{\text{COLDER}}$$

The three most common techniques used to build them are:

- Singular value decomposition: it works by computing a large matrix $X_{k \times k}$ with k being the size of the vocabulary, which stores the *co-occurrence* weights of each word against all others. This matrix is split into multiple matrices using the single value decomposition technique $X = UDV^T$ [7], which are stored and used when necessary. The problem of this technique it that it also quickly scales to an intractable dimension when working with large vocabularies and that it does not consider the order of words in the sentence during its construction.

- word2vec [10]: in short, *word2vec* generates word embeddings by learning for each word its probability conditioned of the surrounding words $P(w_i = \text{"orange"} \mid w_{i-1} = \text{"cold"}, w_{i+1} = \text{"juice"})$ using unsupervised learning techniques. The surrounding words we choose as conditions can be sampled using different techniques, such as the surrounding n-gram or randomly sampled words from a fixed size window. These types of embeddings have become very popular for their performance and because pre-trained vectors can just be downloaded and used in any architecture.

- GloVe [12]: *Global vectors for word representation* is a technique similar to word2vec, which uses ratios of co-occurrence probabilities, rather than the co-occurrence probabilities themselves. They are faster to train, scale well to large corpora and achieve good performance even with small vectors.

On a final note, also word embeddings have some limitations: they are vulnerable to attacks and are not a robust concept, they can take a very long time to train, and they are not able to work with unseen words.

In our work, we are using embeddings which are fixed and can be manually built by looking at the target labels, as described in our paper. Their dimension is 256, but only the first elements contain information on the labels. We have decided to use this technique because we believed that the label information we wanted to pass to the architecture could be directly represented using its mathematical form. A point can be fully represented using its coordinates in the image, for which a vector of dimension 2 is already large enough. However, we believe that testing out learned embeddings could also be beneficial for our task as the model could incorporate additional information to this vectors to use it at different stages of the decoder. We leave this analysis for future work.

3 Evaluation metrics

This chapter is intended as an overview of the evaluation metrics used for our experiments and shows why they are able to provide, as a single value, a meaningful average between the percentage of correct predictions and the precision of the predictions.

I will begin this discussion by providing the definition of IoU, L_1 and L_2 distances, which act as a threshold to determine if the model prediction is correct based on the spacial dimension. Then, I will explain how these threshold are incorporated into the final evaluation metric. My code and part of this explanation takes inspiration from [11].

3.1. IoU

Intersection over Union (IoU) is a metric based on the Jaccard similarity index which measures the precision of a position based prediction A_p with respect to the location of the ground truth A_{gt}. A representation of IoU can be seen in figure 3.1. IoU is defined by the area between the prediction and the ground truth divided by the total area covered by both:

$$IoU = \frac{area(A_p) \cap area(A_{gt})}{area(A_p) \cup area(A_{gt})} \tag{3.1}$$

Using different IoU thresholds, we can evaluate upon the precision of the predictions, as shown in figure 3.2.

3.2. L_1 and L_2 point distances

The L_1 distance of two points $\mathbf{x}$ and $\mathbf{y}$ with $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$ is defined as

$$L_1(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{2} \mid x_i - y_i \mid \tag{3.2}$$

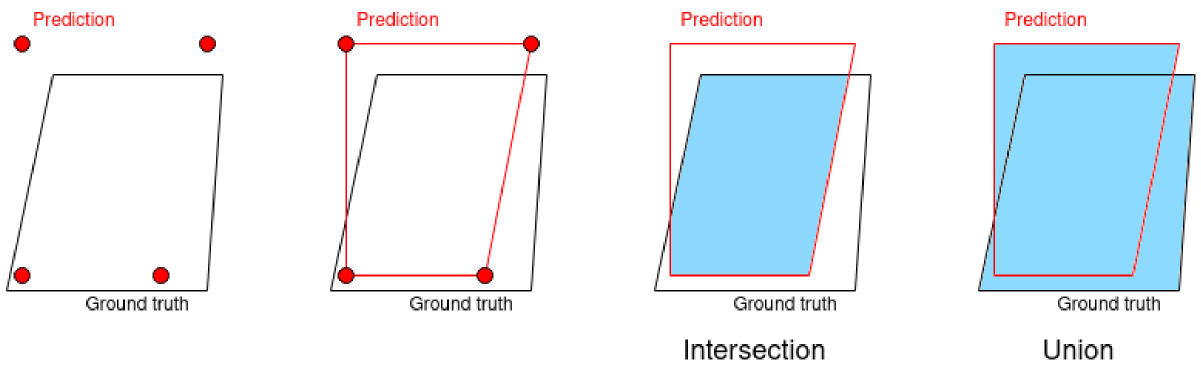


Figure 3.1: Computation of IoU given the model predictions of the polygon corners. An IoU of 1 indicates a perfect prediction.

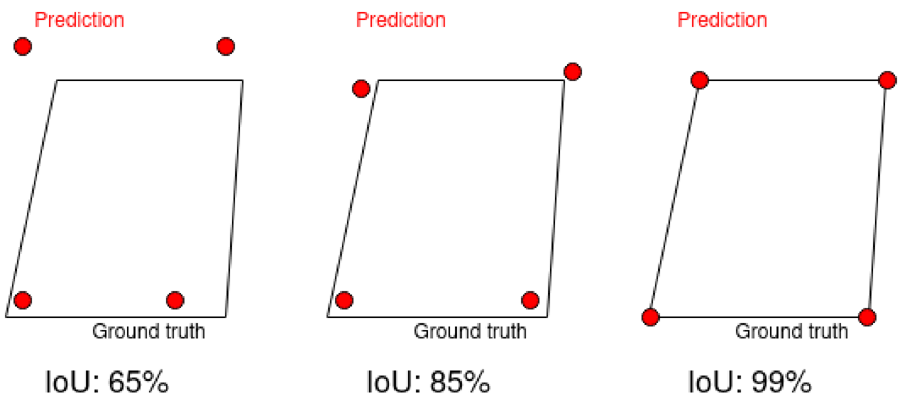


Figure 3.2: A threshold for IoU can be used to determine if a prediction is a true positive or a false positive. For example, if the IoU threshold is set to 0.7, the first prediction of this figure would be considered a mistake, while the other two would be considered correct. Varying the IoU threshold and evaluating the model predictions can tell us about the precision of the architecture.

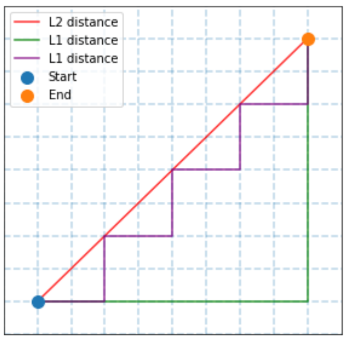


Figure 3.3: The difference between the L_1 and L_2 distance. The L_1 distance is also referred to as the *city block distance* as it represent the distance one would have to travel to go from point A to point B on a city grid.

while the L_2 distance is defined as

$$L_2(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{2} (x_i - y_i)^2} \tag{3.3}$$

The L_1 is also knows as the *city block distance* or *Manhattan distance* as it also refers to the space that is necessary to travel to go from $\mathbf{x}$ to $\mathbf{y}$ on a grid, as shown in figure 3.3. Although both of these metrics could be used to compare a point prediction to a ground truth, we have decided to use the L_1 as we directly compare the relative coordinates of the point prediction. Since $x_i, y_i \in [0, 1] \forall i$, the L_2 distance would quickly go to zero as the predictions become more and more accurate, without leaving enough space for evaluation under different thresholds.

3.3. True positives, False Negatives

Once we have defined a method to determine if a prediction is correct and with which precision, we can introduce different definitions. They are based on IoU, but can be easily defined on point distances too:

- True positive (TP): A correct detection, a detection with IoU greater than the threshold.
- False positive (FP): A wrong detection, a detection with IoU smaller than the threshold or a detection of a polygon that was already detected by the same model.
- False negative (FN): A ground truth object that has not been detected.
- True negative (TN): in general, this represents a correct misdetection. For the object detection task, this is not used by the metric as it would have to consider all possible prediction that the model could have made and correctly did not do so.

3.4. Metrics

Using these definitions we can define different evaluation metrics:

- **Precision**: this metric is defined as the percentage of correct predictions out of all the predictions that the model attempted to make:

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{all\ detections} \tag{3.4}$$

- **Recall**: this metric is defined as the percentage of ground truth objects that were found by the model:

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{all\ ground\ truths} \tag{3.5}$$

- **F1 score**: this metric is defined as the harmonic mean of precision and recall, and can be used to provide a single performance value that incorporates information from both metrics:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3.6}$$

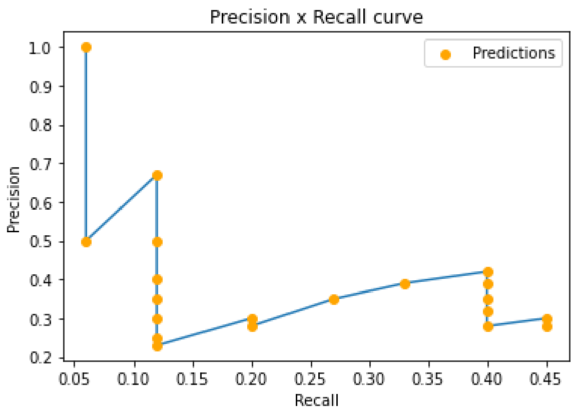


Figure 3.4: Precision x Recall curve

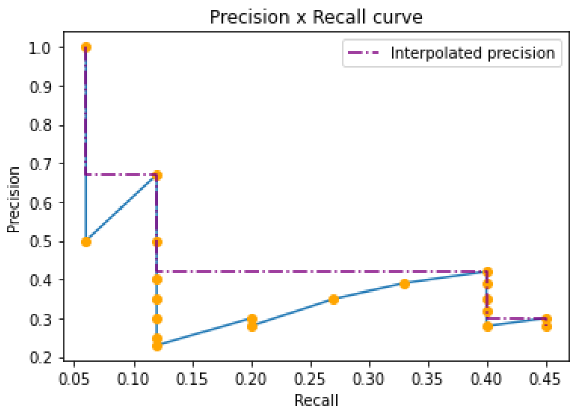


Figure 3.5: Interpolated Precision x Recall curve at all points

- **Precision x Recall curve**: to compare different detectors or the performance of a single detector on different classes, plotting the precision x recall curve can provide interesting information. A good detector is one whose precision remains high as recall increases, meaning that most of its predictions are correct and it also finds most of the relevant objects. An example of a precision x recall curve is shown in figure 3.4.

- **Average Precision (AP)**: AP is based on the Precision x Recall curve and summarizes the performance of the detector in a single numerical metric. It is defined as the area under the curve (AUC) of the Precision x Recall curve. Since Precision x Recall curves are often zigzag curves going up and down, thus making it hard to easily compare different curves, AP is in practice calculated by using precision averaged across all recall values between 0 and 1. This can be done by averaging the precision at 11 equally spaced recall levels $[0, 0.1, ..., 1]$, which has been the approach used by the PASCAL VOC challenge [5] until 2010.

  Instead of interpolating at 11 points, one could interpolate through all n points such that:

$$\sum_{n=0} (r_{n+1} - r_n)\, p_{interp}(r_{n+1}) \tag{3.7}$$

  with

$$p_{interp}(r_{n+1}) = max_{\tilde{r} > r_{n+1}} p(\tilde{r}) \tag{3.8}$$

  What this means is that AP is now obtained by interpolating the precision at each recall level r by taking the maximum precision at levels greater than r. Figure 3.5 shows an example of how the new interpolated curve looks like. The advantage of interpolating at all recall levels is that the new curve is less sensitive to noise in the predictions as these are equally averaged.

- **Mean Average Precision (mAP)**: this metric is often used by important competitions such as the Google Open Images Dataset V4 Competition [8] or the COCO object detection challenge [9] as their primary metric. It is defined as the average of the Average Precision metric over all classes or over different thresholds. In our work, we evaluate our models using mAP averaged over 10 different IoU thresholds $[0.5, 0.55, ..., 0.95]$ as we believe this value to be a good approximation of both the detection performance of our models and the spacial precision of our detections. Figure 3.6 and 3.7 show a comparison of the AP curves at different IoU thresholds. These curves show that Mask R-CNN is able to find more ground truth gates, but is much less precise in its detection, achieving an AP of only 48.09% when the threshold goes up to 0.85.

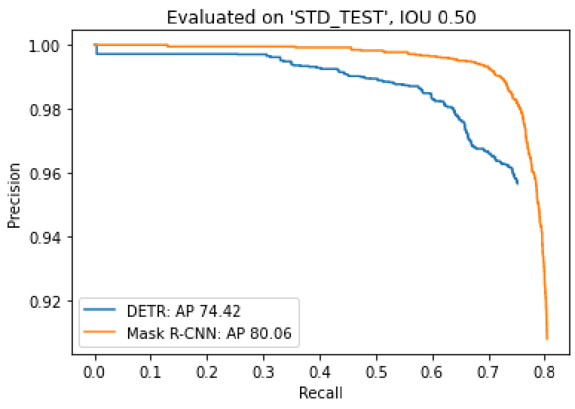


Figure 3.6: Average Precision at IoU threshold of 0.5

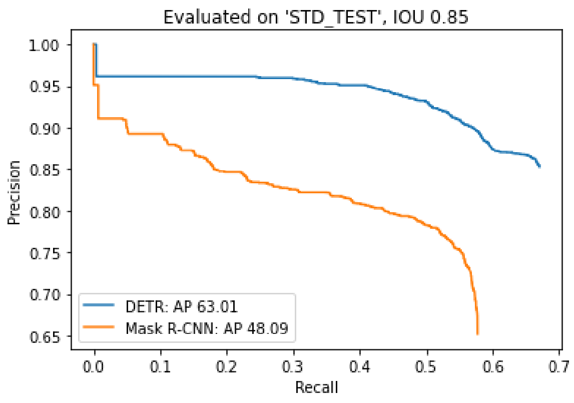


Figure 3.7: Average Precision at IoU threshold of 0.85

4 Datasets

This chapter will serve as a description of the datasets used in our experiments and will provide general guidelines for their reproducibility. First, we discuss how the gates and polygons toy setting images are generated, followed by the discussion regarding the lines toy setting. Then, we provide examples from all datasets to show the reader visual clues about the difficulty of these types of predictions.

4.1. Gates and polygons generation

The polygon and gate generation techniques are identical, as gates are generated by fixing the number of corners to 4 during the process. The most important steps can be seen in figure 4.1. First, a random point $c = (x_c, y_c)$ is sampled with $x_c, y_c \in \mathcal{U}(0.20, 0.80)$ and acts as the center of a circle of radius $r \in \mathcal{U}(0.05, 0.40)$. All coordinates and lengths are given as percentage with respect to the image size. Once we have the random circle, we divide its circumference using $N \in \mathbb{N}$ equally distributed points, where $N = 4$ for gates and $N \in \mathcal{U}(3, 7)$ for polygons. All points are equally shifted to the right by a small $\alpha \in \mathcal{U}(-\frac{\pi}{4}, \frac{\pi}{4})$ to avoid generating polygons that are too similar. Then, each point is shifted in the direction of its corresponding radius by $\Delta \in \mathcal{U}(-\frac{\sqrt{r}}{4}, \frac{\sqrt{r}}{4})$. All points are finally connected to define the final polygon. The line width is sampled uniformly from three possibles sizes $\{1, 2, 3\}$ expressed in pixel size.

4.2. Lines generation

The steps required to generate the images for the line toy setting dataset are shown in figure 4.2. First, a line going from $p_{start} = (0.10, 0.10)$ to $p_{end} = (0.90, 0.90)$ is drawn and split by 8 equally distributed points. Then, each points is randomly shifted in a direction perpendicular to the original line. This is done by shifting each point horizontally by $\delta \in \mathcal{U}(-0.15, 0.15)$ and vertically by $-\delta$. Finally, all points are connected to form the final lines as shown in figure 4.5. For our experiments, we draw a single line of 8 points for each image.

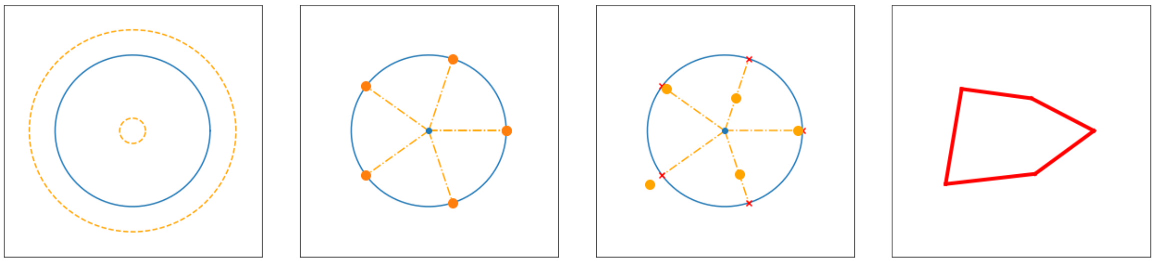


Figure 4.1: The steps required to generate a polygon. The first image shows the smallest and greatest circle radius that can be sampled in orange. The orange points are then the results of the splitting and of the random shift in the direction of the radius. The last image represents the final polygon generated by this example.

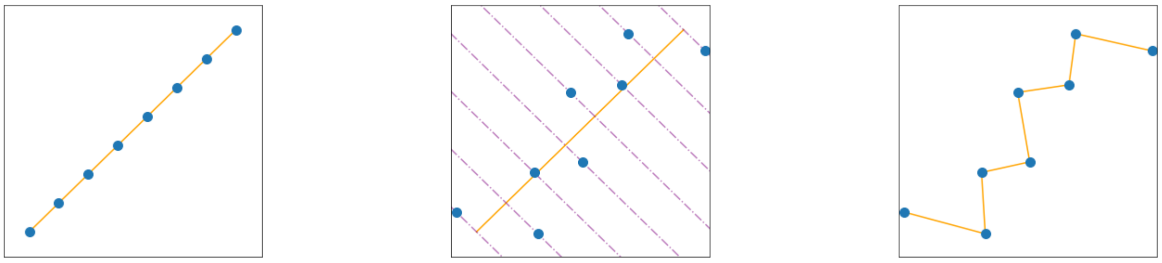


Figure 4.2: The steps required to generate a line for the lines toy setting dataset.

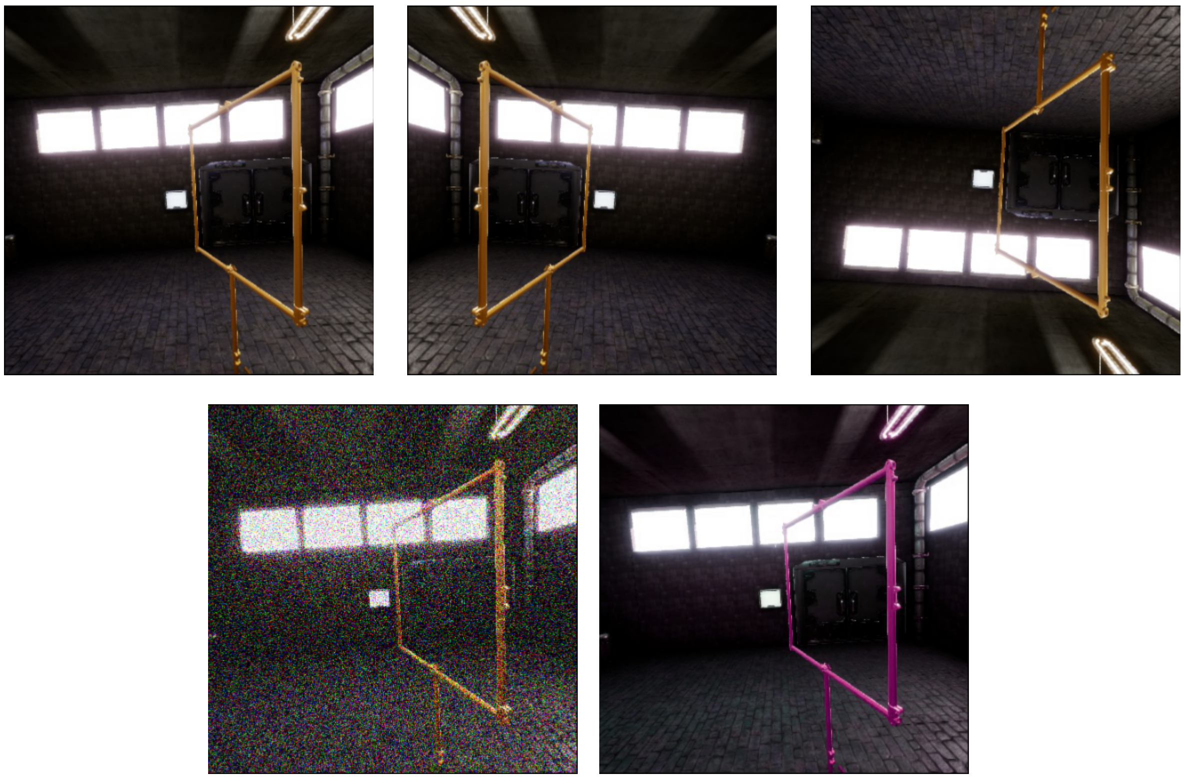


Figure 4.3: The data augmentation techniques used to train on the real gates dataset. From the top-left: the original image, horizontal flip, vertical flip, Gaussian noising, hue shift.

4.3. Data augmentation and dataset examples

Figure 4.3 shows the data augmentation techniques applied to the real gates dataset during training. Each type of augmentation is applied on each image with probability p. The techniques we used are: flipping the image horizontally ($p = 40\%$), flipping the image vertically ($p = 40\%$), shifting the image hue ($p = 10\%,\ \text{factor} = 0.25$) and adding random Gaussian noise to the image ($p = 10\%,\ \mu = 0,\ \sigma = 0.1$). The main reason for data augmentation is to upscale the dataset size by generating new useful images. The flips are applied because models should be able to detect gates from all angles. The Hue shift is a particular data augmentation technique which changes the colors of the image while maintaining its lightness and saturation. We apply this to prevent our models from overfitting on the yellow color of the gates. Finally, the Gaussian noise simulates situations of low image quality. Data augmentation is an important factor to consider when training Transformer based architectures as they are notoriously data hungry machines.

The following two pages provide examples of images from all the datasets used in this project. First, the toy settings are pictured, followed by images taken from all three different scenes of the real gates dataset.

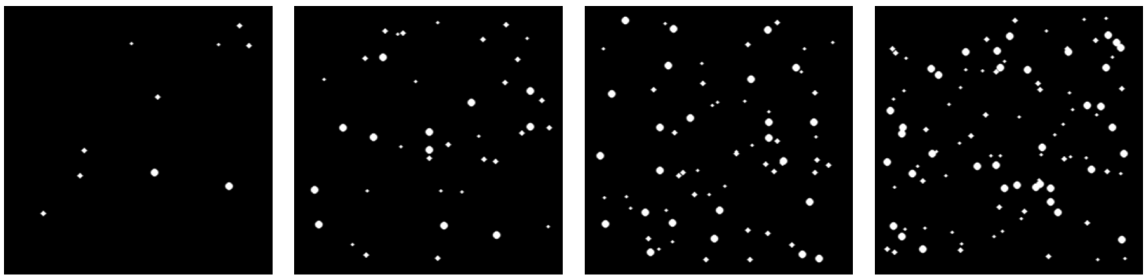


Figure 4.4: The points toy setting dataset

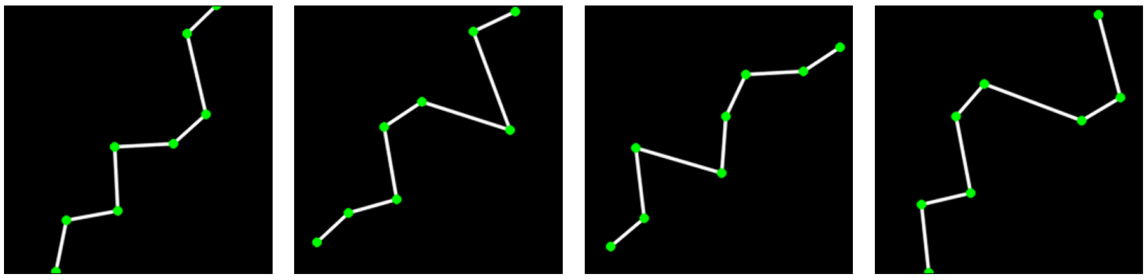


Figure 4.5: The line toy setting dataset

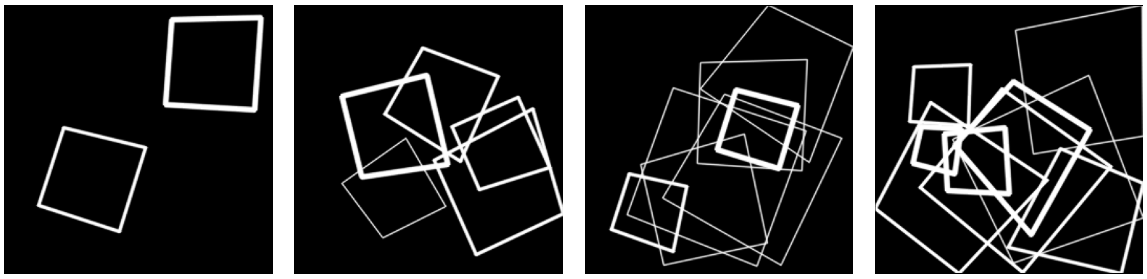


Figure 4.6: The gates toy setting dataset

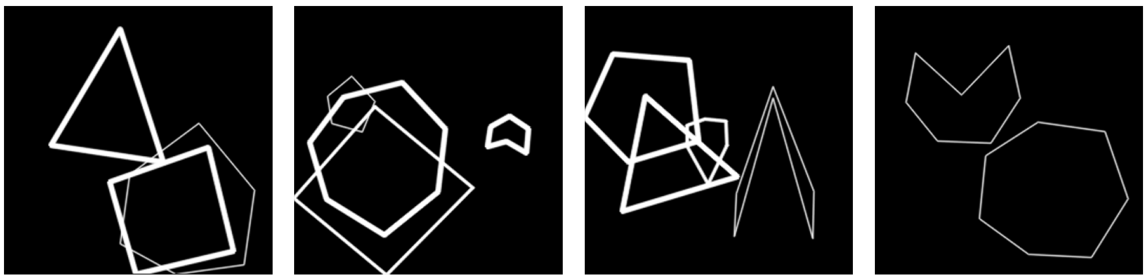


Figure 4.7: The polygons toy setting dataset

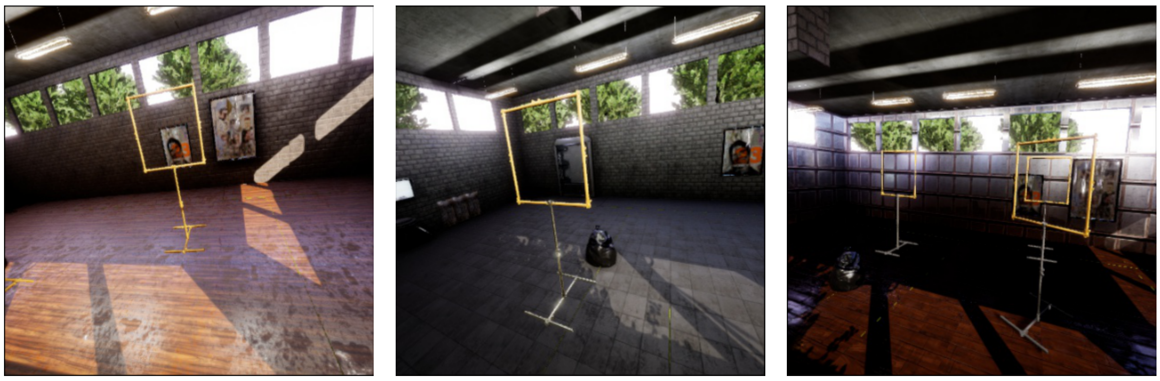


Figure 4.8: The real gates dataset, images from the first scene of the training set.

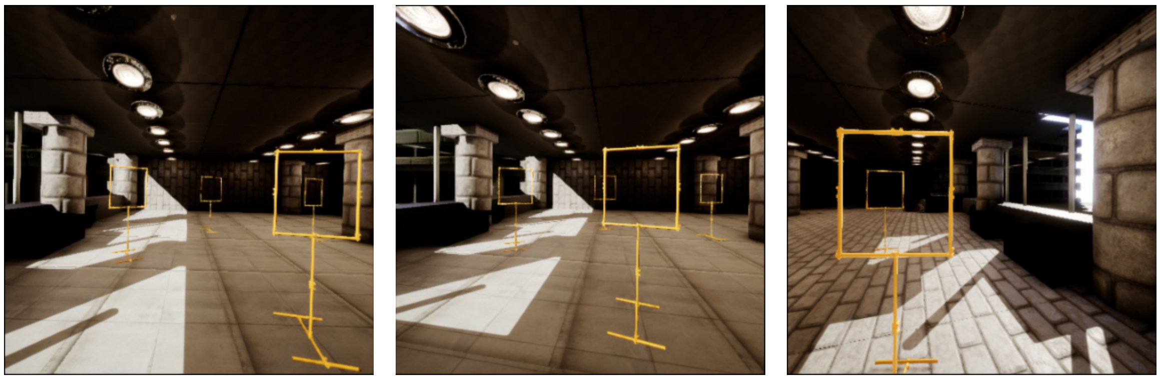


Figure 4.9: The real gates dataset, images from the second scene of the training set.

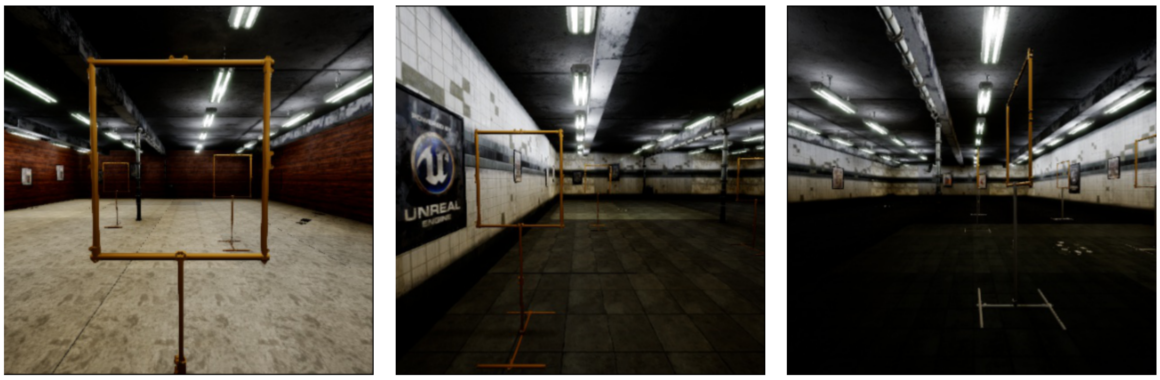


Figure 4.10: The real gates dataset, images from the test set.

5 Additional results

This chapter contains additional results that were not of any interest for the scope of our paper, but which might contain some interesting information for the reader. Section 5.1 shows our experiments on the optimal setting for the Transformer, and mirror the ones showed by [2] on a different dataset. Section 5.2 provides additional insights on the real gates dataset results.

5.1. Optimal settings for the Transformer

Table 5.1, 5.2 and 5.3 report the results of our study upon the importance of the most relevant components of the Transformer model: the number of encoder-decoder layers, the number of attention heads and a set of different backbones. The evaluation is based on the gates toy setting dataset with images containing 1 to 8 gates, with IoU threshold at 90%. They reflect the same results of DETR on the COCO object detection dataset [9]. Table 5.4 reports a comparison of the architecture with the minimal settings and the standard one used for all the experiments of our paper. The final setup that we implemented for the experiments discussed in our paper uses 6 encoder layers, 6 decoder layers, 8 attention heads and Resnet-50 as backbone.

5.2. Additional results on the real gates dataset

Table 5.5 shows how the different models improve from 100 to 300 epochs of training time. Table 5.6 is the result of our study on the performance of both models on different object sizes. The Transformer is stronger on small object while Mask R-CNN wins on medium and large objects. As medium and large objects represent the vast minority of the dataset samples, we believe these results to highlight how Transformers require a lot more data that CNNs, rather than their preference towards certain types of polygons.

Backbone	Precision	Recall	F1 score
Resnet-18	84.21%	97.49%	90.37%
Resnet-34	88.70%	97.00%	92.67%
*Resnet-50**	92.44%	98.06%	95.17%
Resnet-101	93.59%	97.86%	95.68%

Table 5.1: **Comparison of different backbones.** Resnet-50 achieves the best balance between performance and training time and is therefore the backbone used for the rest of our experiments.

# att. heads	Precision	Recall	F1 score
1	81.69%	97.30%	88.82%
2	89.00%	98.10%	93.33%
4	92.08%	98.01%	94.95%
8*	92.44%	98.06%	95.17%

Table 5.2: **Comparison of different amounts of attention heads**. 8 heads achieves the best performance and is therefore used in the studied model.

# layers	Precision	Recall	F1 score
1	72.94%	98.31%	83.74%
2	84.29%	98.27%	90.74%
3	89.54%	97.72%	93.45%
4	89.17%	98.15%	93.44%
6*	92.44%	98.06%	95.17%

Table 5.3: **Influence of the number of encoder and decoder layers on the performance of the Transformer.** 6 encoders and 6 decoder is the setting used for the rest of our experiments.

Backbone	# att. heads	# enc-dec layers	Precision	Recall	F1 Score
Resnet-18	1	1	42.01%	97.63%	58.74%
Resnet-50	8	6	92.44%	98.06%	95.17%

Table 5.4: Comparison of the minimal available architecture against the standard one used for the rest of our experiments

Architecture	Epochs	Precision	Recall	F1 Score	AP	mAP
Mask R-CNN	100	2.05%	4.00%	2.71%	0.22%	55.24%
Mask R-CNN	300	17.60%	15.59%	16.54%	6.09%	61.30%
Transformer	100	41.73%	34.36%	37.69%	24.55%	60.76%
Transformer	300	54.54%	42.89%	48.02%	29.89%	65.00%

Table 5.5: **Samples taken from the learning curve of Mask R-CNN and the Transformer architectures.** Precision, recall, F1 score and AP are based on IoU @ 0.95. mAP as average of IoU at [0.50, 0.55, …, 0.95].

Architecture	Object size	Precision	Recall	F1 Score	AP	mAP
Mask R-CNN	S	16.48%	14.59%	15.48%	5.33%	60.77%
Transformer		53.68%	42.42%	47.39%	29.50%	65.05%
Mask R-CNN	M	39.32%	37.55%	38.41%	27.40%	79.20%
Transformer		69.95%	57.96%	63.39%	53.34%	77.21%
Mask R-CNN	L	19.23%	13.70%	16.00%	12.35%	46.98%
Transformer		70.37%	26.03%	38.00%	24.18%	33.08%

Table 5.6: **Comparison of performance of Mask R-CNN and the Transformer architectures against object sizes.** Precision, recall, F1 score and AP are based on IoU @ 0.95. mAP as average of IoU at [0.50, 0.55, …, 0.95].

Bibliography

[1] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[2] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020.

[3] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] 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.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[7] V. Klema and A. Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on Automatic Control*, 25(2):164–176, 1980. doi: 10.1109/TAC.1980.1102314.

[8] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, et al. The open images dataset v4. *International Journal of Computer Vision*, pages 1–26, 2020.

[9] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014. URL http://arxiv.org/abs/1405.0312. cite arxiv:1405.0312Comment: 1) updated annotation pipeline description and figures; 2) added new section describing datasets splits; 3) updated author list.

[10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[11] Rafael Padilla, Wesley L. Passos, Thadeu L. B. Dias, Sergio L. Netto, and Eduardo A. B. da Silva. A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, 10(3), 2021. ISSN 2079-9292. doi: 10.3390/electronics10030279. URL https://www.mdpi.com/2079-9292/10/3/279.

[12] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL `https://www.aclweb.org/anthology/D14-1162`.

[13] Josef Sivic and Andrew Zisserman. Efficient visual search of videos cast as text retrieval. *IEEE transactions on pattern analysis and machine intelligence*, 31(4):591–606, 2008.

[14] Richard Socher. Lecture notes from Standford's Deep Learning for NLP course, March 2016.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.