DePerceiver: Improving Small Object Detection in DETR

Edward Li, Aditya Kannan
Carnegie Mellon University
{edwardli, akannan2}@cs.cmu.edu

Abstract

DETR is a recently proposed method for end-to-end object detection, eliminating the need for hand-crafted model components while still achieving good performance. However, performance on small objects remains lacking, due to the inability for transformers to efficiently process high resolution feature maps. We propose DePerceiver, a method that combines DETR and Perceiver IO to enable the use of higher resolution feature maps. We find that DePerceiver results in higher relative small object performance when evaluated on the COCO dataset. Code can be found at `https://github.com/mooey5775/DePerceiver`.

1. Introduction

Historically, modern object detection advances have come through the creation of domain-specific handcrafted components [9], such as complicated region proposal methods, use of anchors, and post-processing heuristics like non-maxima suppression in techniques such as Faster R-CNN [16], SSD [11], and YOLO [14, 15]. These methods perform well, but are not end-to-end and not as generalizable due to domain-specific engineering.

Parallel to the development of object detection methods has been the increasing use of transformers [18] in vision tasks, spurred by the initial success of ViT [2]. The free long-range relationships modeled by transformers, as well as its low inductive bias, make it a desirable model for a large variety of modalities.

In a recent work by Carion et al. [1], the use of transformers are proposed for object detection in DETR, leveraging the long-range properties of self-attention to build the first simple end-to-end object detection method. More concretely, DETR combines a convolutional feature extractor with a transformer encoder/decoder, with a complex loss function that directly optimizes for the object detection objective.

DETR achieves extremely competitive performance, outperforming models like Faster R-CNN in a variety of metrics. However, DETR underperforms significantly in small object detection performance. We hypothesize that this underperformance is due primarily to the use of an extremely coarse feature map from the convolutional feature extractor. More concretely, the features used by DETR are taken after a *downscaling factor* of 32x, meaning that the feature map resolution has been downscaled by 32x on each side. While other competitive object detection methods are able to use multiple feature scales to detect smaller objects, DETR is left with low-resolution information insufficient for good performance. Our intuition here is supported by other recent work such as Deformable DETR [20].

Moreover, the use of multiscale feature maps or lower downscaling factors are computationally intractable for DETR (as expanded upon in section 5.1), due to the quadratic time and space complexity for self-attention w.r.t. input size. It seems that finding a *linear complexity* attention method is necessary for good small object performance.

In this paper, we propose *DePerceiver*, a method that addresses DETR's high complexity issues (Figure 1). We replace the quadratic-time transformer with the recently proposed linear-time attention-based method Perceiver IO [4]. Perceiver IO performs the vast majority of computation in a constant-sized latent space, decoupling input size and model complexity. Additionally, it retains the lack of spatial or local inductive bias that is an attractive feature of transformers.

DePerceiver allows us to experiment with features maps with lower downscaling, as well as multiscale feature maps. We evaluate versions of DePerceiver using several *downscaling factors* against DETR on the popular object detection benchmark COCO [8]. Through these experiments, we demonstrate DePerceiver achieves better relative small object performance (AP_s) when compared to DETR. Code is released at `https://github.com/mooey5775/DePerceiver`

2. Related Work

Our work is motivated by prior work in linear-cost attention mechanisms, as well as the utility of multiscale features in vision transformer models.

2.1. Linear-cost Attention Mechanism

Transformers, as used in DETR [1], are extremely flexible and come with little inductive bias. However, self-attention's comparison of each input to each other results in an inefficient quadratic time and space complexity per layer. The standard response to this limitation is to downscale the input in some way. Vision Transformer (ViT) [2] proposes splitting an image into a grid of $k \times k$ patches, using the embedded patches as input tokens for a transformer. However, a simple embedding of a high-frequency image patch is quite limiting. Methods like DETR use an intermediate activation map of a pretrained convolutional feature extractor (such as ResNet50 [3]), which is more expressive than a simple embedding.

However, because this downscaling results in poor small object performance, we would like to modify the attention mechanism to be linear-cost. Several previous works have explored similar concepts. Most closely related to our work is Deformable DETR [20], which introduces the deformable attention layer. The deformable attention layer is explicitly sparse, attending on a fixed size k other inputs. This linear-cost attention layer allows Deformable DETR to outperform DETR in sample efficiency and small object detection. However, deformable attention adds some local inductive bias back into the transformer model, which is a feature we aim to avoid in this work.

Several other methods have been proposed that avoid adding additional inductive bias. First is the Set Transformer [6], which projects the large input array into a smaller array using cross-attention in their Induced Set Attention Block. The Set Transformer is able to achieve linear scaling w.r.t input size, although each induced set attention block reconstructs the original input size after attention, limiting scalability.

Perceiver [5] is based on a similar principle, using cross-attention to project inputs into a smaller latent space. However, Perceiver remains in the latent space in subsequent self-attentions, meaning further computation is completely decoupled from the input size. Therefore, Perceiver's depth scaling is strictly better than Set Transformer. In this work, we use Perceiver IO [4], an improvement to Perceiver that allows for arbitrary sized outputs using a cross-attention decoding step, which will be discussed in further length in section 3.

2.2. Multi-scale features in vision transformers

As our primary intuition in this work is to improve small object detection performance by exploiting higher resolution features and multi-scale features, we first turn to some previous works to explore the effect of higher resolution features on object detectors and vision transformers.

Most obviously, the work of Feature Pyramid Networks (FPNs) [7] pioneered the use of multiscale features in object detection. Through the addition of a top-down pathway and lateral connections to combine multiscale features, FPNs outperformed previous methods in both AP and strongly in AP_s. Several follow-up works to FPNs exist as well, including PANet [10] and BiFPN [17], further expanding on and simplifying multiscale feature fusion. Overall, these papers confirm our intuition for using higher resolution and multiscale features in object detection.

Additionally, multiscale features have been shown to help in vision transformer context as well. Most relevantly, Deformable DETR [20] exploited multiple features to achieve higher object detection performance. However, Deformable DETR did not present a strong set of ablations on the affect of feature resolution on object detection performance, making it difficult to draw conclusions or insights on the impact of feature resolution.

One other notable work to mention is the Pyramid Vision Transformer (PVT) [19], which iteratively downscales features after each transformer encoder layer, similar to the use of max pooling in CNN-based architectures. This way, the quadratic cost of higher feature resolutions is only paid on the first several layers. PVT was shown to boost the performance on a variety of vision tasks, including object detection.

Overall, it seems that multiscale or higher resolution features will indeed be helpful for small object detection, which we aim to explore in this work through a set of directly comparable experiments.

3. Background

Before detailing the DePerceiver model architecture, it is worth exploring both DETR and Perceiver IO.

3.1. DETR

DETR [1] is an object detection model based on an encoder/decoder transformer. However, the main contribution of the authors seems to be the complicated Hungarian loss, which forces unique matching between prediction and ground truth, a problem that has long plagued end-to-end object detection. We briefly detail the network architecture, as well as some of the components of the loss function here.

DETR begins with a CNN backbone to extract image features. From an image $x_{img} \in \mathbb{R}^{3\times H_0 \times W_0}$, DETR uses the activations of a deeper convolutional layer as input to the transformer $x \in \mathbb{R}^{C\times H\times W}$. Normally, DETR uses a ResNet50 [3] backbone, with $C = 2048, H = \frac{H_0}{32}, W = \frac{W_0}{32}$.

DETR uses some further preprocessing before input into the transformer encoder. A 1×1 convolution embeds the feature map into a smaller dimension $z_0 \in \mathbb{R}^{d\times H\times W}$, which is then reshaped into a flat $d \times HW$ tensor. Due to the permutation invariance of transformers, DETR adds a position embedding, either learned or from fourier features.

The transformer encoders and decoders are a standard transformer architecture, with self-attention in the encoder having complexity $O(H^2W^2d)$, quadratic with respect to input size. To extract object detections, the decoder takes N object queries as input, represented as learnable position embeddings. Intuitively, these learnable position embeddings can learn some prior for object locations or sizes for diverse classes of objects. Because of the self-attention structure in the decoder, uniqueness can more easily be enforced between object detections. Because N is fixed across inputs, the only impact of input size on the decoder is in cross-attentions, which are linear w.r.t input size.

The transformer decoder outputs are postprocessed by a 3-layer MLP, directly predicting the normalized center coordinate, height, and width of the box. Additionally, a linear projection layer is used to compute the class of the predicted box, which include the 90 COCO classes and 1 no object class, as N is often much larger than the total number of objects.

For successful learning, DETR's Hungarian loss function is quite complicated. The authors begin by defining a pairwise loss function between a single ground truth and prediction box. This loss involves a negative log-likelihood class error, as well as a generalized IOU loss for bounding box loss. Then, the final loss is computed as the optimal matching of predicted and ground truth boxes w.r.t pairwise loss, using a Hungarian matching algorithm.

Overall, DETR's architecture is quite elegant and simple. However, the main issue that we aim to resolve in this work is through the large downscaling DETR uses from its ResNet50 backbone, resulting in low small object detection performance.

3.2. Perceiver IO

Perceiver IO [4] is a transformer-like architecture that uses attention to perform computation. It was specifically designed to make no assumptions on the input and output structure, built for various combinations of multi-modal inputs. Because multi-modal inputs can often be quite large in input size, Perceiver IO aims to be computationally efficient, with its first layer being linear in input size, and all other layers completely decoupled from input size. To achieve this, Perceiver IO first *encodes* inputs into a fixed size latent space, then *processes* in the latent space, with a final *decode* step which queries the latent space with a set of learned or fixed embeddings.

Perceiver IO's encoder uses cross-attention to encode an input $x \in \mathbb{R}^{M\times C}$ into a latent space of size $z \in \mathbb{R}^{N\times D}$ The model input is used as key and value, while latents are used as the query. The processing step uses multi-head self-attention on the latent space alone. Note that N nor D are related to the input size, making the latent processing step a fixed cost no matter the input size. The decoder is once again cross-attention, with an output query array ($\mathbb{R}^{OE}$) used as a query, with latents used as key and value.

Notably, the authors of Perceiver IO claim that only a single encoding and decoding step are required, as multiple steps lead to significant increase in computation, while not significantly improving performance in tasks such as image classification and StarCraft II. However, the truth of this statement in an object detection context remains to be seen.

4. DePerceiver

Our architecture design is based on the results of DETR [1] and Perceiver IO [4]. In order to bypass the quadratic time complexity constraints of DETR's transformer architecture, we utilize Perceiver IO in its place.

Fiver major components are used in the architecture for our DePerceiver model: (1) a backbone to extract image features, (2) the Perceiver IO encoder, (3) the Perceiver IO decoder, (4) prediction heads for bounding box and class characteristics, and (5) the Hungarian loss for evaluating the model's predicted bounding box classes.

4.1. Backbone

We first make use of a standard CNN backbone to extract important features of our input image. We used ResNet-50 [3] to take an input image of size $3 \times H \times W$ and downsample it to a smaller feature space of size $C \times \frac{H}{2^k} \times \frac{W}{2^k}$.

ResNet-50 has five blocks, each of which reduces both the height and width by a factor of 2. The final layer outputs features of size $\frac{H}{32} \times \frac{W}{32}$, but we can also use larger features from intermediate layers to produce features of size $\frac{H}{8} \times \frac{W}{8}$ or of size $\frac{H}{16} \times \frac{W}{16}$.

There will be different number of channels based on the number of layers at which we extract the image features. For features of size $\frac{H}{8} \times \frac{W}{8}$, we will have $C = 512$, for size $\frac{H}{16} \times \frac{W}{16}$, we will have $C = 1024$, and for size $\frac{H}{32} \times \frac{W}{32}$, we will have $C = 2048$.

4.2. Perceiver IO Encoder

The input of the Perceiver IO encoder is a flattened array of features by channel. In other words, the backbone's features in the shape $C \times H \times W$ is transformed to be $HW \times C$.

Next, we generate a set of Fourier positional encodings that are added to the flattened features. We can include these encodings because attention is a permutation-invariant operation. We use a parameterization scheme described in the original Perceiver [5] paper that is similar to that of NeRF [13]. Given a value x_d from the input in the d-th dimension and a frequency f_k, our positional embedding will be $[\sin(f_k \pi x_d), \cos(f_k \pi x_d)]$. The value f_k is determined in relation to the Nyquist frequency μ. This result is added to the features and fed into the Perceiver encoder.

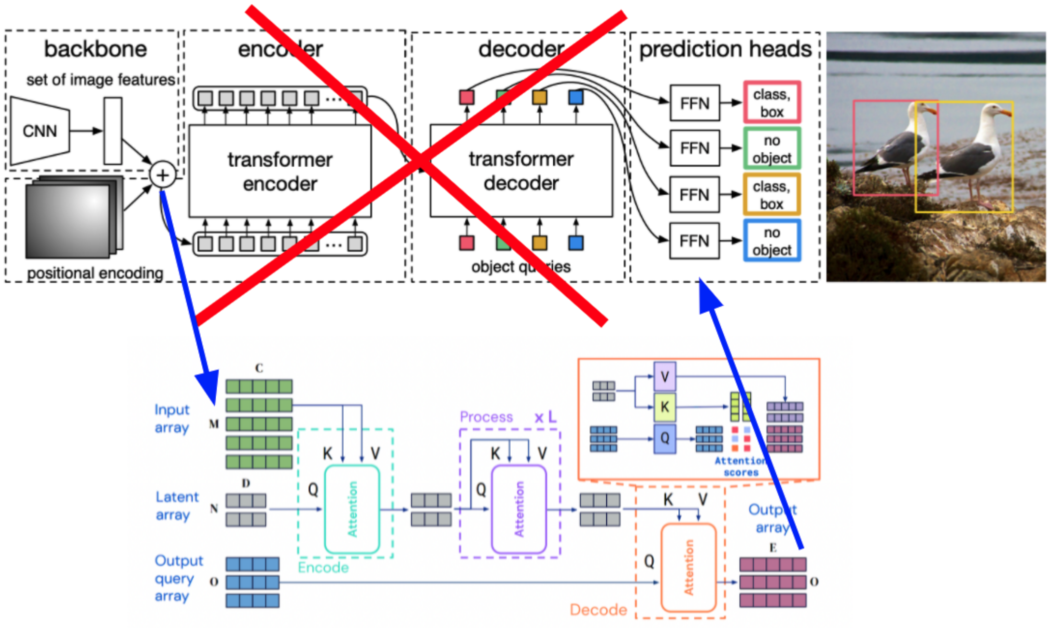

Figure 1. The DePerceiver model modifies the DETR architecture by replacing the transformed encoder-decoder with Perceiver IO.

The Perceiver model encodes the input with a small, fixed size latent space. Specifically, the Perceiver encoder uses a cross-attention module to maps the input of size $HW \times C$ to a latent space of size $N \times M$, where N, M are hyperparameters that we can choose. We choose our latent space to be 2048×256. In applying this cross-attention module, we make this step occur in linear time with respect to the image space.

After we generate our latent space using the image features, we use a series of self-attention blocks to process the latent space in transformer-like fashion. The key observation here is that our latent space has a fixed size, so the self-attention modules run in $O(1)$ time.

4.3. Perceiver IO Decoder

The Perceiver IO decoder employs a similar strategy as the encoder. In the same way that the encoder uses a cross-attention module to transform the inputs to the latent space, the decoder uses a cross-attention module to project the latent features to the output space. Latents are used as keys and values, and a set of learned positional embeddings are used to query the Perceiver IO decoder.

Each position corresponds to an output embedding that is generated by the decoder. Each of these output embeddings is individually used as input to the prediction feed forward networks described in the section below. In our implementation, the query embeddings are size 256.

4.4. Feed Forward Networks

The outputs from the decoder are each sent through two networks. The first network predicts the bounding box coordinates for a detected object in the image, and the second network predicts the class. If the queried position does not correspond to an object in the image, the class prediction network will indicate that no object is present.

The bounding box network is a three layer neural network that takes an input of size 256 and predicts the center x and y coordinates, as well as the width and the height. The class prediction network is a one layer linear perceptron that predicts the class of the object (or lack thereof).

4.5. Set Prediction Loss

To train our model, we used the Set Prediction Loss that is introduced in DETR [1]. DETR uses the Hungarian Matching algorithm to find an optimal bipartite matching between predicted and ground truth objects based on their class and position in the image. After generating the bipartite matching, we can use bounding box losses. We refer the reader to a detailed discussion of Hungarian loss in the DETR paper.

4.6. Multi-scale Inputs

By leveraging the linear complexity of our model with respect to image size, we can use larger feature maps as inputs into our encoder. We aim to use multi-scale inputs to be help our model detect small objects.

Specifically, our multi-scale inputs will take the features downsampled by three different factors. We obtain three sets of features from our backbone CNN model: features of size $\frac{H}{8} \times \frac{W}{8}$, features of size $\frac{H}{16} \times \frac{W}{16}$, and features of size $\frac{H}{32} \times \frac{W}{32}$. We supplement these features with a learned scale encoding corresponding to their downsampling factor. We add a positional encoding based on their size as well.

We concatenate these three features and use the result as input for our encoder.

5. Experiments

The dataset that we trained and evaluated our models on is COCO 2017 [8]. We trained our models on the training set and evaluated on the validation set.

We ran three main experiments based on the size of the features generated by the backbone. We considered backbones that downsampled the image by a factor of 8, 16, and 32. In addition, we attempted to implement a working model that used features at multiple scales.

Our experiments were run on a cluster with 10 GPUs. The models were run for 50 epochs each. We trained our models with the AdamW [12] optimizer and a base learning rate of 10^{-4} with a weight decay of 10^{-4}. The learning rate was dropped to 10^{-5} after 33 epochs.

We used a pre-trained ResNet-50 [3] for our backbone. Our models use a latent space of size 2048×256, following the image classification baseline presented in Perceiver IO. We chose this size based on the limitations of the GPU we were using. Larger latent spaces would require us to use a batch size of 1. The query embeddings for our decoder were size 256.

5.1. Time Complexity Analysis

We compared the efficiency of the Perceiver IO encoder-decoder in the DePerceiver model with the transformer in the DETR model. As shown in figure 2, in each of our experiment configurations, the Perceiver architecture has a smaller memory footprint than the transformer.

The results are as expected; for Perceiver, the number of activations should be constant among the self-attention modules whereas for DETR's transformer, the number of activations should be increasing quadratically for larger feature maps. The multi-scale model is computationally intractable with DETR's transformer–as indicated in the figure, the model ran out of memory on our CPU.

5.2. Absolute Performance

We compare the performance of our DePerceiver models with DETR using Average Precision for the detections. Our DePerceiver models are not able to match the DETR baseline as shown in table 1. Among the three single scale DePerceiver models, the model downsampled by 16x performs the best after 50 epochs. The DETR model uses features that are downsampled by 32x.

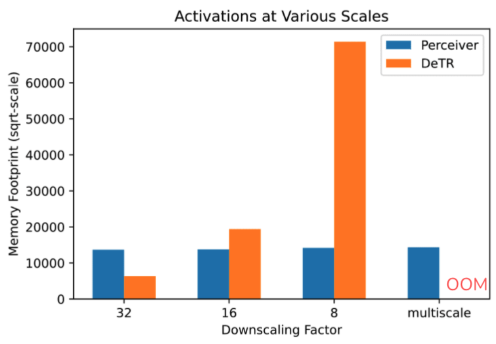

Figure 2. Memory footprint of encoder-decoder for one iteration of model. The x-axis denotes the ratio between the size of the image and the feature map. The y-axis is in square root scale to account for image area as opposed to the image length. The multi-scale model for DETR ran out of memory on our CPU.

Model	AP	AP$_{50}$	AP$_{75}$	AP$_s$	AP$_m$	AP$_l$
DETR	34.1	54.9	35.3	13.2	36.5	53.0
DePerceiver-32	10.5	20.8	9.3	4.3	11.8	15.6
DePerceiver-16	15.3	27.1	16.3	7.1	16.3	22.3
DePerceiver-8	11.4	21.5	10.5	3.6	10.8	19.4

Table 1. Average Precision of DePerceiver models compared with DETR after 50 epochs. The number after the model name refers to DePerceiver trained with features downsampled by that factor.

The performance of our DePerceiver models can be attributed to a few issues. Whereas the original Perceiver paper was able to train a model to attain baseline performance on ImageNet classification tasks, higher resolution tasks like object detection might be more difficult for the latent space bottleneck to capture effectively. Increasing the complexity of the model to capture more features from the image would help the model perform better. Increasing the latent dimension or the number of heads for the Perceiver IO encoder could help. Another possibility is to use multiple cross-attention layers for the encoder or decoder. Overall, it seems that the original author's claims that a single encoder or decoder layer are sufficient might not be correct in higher resolution tasks.

We also attempted to train the multi-scale DePerceiver model we described previously. Unfortunately, the model was not able to effectively improve its detection performance over 10 epochs as training was unstable. A multi-scale model would require a larger model to train and experimenting with the suggestions expressed above may be a viable option.

5.3. Downscaling Effects on Relative AP_s performance

Based on the heuristics we established previously, it makes sense that DePerceiver-16 outperforms DePerceiver-

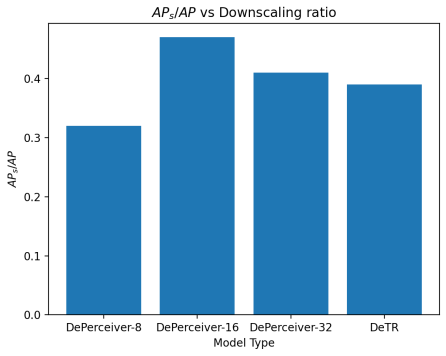

Figure 3. The ratio of AP_s/AP in each of our three experiments as well as DETR after 50 epochs. DETR downscales the input by a factor of 32.

32 with regards to AP_s. However, it is surprising that DePerceiver-8 performs worse than DePerceiver-32 and DePerceiver-16 in AP_s.

Moreover, we can see this effect appear again when we consider the ratio AP_s/AP in figure 3. The AP_s/AP value peaks at DePerceiver-16, outperforming DETR. The value is about the same for DePerceiver-32 and DETR, which makes sense because DETR downsamples its images by 32 as well. Overall, an interesting conclusion here seems to be that relative small object detection performance is almost intrinsic to the *downscaling factor* we use.

We believe that this result shows a downscaling-compression tradeoff. Whereas downsampling by a smaller factor than 32 produces features with greater resolution, it also results in features that have less preprocessing from the ResNet backbone. The DePerceiver-8 model requires significant compression into the latent space of the encoder, and this removes much of the high resolution information.

This effect seems to indicate that simply using higher resolution features in the backbone does not guarantee better AP_s. Considering the tradeoff, it makes sense that DePerceiver-16 has the best performance out of all our experiments.

6. Conclusion

We present DePerceiver, which exploits the linear time complexity of the Perceiver architecture to increase the efficiency of the DETR model for object detection. It does not outperform the DETR baselines in absolute terms, but it does outperform in relative small object detection performance. We also present some new directions and intuitions for how to construct a multi-scale model or a model with larger features. We find that relative small object detection performance seems intrinsically tied to feature downscaling. Higher resolution features can improve AP_s but can also lose information when fed into the latent space bottleneck. It is important to find the sweet spot to maximize performance.

References

[1] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers, 2020.

[2] 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, 2021.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[4] A. Jaegle, S. Borgeaud, J.-B. Alayrac, C. Doersch, C. Ionescu, D. Ding, S. Koppula, D. Zoran, A. Brock, E. Shelhamer, O. Hénaff, M. M. Botvinick, A. Zisserman, O. Vinyals, and J. Carreira. Perceiver io: A general architecture for structured inputs outputs, 2021.

[5] A. Jaegle, F. Gimeno, A. Brock, A. Zisserman, O. Vinyals, and J. Carreira. Perceiver: General perception with iterative attention, 2021.

[6] J. Lee, Y. Lee, J. Kim, A. R. Kosiorek, S. Choi, and Y. W. Teh. Set transformer: A framework for attention-based permutation-invariant neural networks, 2019.

[7] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection, 2017.

[8] 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, 2015.

[9] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen. Deep learning for generic object detection: A survey, 2019.

[10] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation, 2018.

[11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016.

[12] I. Loshchilov and F. Hutter. Decoupled weight decay regularization, 2019.

[13] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020.

[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection, 2016.

[15] J. Redmon and A. Farhadi. Yolov3: An incremental improvement, 2018.

[16] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[17] M. Tan, R. Pang, and Q. V. Le. Efficientdet: Scalable and efficient object detection, 2020.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.

[19] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions, 2021.

[20] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai. Deformable detr: Deformable transformers for end-to-end object detection, 2021.