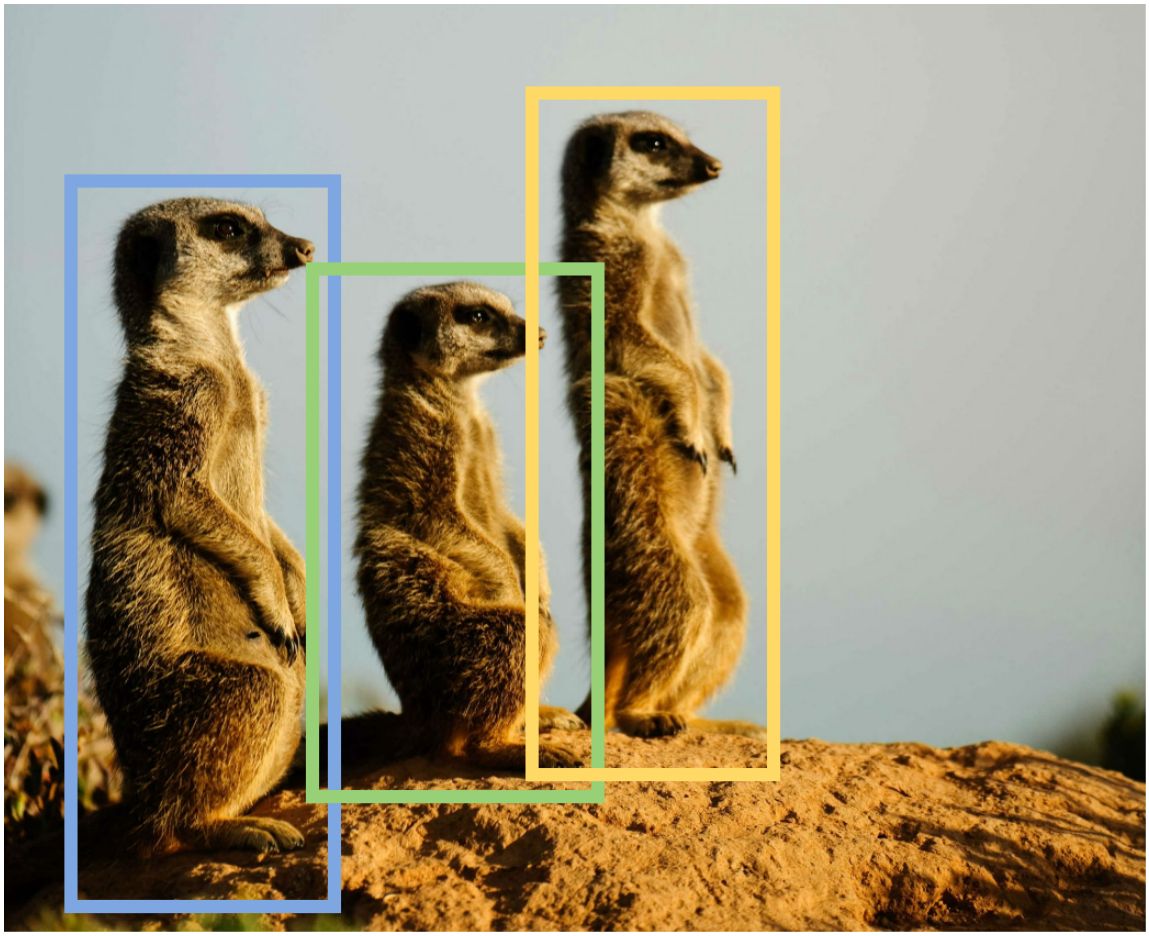
End-to-End Object Detection with Transformers on Edge Devices



Sam Titarsolej

Layout: typeset by the author using LaTeX.
Cover image: atlasreizen.be

End-to-End Object Detection with Transformers on Edge Devices

Sam Titarsolej
12206385

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
MSc. Xiaotian Guo

Parallel Computing Systems
Faculty of Science
University of Amsterdam
Science Park 907
1098 XG Amsterdam

Jun, 2021

Abstract

End-to-End Object Detection with Transformers (DETR) [1] shows great performance in the task of object detection, without the need for complex and manual fine-tuning for merging duplicate predictions. The deployment of systems such as DETR on devices with limited computational resources, such as edge devices, is still a challenge however. In recent years, the computer science paradigm of edge computing has shown great potential in reducing the gap between data sources and the location at which the data streams are processed. The research described in this thesis aims to reduce the trade-off between inference accuracy and inference speed of DETR on edge devices. We study knowledge distillation techniques for simplification of the DETR architecture, as well as parallelization and model splitting approaches to enable DETR to be deployed on an NVIDA Jetson TX2 edge device. The results of our experiments with distillation and parallelization of DETR provide insight in bottlenecks of our proposed pipeline, as well as a baseline to build upon in future research.

Contents

Chapter 1

Introduction

Object detection is the task of predicting a set of bounding boxes and classes for a certain number of objects in an image. State of the art approaches to object detection require complex post-processing of predictions, and are often manually fine tuned to merge duplicate bounding box predictions [1]. DEtection TRansformer (DETR) [1] approaches object detection as a closed set problem, meaning that the number of predictions by DETR is fixed. Through this approach of object detection, DETR reduces the complexity needed to produce the same state of the art results compared to more complex systems.

While DETR achieves state of the art results in object detection on the COCO dataset [2] with this reduced complexity, deploying object detection systems to devices with computational resource constraints is still a challenge. Modern architectures such as DETR come with a significant trade-off between the speed of the system and the performance of the system [3]. By studying inference optimization techniques on DETR, this thesis aims to answer the following research question: How to perform object detection with Transformers on a computational resource-constrained device? We propose new methodology that allows deep learning architectures such as DETR to be deployed on edge devices improving execution speed without significantly losing prediction accuracy.

In many real-world scenarios, systems such as DETR are deployed on cloud servers, usually introducing high latency due a long data transmission path. Thus, in time-sensitive cases, ideally the data streams would ideally be (pre)processed closer to the data source to reduce this latency. We propose a pipeline prototype to model and test the latency of end to end object detection on edge devices.

Such a pipeline would be beneficial towards both scientific research, as well as societal applications of deep learning systems. Reducing the trade-off described by Bianco et al. [3] could help both machine learning related research as well as non machine learning related research, through faster prototyping on less computational resource constrained hardware. It would also be helpful in granting access to deep learning systems to a wider audience through deployment on for example mobile devices.

To achieve such a pipeline we propose and test a set of optimization techniques. These techniques include both parallelization of the DETR architecture across multiple devices and distillation of the DETR architecture. Previous work on deployment of deep learning models on devices with limited resources, focuses solely on either one of these aspects [4] [5]. By combining these techniques we expect to achieve faster inference time compared to the original DETR model, while maintaining the performance produced by DETR.

1.1 Related Work

Plastiras et al. [4] achieve high precision and low processing time object detection on edge devices with their proposed Convolutional Neural Network (CNN) based EdgeNet architecture. This architecture involves multiple CNN detectors, improving the accuracy and processing time of their framework. The first CNN in their system is responsible for the estimation of possible object locations in an input image. Using a CNN-based tiling and selection technique, the outputs of this first stage are refined. The tiling technique reduces the processing time needed to process the image by finding the smallest region of an input image that needs to be processed, using the outputs of the first stage. Plastiras et al. [4] describe 90% precision at 0.015 seconds of processing time required using EdgeNet.

Research towards the distribution of object detection across multiple edge devices and cloud computers (Ren et al., 2018) [6] proposes a network of distributed layers of image feed compression and processing. Through this network, both the size and the amount of the traffic between surveillance cameras and cloud processing facilities are minimized. This work is mostly aimed at studying compression of less relevant sections of an image for communication between the edge and the cloud facility, and exposes a significant trade-off between compression ratio and detection accuracy.

Chapter 2

Theoretial Framework

The experimental research described in this thesis builds upon research and studies of others. To this end, the following sections describe these studies, to build a fundamental theoretical framework for the experiments described in this thesis.

2.1 Edge Computing

Edge computing is a paradigm in the field of computer science that moves data processing closer to the source of the data; the computing edge. Through edge computing, the response and processing time of data pipelines can be improved, and communication bandwidth with centralized data centers can be reduced. Edge computing is made possible through edge devices (figure 2.1), which are often directly connected to a data source. Using edge devices, a topology of distributed computing and processing nodes can be built, removing dependencies on centralized services and internet connections. The edge devices in this distributed network however, often have limited computational resources, creating challenges for deployment of computationally heavy tasks such as deep learning on these machines.

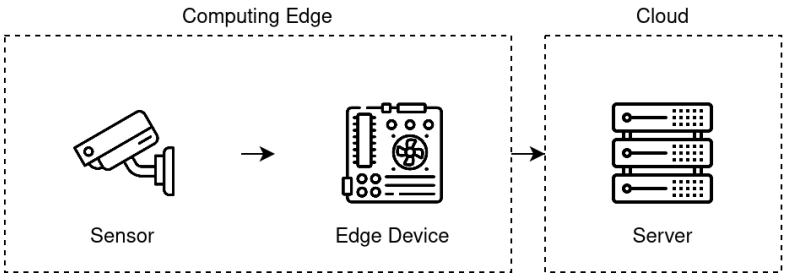


Figure 2.1: The role of edge devices in edge computing.

2.1.1 Deep Learning at the Edge

As shown by Bianco et al. [3], to perform inference modern deep learning systems, modern hardware is required. This requirement introduces challenges for the inference on edge devices with limited computational resources. Bianco et al. also demonstrate that the inference time of a ResNet50 image classifier is roughly ten times slower on an edge device compared to inference on a high-end modern graphics card. This reduction in speed makes for example real time video feed processing with high accuracy close to impossible on such devices with limited computational resources.

2.2 DETR Revised

The focus of our research is the optimization of DETR [1] for deployment on edge devices. The DETR architecture (figure 2.2) uses a ResNet50 (He et al.) [7] backbone and a Transformer (Vaswani et al.) [8] feature interpreter. In the DETR model, the backbone generates a set of image features from an input image, forming an internal representation of an image. Images that contain similar information, result in a similar set of features. These features are then processed and interpreted by the Transformer encoder-decoder blocks, resulting in a set of bounding boxes and classes for the input image. Since DETR approaches object detection as a closed set problem, the set of outputs is of a fixed size. Not only does this imply that the model can only detect a limited number of bounding boxes, it also means that the outputs of the system can contain no object detections. No object detections occur when DETR detects less objects than the length of the closed output set.

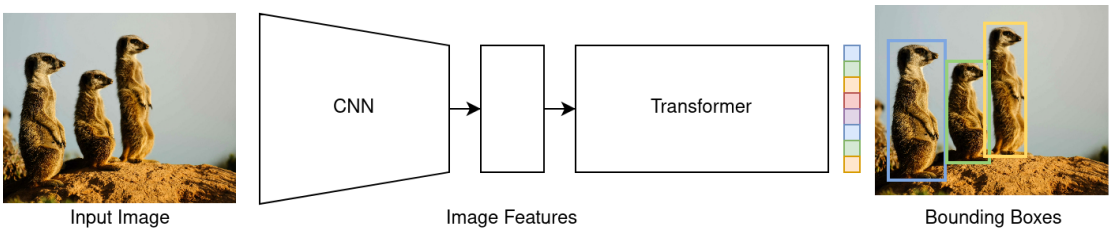


Figure 2.2: The DETR architecture diagrammatically, with a CNN backbone producing image features and a Transformer interpreting these image features into bounding box predictions.

For a better understanding of the DETR architecture, the individual components are described in more depth in the following sections.

2.2.1 ResNet Revised

When training deep learning models, the vanishing gradient problem is often encountered, resulting in sub-optimal training results. The vanishing gradient problem occurs when the error signal of the output of a model travels backwards through a model to calculate parameter updates. When the error signal reaches the deeper layers of a model, the strength of the signal decays. As the error signal decays, the parameter updates of these deeper layers become smaller, meaning that these deeper layers are not updated proportionally.

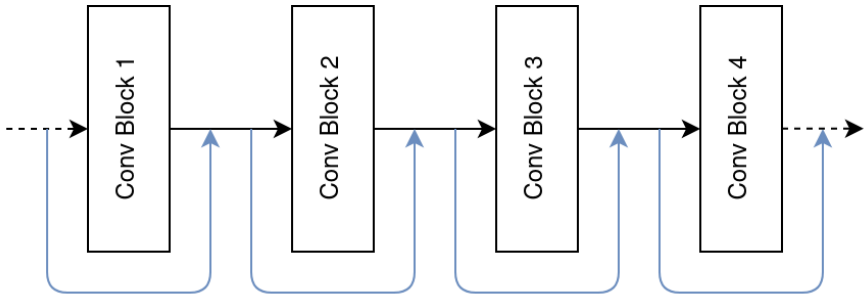


Figure 2.3: Residual connections between convolutional blocks in the ResNet [7] architecture, enabling error propagation throughout deeper layers of the model.

The ResNet architecture aims to eliminate the vanishing gradient problem through residual connections. Residual connections (figure 2.3) in a model allow the error signal to back propagate through the network without passing through non-linear activation functions. By bypassing non-linear activation functions, the strength of the error signal is maintained throughout the deeper layers of the model. Using residual connections, the ResNet architecture allows for exceptionally deep models, resulting in state of the art results in various computer vision tasks, as demonstrated by He et al. [7]. He et al. propose multiple ResNet architectures of various depths. ResNet50 is a deep Convolutional Neural Network (CNN) (LeCun et al.) [9] with 48 convolutional layers and 50 layers overall. The depth of the architecture provides systems using ResNet50 with a solid and low dimensional representation of images, where similar images have a similar representation.

A CNN employs learnable convolutional kernels to learn relevant filters for image processing. Each convolutional layer in a CNN consists of n such kernels and reduces the dimensionality of the input image. When trained, the learnable kernels in a CNN represent feature filters for the input images. For instance, one such feature filter might detect edges in the input image. By performing multiple convolutions sequentially, the CNN learns to apply both low level filters such as edge detection, as well as high level filters such as facial feature detection.

2.2.2 Transformers Revised

For the interpretation of the image features produced by the ResNet50 backbone, DETR uses a Transformer [8]. The Transformer is a model architecture that uses attention mechanisms [10] [11] to learn to distinguish relevant information of an input sequence, such as a text document, from irrelevant information. Originally, the Transformer architecture was developed as a language model. In language modelling the attention mechanism in the Transformer architecture enables a system to model longer sequences as opposed to earlier techniques such as the Long Short Term Memory (LSTM) architecture (Hochreiter & Schmidhuber) [12]. A LSTM models an input sequence step by step. At each step the model stores relevant information about the current step and about previous steps, and forgets irrelevant information. By processing input sequences sequentially, the LSTM architecture is unable to learn long term dependencies and references in an input sequence.

Transformers approach sequence modelling differently. A Transformer models an entire sequence at once, and filters relevant information from irrelevant information through attention. Each element in a sequence attends to all other elements in the sequence to a certain degree. The amount that an element attends to another element describes how closely these two elements are related. In Language modelling, referential words highly attend to the words they refer to for example.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{n}})V \tag{2.1}$$

Attention mechanisms work through three important features: query vectors, key vectors and value vectors. The query vector describes what information needs to be retrieved from an input sequence to process the sequence, while the key and value vectors describe the information in the input sequence. More specifically, the key vector is a representation of the input sequence and the value vector is a repre-

sentation of what these keys represent semantically. In the attention mechanism, the query vector queries value vectors through the key vector. Mathematically, the attention mechanism is a normalized dot product between the query vector and the key vector resulting in weights for the value vector. The value vector is then multiplied with this normalized dot product in an element-wise manner, which in turn results in a weighted sum of the value vector (equation 2.1).

Attention mechanisms (top left of figure 2.4) learn to attend to certain parts of a sequence by learning to generate these query, key and value vectors from the input vectors. The query, key and value vectors are the result of a dot product between the input sequence and a query, key and value weight matrix. The weights in these matrices are subject to optimization during the training process of the model, enabling the model to learn to attend to certain parts of a sequence more than others. After the scaled dot product is applied to the query, key and value vectors, the results are concatenated and fed through a single feed forward layer.

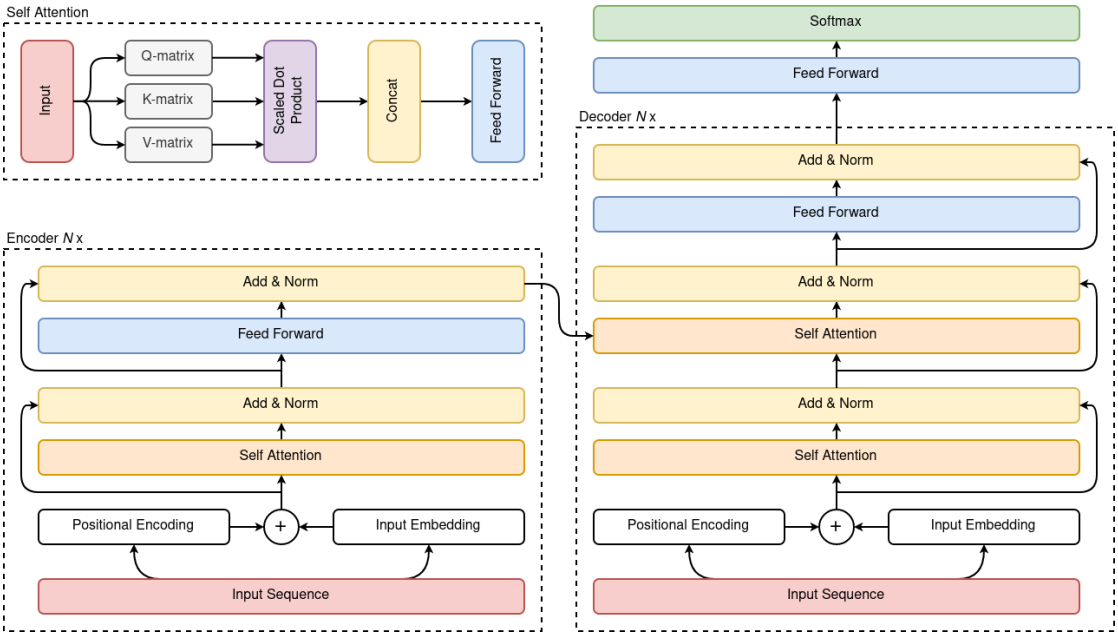


Figure 2.4: Transformer architecture with N encoders & decoders and the attention mechanism diagrammatically (top left).

Transformers are composed of encoder and decoder decoder blocks, as shown in figure 2.4. Encoder blocks encode an input sequence using the attention mechanism into an internal representation of the input sequence. An encoder block employs an attention block with a residual connection, followed by a feed forward linear layer with a residual connection. Similar to residual connections in the ResNet architecture, residual connections allow information to exist in deeper layers of the Transformer architecture. Decoder blocks decode the internal representation of the input sequence to make a prediction depending on the task at hand. The input for the encoder block is the input sequence, while decoder blocks take both the input sequence to generate queries, as well as the output of an encoder block as input to apply these queries on. To allow for more complex sequence modelling, the encoder decoder block architecture is chained N times, making the Transformer model deeper.

2.2.3 Learning Object Queries

A key part in the success of Transformers on sequence processing is the query vector of the attention mechanism. In language modelling the query vector is an internal representation of what information needs to be retrieved from a sequence of words or characters. Since the DETR architecture processes images and not sentences or text documents, these query vectors play a different role in the Transformer of the DETR model. The query vectors represent what information needs to be retrieved from an input image for the Transformer in DETR. In language modelling the Transformer learns to generate these query vectors based on the input sequence. DETR learns to compose more general query vectors during the training process that are not based on single input images, but are applicable to all images. These queries are the input for the decoders in the DETR Transformer. Visualizing the learned queries after training DETR shows that the model learns to examine different locations of an image on different scales.

2.2.4 Bipartite Matching Loss

As DETR predicts a closed set of bounding boxes and classes, the positions of these predictions in the output set should not influence how the predictions of DETR are interpreted. In order not to penalize the model for incorrect prediction positions, a bipartite matching loss is employed (equation 2.2). By finding the minimal error between a prediction and ground truths in a training set, the bipartite matching loss matches predictions and targets. The set of relations (matches) between the set of predictions and the set of ground truths is found, such that the sum of errors between the predictions and the ground truths is minimized. Each element in the set of predictions is matched to a element in the set of ground truths.

$$\hat{\sigma} = \arg\min_{\sigma \in S_N} \sum_{i=1}^{N} L_i(y_i \hat{y}_{\sigma i}) \tag{2.2}$$

All predictions in the closed set of predictions consist of a class prediction and a bounding box prediction. A class prediction is a probability distribution over the set of classes, meaning that the model predicts a probability for each class. A bounding box prediction is represented by the width, the height and the center coordinates (x, y) of the bounding box in the image. Since DETR predicts both classes and bounding boxes, two separate error functions are combined to calculate the prediction error of the model as shown in equation 2.3; a cross entropy loss for the class, and a box-loss for the bounding box. When the prediction for a bounding box is the empty set, and the ground truth value for that bounding box is also the empty set, the value of this match loss is 0.

$$\mathcal{L}_{match}(y_i, \hat{y}_{\sigma(i)}) = -\mathbb{I}_{\{c_i \neq \emptyset\}} \hat{p}(c_i) + \mathbb{I}_{\{c_i \neq \emptyset\}} \mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)}) \tag{2.3}$$

The loss function of the bounding box prediction (equation 2.4) is a combination of the L1-loss and the Generalized Intersection-Over-Union (GIOU) loss proposed by Rezatofighi et al. [13]. This combined loss function takes into consideration both the absolute displacement (L1) and overlap in area between two bounding boxes (GIOU). Both elements in this loss function are scaled with hyperparameters λ_{GIOU} and λ_{L1}.

$$\mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)}) = \lambda_{giou} \mathcal{L}_{giou}(b_i, \hat{b}_{\sigma(i)}) + \lambda_{L1} \mathcal{L}_{L1}(b_i, \hat{b}_{\sigma(i)}) \tag{2.4}$$

2.3 Knowledge Distillation

As deep learning models become more complex, expressed in the amount of layers of an architecture, the information learned during the training process of the model is stored in the model less efficiently [14] [15] [16]. Knowledge Distillation (Hinton et al.) [15] (Gou et al.) [16] aims to compress the information stored in a large, complex model into a smaller model with reduced complexity. During the process of knowledge distillation, a larger model acts as a teacher for a smaller model. The smaller model tries to mimic the outputs of the larger model given a certain input.

More recent research towards knowledge distillation on the ResNet architecture specifically, (Gao et al.) [17], has demonstrated an increase of 1.8% in accuracy on the task of image classification. The method proposed by Gao et al. [17] uses Stage-by-Stage Knowledge Distillation (SSKD) to distill the information contained in a ResNet34 (with 34 layers) into a ResNet18 (with 18 layers). Instead of attempting to mimic the output of the larger ResNet34 model, the ResNet18 model learns to mimic the internal representations of ResNet34 of an input image through SSKD. Each stage in the ResNet18 model is trained to mimic the corresponding stage in the ResNet34 model. When a certain stage of the ResNet18 model is trained, the preceding stages are frozen and not optimized any further. The process of SSKD is schematically in figure 2.5

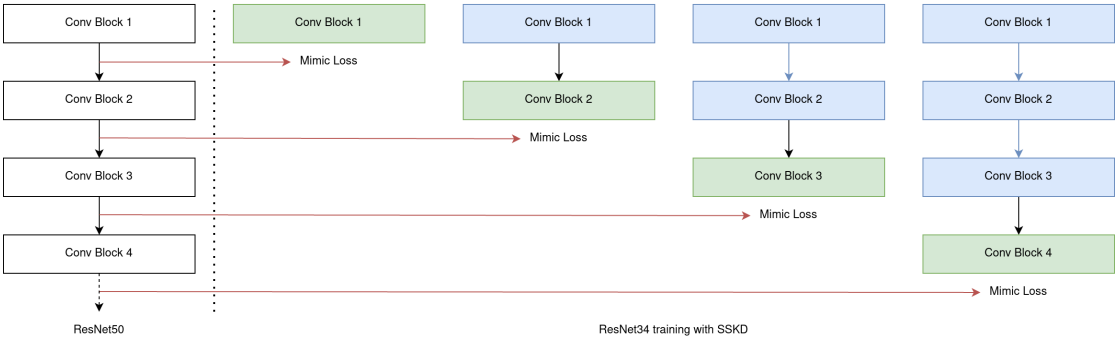


Figure 2.5: SSKD of a ResNet50 to a ResNet34. The green convolutional blocks of the ResNet34 are being optimized in each stage, while the blue convolutional blocks are frozen during the optimitzation of the green convolutional blocks.

Chapter 3

Experiments

The following sections provide an overview of our proposed techniques to optimize the inference time of the DETR object detection system on hardware with scarce resources. In the first of these sections, the general setup and hardware used in our experiments is described. This setup applies to the experiments described in the succeeding two sections. The first experiment is described in the second section which involves the optimization of the image feature extraction backbone in DETR (a ResNet50) through distillation. Our second experiment is described in section three, and explains our approach to the parallelization of performing inference on the hardware described in section 3.1.

3.1 Experimental Setup

Each of the experiments described in this section is conducted on a NVIDIA Jetson TX2. The TX2 is an AI edge computing device with a 256-core NVIDIA Pascal GPU and 8GB of memory. NVIDIA provides a Software Development Kit (SDK) for inference optimization of deep learning models on NVIDIA hardware; TensorRT [18]. Before deployment on the TX2, all models used in our experiments are optimized through TensorRT. The models are converted to TensorRT engines through the Open Neural Network eXchange (ONNX) [19] format.

To evaluate our proposed techniques, the performance of our deployed pipelines on the TX2 is measured in inference time in seconds, which is also converted to images processed per second. The memory usage of our pipeline on the TX2 is tracked during each experiment, as well as the power usage of the GPU of the TX2.

3.2 Backbone Distillation

In our first experiment, we studied the feasibility of deploying the DETR architecture with a ResNet34 backbone without significantly losing performance, through SSKD of the original ResNet50 backbone in DETR. While the ResNet34 backbone is trained to mimic the representation of the ResNet50 backbone, the Transformer layers of DETR are frozen. By freezing the parameters of the Transformer, the target features for the ResNet34 backbone remain constant throughout the distillation process.

The performance of the ResNet34 backbone is measured through average precision (AP) of the entire DETR pipeline, and compared to the AP of the DETR architecture with ResNet50. Comparing the AP of DETR with ResNet34 to DETR with ResNet50 will provide insight into how much knowledge is lost in the distillation process. The resulting AP of DETR with ResNet34 is shown in table 3.1, along with the number of parameters of our distilled DETR model.

Model	**# of Parameters**	**AP**
DETR-R50	49.2M	42.0
DETR-R34 (distilled)	46.8M	41.2
Difference	2.4M (4.8%)	0.8 (1.9%)

Table 3.1: The results of our SSKD experiment of the backbone of DETR from a ResNet50 to a ResNet34. The number of parameters as well as the AP of the DETR model with both backbones are shown.

3.3 Inference Parallelization

Our second experiment focuses on the parallelization of DETR across multiple devices. To perform inference with DETR across multiple devices, the Message Passing Interface (MPI) [20] is utilized. MPI allows multiple processes to communicate with each other on any scale and on multiple devices with minimal overhead. Using MPI, the DETR architecture is split across two devices; the Jetson TX2, and a workstation computer with a NVIDIA RTX 2070 Super. The workstation and the edge device are connected over an internet connection.

Configuration	Backbone on	Transformer on
Full	Jetson TX2	Jetson TX2
Backbone	Jetson TX2	Workstation
Transformer	Workstation	Jetson TX2

Table 3.2: The configurations of our parallelization experiment. Each row shows a different configuration, where the backbone and Transformer of DETR is deployed on the specified devices.

DETR is split between the backbone and the Transformer. All permutations of deploying these backbone and Transformer components on either the TX2 or the workstation are tested except for running the entire architecture on the workstation. These permutations result in three different configurations as shown in table 3.2.

To take advantage of splitting the model across two devices, the backbone and the Transformer run in parallel when processing a sequence of input images, such as a live video feed, as demonstrated by figure 3.1. Through this parallelization setup, the time spent computing features for image n using the backbone is also used to predict bounding boxes with the Transformer for image $n-1$. The results from the parallelization experiment are obtained by processing the first 1000 images from the COCO [2] 2017 validation dataset. During the experiment, inference time of both the backbone and the Transformer are measured, as well as the memory usage and the power consumption. ResNet backbones of multiple sizes are tested as well to measure the change in inference time of the backbone.

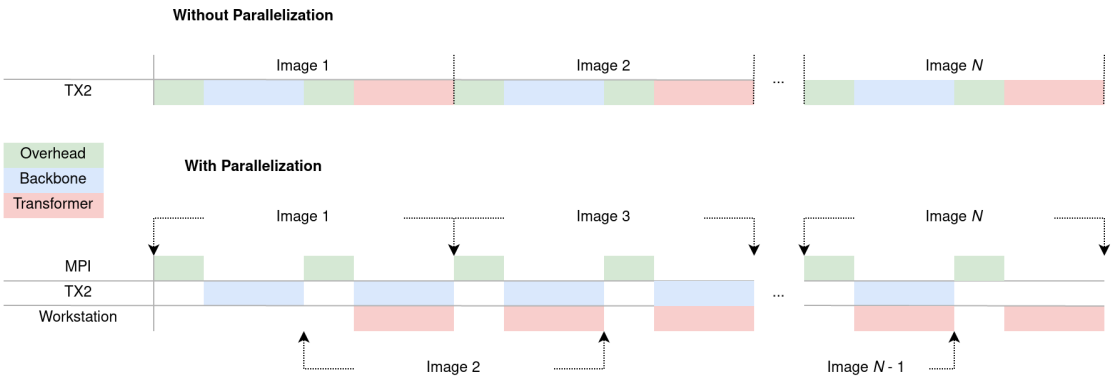


Figure 3.1: Our pipeline processes an image feed in parallel, with the backbone and the Transformer of DETR processing images simultaneously.

Figures 3.2 and 3.3 show the results of our parallelization experiment expressed in frames per second (FPS) and the inference times of different configuration of this experiment. While the first figure shows the FPS of the parallelization pipeline with different ResNet backbones, the second figure only show the results using a ResNet50 backbone. The second figure also shows how long each individual component (TX2, MPI & Workstation) of our inference pipeline takes.

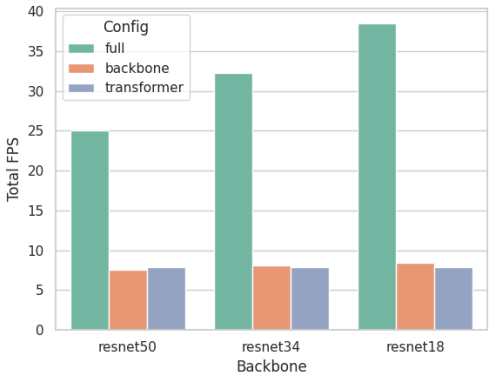


Figure 3.2: FPS of our parallelized inference pipeline for different ResNet backbones

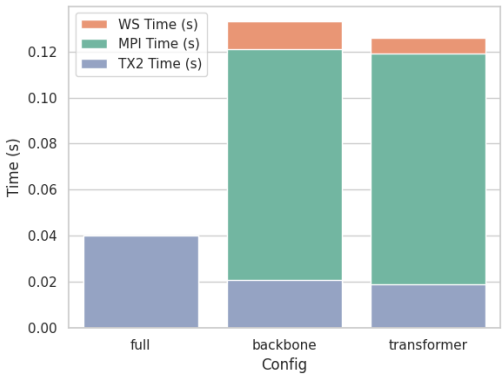


Figure 3.3: Inference time of our parallelized inference pipeline of different configurations

The data in figures 3.4 and 3.5 show the throughput of each deployed component on the TX2 and the inference time spent on the TX2 respectively. Both figures display these measures for three different sized ResNet backbones.

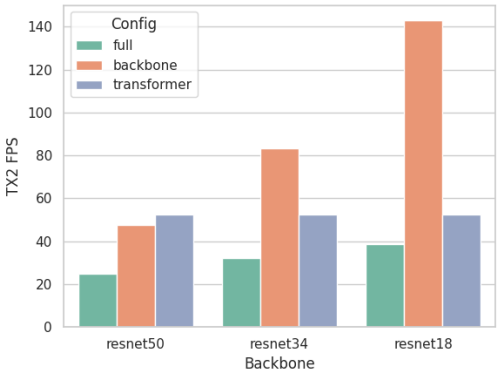


Figure 3.4: Inference speed of TX2 component in FPS

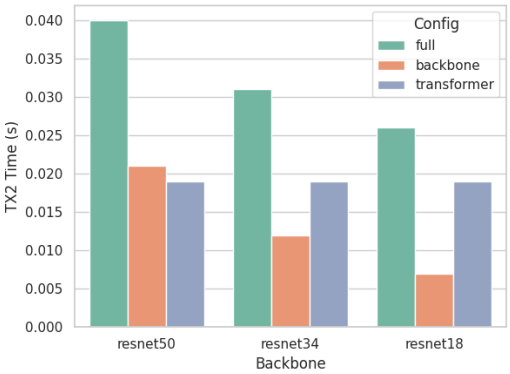


Figure 3.5: Inference time of TX2 component in seconds

Finally, figures 3.6 and 3.7 show the power usage and the memory usage of each DETR component on the TX2 with different sized ResNet backbones.

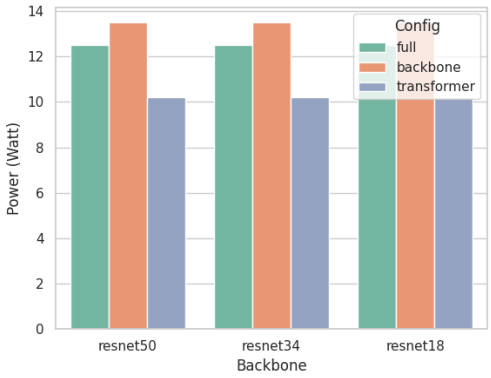


Figure 3.6: TX2 power usage in watts

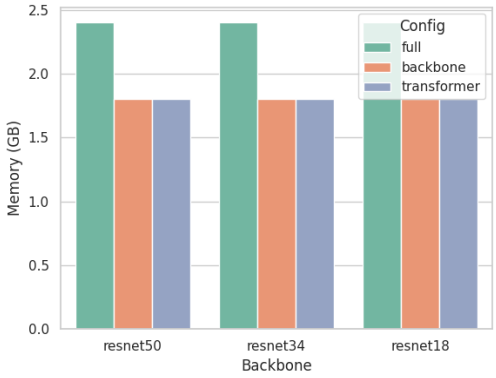


Figure 3.7: TX2 memory usage in GB

Chapter 4

Discussion

Our knowledge distillation experiment of the ResNet50 backbone of DETR shows that it is feasible to reduce the depth of the backbone from 50 layers to 34 layers, while staying within a 2% performance margin expressed in AP. Moreover, the paralellization experiments show that replacing the ResNet50 backbone with a ResNet34 backbone decreases the inference time of the backbone on the edge device from 0.021 seconds to 0.012 seconds, or 42.8%. In terms of throughput (FPS), the edge device is able to process over 80 images per second. In a real-world scenario, with for example surveillance cameras providing a video feed with 30 FPS, a distilled ResNet34 backbone could process multiple such video feeds on a single edge device. Distilling the ResNet50 backbone does not significantly change the power usage and the memory usage of the Jetson TX2.

The results of the parallelization experiment also show that the main bottleneck of our parallelization pipeline is the internet connection between the edge device and the workstation. However, by interpreting this overhead as latency in inference, instead of as a bottleneck in inference time directly, the actual inference time of our parallelization pipeline is reduced from 0.04 seconds in the 'full' configuration to 0.033 seconds in the 'backbone' configuration. Our pipeline decreases the total inference time of DETR by 17.5% by running the Transformer of the DETR architecture from the edge device to a powerful workstation.

Combining the results of the distillation feasibility study, and the parallelization experiment, the inference time of DETR on edge devices is reduced from 0.031 of the 'full' ResNet34 configuration to 0.024 seconds of the 'backbone' ResNet34 configuration. These combined results show a 22.5% decrease in inference time. In terms of throughput, combining our experiments achieves 41.6 FPS on the entire DETR architecture when partially deployed on a NVIDIA Jetson TX2.

4.1 Future Work

Our parallelization experiment shows that the latency caused by the internet connection is of significant impact on our proposed pipeline. In some applications this latency might not be acceptable. In other applications, for example when the edge device is deployed in remote locations with slow internet connectivity, the latency could be even larger than described in our experiment. The latency could be reduced significantly by splitting the DETR architecture using our parallelization methodology on two edge devices directly, instead of on an edge device and a workstation computer. The results of our parallelization experiment show that such a split could drastically decrease the inference speed of DETR on multiple edge devices compared to performing inference on a single device.

For futher optimization the DETR architecture for (partial) deployment on edge devices, the complexity of the Transformer of the DETR model could be optimized as well. For optimal perfomance in AP, the Transformer in DETR uses 12 encoder and 12 decoder layers. Carion et al. [1] show that reducing the number of encoder layers from 12 to 6, increases the FPS of DETR by 15%, while decreasing the AP of DETR by 2.4%. Potentially, performing knowledge distillation on the DETR Transformer, similarly to our proposed method of knowledge distillation of the DETR backbone, could improve the AP of DETR with less Transformer encoder layers.

4.2 Conclusion

Through our distillation experiment we have demonstrated the feasibility of performing knowledge distillation of the backbone of the DETR architecture from ResNet50 to ResNet34. We have also shown that replacing the ResNet50 backbone with a ResNet34 backbone decreases the inference time of the DETR architecture on an edge device by 42.8%. Our model splitting methodology and parallelization experiment strongly indicate great potential of inference speedups through parallelization of DETR across multiple devices.

References

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

[2] 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.

[3] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, p. 64270–64277, 2018.

[4] G. Plastiras, C. Kyrkou, and T. Theocharides, "Edgenet," *Proceedings of the 13th International Conference on Distributed Smart Cameras*, Sep 2019.

[5] S. Tuli, N. Basumatary, and R. Buyya, "Edgelens: Deep learning based object detection in integrated iot, fog and cloud computing environments," 2019.

[6] J. Ren, Y. Guo, D. Zhang, Q. Liu, and Y. Zhang, "Distributed and efficient object detection in edge computing: Challenges and solutions," *IEEE Network*, vol. 32, no. 6, pp. 137–143, 2018.

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

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

[9] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," Springer-Verlag, 1999.

[10] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016.

[11] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," 2015.

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, p. 1735–1780, Nov. 1997.

[13] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," 2019.

[14] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," 2020.

[15] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, p. 1789–1819, Mar 2021.

[16] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[17] M. Gao, Y. Shen, Q. Li, J. Yan, L. Wan, D. Lin, C. C. Loy, and X. Tang, "An embarrassingly simple approach for knowledge distillation," 2019.

[18] NVIDIA, "Tensorrt." https://github.com/NVIDIA/TensorRT, 2021.

[19] Microsoft, "Open neural network exchange." https://github.com/onnx/onnx, 2021.

[20] "Mpi: A message-passing interface standard," tech. rep., USA, 1994.