Darmstadt University of Applied Sciences

Computer Vision Seminar

Unearthing Surgical-Dino

Author
Tilman Seeßelberg

Reviewer
Prof. Dr. Andreas Weinmann, Darmstadt University of Applied Sciences
Vladyslav Polushko, Darmstadt University of Applied Sciences

Date of Submission: June 26, 2024

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Abstract

Kurzzusammenfassung

Contents

List of Figures

List of Tables

List of abbreviations

MLP Multi Layer Perceptron

Chapter 1
Introduction

Chapter 2
Theoretical Foundations

2.1 Transformers

Transformers [1] were first introduced in 2017 as successors to neural machine translation (NMT) algorithms such as long short-term memory (LSTM) [2] and Recurrent Neural Networks (RNNs) [3] . The transformer architecture follows an encoder-decoder structure, where the encoder maps an input sequence to a continuous representation, and the decoder generates the output sequence from this representation.

Transformers improve on several short commings that previous state-of-the-art algorithms at that time such as LSTM and RNNs have struggled with, including computational efficiency and context window size [4, p 609]. The context window refers to the length of the input sequence considered for predicting the next word or token. A small context window limits the ability to attend to words that are far apart, e.g. 30 words, which can be problematic for understanding long texts or complete documents. The transformer model efficiently handles a much larger, theoretically infinite, context window using a mechanism called attention, which comes for the transformer is used in two kinds: self-attention and cross-attention (an explanation for attention follows further down).

Typically, a transformer is composed of multiple distinct components: the embedding layer, positional encoding, attention mechanisms, and feed-forward neural networks as can be seen in Figure 2.2.

Before sequential data is input into the transformer model, it needs to be embedded. The embedding model is a dictionary that maps known small patterns such as words, syllables, or other snippets in sequences to so-called embedding vectors. This way, a sentence can be represented as a matrix where each column is an embedding vector of a word in the sentence.

Given the significance of a word's position in providing context within a sentence, an additional pattern, known as positional encoding, is incorporated for the model to learn. Positional encoding becomes crucial in transformers due to their non-sequential processing nature, which contrasts with RNNs and CNNs that inherently understand the order of inputs. The positional encoding for a given embedded token vector $\mathbf{x} \in \mathbb{R}^n$ is calculated by adding a positional encoding vector $\text{PE} \in \mathbb{R}^n$ to the embedded token vector $\mathbf{x}$. The values of PE for each token are calculated with regard to their position pos inside the input sequence as follows: Every element in the encoding vector with an even index i gets calculated using equation 2.1, and every element at an odd index is computed using equation 2.2.

s

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{2.1}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{2.2}$$

With the input sequence embedded and the positional encoding added, the sequence is ready to be fed into the attention layers.

Attention Layers, also known as **Alignment Models** in machine learning, are a type of trainable memory that can determine the relationship between an embedding vector and other

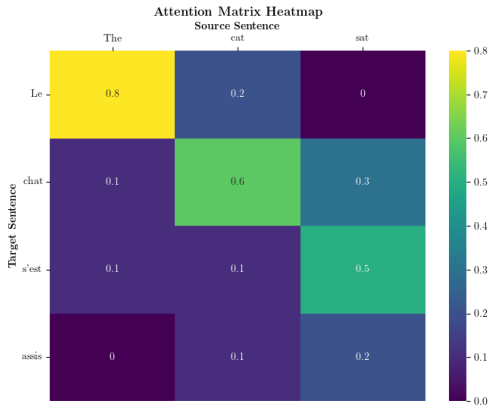

Figure 2.1: Arbitrary example of an Attention Matrix in context of an english-french translation

embedding vectors given an arbitrarily long input sequence. This concept was first introduced in 2014 by Bahdanau et al. [5] as additive attention in the context of machine translation and refined by Luong et al. [6] n 2015 as multiplicative attention, which is more computationally efficient and is used in transformers.

The attention layer outputs an $n \times n$ matrix where n is the number of tokens in the input sequence. Each element in that matrix represents an energy score, illustratet as a heatmap in 2.1, which can be interpreted as how much the word in the row is related to the word in the column, similar to a correlation matrix. Typically, attention is only given to preceding words in the sequence. This is accomplished by masking the upper right triangle of the attention matrix, setting these values to $-\infty$. When the softmax function is subsequently applied, these become zero. The attention layers implemented by Vaswani et al. [1] in the original transformer paper uses trainable weights in the form of three parallel linear layers, which produce three matrixes called query matrix Q, key matrix K and value matrix V. These matrices have the same dimensionality $d_k \times d_k$ where d_k is the dimensionality of the embedding vector. The attention is calculated using equation 2.3 by multiplying Q and K, resulting in a square matrix called the attention score. This score is then normalized to prevent exploding gradients before being multiplied with the value matrix V. The softmax function is applied to ensure the sum of the values in each **row is 1(CHECK IF ITS NOT COLUMNS)**.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.3}$$

Multi-head attention allows the model to learn different types of relationships, such as grammatical structure or contextual changes in word meaning. Multiple attention heads in parallel, each learning independently, are concatenated and processed by a final linear layer. The entire process is illustrated in figure 2.2.

Transformers excel in managing long-distance dependencies, offer substantial parallelization for accelerated training, and outperform tasks such as machine translation. Vaswani et al. [1] further assert that in the field of Neural Machine Translation (NMT), transformers can be trained only using up to a tenth of the floating point operations (FLOPS) required by preceding top-tier models.

WORK IN PROGRESS

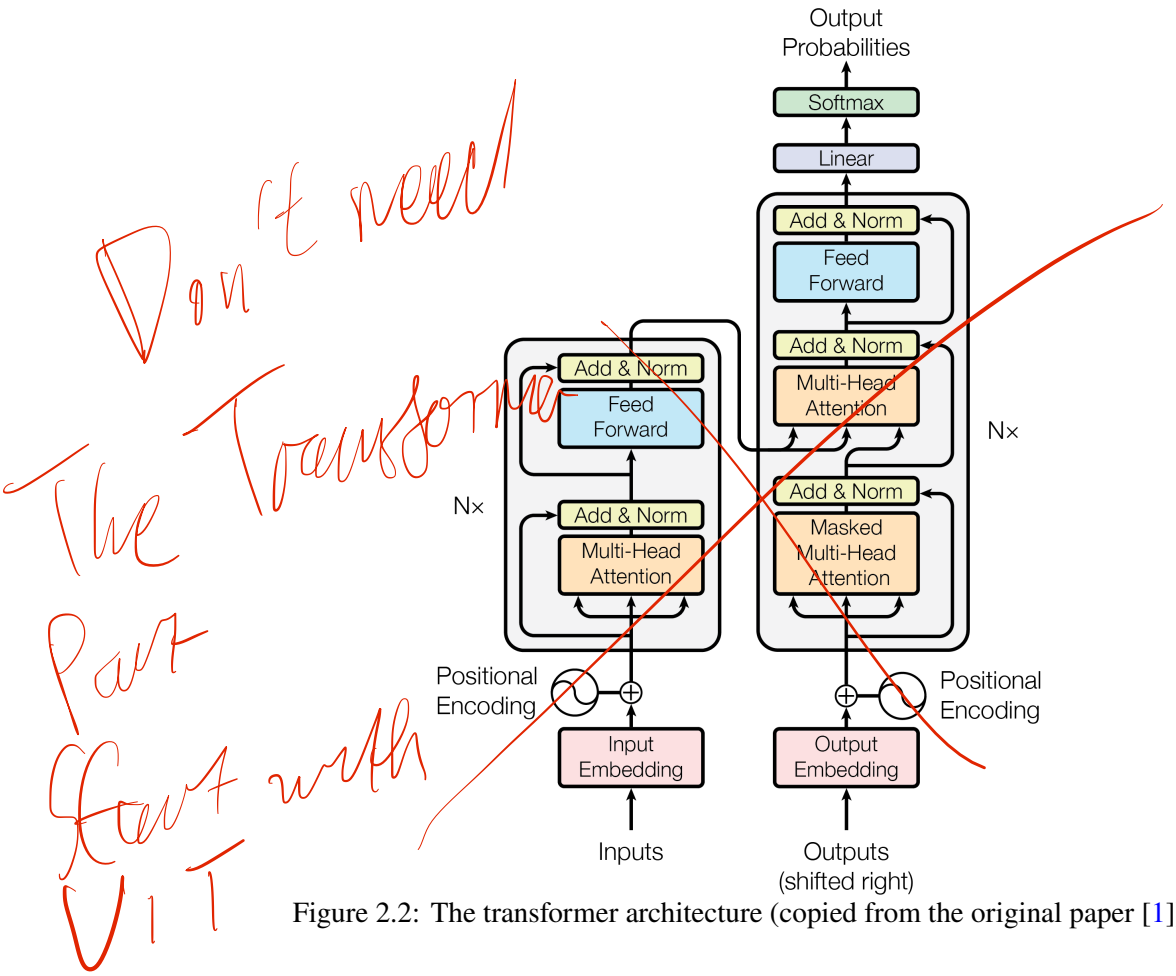

Figure 2.2: The transformer architecture (copied from the original paper [1])

2.2 Vision Transformer

2.3 Vision Transformer

One of the first applications of attention mechanisms in the computer vision field was developed by Xu et al. in 2015 [7], who used a CNN and a downstream RNN with an attention mechanism to generate image descriptions. Vision Transformers (ViTs) then followed in 2020 as a novel approach for image classification developed by Dosovitskiy et al. [8] ViTs are an evolution of transformers as discussed in 2.1.

The ViT, while reusing most parts of the original transformer, presents several key differences from the original transformer architecture. Unlike the original transformer which employs both an encoder and a decoder, the ViT only uses an encoder, as the task of image classification does not necessitate a decoder. Just like the transformer the ViT processes 1d tokens. These however are not generated using a dictionary like embedding layer as the transformer, but simply by splitting the image into 16×16 patches and flattening them into a 1D vector. These flattened patches undergo a learnable linear transformation and are treated as embedding vectors, like in the original architecture. Another significant difference lies in the positional embedding. While the original transformer uses fixed positional embedding, the ViT implements a learnable one. Additionally a so called class token is prepended to the embedding vector matrix, which is meant to destil the information of the image into a single vector, which later is used for classification. Despite these differences, the ViT has demonstrated superior performance over state-of-the-art convolutional neural networks (CNNs) in several image classification tasks. However, it requires

significantly more training data and computational resources, unlike CNNs, transformers do not inherently leverage spatial information in images.

ViTs involves several key steps and components:

- **Patch Embedding**: The input image $x \in \mathbb{R}^{H\times W\times C}$ devided into non-overlapping patches $x_p \in \mathbb{R}^{N\times(P^2\cdot C)}$, with the original resolution (H, W), the color channels C, the path resulution (P, P) and $N = HW/P^2$ as the number of patches. Each patch is flattend and linearly embedded into a embedding vector.
- **Position Embedding**: To retain positional information, standard learnable 1D position embeddings are added to the patch embeddings as 2D position embeddings did not yield any advantage.
- **Transformer Encoder**: The embedded patches, along with positional embeddings, are fed into the transformer encoder. The encoder consists of alternating layers of multi-headed self-attention (MSA) and multi-layer perceptron (MLP) blocks. Each block applies layer normalization (LN) before and residual connections after every block.

The architecture of the Vision Transformer is mathematically described in the paper [1] as follows:

$$z_0 = [x_{\text{class}}; x_1^p E; x_2^p E; \cdots; x_N^p E] + E_{\text{pos}} \tag{2.4}$$

where $E \in \mathbb{R}^{(P^2\cdot C)\times D}$ is the trainable linear projection for patch embedding, and $E_{\text{pos}} \in \mathbb{R}^{(N+1)\times D}$ are the position embeddings.

The transformer encoder operates on these embeddings through L attention heads:

$$z'_l = \text{MSA}(\text{LN}(z_{l-1})) + z_{l-1}, \quad l = 1 \ldots L \tag{2.5}$$

$$z_l = \text{MLP}(\text{LN}(z'_l)) + z'_l, \quad l = 1 \ldots L \tag{2.6}$$

where z_l represents the output of the l-th layer.

Finally, the output representation for classification is obtained from the embedded class token:

$$y = \text{LN}(z_L^0) \tag{2.7}$$

The Vision Transformer (ViT) demonstrates that a pure transformer model applied directly to sequences of image patches can achieve state-of-the-art performance on image classification tasks. The success of ViTs underscores the importance of pre-training on large datasets and the flexibility of transformer architectures in processing different types of data beyond natural language.

2.4 Dino and DinoV2

So far ViTs were mostly trained in a supervised manner and needed a lot of labeled data to achieve state-of-the-art performance. As labeled training data is expensive and time consuming to generate, it is very desirable to train models in an unsupervised manner. Just a year after the introduction of the Vision Transformer, Caron et al. [9] introduced an approach to achieve exactly this called DINO (DIstilled NOt labeled), which is a framework for training in theory any network capable of ingesting image data using self-distillation without labels. However the paper in particular emphasizes the synergy of Dino with Vision Transformers as a good compromise between performance and computational cost. Dino is a usefull framework for tasks such as

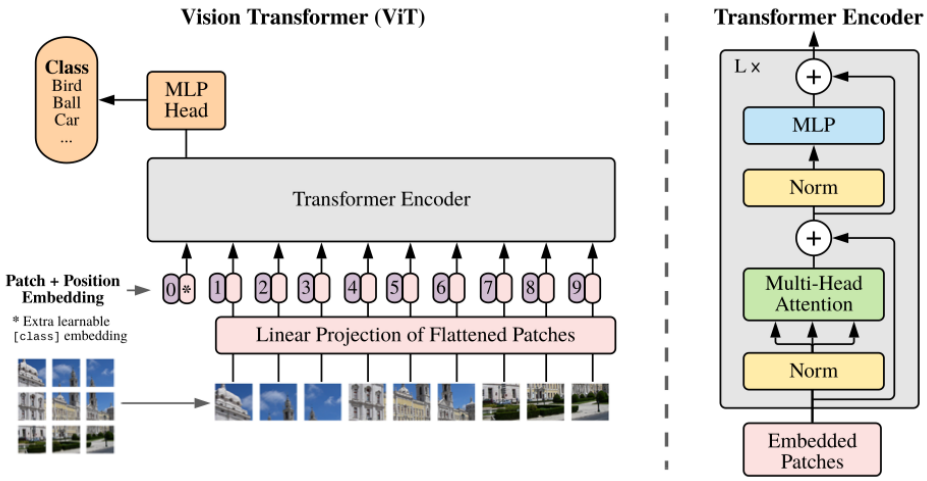

Figure 2.3: The Vision Transformer architecture (copied from [8]).

segmentation, depth estimation, and object detection, as it provides a good initialization for supervised fine-tuning.

As explained in [10] destillation in machine learning is a method facilitating the transfer of knowledge from one model to another using a teacher-student model approach. Often the teacher is a larger complex pretrained model which is used to train a much smaller stundent model. However, in the case of Dino the teacher is a non-pretrained model being trained in parallel with the student model. This process is called self-distillation, as no pretrained model is used to inject prexisting knowledge into the student model.

In a teacher stundent architecture the student tries to predict the output of the teacher model. In Dino the students weights get updated using the classical gradient decend approach, while the teacher model is updated using a exponential moving average of the student model weights. Caron et al. figured that in order to prevent a trivial solution where both teacher and student would start outputing the same output no regard to the input, it is enough to center the output of the teacher model to a mean of 0 and to apply a sharpened softmax function to the output by applying a temperatur parameter τ called temperatur to the softmax function as shown in equation 2.8. It makes very large values even larger and very small values even smaller.

Architecture

Dino is trained using a classical student teacher network approach, though the teacher is not a pretrained model but a model trained in parallel to the student model. The Tracher model is largely the same as the student model, but the outputs follow a normalisation applying centering subtracting the mean and running the resulting outputs thorugh a shapened softmax function as shown in equation 2.8.

$$q_i = \frac{f_i}{|f_i|_2} \cdot \tau \tag{2.8}$$

- Attention Map right after training puts much attention on the main object of the image and less on the background. This already is a good start for segmentation - Dino is trained unsupervised and places, when trained, images in to a latent vector, where similar images are close to each other, similar to how similar words are embedded close to each other when training a embedding on them. This could already be used for a k-nearest neighbor approach to classify images.

2.4.1 Depth Estimation

2.4.2 Segmentation

2.5 Model Fine-Tuning

Foundation Models (FM) are, as defined in [11], models trained on a large variety of data using self-supervised learning sometimes even for weeks on hunderds of graphic processing units (GPUs) as e.g. the GPT3 model [12] or the Florence CV Foundation Model by Yuan et al. [13]. These FM are then fine-tuned doing few shot training, to tailor the FM to a specific task using by several magnitutes less data as would be needed for training a model from scratch. This saves a lot of computational cost and time as usually only small parts of the FM are retrained during fine tuning, and most of the parameters get frozen as well as that cost can be saved creating the large dataset. Mathematically, the straight forward way of fine-tuning a model by retraining all parameters, which can be written as in equation 2.9 with W_0 being the pretrained weights and δW the change in weights during fine-tuning.

$$W' = W_0 + \delta W \tag{2.9}$$

This However still is relativley computationally intesive. **DISCUSS IN GENERALL HOW PRETRAINING WORKS???**

A more efficient way of fine-tuning weight matrices is Low Rank Adaption (LoRA) as suggested by Hu et al. [14] in the context of transformer fine-tuning. Hut et al. assume based on the the work of Aghajanyan et al. [15], that the updated weight matrices δW_n inside the attention heads of the transformer have a rank much lower then their dimensionality. In order to utilize this assumption to more efficiently fine-tune a transformer the updated weight matrices δW_n are decomposed into two matrices $A \in \mathbb{R}^{d \times r}$ and $B \in \mathbb{R}^{r \times k}$ with $r << d$ and $r << k$ as shown in equation 2.10. Hu et al. set the goal to use a maximum of 18M parameters for fine tuning the 173B parameter GPT3 model and created a grid search for optimizing the rank r and which weight matrices to update. Eventually they figured for their usecaser a rank $r = 4$ and only applying updates to all key matrices K and value matrices V in the attention heads of the transformer performed the best. Besides advantage of significantly lowering the computational cost of only optimizing the A and B matrices, fine-tuning using LoRa does not increase inference times as the decomposed matrices are recomposed and added onto the original weight matrices W_0 and therefor do not add any additional computation to the forward pass of the model.

$$W = W_0 + BA \tag{2.10}$$

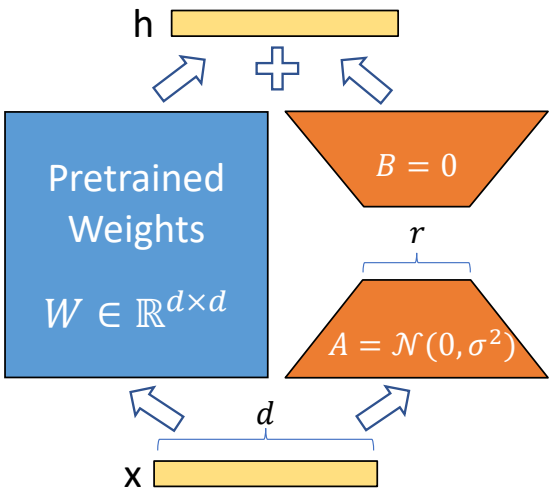

Figure 2.4: LoRa architecture with the docomposed $\delta W_n = AB$ and their initialization method (graphic lifted from [14])

Chapter 3
Unearthing Surgical-Dino

3.1 Improvements of Surgical-Dino

Chapter 4
Discussion

Chapter 5
Conclusion

Appendix A
Code Tools

Bibliography

[1] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2017. DOI: 10.48550/ARXIV.1706.03762.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 1530-888X. DOI: 10.1162/neco.1997.9.8.1735.

[3] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, and S. Valaee, *Recent advances in recurrent neural networks*, 2018. DOI: 10.48550/ARXIV.1801.01078.

[4] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems, Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Incorporated, 2022, ISBN: 9781098125974.

[5] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, 2014. DOI: 10.48550/ARXIV.1409.0473.

[6] M.-T. Luong, H. Pham, and C. D. Manning, *Effective approaches to attention-based neural machine translation*, 2015. DOI: 10.48550/ARXIV.1508.04025.

[7] K. Xu, J. Ba, R. Kiros, *et al.*, "Show, attend and tell: Neural image caption generation with visual attention," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., ser. Proceedings of Machine Learning Research, vol. 37, Lille, France: PMLR, Jul. 2015, pp. 2048–2057. [Online]. Available: https://proceedings.mlr.press/v37/xuc15.html.

[8] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, *An image is worth 16x16 words: Transformers for image recognition at scale*, 2020. DOI: 10.48550/ARXIV.2010.11929.

[9] M. Caron, H. Touvron, I. Misra, *et al.*, "Emerging properties in self-supervised vision transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2021, pp. 9650–9660.

[10] M. Phuong and C. Lampert, "Towards understanding knowledge distillation," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, Jul. 2019, pp. 5142–5151. [Online]. Available: https://proceedings.mlr.press/v97/phuong19a.html.

[11] R. Bommasani, D. A. Hudson, E. Adeli, *et al.*, *On the opportunities and risks of foundation models*, 2021. DOI: 10.48550/ARXIV.2108.07258.

[12] B. Yuan, Y. He, J. Davis, *et al.*, "Decentralized training of foundation models in heterogeneous environments," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, Curran Associates, Inc., 2022, pp. 25 464–25 477. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/a37d615b61f999a5fa276adb14643476-Paper-Conference.pdf.

[13] L. Yuan, D. Chen, Y.-L. Chen, *et al.*, *Florence: A new foundation model for computer vision*, 2021. DOI: 10.48550/ARXIV.2111.11432.

[14] E. J. Hu, Y. Shen, P. Wallis, *et al.*, *Lora: Low-rank adaptation of large language models*, 2021. DOI: 10.48550/ARXIV.2106.09685.

[15] A. Aghajanyan, L. Zettlemoyer, and S. Gupta, *Intrinsic dimensionality explains the effectiveness of language model fine-tuning*, 2020. DOI: 10.48550/ARXIV.2012.13255.