Data C182 Designing, Visualizing & Understanding DNN
Fall 2024 Eric Kim, Naveen Ashish

Discussion 09

This discussion covers Vision Transformer (ViT) and Masked Autoencoder (MAE).

1. Vision Transformer

Vision transformers (ViTs) apply transformers to image data by following the following procedure:

(a) **Split image into patches** - The original ViT paper split images into a 16x16 grid of patches.

(b) **Convert each patch into a single vector** - In the original paper, they flattened the patch and applied a linear projection.

(c) **Stack the patches into a sequence, concatenate a CLS token, and add in positional embeddings.** Absolute learned positional embeddings are most common here.

(d) **Pass the sequence through a transformer as usual.**

Below is a diagram of ViT for supervised image classification.

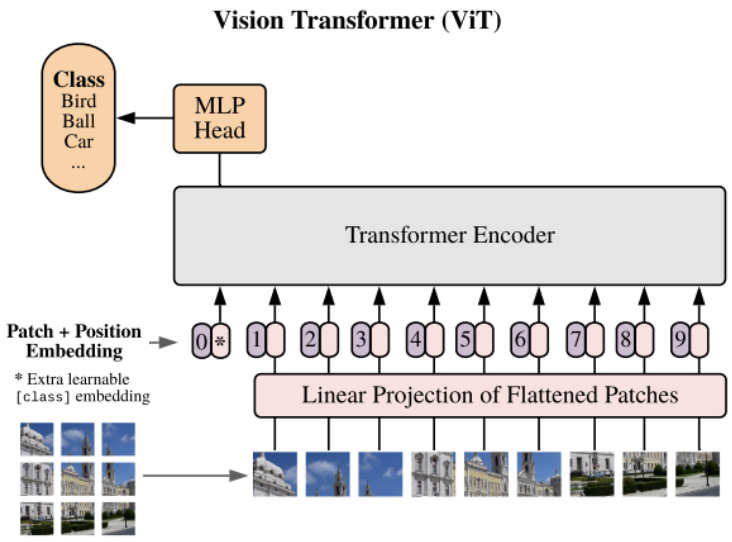

Figure 1: Vision Transformer (Link to Google blog)

(a) We are not going to cover this in detail, but for Homework 4, you will need to use the `einops` Python library. It is a better way to reshape Python tensors, and frequently used to simplify the image-patchify implementation for ViT.

We point you to the good official tutorials:

- https://einops.rocks/pytorch-examples.html
- https://github.com/arogozhnikov/einops/tree/main/docs
- https://githubtocolab.com/arogozhnikov/einops/blob/main/docs/1-einops-basics.ipynb

And if you are adventurous, try lucidrain's very clean implementation of ViT.

(b) Does it matter which order you flatten the sequence of patches?

Solution: No, the positional encoding will tell the model where the patch came from, as long as we use the same flattening order (ie. row-order or column-order) throughout training.

(c) What is the complexity of the vision transformer attention operation? Assume you have an image of size $H \times W$ and patches of size $P \times P$. Only consider the time of the attention operation, not the time to produce queries, keys, and values. Queries, keys, and values are each size D.

Solution: We have $(H/P) \times (W/P)$ patches, so the complexity is $O(\text{seq}^2 D) = O((HW/P^2)^2 D)$.

(d) The receptive field of a neuron in a CNN is the subset of all input entries that can affect what the neuron's activation is. That is, if you change any pixel outside the receptive field, then the neuron's activation would stay the same, but if you change any pixel inside the receptive field, then the neuron's activation can change. This is shown in Figure 3.

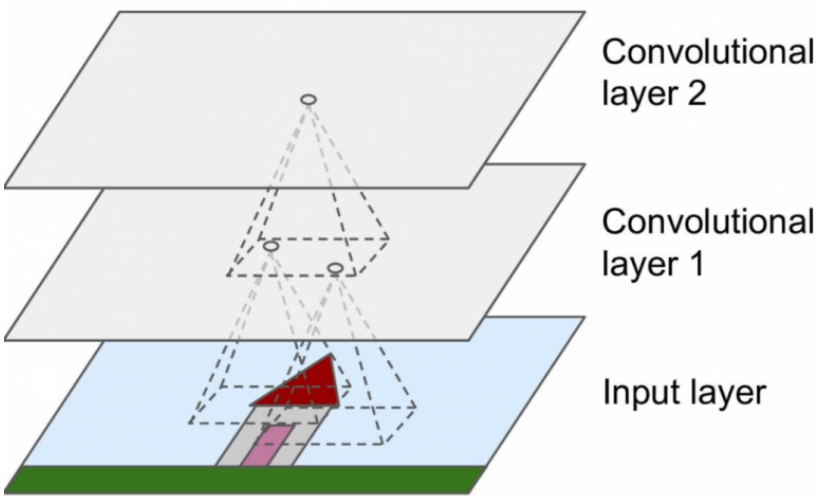

Figure 2: Receptive field of a neuron in a CNN.

What is the receptive field of one sequence item after the first layer of the transformer? How does this compare to a CNN, and what are the pros and cons of this?

Solution: After a single transformer layer, the receptive field is the entire image. In contrast, a CNN's receptive field grows slowly through the layers. This change allows transformers to pick up on spatially distant patterns more easily. However, the CNN's inductive bias toward spatially local patterns can be helpful for learning, especially when training from scratch on a small dataset. Some transformers (like Swin Transformer) have also explored only attending to spatially-local patches in early layers of the transformer to recover this inductive bias.

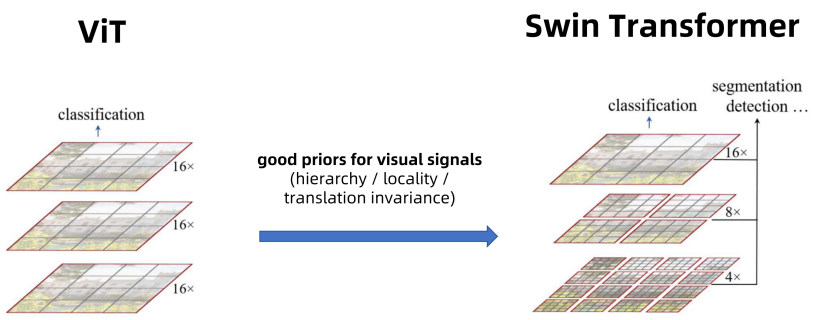

Figure 3: In the Swin Transformer (Swin = "Sliding WINdow"), there are attention mechanisms that process rectangular sub-regions ("sliding window") of the image. This enforces locality, like a convolution.

(e) If we forgot to include positional encodings, could the model learn anything at all? State one task where a model could perform well without positional encoding, and one task where it would do poorly.

Solution: The model would see the image as a bag of patches (similar to a bag of words in NLP). It could still perform well on tasks where patches are sufficient to understand the content of the image (e.g. classifying trafic lights as red, yellow, or green), but it would perform poorly on patches that rely on the order of patches (e.g. reading text from a screen).

Bag of visual words used to be a common technique in computer vision before 2010. It is still used as an extremely low-cost baseline.

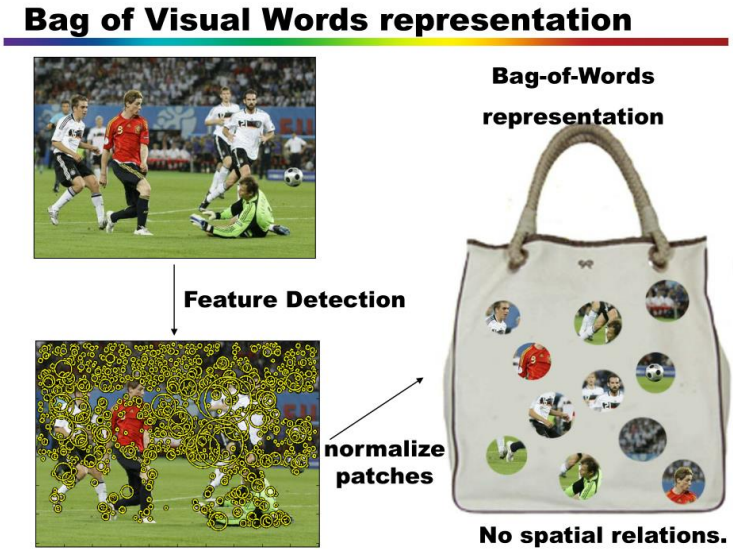

Figure 4: Bag of Visual Words

(f) (Pooling) The original ViT described above is almost exactly the same as BERT. In particular, each image is divided into 16×16 "words" [1], and a single special token `<CLS>` is appended to the start. This then allows us to use a small layernorm-linear-softmax module on the output vector of `<CLS>` to predict the probability of the given image being in the categories. Equivalently, we can remove the softmax to predict the logits.

[1] Thus the title "An Image is Worth 16x16 Words"

More recently, it has been shown that the special token is unnecessary, and better performance can be achieved by simply pooling the output vectors of the tokens. Let $x_1, \ldots, x_{256}$ be the 256 input image patches, and let $y_1, \ldots, y_{256}$ be the 256 output vectors. The task of pooling is to compute a single y from the 256 output vectors.

- Global Average Pooling (GAP) is essentially the same as average pooling in convolutional networks. What is y if we use GAP?
- Multiheaded Attention Pooling (MAP) is essentially using a query vector to "query" the output vectors in an attention mechanism. Let the query vector be q. What is y? What are the learned parameters? What should be the type of attention?

Solution: For GAP, $y = \frac{1}{256}\sum_{i=1}^{256} y_i$.
For MAP, the type of attention mechanism is a cross-attention mechanism, with no masking. It should have the form

$$y = \text{MultiheadAttention}(q, Y, Y)$$

where q is the query vector, and Y is the matrix with rows being $y_1, \ldots, y_{256}$.
The trainable parameters are q, as well as the weight matrices inside the multihead attention mechanism.

(g) So far the ViT training procedure has been fully-supervised. Yet we know that most of the available data are unlabelled. How can we do BERT-style self-supervised **representation** learning with vision transformers?

- *Hint 1*: Think about how BERT is trained. How should the input image be modified? What should be the target?
- *Hint 2*: ViT in this question only has an encoder. For BERT-style training, you will need a decoder. Why is this the case?

Solution:
This question also does not have a unique solution. Instead, the following solution provides an overview of proven effective methods in past research.

BERT is able to obtain useful representations for NLP tasks using two pretext tasks: (1) mask prediction (2) next sentence prediction, with the mask prediction objective being the more effective one. In vision transformer, the analog of mask prediction will be **predicting the masked patches**.

Specifically, we can mask out, say, 50% of the input patches and treat the masked image as input to our model. The model will be trained to predict the masked region of the original image, by optimizing a reconstruction loss (e.g. mean squared error).

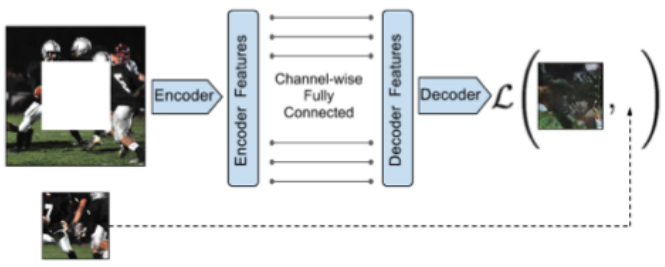

Figure 5: Diagram of Context Encoder

This objective of predicting masked patches or pixels has a techincal name called **inpainting**, and the aforementioned procedure is a simplified version of **Context Encoder** (Pathak et al), which takes in a masked image (with masked pixels filled with mean value of the image) and predicts the missing region. While Context Encoder was an effective self-supervised representation learning method at the time, there's a design choice that adversely impact its performance: **mask tokens should not be fed into the image encoder**.

The intuition behind why mask tokens should not be fed into image encoder is that at test-time, real images do not contain such a huge portion of mask values. This creates a huge distribution difference between train- and test-time. To learn useful representations, a better design would be to only feed the encoder *real* patches and then append the mask tokens / values right before the decoder input for image reconstruction.

One way to do this would be via **image inpainting**. Specifically, we set the pixels that we wish to mask to a constant value (e.g., zero). A downside of this approach is that it requires spending compute on masked out regions, which can be large and contain little useful information. A better design would be use an *asymmetric* encoder-decoder model and only feed the *visible* patches to the encoder and include the mask tokens as the decoder inputs for image reconstruction.

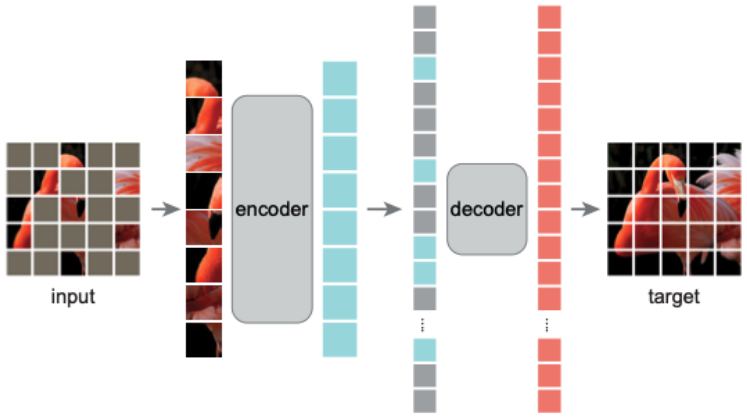

Figure 6: Diagram of Masked Autoencoder

This is the idea of Mask Autoencoder or MAE[2]. MAE uses a Vision Transformer encoder-decoder structure, which takes in an image, patchify the image, randomly mask out 75% of the patches, pass the un-masked patches into the encoder, and lastly append the mask tokens right before the decoder input.

(h) (Bonus) If you wanted to add a few convolutional layers into ViT, how would you incorporate them?

Solution: There's no one unique solution to this. One way is to first apply a few conv layers to the input image, and then split the resulting feature map into patches for the rest of the ViT operations[3]. Research found that applying convolution for early "patchify" layer leads to more robust

[2]He, Kaiming, et al. "Masked autoencoders are scalable vision learners." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2022.

[3]Xiao, Tete, et al. "Early convolutions help transformers see better." Advances in neural information processing systems 34 (2021): 30392-30400.

ViT training. Another way is to apply a small convolution filter after each ViT block to "reduce the spatial dimension" of the feature map[4], which can be thought of as intentionally adding a *locality bias* regularly after every few self-attention layers, which have low inductive bias and a receptive field of global.

(i) (Bonus) How would you use ViT to do GPT-style autoregressive generation of images?

Solution: You would need to pass the transformer's output into a decoder which can generate image patches. Like with language, you could then feed the generated patch back into the model to generate the next patch. Train with MSE loss against the ground-truth patches. (Or cross-entropy loss if you're using discrete tokens, e.g. Parti `https://parti.research.google/`. I don't think discrete tokenization e.g. VQVAE has been covered yet.)

[4]Xie, Enze, et al. "SegFormer: Simple and efficient design for semantic segmentation with transformers." Advances in Neural Information Processing Systems 34 (2021): 12077-12090.

