EEEM068: Interpreting classic Vision Transformers

Nam Tran

Abstract

Other than looking at accuracy, Viewing the weight allocation of CNN is one of the most crucial aspects to quantitively assess a pretrained model's performance for a given task and dataset where the neural network architecture and learning (i.e., Representation Learning, Meta Learning) all play major parts to the overall performance. Same principle applies for Vision Transformer (ViT). To view weight allocation of ViT, a well-known baseline technique in ViT community is Attention Rollout [1]. *In this report, I reimplement the technique and discussed its incompatibility and relevance in assessing modern ViTs.*

1. Introduction

Attention Rollout works as follows: As the input image forward pass ViT, [1] obtained the output of each attention layer within each multi-head within each encoder block whenever the input image pass through that layer; then for each multi-head [1] extracted the most attend layer; then averaged all most-attended layer to obtain a mask, which is used as a heatmap (visualization for weight allocation) to overlay on the original input.

1.1. Reimplementation method

1. Provided with hook function[1], I use it to obtain the score of each attention layer and create the Rollout function to compute the mask from the obtained outputs, code provided here[2].
2. With the output mask of Attention Rollout, I rescale a single sample from DataLoader and create the heatmap function to overlay the mask on the input (see provided code).

Hook by itself cannot hook on Vision Transformer as the latter is a nested module. As given a module as input, hook can only hook on its nearest child module. Fortunately, the line of code *for name, m in self.model.named_modules()* can get a chain nested module to the desire module as string like a file directory. So, I can use *name.split('.')* and a loop to hook into encoder block. With hook, I can get 3D-score tensor for each head per encoder for all encoders in ViT stored *self.outputs[]* from a single forward pass. *self.outputs[]* = input for the Attention Rollout.

Computing the mask is explained in-depth in section 2.1. Here I only pinpoint that the mean of max score across all heads is done via *torch.max* and *mean = attention / attention.sum(dim=-1, keepdim=True)*. Then there are 1 necessary add-on: Not every multi-head will score tensor. So, I rescale the values for score tensor for each multi-head with *a = (attention_heads_fused + 1*I)/2* where I is an identity matrix.

The heatmap is itself is self-explanatory as it is just a matter of converting $[Tokens, Tokens]$ into $[Width, Height, 3]$ numpy array with the line applyColorMap to overlay on the input image as numpy array. The reason to use numpy array is for *plt.show* or *cv2.show*. Because applyColorMap only produces $[Width, Height, 3]$, a single input sample that has the shape $[1,3, Width, Height]$ via unsqueeze line to forward pass through ViT needs to change to $[Width, Height, 3]$ to match with the mask.

2. Experiments

The initial plan to is to experiment Attention Rollout with vit-base-16, vit-large-32, vit-huge-14 for 10 epochs to test accuracy w.r.t attention score w.r.t patch size, number of head, number of encoders of each ViT. However, because the RAM provided by Colab is insufficient for Colab Pro and Colab Pro++ is too expensive to pay for a coursework, vit-huge-14 is discarded.

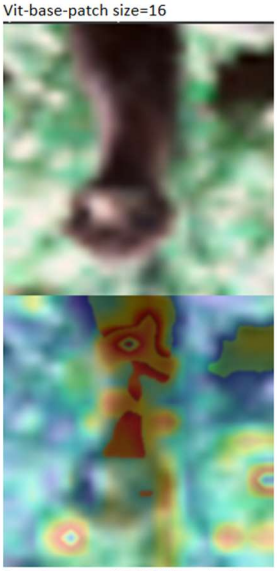
My routine to load pretrained ViT from PyTorch is as follows: First, create a ViT model that has the exact same number of classes as the pretrained ViT (num_classes=1000) just to load pretrained weights onto newly created ViT. Then, replace linear classifier head of ViT with a new linear classifier head where output = number of classes of finetuning dataset instead of 1000. Given the rest of my finetuning pipeline is just choosing the right optimizer, lr and lr-scheduler which I've followed

[1] https://gitlab.surrey.ac.uk/jw02425/vitoolkit

[2] https://colab.research.google.com/drive/1Yb1VE6DpwGhQgvg7aXKIyog6BxXPMJ0h?usp=drive_link

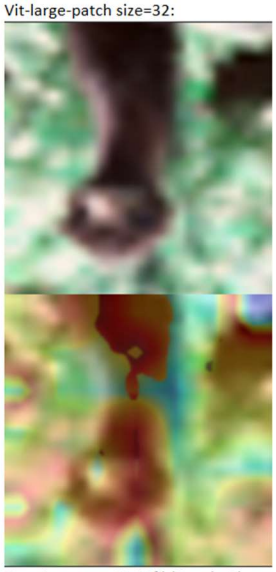
from [2] as their new training recipe has given me very good results when finetuning ResNet50 for 10 epochs. So, all in all, the test accuracies have been very baffling (see snippets below), and I leave it for the markers to give me feedback on what I've done wrong provided the polished code.

CIFAR10 (50k samples)
Vit-base-patch size=16



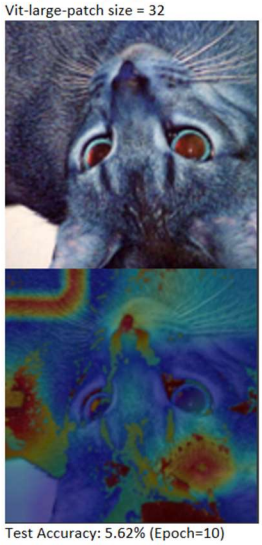
Test Accuracy: 24.83% (Epoch=5)

CIFAR10 (50k samples)
Vit-large-patch size=32:



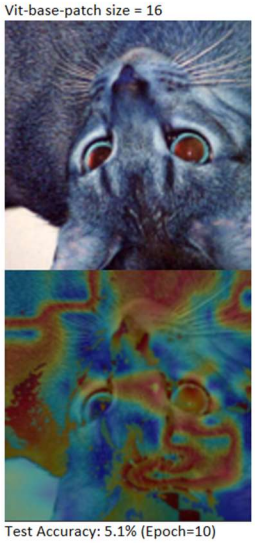
Test Accuracy: 42.01% (Epoch=5)

Pets (3680 samples)
Vit-large-patch size = 32



Test Accuracy: 5.62% (Epoch=10)

Pets
Vit-base-patch size = 16



Test Accuracy: 5.1% (Epoch=10)

Flowers102(1020 samples)
Vit-base-patch size=16

Test Accuracy: 12.50% (Epoch=10)

Flowers102
Vit-large-patch size=32

Test Accuracy: 9.16% (Epoch=10)

3. Discussion

Despite the failure to get decent accuracy, there is at least a correlation between having a very hot heatmap and accuracy, and another correlation between accuracy and larger patch size, more heads, more encoder block which supports the scaling rule in transformer. But given that CIFAR10 gives significantly better accuracy for having significantly larger dataset size, can all the blame for bad accuracies be attributed to the lack of samples in Pets and Flowers? Indeed, more experiments are needed to answer this question but I have ran out compute unit on Colab.

3.1. Self-Attention ViTs

Let's delve down to how Self-Attention deals with batch input $[Batch, Tokens, Features]$ because it helps when explaining why Attention Rollout function doesn't work on other ViTs. The 2nd line of def forward of Self-Attention does the following thing, first it puts batch input through a linear layer (learnable weights for Q, K, V in Self-Attention) to output $[Batch, Tokens, 3 \times Features]$, then $[Batch, Tokens, 3 \times Features]$ is reshaped into $[Batch, Tokens, 3, Heads, Head\ dim]$
where Head dim = Features / Heads.

Then permute $[Batch, Tokens, 3, Heads, Head\ dim]$ into $[3, Batch, Heads, Tokens, Head\ dim]$ where Q, K, V each get $[Batch, Heads, Tokens, Head\ dim]$ by index. Computing the score for Q, K with following formula get $[Batch, Heads, Tokens, Tokens]$

$$Score = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

for each multi-head per encoder block.

For each multi-head per encoder block, the Attention Rollout first takes maximum $[Batch, Tokens, Tokens]$, tensor score across all heads, then Identity matrix $[Tokens, Tokens]$ is created from this output and adds to $[Batch, Tokens, Tokens]$ via PyTorch broadcasting to rescale the score for reason as discussed in section 1.1. This output multiplies before being squeezed and downsized to mask $[Tokens, Tokens]$ **which has the same shape as score matrix of each attention layer because the goal of Attention Rollout is to compute the cumulative effect of multiple layers of attention in a deep transformer network.**

3.2. Hydra-Attention ViT

Hydra Attention [3] is one of many recent attempts to achieve linear complexity for ViT while surpass or retain competitive accuracy comparing to the based model [3]. Given a batch input $[Batch, Tokens, Features]$, Hydra Attention works as follows:

Q, K, V are equally split along the Features dimension from batch input to have $\left[Batch, Tokens, \frac{Features}{3}\right]$. Then element-wise multiply K, V to get kv tensor of shape $\left[Batch, Tokens, \frac{Features}{3}\right]$, then element-wise multiply of q and kv to get output tensor of shape $\left[Batch, Tokens, \frac{Features}{3}\right]$.

As there is no Head dimension, Attention Rollout is incompatible. But assuming we don't take the max values across all heads, Attention Rollout still wouldn't work because squeezing and downsizing of (mean of max score across all heads) was implemented in the original paper to explicitly work with score matrix, which Hydra Attention never has so forcefully viewed the weight allocation of Hydra Attention with Attention Rollout only resulting unintelligible heatmap.

These two arguments apply to other recent ViTs as well whenever Attention Rollout cannot work meaningfully with them.

3.3. Weight dynamic across the channels

Self-Attention aims to make each entry in a tensor influenced by every other entry in the same tensor in head, yet every multi-head uses the same linear layers whereas Convolution sweeps [3x3xchannel] tensor across the input (see figure 1 below).

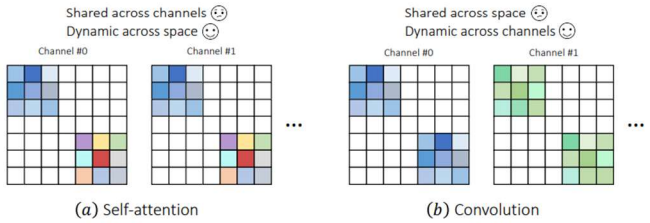


Figure 1 from [4]: Weight allocation in Self-Attention and Conv. Different colors denote different weights. In self-attention, the weight matrix is dynamic (varied) across space (height and width) but shared across different channels thanks to the linear layer. In contrast, the weight matrix of convolution is shared across space (thanks to Conv filters) but varies in channels thanks to having many Conv filters for every Conv layer.

Given the recent research trend to combine the benefits of Conv and Self-Attention, and [4] has demonstrated the significant advantage (see their table 1,2) and in varying weights along channel-wise and space-wise in designing U-Net for medical image segmentation task (see figure 2):

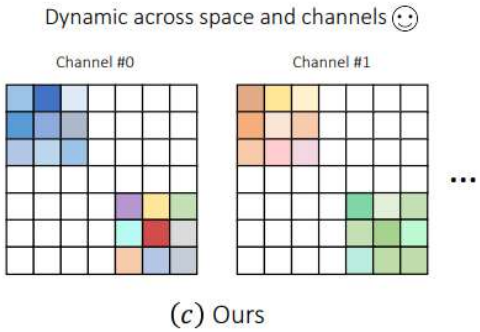


Figure 2 from [4] which the authors integrate the advantages of self-attention and convolution by assigning dynamic weights to different positions and channels.

Assuming future research will follow this line of thinking because the weight allocations of Self-Attention and Convolution are fundamentally as illustrated like in figure 1 so any attempt to combine the features between the twos should follow this line of thinking, I'd argue that the procedure of Attention Rollout which collapses the "channel" dimension by finding the most attended head in head multi-head as well as averaging all multi-head together to create the mask dismiss the opportunity to view the dynamic along the "channel". Therefore future Attention Rollout must be able to view the dynamic space-wise and channel-wise by either having a 3D mask or having 2 views of 2D mask: One for space-wise and one for channel-wise.

4. Conclusion

Despite failing to achieve desired baseline finetuning accuracy, my reimplementation of the Attention Rollout is

good enough to identify the scaling trend in designing Transformer model and I have identified the motivation to design a 3D Attention Rollout to accommodate the recent trend of combing Conv with Self-Attention.

5. References

[1] S. Abnar and W. Zuidema, "Quantifying Attention Flow in Transformers," 2 May 2020. [Online]. Available: https://arxiv.org/abs/2005.00928.

[2] PyTorch, "How to Train State-Of-The-Art Models Using TorchVision's Latest Primitives," 18 November 2021. [Online]. Available: https://pytorch.org/blog/how-to-train-state-of-the-art-models-using-torchvision-latest-primitives/.

[3] D. Bolya, C.-Y. Fu, X. Dai, P. Zhang and J. Hoffman, "Hydra Attention: Efficient Attention with Many Heads," 15 September 2022. [Online]. Available: https://arxiv.org/abs/2209.07484.

[4] J. Guo, H.-Y. Zhou, L. Wang and Y. Yu, "UNet-2022: Exploring Dynamics in Non-isomorphic Architecture," 27 October 2022. [Online]. Available: https://arxiv.org/abs/2210.15566.