I had started a while an article study about the Transformers in the paper Attention is all you need for NLP tasks; I was going to move on to VIT models and eventually toward the SWIN Transformers.

The state of art studying process is rather straight forward. From trying to understand the intuition behind the architecture, to understand how we get from input to output. How best can we utilize this architecture? To figure this out we also check for GitHub repo, available public weights and dataset used to.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Git | Weights | dataset | Params | Performance |
| available | available | available | - | - |

The model is a little on the heavy side for embedded applications, but there have been some detection models that integrated the SWIN transformers into their backbone. We d have to review those models to test their suitability on embedded systems.

**I AN IMAGE IS WORTH 16X16 WORDS: Transformer for image recognition at Scale**  
The default approach for Transformers that consists of representing each token from the input with a linear embedding to feed them through the attention layer was tested as is firsthand by using image patches (image split into a grid) as tokens, such test didn’t yield encouraging results due to inability of transformers to mimic CNNs, because VIT doesn’t handle image straight to it has less inductive bias. In CNNs, locality, two-dimensional neighborhood structure, and translation equivariance are baked into each layer throughout the whole model, but it is not the case in VITs which will be illustrated below.

Une image contenant diagramme

Description générée automatiquementHowever, this was overcome by supplying more data for training, which is a little inconvenient in the cases where one wants to make a fresh model from scratch for a certain use case due to usually lack of data or resource to procure data.

We review how attention works. Transformers essentially are attention layers stacked on top of each other, it is split into 2 parts encoders and decoders, and we take interest in the encoder of course (feature extraction).   
In NLP, for token in the input which are limited to 500 token tops at once, a linear embedding vector is generated using the positional embedding (regular embedding vector + position in the input vector). In vision such linear vector is generated through a flatten layer, so each image patch is flattened into a single vector.

Une image contenant texte, horloge, capture d’écran, graphique vectoriel

Description générée automatiquement

So, post this stage the attention calculation is similar. The first step is to create 3 vectors query, key and value for each linear vector received from the input. The vectors are created using the dot product between the linear vector and the weight matrices. Notice that there are 7 different matrices, each one is what is known a head of attention thus the famous deep learning layer multi headed attention. So, these query key and value vectors is calculated in every head. Within each head, we have Q, K and V vectors for each patch from the input for which we need to calculate attention now, how? for each patch’s query vector we calculate dot product with all other K vectors (including patch s K vector in question) the result which should be scalar is divided by square root of K’s size (64 by default) and fed to a SoftMax function. The SoftMax values determine how much the patch t is expressed in each corresponding patch from the sequence, by multiplying it by its value vector allows to conserve the words with the highest SoftMax score and draw out the ones with the lowest scores. In such manner attention matrix for each patch are generated and stacked. As there is a mismatch between the stacked matrices and the input for the next layer Feed forward, a weight (WO) matrix is added for a dot product. This WO is trained alongside the model.

Une image contenant diagramme

Description générée automatiquementWe have gotten through how attention layers work. We circle back to ViTs which is almost identical in process, difference lies in the initial embedding input and of course it is not a sequence-to-sequence architecture although I believe it can be done. One proposed variant consists of swapping out the flatten layers with a CNNs to extract feature vectors for each patch (mainly tested extracted was ResNet).

|  |  |  |
| --- | --- | --- |
| Model | Layers | Params (m) |
| ViT-Base | 12 | 86 |
| ViT-Large | 24 | 307 |
| ViT-Huge | 32 | 632 |