Enhancing Vision Transformers Through Recursive Tansformers inside Transformers

Abstract

Transformer-based models have recently revolutionized image classification by leveraging self-attention mechanisms. However, many of these approaches rely on a sequence of fixed-size independent patch division, which often lacks the ability to represent local spatial relations essential for precise image understanding. This project aims to re-implement and extend the Transformer in Transformer (TNT) architecture—an approach that addresses these limitations through a nested framework—by further refining local feature extraction. Our proposed extension, referred to as RTNT, explores recursive transformers in transformers as a means to capture even finer spatial relationships without incurring prohibitive data or computational costs and/or overfitting. Through these modifications, we seek to enhance the overall robustness of vision transformers, making them more effective and practical for diverse computer vision tasks, such as medical imaging, autonomous driving, etc. However, experimental results proved that this architecture even though good in theory, it's not better than the original TNT architecture.

https://anonymous.4open.science/r/ece570projectd/README.md

1 Introduction

Vision transformers have rapidly emerged as a viable alternative to convolutional neural networks in image recognition tasks, yet their ability to capture local spatial details remains a persistent challenge as they lack inductive bias present in traditional CNNs. Traditional Vision transformers typically treat images as sequences of local patch divisions, then attention can be calculated normally between any two image patches, inadvertently sacrificing spatial relationships that are crucial for precise visual understanding. Recent advances, such as the Transformer-in-Transformer (TNT) architecture (Han et al., 2021), have sought to mitigate this limitation by embedding an inner transformer module within the conventional framework, further dividing patches into smaller sub-patch, then attention of each sub-patch will be calculated with other sub-patch within the given patch, hence preserving local structure and spatial information of the image.

In this work, we propose an extension to the TNT model—termed RTNT—that adopts a recursive scheme. By iteratively subdividing image patches into increasingly finer sub-patches and applying localized self-attention at each level. Our objective is to enhance the extraction of nuanced local features without imposing significant additional computational burdens or data requirements. This recursive refinement may has the potential to not only improve classification accuracy but also to bolster the model's applicability to tasks demanding detailed spatial analysis, such as medical imaging and autonomous driving. The end goal aims to further bridge the gap between the global modeling strength of transformers and the fine-grained local sensitivity traditionally offered by CNNs, offering a scalable approach for vision tasks that demand high spatial precision.

2 Related Works

Transformer-based architectures have revolutionized natural language processing, which motivated many researchers to try and adopt the transformer model in computer vision tasks as an alternative to traditional models, such as conventional CNNs. For example, Carion et al. (2020) introduce DETR, which deals with object detection as a direct set prediction task solved via a transformer encoder-decoder architecture. In a similar vein, Chen et al. (2020) propose iGPT, the first work to employ a pure transformer model (i.e., without convolution) for image recognition through self-supervised pre-training. This models, while theoretically efficient, have not yet been scaled effectively on modern hardware accelerators due to the use of specialized attention patterns.

Unlike data in NLP, there is a semantic gap between input images and ground-truth lables in Computer

Vision tasks. Consequently, Dosovitskiy et al. (2021) develop ViT, which paves the way for transferring the success of transformer-based NLP models. A ViT treats an image as a sequence of patches; the image is divided into several local patches (typically 16x16 pixels) as visual sentences; each of which is flattened into a vector and linearly projected into a high-dimensional embedding space. Then, the attention can be calculated normally between any two image patches; with fewest possible modifications to the Transformer architecture, which makes it simple and scalable (unlike the previous models which could not scale well). A key aspect of Vit is that it eliminates the inductive biases inherent in CNNs, so the model relies solely on its self-attention mechanism to learn; which makes the model highly data hungry and lacks local features representations.

These challenges have driven research efforts toward improving both data efficiency and local feature extraction in vision transformers. Touvron et al. (2021) explore the data-efficient training and distillation (DeiT) to enhance the performance of ViT. Unlike Vit which is generally unsupervised, DeiT leverages a teacher-student training paradigm. During training, a strong teacher model provides additional supervisory signals via a distillation loss, guiding the transformer to learn more effectively on smaller datasets such as ImageNet-1K and obtain an about 81.8% ImageNet top-1 accuracy, which is comparable to that of the state-of-the-art convolutional networks. Additionally, Liu et al. (2021) introduce Swin Transformer; which enhances local feature extraction by using a shifted window mechanism that limits self-attention to local regions while still allowing information flow across windows. This creates a hierarchical feature map that captures fine-grained details efficiently. Furthermore, Han et al. (2021) explore nested Transformer iN Transformer (TNT) approach, aiming to improve local feature extraction of ViT. Specifically, it regard local patches (e.g., 16x16) as 'visual sentences' and pres to further divide them into smaller patches (e.g., 4x4) as 'visual words, The attention of each word will be calculated with other words in the given visual sentence with negligible computational costs. Features of both words and sentences will be aggregated to enhance the representation ability. Nowadays, transformer architectures have been used in a growing number of computer vision tasks (Han et al., 2022) such as image recognition, object detection , and segmentation.

Building upon these prior advancements, our work adds to the ongoing effort to address the locality gap in vision transformers. While models such as Swin Transformer and TNT have introduced mechanisms to preserve spatial hierarchies—through shifted windows or nested attention within patches—these approaches typically employ a single level of local enhancement. In contrast, we propose a recursive refinement of the TNT architecture, where local patches are subdivided repeatedly, enabling multi-level patch attention.

3 Problem Definition

The problem addressed by this project is twofold: enhancing the capture of fine-grained local spatial information in vision transformers and maintaining computational efficiency. Although transformer-based models like ViT have redefined image classification by processing images as sequences of patches; it lacks for inductive bias often leading to a loss of detailed spatial context, and does not generalize well unless trained on huge datasets hence higher computational costs.

The central research question is: **How can we refine the patch processing in transformer-based models by recursively subdividing patches—effectively implementing transformers within transformers—to capture even finer local and spatial details while preserving data and computational efficiency and mitigating overfitting?**

This work tackles two interrelated challenges:

1. **Local Feature Extraction:** Investigating how recursive subdivision of patches can enhance the model's ability to capture nuanced spatial dependencies that are otherwise lost in a sequence of independent patches framework.
2. **Computational Efficiency:** Ensuring that improvements in local feature extraction do not come at the cost of increased computational cost, overfitting, data requirements, thereby preserving the model's ability to perform well even on smaller datasets.

4 Methodology

In model design and methodology, we follow the original TNT (Han et al., 2021) as closely as possible. Anadvantage of this simple setup is that the TNT MSA and MLP layers can be used almost out of the box on all further subdivision of patches.

4.1 Preliminaries

We first describe the basic components in transformer (Vaswani et al., 2017), including MSA (Multi-head Self-Attention), MLP (Multi-Layer Perceptron) and LN (Layer Normalization)

MSA: In the self-attention module, the inputs $X \in \mathbb{R}^{n\times d}$ are linearly transformed into three parts, *i.e.*, queries $Q \in \mathbb{R}^{n\times d_k}$, keys $K \in \mathbb{R}^{n\times d_k}$, and values $V \in \mathbb{R}^{n\times d_v}$. where n is the sequence length, and d, d_k, d_v are the dimensions of inputs, queries/keys, and values, respectively. The scaled dot-product attention is computed as:

$$\text{Attention}(Q,K,V) = \text{softmax}\Big(\frac{QK^T}{\sqrt{d_k}}\Big)V. \quad (1)$$

A linear layer then produces the final output. Multi-head self-attention splits Q, K, V into h parts, computes attention in parallel, and concatenates the results:

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\ldots,\text{head}_h)W^O,$$

where W^O is a learnable projection matrix.

MLP is placed between self-attention layers for feature transformation and non-linearity:

$$\text{MLP}(X) = \text{FC}\big(\sigma(\text{FC}(X))\big), \quad \text{FC}(X) = XW + b, \quad (2)$$

where W and b are the weight and bias of a fully-connected layer, respectively, and σ is a non-linear activation function (such as GELU[]).

LN: Layer normalizations is a key component in the Transformer for stable training and faster convergence. LN is applied over each sample $x \in \mathbb{R}^d$ as follows:

$$\text{LN}(x) = \frac{x-\mu}{\sigma}\gamma + \beta, \quad (3)$$

where μ and σ denote the mean and standard deviation of x, respectively, and $\gamma, \beta \in \mathbb{R}^d$ are learnable affine transformation parameters.

4.2 Recursive Transformers

An overview of the model is depicted in Figure [1]. In this Implementation, we do three recursive sub-divisions, regarded as visual sentences, words, and letters.

Given a 2D image of three color channels, we uniformly split it into n patches:

$$\mathscr{X} = \{X^1, X^2, \ldots, X^n\} \in \mathbb{R}^{n\times p\times p\times 3}, \quad (4)$$

where $X^i \in \mathbb{R}^{p\times p\times 3}$ is the i-th *sentence* of the image, (p,p) is the resolution of each image patch and $i = 1,2,\ldots,n$.

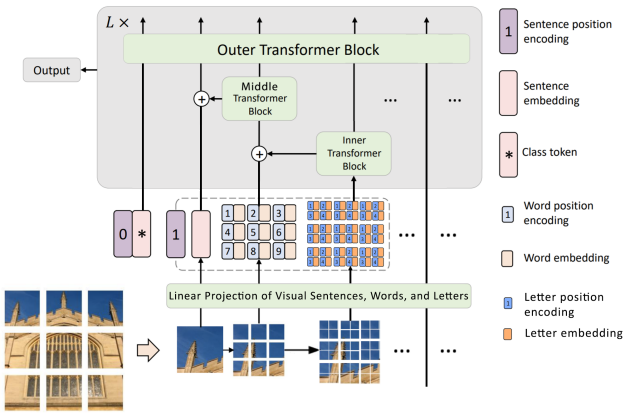

Figure 1. Model overview. The current implementation consists of three recursive sun-divisions, regarded as visual sentences, words, and letters. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" on the sentence level. The illustration was inspired by Han et al. (2021)

Each patch is further divided into m sub-patches; in other words, each visual sentence is composed of a sequence of visual words. Formally:

$$X^i = \{x^{i,1}, x^{i,2}, \ldots, x^{i,m}\}, \quad (5)$$

where $x^{i,j} \in \mathbb{R}^{s\times s\times 3}$ is the j-th *visual word* of the i-th visual sentence. Here, (s,s) is the spatial size of each sub-patch and $j = 1,2,\ldots,m$

Furthermore, Each sub-patch is divided into g divisions, i.e., each word is composed of a sequence of visual letters:

$$x^{i,j} = \{\mathbf{x}^{i,j,1}, \mathbf{x}^{i,j,2}, \ldots, \mathbf{x}^{i,j,g}\}, \quad (6)$$

where $\mathbf{x}^{i,j,k} \in \mathbb{R}^{r\times r\times 3}$ is the k-th *visual letter* of the i,j-th visual sentence. Also, (r,r) is the spatial size of each sub-patch-division and $k = 1,2,\ldots,g$. With a linear projection, we transform the visual letters into a sequence of *letter embeddings*:

$$y^{i,j} = \{\mathrm{y}^{i,j,1}, \mathrm{y}^{i,j,2}, \ldots, \mathrm{y}^{i,j,g}\} = \text{FC}\big(\text{Vec}(x^{i,j})\big), \quad (7)$$

where $\mathrm{y}_k^{i,j} \in \mathbb{R}^{di}$ is the k-th letter embedding, di is the dimension of the letter embedding, and $\text{Vec}(\cdot)$ is the vectorization operation.

In RTNT, we have three data flows:

1. One flow operates across the *visual sentences* (i.e., patch embeddings)

2. The other flow operates across the visual words within each sentence.

3. The last flow operates on the visual letters in each word.

For the letter embeddings, we utilize a transformer block to explore the relation between them:

$$y\prime_l^{i,j} = y_{l-1}^{i,j} + \mathrm{MSA}\big(\mathrm{LN}(y_{l-1}^{i,j})\big), \tag{8}$$

$$y_l^{i,j} = y\prime_l^{i,j} + \mathrm{MLP}\big(\mathrm{LN}(y\prime_l^{i,j})\big), \tag{9}$$

where $l = 1, 2, \ldots, L$ L is the index of the l-th block, and L is the total number of stacked blocks. The input of the first block $y_0^{i,j}$ is just $y^{i,j}$ in Eq. 7. All letter embeddings in the image after transformation are $\mathcal{Y}_l = \{y_l^{1,1}, y_l^{1,2}, y_l^{1,3}, \ldots, y_l^{n,m}\}$. This can be viewed as an inner transformer block, denoted as T_{in}. This process builds the relationships among visual words by computing interactions between any two visual letters. For example, in a patch of human face, a letter corresponding to the eye is more related to other letters of eyes while interacts less with forehead part.

For the word level, we create the word embedding memories to store the sequence of sentence level representations: $Y = \{\mathrm{Y}^{1,1}, \mathrm{Y}^{1,2}, \mathrm{Y}^{1,3}, \ldots, \mathrm{Y}^{n,m}\}$, and all of them are initialized as zero. In each layer, the sequence of word embeddings are transformed into the domain of sentence embedding by linear projection and added into the sentence embedding:

$$\mathrm{Y}_{l-1}^{i,j} = \mathrm{Y}_{l-1}^{i,j} + \mathrm{FC}\big(\mathrm{Vec}(y_l^{i,j})\big), \tag{10}$$

where $\mathrm{Y}^{i,j}$ and the fully-connected layer FC makes the dimension match for addition. With the above addition operation, the representation of word embedding is augmented by the letter-level features. We use the standard transformer block for transforming the word embeddings:

$$Y\prime_l^i = Y_{l-1}^i + \mathrm{MSA}\big(\mathrm{LN}(Y_{l-1}^i)\big), \tag{11}$$

$$Y_l^i = Y\prime_l^i + \mathrm{MLP}\big(\mathrm{LN}(Y\prime_l^i)\big), \tag{12}$$

All word embeddings in the image after transformation are $\mathscr{Y} = \{Y_l^1, Y_l^2, , \ldots, Y_l^n\}$. This can be viewed as a middle transformer block, denoted as T_{mid}.

For the sentence level, we create the sentence embedding memories to store the sequence of sentence level representations: $\mathscr{Z} = \{Z_{class}, Z^1, Z^2, \ldots, Z_0^n\}$ where Z_{class} is the class token similar to TNT [], and all of them are initialized as zeros. In each layer, the sequence of word embeddings are transformed into the domain of sentence embedding by linear projection and added into the sentence embedding:

$$Z_{l-1}^i = Z_{l-1}^i + \mathrm{FC}\big(\mathrm{Vec}(Y_l^i)\big), \tag{13}$$

With the above addition operation, the representation of sentence embedding is augmented by the word-level features. We use the standard transformer block for transforming the sentence embeddings:

$$\mathscr{Z}\prime_l = \mathscr{Z}_{l-1} + \mathrm{MSA}\big(\mathrm{LN}(\mathscr{Z}_{l-1})\big), \tag{14}$$

$$\mathscr{Z}_l = \mathscr{Z}\prime_l + \mathrm{MLP}\big(\mathrm{LN}(\mathscr{Z}\prime_l)\big), \tag{15}$$

This outer transformer block T_{out} is used for modeling relationships among sentence embeddings.

In summary, the inputs and outputs of the RTNT block include the visual letter embeddings, word embeddings and sentence embeddings as shown in Fig. 1, so the TNT can be formulated as

$$\mathcal{Y}_l, Y_l, \mathscr{Z}_l = RTNT(\mathcal{Y}_{l-1}, Y_{l-1}, \mathscr{Z}_{l-1}), \tag{16}$$

In our RTNT block, the inner transformer block is used to model the spatial nuanced information between visual letters, while the inner transformer block is used to model the relationship between visual words for local feature extraction, and the outer transformer block captures the intrinsic information from the sequence of sentences. By stacking the RTNT blocks for L times, we build the recursive transformer network. Finally, the classification token serves as the image representation and a fully-connected layer is applied for classification.

Positional encoding: Spatial information plays a crucial role in image recognition. To preserve this information, we add corresponding positional encoding to all sentence, word, and letter embeddings, as illustrated in Fig. 1. In this work, we use standard learnable 1D positional encoding.

5 Experiments and Results

Our objective when experimenting isn't mainly proving that the model is a state-of-art or the absolute results, but it is to see if this extension adds any value on top of the base TNT architecture; specifically, wether the inner transformer block adds any value of the middle transformer block is enough. ViTs usually are pre-trained on very huge datasets like ImageNet-21k with 21k classes and 14M images (Deng et al., 2009), and JFT (Sun et al., 2017) with 18k classes and 303M high-resolution images; then the model is fine-tuned for other small classification tasks. This require enormous computational power, bandwidth, and storage. TNT

follow the distillation approach of Touvron et al. (2021) which allows the model to be pre-trained on ILSVRC-2012 ImageNet dataset with 1k classes and 1.3M images. This is a good approach yet still not feasible in our work. This would require us to implement the whole training pipeline and distillation mechanism introduced in Deit paper. Also, it still require some form of enterprise computational power and resources not available for us. To this end, we arrived at the following:

1. Training will be done on MNIST, and FashionMNIST, and STL10 datasets. All are fairly big for training and testing yet require little computational power. the training pipeline will be normal pipeline as used in Pytorch documentation.
2. We will train our model with the inner transformer applied for all L transformer blocks, and without the it applied to any block (which would mimic the original TNT architecture). Each one will have fairly the same setup and number of parameters.

One may argue that each architecture may require its own set of hyper-parameters, etc, to show its true potential; however, We believe this is the best we can achieve with our available resources.

Model Variants. We have mainly two configurations for our model (Tiny and Nano) as summarized in Table [1]. Patch sizes could vary a bit depending on the datasets, but information loss may occur if size*size*channels is less than the dimension. Note that the Transformer's sequence length is inversely proportional to the square of the patch size, thus models with smaller patch size are computationally more expensive (Dosovitskiy et al., 2021).

Training the RTNT model on MNIST dataset, we observed that the model training loss keep decreasing then it start oscillating as visualized in figure 2, This behavior is observed for TNT model also, but more visible for RTNT. Our assumption is that at a point, the learning rate becomes high and needs to decrease to achieve good results. As a result, we implement learning rate scheduler to decay learning rate by a factor 0.96 after each 200 steps. the accuracy before implementing the change was 95.51% after five epochs, after the changes, the accuracy jumped to 96.20%, and since we are using seeds, therefore this results are likely pure improvement. Testing the RTNT model with same the inner block disabled results in 96.740% accuracy for same number of epochs, which highlight that the extension doesn't actually improve on the normal architecture.

Model	RTNT-N	RTNT-Ti
Depth	12	12
Inner transformer		
dim ci	6	12
#heads	1	2
MLP r	4	4
Middle transformer		
dim c	24	48
#heads	2	4
MLP r	4	4
Outer transformer		
dim d	96	192
#heads	3	3
MLP r	4	4
Params (M)	1.57	6.2

Table 1. Configurations of RTNT model variants.

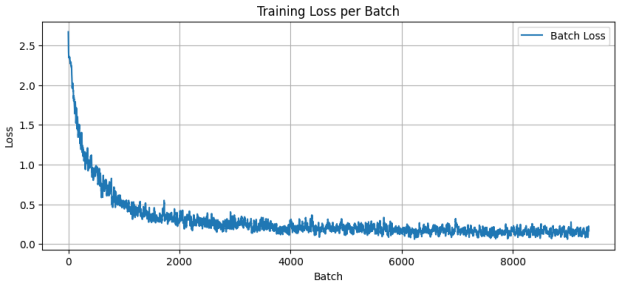

Figure 2. Training loss fluctuation of RTNT model after smoothing

Contributions

In this project, we reimplemented the Transformer in Transformer (TNT) model using PyTorch and applied it to the MNIST dataset. The implementation is organized in a modular and readable way, with clear separation between core components such as patch embedding and the inner and outer transformer blocks. Beyond the original design, we extended the architecture by introducing a recursive Transformer-in-Transformer structure, enhancing the model's ability to capture hierarchical representations. We also developed custom training and evaluation loops with support for learning rate schedulers, device management, and training curve smoothing. The project reflects a strong understanding of the TNT architecture along with thoughtful extensions for further experimentation and analysis.

LLM Acknowledgment

In the preparation of this work, a LLM was used as a supplementary tool under the guidance of the course syllabus' acceptable AI/LLM usage policy. Specifically, the LLM assisted with:

- **Technical formatting**: Ensuring proper LaTeX syntax for equations, document structure, and cross-referencing.
- **Draft ideation**: Generating preliminary phrasing and organizational suggestions for sections.

The LLM was not involved in conducting research, formulating conclusions, or contributing original ideas. All analytical work, critical reasoning, and final interpretations remain entirely my own.

etc.

$$y^{i,j} \leftarrow y^{i,j} + E_{\text{pos}},$$

References

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. End-to-end object detection with transformers. In *European conference on computer vision*, pp. 213–229. Springer, 2020.

Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Luan, D., and Sutskever, I. Generative pretraining from pixels. In *International conference on machine learning*, pp. 1691–1703. PMLR, 2020.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Han, K., Xiao, A., Wu, E., Guo, J., Xu, C., and Wang, Y. Transformer in transformer. In *Advances in Neural Information Processing Systems*, volume 34, pp. 15908–15919. Curran Associates, Inc., 2021.

Han, K., Wang, Y., Chen, H., Chen, X., Guo, J., Liu, Z., Tang, Y., Xiao, A., Xu, C., Xu, Y., et al. A survey on vision transformer. *IEEE transactions on pattern analysis and machine intelligence*, 45(1):87–110, 2022.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 10012–10022, 2021.

Sun, C., Shrivastava, A., Singh, S., and Gupta, A. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pp. 843–852, 2017.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jegou, H. Training data-efficient image transformers & distillation through attention. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10347–10357. PMLR, 18–24 Jul 2021.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

()