TinyViT: A Small Vision Transformer for Image Classification

Lucian Dorin Crainic
Department of Computer Science
crainic.1938430@studenti.uniroma1.it

La Sapienza University of Rome

Abstract

Vision Transformers (ViTs) excel in image classification but require large datasets and complex architectures. We introduce TinyViT, a minimalist ViT that retains core components patch embedding and attention while drastically reducing scale. By simplifying token processing and prioritizing parameter efficiency, we test whether a tiny ViT can achieve good accuracy despite limited complexity. Experiments on standard benchmarks show TinyViT delivers acceptable results even with reduced data requirements, challenging the assumption that ViTs inherently demand heavy resources. This work demonstrates that minimalist transformer architectures can learn meaningful representations, offering a pathway to simpler, more accessible models without sacrificing core functionality.

1 Introduction

Transformers, introduced by [Vaswani, 2017] for natural language processing (NLP), have become the dominant architecture for sequence modeling due to their scalability and self-attention mechanisms. Inspired by their success in NLP, [Alexey, 2020] pioneered the Vision Transformer (ViT), demonstrating that transformers can achieve state-of-the-art results in image recognition by treating images as sequences of patch tokens. By splitting an image into fixed-size patches, linearly embedding them, and processing the sequence with a standard transformer encoder, ViT outperformed convolutional neural networks (CNNs) [He et al., 2016] on large-scale datasets like ImageNet when pretrained on massive datasets (e.g., JFT-300M). However, ViT's strong performance comes at a cost: it requires extensive computational resources and large pretraining datasets, raising practical barriers for adoption in settings where such infrastructure is unavailable.

In this work, we aim to (1) elucidate the foundational mechanics of Vision Transformers and (2) present TinyViT, a minimalist implementation designed to test the viability of ViTs in simplified, resource-efficient settings. Unlike the original ViT, which emphasizes scaling to massive datasets, TinyViT reduces architectural complexity employing fewer transformer layers, smaller embedding dimensions, and streamlined attention mechanisms while retaining the core principles of patch-based processing and self-attention. We evaluate TinyViT on widely adopted benchmarks like CIFAR-10 and CIFAR-100 [Krizhevsky et al., 2009], and STL-10 [Coates et al., 2011].

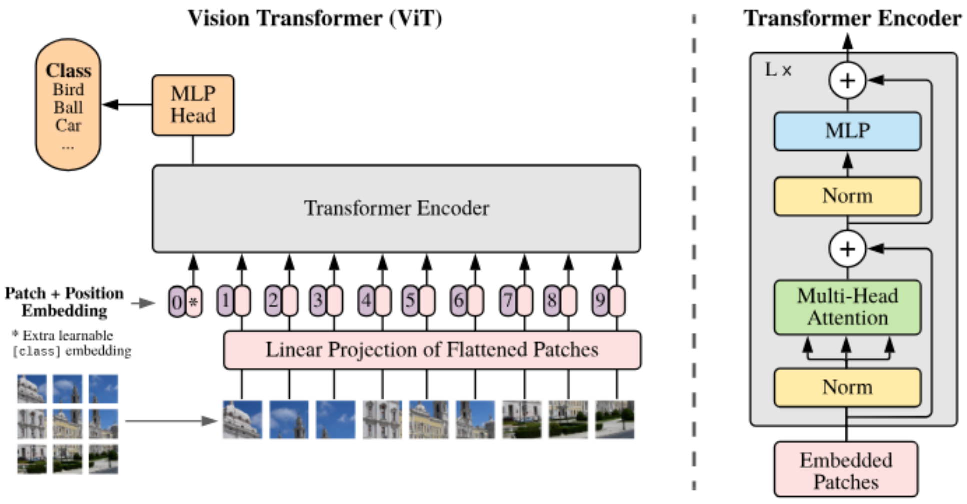

Figure 1: The Vision Transformer (ViT) architecture from [Alexey, 2020]. On the left the model architecture while on the right the Transformer encoder block architecture from [Vaswani, 2017].

2 Architecture

In recent years, the Transformer architecture originally developed for natural language processing has been successfully applied to computer vision tasks. The Vision Transformer (ViT) rethinks image classification by treating an image as a sequence of patches, much like tokens in a sentence, and processes them using standard Transformer blocks. In this chapter, we detail the architecture of the Vision Transformer and provide the mathematical foundations underlying its design.

2.1 Overview

The Vision Transformer (ViT) adapts the Transformer architecture originally developed for natural language processing to image classification tasks. Instead of processing an image as a whole, ViT divides it into a sequence of patches (analogous to tokens) and processes these patches through a stack of Transformer encoder blocks. The key steps in the ViT pipeline are:

1. **Image Patch Embedding**: The input image is divided into fixed-size patches that are flattened and projected into a latent space.
2. **Positional Encoding**: Since Transformers are permutation-invariant, positional embeddings are added to the patch embeddings.
3. **Transformer Encoder**: A series of Transformer encoder blocks processes the sequence of embeddings using self-attention and feed-forward networks.
4. **Classification Head**: The encoder output is used by a classification head for the final prediction.

2.2 Image Patch Embedding

Patch Extraction

Given an input image

$$\mathbf{X} \in \mathbb{R}^{H\times W\times C}, \tag{1}$$

where H and W denote the height and width of the image, and C denotes the number of channels, the image is divided into N patches of size $P \times P$. Hence, the number of patches is:

$$N = \frac{HW}{P^2}. \tag{2}$$

Flattening and Linear Projection

Each patch, denoted as $\mathbf{x}_p \in \mathbb{R}^{P\times P\times C}$, is flattened into a vector of dimension $P^2 \cdot C$. A learnable linear projection (implemented as a fully connected layer) maps this vector into a D-dimensional embedding space:

$$\mathbf{z}_p = \mathbf{W}_e \mathbf{x}_p + \mathbf{b}_e \tag{3}$$

where $\mathbf{W}_e \in \mathbb{R}^{D \times (P^2 \cdot C)}$ and $\mathbf{b}_e \in \mathbb{R}^D$.

After processing all patches, we obtain a sequence of embeddings:

$$\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_N] \in \mathbb{R}^{N \times D}. \tag{4}$$

2.3 Positional Encoding

Transformers do not inherently capture the order or spatial structure of input data. To inject spatial information, learnable positional embeddings are added to the patch embeddings. Let:

$$\mathbf{E}_{\text{pos}} \in \mathbb{R}^{N \times D} \tag{5}$$

be a set of positional embeddings. The combined input to the Transformer encoder is then:

$$\mathbf{Z}_0 = \mathbf{Z} + \mathbf{E}_{\text{pos}}. \tag{6}$$

In some implementations, a special class token $\mathbf{z}_{\text{cls}} \in \mathbb{R}^D$ is prepended to the sequence:

$$\mathbf{Z}_0 = [\mathbf{z}_{\text{cls}}, \mathbf{z}_1, \ldots, \mathbf{z}_N]. \tag{7}$$

2.4 Transformer Encoder Blocks

The Vision Transformer uses a stack of L Transformer encoder blocks. Each block is composed of two main components: the Multi-Head Self-Attention (MHSA) mechanism and a Feed-Forward Network (FFN). Layer normalization (LN) and residual connections are applied around each component.

Multi-Head Self-Attention (MHSA)

Let $\mathbf{Z}^{(l-1)} \in \mathbb{R}^{M \times D}$ be the input to the l-th encoder block, where M is the number of tokens (including the class token if used).

Query, Key, and Value

The input is projected into query ($\mathbf{Q}$), key ($\mathbf{K}$), and value ($\mathbf{V}$) matrices using learnable projection matrices $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{D \times D}$:

$$\mathbf{Q} = \mathbf{Z}^{(l-1)} \mathbf{W}_Q, \quad \mathbf{K} = \mathbf{Z}^{(l-1)} \mathbf{W}_K, \quad \mathbf{V} = \mathbf{Z}^{(l-1)} \mathbf{W}_V. \tag{8}$$

For multi-head attention, these are divided into h heads. For head i ($i = 1, \ldots, h$), we have:

$$\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i \in \mathbb{R}^{M \times d}, \quad \text{with } d = \frac{D}{h}. \tag{9}$$

Scaled Dot-Product Attention

For head i, the attention is computed as:

$$\text{Attention}_i(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i) = \text{softmax}\left(\frac{\mathbf{Q}_i \mathbf{K}_i^\top}{\sqrt{d}}\right) \mathbf{V}_i.$$

The scaling by $\sqrt{d}$ ensures numerical stability.

Concatenation and Final Projection

The outputs from all h heads are concatenated and projected:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}\left(\text{Attention}_1, \ldots, \text{Attention}_h\right) \mathbf{W}_O,$$

with $\mathbf{W}_O \in \mathbb{R}^{D \times D}$ being a learned projection matrix.

Residual Connection and Normalization

A residual connection and layer normalization are applied:

$$\mathbf{Z}'^{(l)} = \text{LN}\left(\mathbf{Z}^{(l-1)} + \text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V})\right).$$

Feed-Forward Network (FFN)

Each encoder block contains a two-layer FFN applied to each token independently:

$$\text{FFN}(\mathbf{z}) = \text{GELU}\left(\mathbf{z}\mathbf{W}_1 + \mathbf{b}_1\right) \mathbf{W}_2 + \mathbf{b}_2, \tag{10}$$

where:

- $\mathbf{W}_1 \in \mathbb{R}^{D \times D_{\text{ff}}}$,
- $\mathbf{W}_2 \in \mathbb{R}^{D_{\text{ff}} \times D}$,
- D_{ff} is the dimension of the hidden layer,
- GELU is the Gaussian Error Linear Unit activation.

A residual connection and layer normalization follow:

$$\mathbf{Z}^{(l)} = \text{LN}\left(\mathbf{Z}'^{(l)} + \text{FFN}(\mathbf{Z}'^{(l)})\right). \tag{11}$$

2.5 Classification Head

For classification tasks, the output corresponding to the class token (or a pooled representation of all tokens) is used. For instance, if a class token $\mathbf{z}_{\text{cls}}$ is used, the final prediction is obtained as:

$$\hat{y} = \text{softmax}\left(\mathbf{W}_c \mathbf{z}_{\text{cls}} + \mathbf{b}_c\right), \tag{12}$$

where $\mathbf{W}_c \in \mathbb{R}^{K \times D}$ and $\mathbf{b}_c \in \mathbb{R}^{K}$ are the weights and bias of the classification layer, and K is the number of classes. Alternatively, a global average pooling can be applied to the token embeddings before the linear projection.

References

[Alexey, 2020] Alexey, D. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv: 2010.11929.*

[Coates et al., 2011] Coates, A., Ng, A., and Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Krizhevsky et al., 2009] Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

[Vaswani, 2017] Vaswani, A. (2017). Attention is all you need. *Advances in Neural Information Processing Systems.*