Vision Transformers for Image Classification and Segmentation

Alexandre Xavier Falcão

Institute of Computing - UNICAMP

afalcao@ic.unicamp.br

What are Vision Transformers (ViTs)?

A Vision Transformer (ViT) is a neural network architecture adapted from Natural Language Processing to treat images as sequences of patches for Computer Vision tasks.

$$x \in \mathbb{R}^{H \times W \times C}$$

$$\downarrow$$

Patch Embedding

$$\downarrow$$

$+$ Positional Encoding

$$\downarrow$$

$$z_0 \in \mathbb{R}^{n \times d}$$

$$\downarrow$$

Transformer Encoder

$$\downarrow$$

$$z_L \in \mathbb{R}^{n \times d}$$

$$\downarrow$$

Task Head

Agenda

- Patch, Position, and Class Embeddings.
- Transformer Encoder.
- Classification Head (a simple MLP).
- Segmentation Head (more complex).

  Extra challenges:

  - Need dense pixel-wise predictions.
  - Preserve spatial resolution.
  - Handle multiple scales.

Patch Embedding

Converting Images to Sequences:

- Divide image into fixed-size patches (e.g., 16×16 pixels).
- Flatten each patch into a vector.
- Linear projection to embedding dimension d.

Mathematical Formulation

For an image $\mathrm{x} \in \mathbb{R}^{H \times W \times C}$:

- Patch size: $P \times P$
- Number of patches: $n = \frac{HW}{P^2}$
- Flattened patch i: $\mathrm{x}_p^i \in \mathbb{R}^{P^2 \cdot C}$
- Projection matrix: $\mathrm{E} \in \mathbb{R}^{(P^2 \cdot C) \times d}$ (learnable)
- Embedded patch i: $\mathrm{z}_p^i = \mathrm{x}_p^i \mathrm{E} \in \mathbb{R}^d$
- All embeddings: $[\mathrm{z}_p^1; \mathrm{z}_p^2; \ldots; \mathrm{z}_p^n] \in \mathbb{R}^{n \times d}$

Class and Position Embeddings

- ViTs are permutation-invariant, which makes it important to encode the spatial relationship among patches.
- Class and Position embeddings are tensors randomly initialized, which are learned during the encoder's training.
- A class embedding $x_{class} \in \mathbb{R}^d$ is added to sequence $[x_p^1 E; \ldots; x_p^n E]$ of patch embeddings for image classification.
- The position embedding is then added to each term:

$$z_0 = [x_{class}; x_p^1 E; \ldots; x_p^n E] + E_{pos} = [z_0^0; z_0^1; \ldots; z_0^n]$$

where $E_{pos} \in \mathbb{R}^{(n+1) \times d}$.

Class [CLS] Token

Attention at each encoder's block among the $n+1$ patch (token) embeddings aggregates information into the class embedding z_L^0 of the last block L, such that it can represent the image for classification.

Usage

- Input: $\mathrm{z}_0 = [\mathrm{z}_0^0; \mathrm{z}_0^1; \ldots; \mathrm{z}_0^n]$.
- After encoder: Extract $\mathrm{z}_L^0 \in \mathbb{R}^d$ from z_L.
- Classification: $y = \mathrm{MLP}(\mathrm{z}_L^0)$.

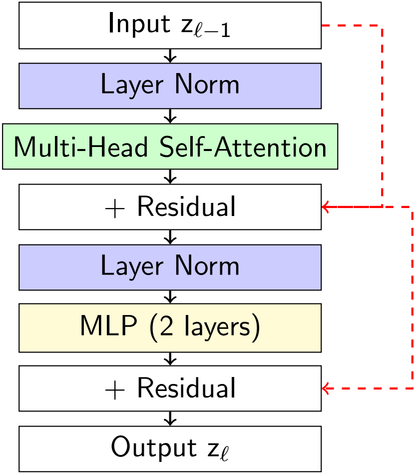
For segmentation, there is no class token: we need per-patch predictions (e.g., labeling each pixel or patch), not a single global representation.

Transformer Encoder Block

Each encoder block $l = 1, 2, \ldots, L$ processes information as follows.

- An input sequence z_{l-1} of token embeddings.
- Layer Normalization + Multi-head self-attention:
  - Each token attends to every other token.
  - Multiple attention heads capture different types of relationships.
- Add: Residual connection from input.
- Layer Normalization + Feed-forward network: a two-layer MLP applied to each token independently.
- Add: Residual connection from the previous residual layer.
- An output sequence z_l of token embeddings.

Transformer Encoder Block



Pre-LN architecture (modern standard): Layer normalization is applied *before* attention and MLP blocks, with residual connections bypassing each sub-block

What do different blocks learn?

For a self-supervised ViT (e.g., ViT-S/8 with 12 blocks), evidence from attention visualization and probing studies suggests:

Early Blocks (1-4):

- Low-level features.
- Colors, textures.
- Local patterns.

Middle (5-9):

- Increasing complexity.
- Object part features.
- Semantic grouping.

Final (10-12):

- Global semantics.
- Object-level features.
- Class-specific patterns.

Finding (Caron et al., 2021)

In **self-supervised** ViTs (DINO), the **last block's** attention maps spontaneously learn to segment objects and object parts **without any labels**.

Layer Normalization (LN)

Normalize each token's features independently to stabilize training.

Input: Token embedding $\mathrm{x} \in \mathbb{R}^d$ where d is the embedding dimension.

Operation:

1. Compute mean: $\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$
2. Compute variance: $\sigma^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2$
3. Normalize: $\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$
4. Scale and shift: $\mathrm{LN}(\mathrm{x})_i = \gamma_i \hat{x}_i + \beta_i$

Parameters: $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^d$ are learnable parameters (trained via backpropagation).

Setup:

- Input sequence: $\mathrm{Z} \in \mathbb{R}^{n \times d}$ where n is number of tokens
- Each row is a token embedding: $\mathrm{z}_i \in \mathbb{R}^d$

Step 1: Create Query, Key, Value matrices

For each token z_i, compute:

$$\begin{aligned} \mathrm{q}_i &= \mathrm{z}_i \mathrm{W}^Q \quad \text{(Query)} \\ \mathrm{k}_i &= \mathrm{z}_i \mathrm{W}^K \quad \text{(Key)} \\ \mathrm{v}_i &= \mathrm{z}_i \mathrm{W}^V \quad \text{(Value)} \end{aligned}$$

where $\mathrm{W}^Q, \mathrm{W}^K, \mathrm{W}^V \in \mathbb{R}^{d \times d_k}$ are learnable weight matrices.

Typically $d_k = d$ (same dimension).

In matrix form:

$$Q = ZW^{Q} \in \mathbb{R}^{n \times d_k}$$
$$K = ZW^{K} \in \mathbb{R}^{n \times d_k}$$
$$V = ZW^{V} \in \mathbb{R}^{n \times d_k}$$

Step 2: Compute attention scores
Measure similarity between queries and keys:

$$A = \frac{QK^{T}}{\sqrt{d_k}} \in \mathbb{R}^{n \times n}$$

Element A_{ij} = similarity between token i's query and token j's key.

Division by $\sqrt{d_k}$ prevents extremely large values (scaled dot-product).

Self-Attention: Single Head

Step 3: Apply softmax

Normalize scores to get attention weights:

$$\mathsf{A}' = \mathsf{softmax}(\mathsf{A}) \in \mathbb{R}^{n \times n}$$

Each row sums to 1: $\sum_{j=1}^{n} A'_{ij} = 1$

Step 4: Weighted sum of values

Output for each token is weighted combination of all values:

$$\mathsf{Z}' = \mathsf{A}'\mathsf{V} \in \mathbb{R}^{n \times d_k}$$

Complete formula:

$$\boxed{\mathsf{Attention}(\mathsf{Q}, \mathsf{K}, \mathsf{V}) = \mathsf{softmax}\left(\frac{\mathsf{Q}\mathsf{K}^T}{\sqrt{d_k}}\right)\mathsf{V}}$$

Multi-Head Self-Attention

Run h attention operations in parallel, each with different learned projections.

For head j (where $j = 1, \ldots, h$):

$$\begin{aligned} \mathrm{Q}^{(j)} &= \mathrm{Z}\mathrm{W}^{Q(j)} \in \mathbb{R}^{n \times d_h} \\ \mathrm{K}^{(j)} &= \mathrm{Z}\mathrm{W}^{K(j)} \in \mathbb{R}^{n \times d_h} \\ \mathrm{V}^{(j)} &= \mathrm{Z}\mathrm{W}^{V(j)} \in \mathbb{R}^{n \times d_h} \end{aligned}$$

where $d_h = d/h$ (head dimension), and $\mathrm{W}^{Q(j)}, \mathrm{W}^{K(j)}, \mathrm{W}^{V(j)} \in \mathbb{R}^{d \times d_h}$.

Compute attention output for head j:

$$\mathrm{head}_j = \mathrm{Attention}(\mathrm{Q}^{(j)}, \mathrm{K}^{(j)}, \mathrm{V}^{(j)}) \in \mathbb{R}^{n \times d_h}$$

Multi-Head Self-Attention

Concatenate all heads:

$$\text{Concat} = [\text{head}_1 \| \text{head}_2 \| \cdots \| \text{head}_h] \in \mathbb{R}^{n \times d}$$

Project to output:

$$\boxed{\text{MSA}(\text{Z}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\text{W}^O}$$

where $\text{W}^O \in \mathbb{R}^{d \times d}$ is a learnable output projection matrix.

Parameters: $h \times 3d \times d_h + d \times d = 4d^2$ total.

Example: ViT-Base has $d = 768$, $h = 12$, so $d_h = 64$.

Multi-Head Attention: Why multiple heads?

Different heads learn different relationships:

- **Head 1:** Spatial proximity (neighboring patches).
- **Head 2:** Color similarity (patches with similar hues).
- **Head 3:** Texture patterns (similar textures).
- **Head 4:** Object parts (semantically related regions).
- **...and so on.**

Each head has its own parameters ($W^{Q(j)}, W^{K(j)}, W^{V(j)}$), allowing it to specialize in capturing different types of patterns.

The model learns what each head should focus on during training.

Residual Connection

$$Z_{\text{out}} = Z_{\text{in}} + \text{SubLayer}(Z_{\text{in}})$$

where SubLayer can be MSA or FFN.

No learnable parameters — just element-wise addition.

Benefits:

- **Gradient flow:** Gradients can flow directly backward through the identity path, preventing vanishing gradients in deep networks.
- **Learning refinements:** The sublayer learns *changes* to the input, not entirely new representations.
- **Easier optimization:** Network can start with identity mappings and gradually learn transformations.

Feed-Forward Network (FFN)

Two-layer MLP applied independently to each token.

Formula:

$$\mathrm{FFN}(\mathrm{z}) = \mathrm{W}_2 \cdot \mathrm{GELU}(\mathrm{W}_1\mathrm{z} + \mathrm{b}_1) + \mathrm{b}_2$$

Parameters:

- $\mathrm{W}_1 \in \mathbb{R}^{d \times d_{\mathrm{ff}}}$: First layer weights (expand).
- $\mathrm{b}_1 \in \mathbb{R}^{d_{\mathrm{ff}}}$: First layer bias.
- $\mathrm{W}_2 \in \mathbb{R}^{d_{\mathrm{ff}} \times d}$: Second layer weights (project back).
- $\mathrm{b}_2 \in \mathbb{R}^{d}$: Second layer bias.

Typically $d_{\mathrm{ff}} = 4d$ (expansion factor of 4).

Total parameters: $2 \times d \times d_{\mathrm{ff}} + d_{\mathrm{ff}} + d \approx 8d^2$.

FFN: Why expand then compress?

Dimension expansion creates capacity:

Think of $d_{\text{ff}} = 4d$ as creating a "higher-dimensional space" where:

- More complex transformations are possible.
- Different features can be processed independently.
- Non-linear interactions are learned.

Analogy: Like a bottleneck in opposite direction:

$$\begin{aligned} \text{Token} \in \mathbb{R}^d. &\xrightarrow{\text{expand}}. \mathbb{R}^{4d} \text{ (more room for computation).} \\ &\xrightarrow{\text{compress}}. \mathbb{R}^d \text{ (back to original size).} \end{aligned}$$

The intermediate $4d$ representation allows the network to compute complex functions that would be impossible with just a single linear layer.

ViT for Classification

Standard Classification Pipeline:

1. Image $\rightarrow$ Patch embeddings + position embeddings.
2. Add [CLS] token at the beginning.
3. Pass through L transformer encoder blocks.
4. Extract [CLS] token representation.
5. Classification head (MLP) for final prediction.

Training

- Pre-trained on large datasets (ImageNet-21k, JFT-300M).
- Fine-tuned on downstream tasks.
- Cross-entropy loss.
- Strong data augmentation crucial.

ViT Model Variants

Model	Blocks	Hidden Size	Heads	Params
ViT-Base	12	768	12	86M
ViT-Large	24	1024	16	307M
ViT-Huge	32	1280	16	632M

Patch Sizes:

- ViT/16: 16×16 patches (most common).
- ViT/32: 32×32 patches (faster, less accurate).
- ViT/14: 14×14 patches (higher resolution).

ViT for Semantic Segmentation

Exist several approaches of segmentation heads.

- **Linear Head** (Simplest).
  - Single linear layer per token.
  - Fast but limited expressiveness.
- **CNN-based Decoders** (Traditional).
  - U-Net style: progressive upsampling with skip connections.
  - Good for fine details but computationally expensive.
  - Example: SETR-PUP, UPerNet.
- **Transformer Decoders**.
  - Use attention mechanisms.
  - Example: Segmenter (mask transformer).
- **All-MLP Decoder** (Modern best practice) ← SegFormer
  - Lightweight, efficient, and powerful
  - We'll focus on this approach!

Efficient Hierarchical Transformer (NeurIPS 2021)

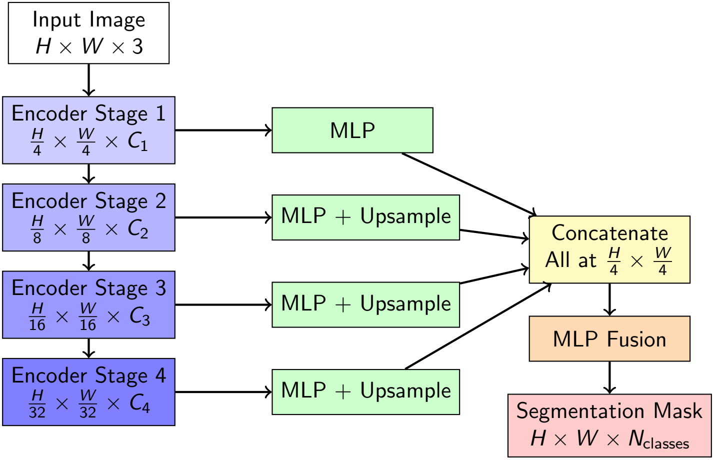
Encoder:

- Hierarchical structure.
- Mix-FFN layers.
- Overlapping patches.
- Multi-scale features.

Decoder:

- Lightweight All-MLP.
- Fuses multi-level features.
- No positional encoding.
- Efficient design.

SegFormer Architecture



SegFormer Encoder: What Changes from ViT?

Four Key Modifications to Standard ViT:

1. **Hierarchical Structure**
   ViT: Single-scale features $\rightarrow$ SegFormer: Multi-scale pyramid.
2. **Efficient Self-Attention (SRA)**
   ViT: $O(n^2)$ complexity $\rightarrow$ SegFormer: Reduced via spatial downsampling.
3. **Mix-FFN (replaces positional encoding)**
   ViT: Fixed PE $\rightarrow$ SegFormer: 3×3 conv for local information.
4. **Overlapping Patch Embedding**
   ViT: Non-overlapping patches $\rightarrow$ SegFormer: Overlapping for local continuity.

Difference 1: Hierarchical Multi-Scale Encoder

ViT (Single-Scale):

- One resolution: $\frac{H}{16} \times \frac{W}{16}$.
- All blocks at same scale.
- Single feature map output.

SegFormer (Multi-Scale):

- 4 stages: $\frac{H}{4}, \frac{H}{8}, \frac{H}{16}, \frac{H}{32}$.
- Progressive downsampling.
- Multi-level feature pyramid.

Stage Configuration (MiT-B0):

Stage	**Resolution**	**Channels**	**Blocks**	**Heads**
1	$H/4 \times W/4$	32	2	1
2	$H/8 \times W/8$	64	2	2
3	$H/16 \times W/16$	160	2	5
4	$H/32 \times W/32$	256	2	8

Difference 2: Spatial-Reduction Attention (SRA)

Problem with ViT: Self-attention has $O(n^2)$ complexity.

$$\text{Attention}(\mathrm{Q}, \mathrm{K}, \mathrm{V}) = \text{Softmax}\left(\frac{\mathrm{QK}^T}{\sqrt{d_k}}\right)\mathrm{V}$$

SegFormer Solution: Reduce spatial dimensions of K and V.
Given input $\mathrm{X} \in \mathbb{R}^{n \times d}$ (where n is number of tokens):

$$\begin{aligned}
\mathrm{Q} &= \mathrm{XW}_Q \in \mathbb{R}^{n \times d} \quad \textcolor{blue}{\text{(unchanged)}} \\
\mathrm{K} &= \mathrm{SR(X)W}_K \in \mathbb{R}^{\frac{n}{R^2} \times d} \quad \textcolor{red}{\text{(reduced by factor } R^2\text{)}} \\
\mathrm{V} &= \mathrm{SR(X)W}_V \in \mathbb{R}^{\frac{n}{R^2} \times d} \quad \textcolor{red}{\text{(reduced by factor } R^2\text{)}}
\end{aligned}$$

where SR(X) is spatial reduction via convolution with stride R (e.g., kernel size 7×7, stride $R = 4$).

Complexity: $O(n^2) \rightarrow O\left(n \cdot \frac{n}{R^2}\right) = O\left(\frac{n^2}{R^2}\right)$.

SRA: Stage-wise Reduction Ratios

Insight: Higher resolution stages need more reduction.

Stage	Resolution	Tokens n	Reduction R	K,V size
1	$H/4 \times W/4$	$\frac{HW}{16}$	8	$\frac{n}{64}$
2	$H/8 \times W/8$	$\frac{HW}{64}$	4	$\frac{n}{16}$
3	$H/16 \times W/16$	$\frac{HW}{256}$	2	$\frac{n}{4}$
4	$H/32 \times W/32$	$\frac{HW}{1024}$	1	n (no reduction)

Design Principle:

- Early stages (high-res): Aggressive reduction to save computation.
- Later stages (low-res): Less/no reduction for global context.

Difference 3: Mix-FFN (No Positional Encoding!)

ViT FFN:

$$\mathrm{FFN}(z) = W_2 \cdot \mathrm{GELU}(W_1 z + b_1) + b_2$$

- Requires fixed positional encoding
- Problem: PE must be interpolated for different resolutions

SegFormer Mix-FFN:

$$\mathrm{Mix\text{-}FFN}(z) = W_2 \cdot \mathrm{GELU}(\mathrm{DWConv}_{3\times3}(W_1 z + b_1)) + b_2$$

- 3×3 depthwise convolution.
- Provides positional info implicitly.
- Resolution-agnostic!

Benefit: Mix-FFN introduces zero-padded 3×3 convolution that:

- Leaks location information to each token.
- Eliminates need for explicit positional encoding.
- Works seamlessly across different input resolutions.

Patch Embedding in Each Stage:

ViT:

- Kernel: 16×16.
- Stride: 16.
- Non-overlapping patches.
- Loss of local continuity.

SegFormer:

- Kernel: 7×7 (stage 1), 3×3 (others).
- Stride: 4 (stage 1), 2 (others).
- **Overlapping patches**.
- Preserves local structure.

Stage 1 Patch Embedding:

$$\text{Input: } H \times W \times 3 \xrightarrow{\text{Conv } 7\times 7\text{, stride } 4} \frac{H}{4} \times \frac{W}{4} \times C_1$$

Subsequent Stages (2-4):

$$\text{Input: } \frac{H}{2^{i-1}} \times \frac{W}{2^{i-1}} \times C_{i-1} \xrightarrow{\text{Conv } 3\times 3\text{, stride } 2} \frac{H}{2^i} \times \frac{W}{2^i} \times C_i$$

Modified Transformer Block Structure:

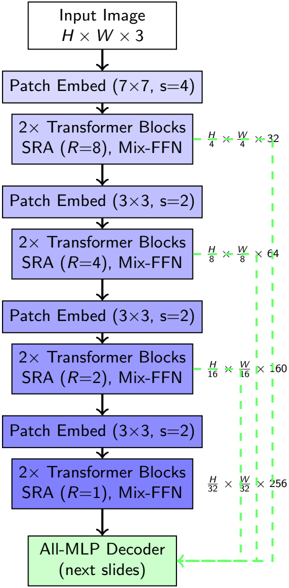
$$\begin{aligned}
\hat{z} &= \text{LayerNorm}(z) \\
z' &= z + \text{SRA-MHSA}(\hat{z}) \quad \text{(vs. standard MHSA in ViT)} \\
\hat{z}' &= \text{LayerNorm}(z') \\
z_{out} &= z' + \text{Mix-FFN}(\hat{z}') \quad \text{(vs. standard FFN in ViT)}
\end{aligned}$$

where:

- **SRA-MHSA**: Spatial-Reduction Multi-Head Self-Attention
- **Mix-FFN**: $W_2 \cdot \text{GELU}(\text{DWConv}_{3\times3}(W_1\hat{z}' + b_1)) + b_2$

Key Point: Same overall structure as ViT (Pre-LN, residual connections), but with efficient attention and implicit positional encoding.

SegFormer Encoder: Complete Architecture



SegFormer Decoder (All-MLP Head)

Given multi-scale features from all 4 stages:

For each stage $i = 1, 2, 3, 4$ with features $\mathrm{F}_i \in \mathbb{R}^{\frac{H}{2^{i+1}} \times \frac{W}{2^{i+1}} \times C_i}$:

1. **Project to unified dimension:**

$$\tilde{\mathrm{F}}_i = \mathrm{Linear}(\mathrm{F}_i) \in \mathbb{R}^{\frac{H}{2^{i+1}} \times \frac{W}{2^{i+1}} \times C}$$

2. **Upsample to resolution $\frac{H}{4} \times \frac{W}{4}$:**

$$\hat{\mathrm{F}}_i = \mathrm{Upsample}(\tilde{\mathrm{F}}_i) \in \mathbb{R}^{\frac{H}{4} \times \frac{W}{4} \times C}$$

3. **Concatenate all upsampled features:**

$$\mathrm{F}_{\mathrm{fused}} = \mathrm{Concat}(\hat{\mathrm{F}}_1, \hat{\mathrm{F}}_2, \hat{\mathrm{F}}_3, \hat{\mathrm{F}}_4) \in \mathbb{R}^{\frac{H}{4} \times \frac{W}{4} \times 4C}$$

4. **Final MLP and upsample:**

$$\mathrm{Logits} = \mathrm{Upsample}(\mathrm{MLP}(\mathrm{F}_{\mathrm{fused}})) \in \mathbb{R}^{H \times W \times N_{\mathrm{classes}}}$$