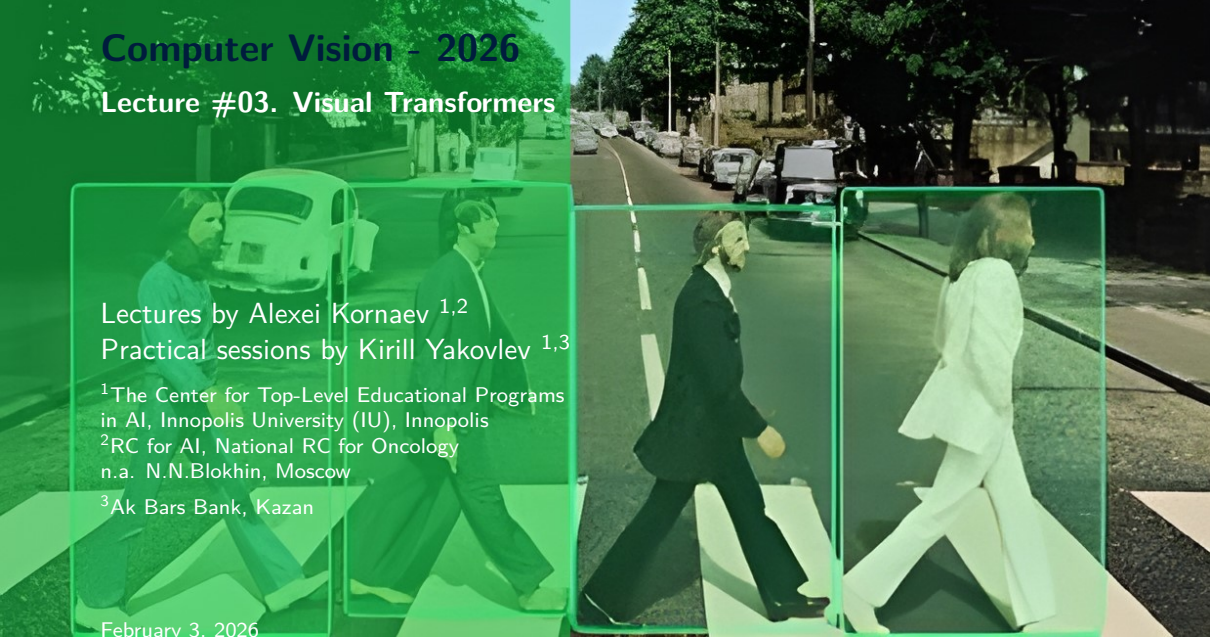
Computer Vision - 2026

Lecture #03. Visual Transformers



Lectures by Alexei Kornaev [1,2]
Practical sessions by Kirill Yakovlev [1,3]

[1]The Center for Top-Level Educational Programs in AI, Innopolis University (IU), Innopolis
[2]RC for AI, National RC for Oncology n.a. N.N.Blokhin, Moscow

[3]Ak Bars Bank, Kazan

February 3, 2026

Agenda

Outcomes

Transformer Architecture Overview

Vision Transformers (ViTs)

Conclusion

1. Outcomes
2. Transformer Architecture Overview
3. Vision Transformers (ViTs)
4. Conclusion

Section 1. Outcomes

Outcomes

This week's lecture on Vision Transformers (ViTs) builds an operational understanding of transformer-based vision models. By the end of this week, students will be able to:

1. Understand the Transformer encoder dataflow: tokens $\rightarrow Q, K, V \rightarrow$ attention weights $\rightarrow$ updated tokens.
2. Describe ViT input formation: patching, linear patch embeddings, class token, and learned positional embeddings.
3. Identify why ViTs scale well and why they can be fragile without strong training recipes/data (augmentation, regularization, pretraining).
4. Distinguish major families at a high level: baseline ViT, data-efficient training ideas (DeiT-style), and hierarchical/window attention (Swin-style), and relate this to modern detector/segmenter backbones.

Key Takeaway: ViTs turn an image into a sequence of patch tokens and iteratively mix global information via self-attention; performance depends strongly on data and training recipe.

Section 2. Transformer Architecture Overview

Transformer Architecture

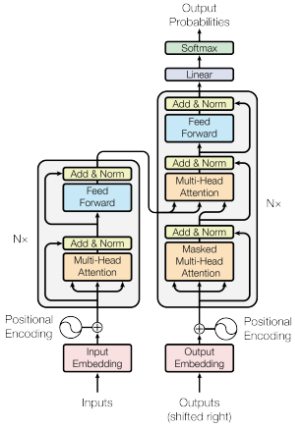


Figure: High-level Transformer architecture, consisting of an encoder and decoder stack [Vaswani et al., 2023].

Transformer Encoder (Vision Perspective)

What We Use in Vision

Vision Transformers rely on the **Transformer encoder only**:

- a sequence of input tokens (image patches),
- repeated **encoder blocks** with self-attention and MLPs,
- a task-specific head (classification, detection, segmentation).

Encoder Block Structure

Each encoder block consists of:

1. Layer Normalization
2. Multi-Head Self-Attention
3. Residual connection
4. Layer Normalization
5. Feedforward Network (MLP)

Key Point: Unlike NLP Transformers, ViTs do **not** use a decoder.

Self-Attention: Core Operation

Self-attention allows each token to update its representation by interacting with **all other tokens** in the sequence.

Scaled Dot-Product Attention

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

- $Q, K, V \in \mathbb{R}^{N \times d}$: queries, keys, values
- N: number of tokens (patches)
- d: embedding dimension

Positional Embeddings

Why Positional Information Is Needed

Self-attention treats tokens as a **set**, not a sequence. Positional embeddings inject information about **order or spatial location**.

Vision Transformers (ViTs)

- An image is split into patches, each patch becomes a token.
- Each token receives a **learned positional embedding**.
- The input to the Transformer is:

$$\mathbf{z}_i = \mathbf{E}_{\text{patch}}(i) + \mathbf{E}_{\text{pos}}(i)$$

- Positional embeddings are **learned parameters**.
- When image resolution changes, embeddings are typically **interpolated**.

Note: Early Transformers used sinusoidal encodings; ViTs almost always use learned ones.

Section 3. Vision Transformers (ViTs)

Image to Patch Splitting

Key Idea

ViTs divide the image into **non-overlapping patches** of fixed size.

- Given an image $x \in \mathbb{R}^{H \times W \times C}$ (height H, width W, channels C), we divide it into N patches of size $P \times P$.
- The number of patches is:

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

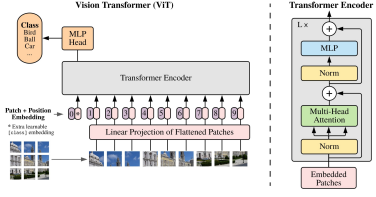


Figure: ViT intuition [Dosovitskiy et al., 2021].

Patch Extraction: Toy Example

Toy Image (Single Channel)

We consider a tiny 4×4 grayscale image ($C = 1$):

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Patch Size $P = 2$

This image is split into $N = 4$ non-overlapping patches:

1 2 5 6	3 4 7 8
9 10 13 14	11 12 15 16

INNOPOLIS UNIVERSITY pretation: Each boxed block is one **patch token**.

Flattening Patches into Vectors

Flatten Each Patch

Each 2×2 patch is flattened into a vector in $\mathbb{R}^4$.

$$x_1 = [1,\ 2,\ 5,\ 6]^T$$
$$x_2 = [3,\ 4,\ 7,\ 8]^T$$
$$x_3 = [9,\ 10,\ 13,\ 14]^T$$
$$x_4 = [11,\ 12,\ 15,\ 16]^T$$

Stacked Patch Matrix

$$X = \begin{bmatrix} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \\ 9 & 10 & 13 & 14 \\ 11 & 12 & 15 & 16 \end{bmatrix} \in \mathbb{R}^{4\times 4}$$

INNOPOLIS UNIVERSITY

Shape: $N = 4$ patches, each of dimension $P^2C = 4$.

Linear Projection of Flattened Patches

Patch Embeddings

Each patch is flattened into a vector and projected into a higher-dimensional space using a linear layer.

- Each patch $x_p \in \mathbb{R}^{P \times P \times C}$ is flattened into a vector $x_p \in \mathbb{R}^{P^2 C}$.
- A trainable weight matrix $E \in \mathbb{R}^{D \times P^2 C}$ projects it into a D-dimensional embedding:

$$z_i = E x_i, \quad i = 1, \ldots, N.$$

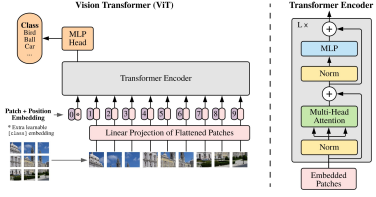


Figure: ViT intuition [Dosovitskiy et al., 2021].

Linear Projection: Patch Embeddings

Projection Matrix

We project each patch vector into a 2D embedding space ($D = 2$):

$$E = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{2\times 4}$$

Patch Embeddings

$$z_i = Ex_i$$

$$z_1 = [\ ,\ ]^T$$

$$z_2 = [\ ,\ ]^T$$

$$z_3 = [\ ,\ ]^T$$

$$z_4 = [\ ,\ ]^T$$

Result: Each patch is now a **token** in $\mathbb{R}^2$.

Linear Projection: Patch Embeddings

Projection Matrix

We project each patch vector into a 2D embedding space ($D = 2$):

$$E = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{2\times 4}$$

Patch Embeddings

$$z_i = Ex_i$$

$$z_1 = [1,\ 5]^T$$

$$z_2 = [3,\ 7]^T$$

$$z_3 = [9,\ 13]^T$$

$$z_4 = [11,\ 15]^T$$

INNOPOLIS UNIVERSITY

Result: Each patch is now a **token** in $\mathbb{R}^2$.

Learned Positional Embeddings in ViTs

Instead of fixed sinusoidal functions, Vision Transformers often use

learned positional embeddings:

$$\mathbf{z}_0 = \texttt{class token}, \quad \mathbf{z}_i = E(x_i) + E_p(i), \quad i = 1, \ldots, N$$

where:

- $E(x_i)$ is the patch embedding, $E_p(i)$ is the learned positional embedding for patch i,
- $\mathbf{z}_0$ is an extra classification token.

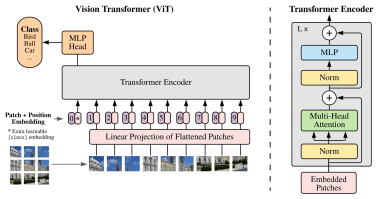


Figure: ViT intuition [Dosovitskiy et al., 2021].

Positional Embeddings: Toy Example

Patch Tokens from the Previous Slide

$$z_1 = [1,\ 5]^T \quad \text{(top-left)}, z_2 = [3,\ 7]^T \quad \text{(top-right)}$$
$$z_3 = [9,\ 13]^T \quad \text{(bottom-left)}, z_4 = [11,\ 15]^T \quad \text{(bottom-right)}$$

Learned Positional Embeddings

We add a learned positional vector to each token:

$$p_1 = [0.1,\ 0.1]^T, \quad p_2 = [0.1,\ 0.2]^T$$
$$p_3 = [0.2,\ 0.1]^T, \quad p_4 = [0.2,\ 0.2]^T$$

Final Input Tokens

$$\tilde{z}_i = z_i + p_i \quad \Rightarrow \quad \tilde{z}_1 = [1.1,\ 5.1]^T,\ \tilde{z}_2 = [3.1,\ 7.2]^T,\ \ldots$$

Positional Embeddings: Toy Example

Patch Tokens (Content Only)

$z_1 = [1,\ 5]^T$ (top-left), $z_2 = [3,\ 7]^T$ (top-right)

$z_3 = [9,\ 13]^T$ (bottom-left), $z_4 = [11,\ 15]^T$ (bottom-right)

Class Token

$z_0 = [0,\ 0]^T$

Learned Positional Embeddings

Each token (including the class token) receives a positional embedding:

$$p_0 = [0.0,\ 0.0]^T$$

$$p_1 = [0.1,\ 0.1]^T, \quad p_2 = [0.1,\ 0.2]^T$$

$$p_3 = [0.2,\ 0.1]^T, \quad p_4 = [0.2,\ 0.2]^T$$

Final Input Sequence

$$\tilde{z}_i = z_i + p_i, \quad i = 0, \ldots, 4$$

$$\tilde{z}_0 = [0,\ 0]^T,\ \tilde{z}_1 = [1.1,\ 5.1]^T,\ \tilde{z}_2 = [3.1,\ 7.2]^T,\ \ldots$$

From Patches to Tokens

What We Have Now

- Image $\rightarrow$ patches
- Patches $\rightarrow$ vectors
- Vectors $\rightarrow$ token embeddings

Next Question

How do these tokens **interact** with each other?

Answer: Self-attention.

Input Representation: Query, Key, and Value

What Are Q, K, V?

Self-attention transforms input embeddings into Query (Q), Key (K), and Value (V) vectors using trainable weight matrices.

- **Input sequence:** $X \in \mathbb{R}^{N \times d}$ (where N is the number of patches, and d is the embedding dimension).
- **Linear projections:**

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

  where $W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$ are learnable.
- **Roles of Q, K, V:** - **Query:** What we are looking for. - **Key:** What is stored in memory. - **Value:** What information is retrieved.

Q, K, V: Numeric Example

Input Tokens (with Position + Class Token)

$$X = \begin{bmatrix} \tilde{z}_0 \\ \tilde{z}_1 \\ \tilde{z}_2 \\ \tilde{z}_3 \\ \tilde{z}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1.1 & 5.1 \\ 3.1 & 7.2 \\ 9.2 & 13.1 \\ 11.2 & 15.2 \end{bmatrix} \in \mathbb{R}^{5\times 2}$$

Projection Matrices (for Simplicity)

$$W_Q = W_K = W_V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Resulting Queries, Keys, Values

$$Q = K = V = X$$

Note: We use identity matrices to focus on **attention mechanics**, not linear algebra.

Similarity Computation: Scaled Dot-Product Attention

How Does Attention Work?

Each query compares itself with all keys to measure similarity.

- **Compute dot-product similarity between Q and K:**

$$S = QK^T \in \mathbb{R}^{N \times N}$$

- **Normalize using $\sqrt{d}$ to stabilize gradients:**

$$A = \frac{S}{\sqrt{d}}$$

- **Apply the softmax function:**

$$A_{ij} = \frac{\exp(S_{ij}/\sqrt{d})}{\sum_k \exp(S_{ik}/\sqrt{d})}$$

- The resulting **attention matrix** A defines how much focus each query has on every key.

Attention Scores: Dot Products

Similarity Matrix

$S = QK^T \in \mathbb{R}^{5\times 5}$

Example: Class Token as Query

We compute similarities between the class token and all keys:

$$S_{0j} = \tilde{z}_0 \cdot \tilde{z}_j$$

$$[0, 0] \cdot [\,\cdot\,] = 0$$

$$\Rightarrow \quad S_{0,:} = [0,\ 0,\ 0,\ 0,\ 0]$$

Example: Patch Token 1 as Query

$$S_{1j} = [1.1,\ 5.1] \cdot [\,\tilde{z}_j\,]$$

$$\Rightarrow S_{1,:} \approx [0,\ 27,\ 40,\ 78,\ 90]$$

Interpretation: Larger dot product $\Rightarrow$ higher attention weight.

Weighted Sum: Generating the Output

How Is the Output Computed?

Each patch's representation is updated based on the weighted sum of all patches.

- **Multiply the attention weights A by the Value matrix V:**

$$Z = AV$$

- Each output row is a **weighted sum of value vectors**.
- Self-attention enables **global interactions between patches**.

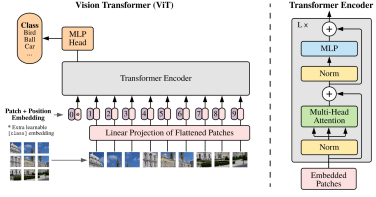


Figure: ViT intuition [Dosovitskiy et al., 2021].

Weighted Sum: Class Token Update

Attention Weights for Class Token

After softmax, the class token assigns weights to all tokens:

$$\alpha_{0j} = \text{softmax}(S_{0j})$$

Updated Class Token

$$z_0^{\text{out}} = \sum_{j=0}^{4} \alpha_{0j}\, v_j$$

- The class token becomes a **weighted mixture of all patches**.
- This vector is used for **classification**.

Key Idea: Classification is done by **reading out the class token**, not individual patches.

Multi-Head Self-Attention (MSA)

Why Multiple Heads?

Different heads can focus on different relations (e.g., local texture vs. long-range structure) in parallel subspaces.

- Input tokens: $X \in \mathbb{R}^{N \times d}$
- Split into h heads, each with dimension $d_h = d/h$.
- Per-head projections:

$$Q_i = XW_{Q_i}, \quad K_i = XW_{K_i}, \quad V_i = XW_{V_i}, \quad W_{Q_i}, W_{K_i}, W_{V_i} \in \mathbb{R}^{d \times d_h}$$

- Per-head attention:

$$\text{head}_i = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_h}}\right) V_i \in \mathbb{R}^{N \times d_h}$$

- Concatenate and mix:

$$\text{MSA}(X) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\, W_O, \quad W_O \in \mathbb{R}^{d \times d}$$

Pre-LN + Residual: Stable Transformer Block

Why Pre-LN + Residuals?

LayerNorm stabilizes gradients; residual connections preserve information and enable deep stacks.

Pre-LN Form (common in ViTs)

$$X_1 = X + \mathrm{MSA}(\mathrm{LayerNorm}(X))$$

$$X_2 = X_1 + \mathrm{FFN}(\mathrm{LayerNorm}(X_1))$$

- **Residual path** keeps a clean signal flow through depth.
- **LayerNorm before** each sublayer improves training stability.

Note: Some early Transformers used post-LN; modern vision recipes typically use pre-LN.

Feedforward Network (FFN / MLP)

What Happens After Self-Attention?

Each token is independently refined by the same 2-layer MLP (position-wise FFN).

Standard FFN Definition

For each token vector $x \in \mathbb{R}^d$:

$$\mathrm{FFN}(x) = W_2\,\sigma(W_1 x + b_1) + b_2$$

where $W_1 \in \mathbb{R}^{d \times d_{\mathrm{ff}}}$, $W_2 \in \mathbb{R}^{d_{\mathrm{ff}} \times d}$, and typically $\sigma = \mathrm{GELU}$.

- Acts on each token **independently** (no token mixing here).
- Provides **nonlinearity** and increases capacity via $d_{\mathrm{ff}} \gg d$.

Key Idea: Token mixing happens in attention; feature transformation happens in the FFN.

Transformer Encoder Block (One Line)

$$\boxed{X_1 = X + \mathrm{MSA}(\mathrm{LN}(X))} \qquad \boxed{X_2 = X_1 + \mathrm{FFN}(\mathrm{LN}(X_1))}$$

Output: X_2 becomes the input to the next block.

Comparison: CNNs vs. ViTs

Key Differences

- CNNs learn spatial hierarchies via local receptive fields and weight sharing.
- ViTs model long-range dependencies using self-attention but lack inductive biases like locality and translation invariance.

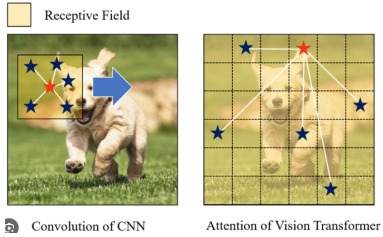


Figure: Comparison of CNNs and ViTs in feature extraction and learning patterns [Baek et al., 2022]

ViT Models Zoo

The following resources should be met:

1. ViTs, Community Computer Vision Course by Hugging Face
2. Vision Transformer (ViT) by Hugging Face

Section 4. Conclusion

Strengths and Weaknesses of ViTs

Strengths

- **Long-range dependencies** captured effectively.
- **Scalability** to large datasets.
- **Flexibility** to adapt across domains (text, vision, multimodal).

Weaknesses

- **Data-hungry** - requires large-scale training.
- **Computationally expensive** due to self-attention complexity.
- **Lack of inductive bias** - struggles with small datasets.

Conclusion

Key Takeaways

- Vision Transformers (ViTs) treat images as sequences of patches and use self-attention for feature extraction.
- Unlike CNNs, ViTs do not rely on convolution but instead learn global relationships from data.
- While powerful, ViTs require large datasets and high computational resources to generalize well.
- Hybrid architectures (CNN + ViT) help balance efficiency and performance.

Looking Ahead: Modern architectures like DeiT, Swin Transformer, and hybrid models address ViT limitations [Li et al., 2021].

Bibliography

A. Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," NeurIPS 2020. [arXiv:2010.11929](https://arxiv.org/abs/2010.11929)

A. Vaswani et al., "Attention Is All You Need," NeurIPS 2017. [arXiv:1706.03762](https://arxiv.org/abs/1706.03762)

H. Touvron et al., "Training Data-efficient Image Transformers & Distillation through Attention," ICML 2021. [arXiv:2012.12877](https://arxiv.org/abs/2012.12877)

Z. Liu et al., "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows," ICCV 2021. [arXiv:2103.14030](https://arxiv.org/abs/2103.14030)

S. Prince, *Understanding Deep Learning*, MIT Press, 2023.

Bibliography

Sihun Baek, Jihong Park, Praneeth Vepakomma, Ramesh Raskar, Mehdi Bennis, and Seong-Lyun Kim. Visual transformer meets cutmix for improved accuracy, communication efficiency, and data privacy in split learning, 2022. URL https://arxiv.org/abs/2207.00234.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. URL https://arxiv.org/abs/2010.11929.

Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. **IEEE transactions on neural networks and learning systems**, 33(12):6999–7019, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.