Computer Vision

Week 12-13

2025-2

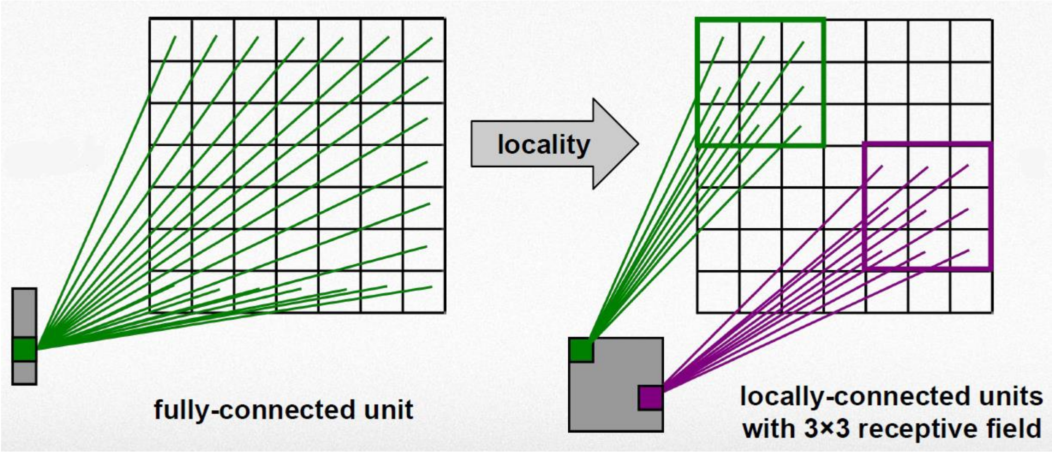
Mobile Systems Engineering

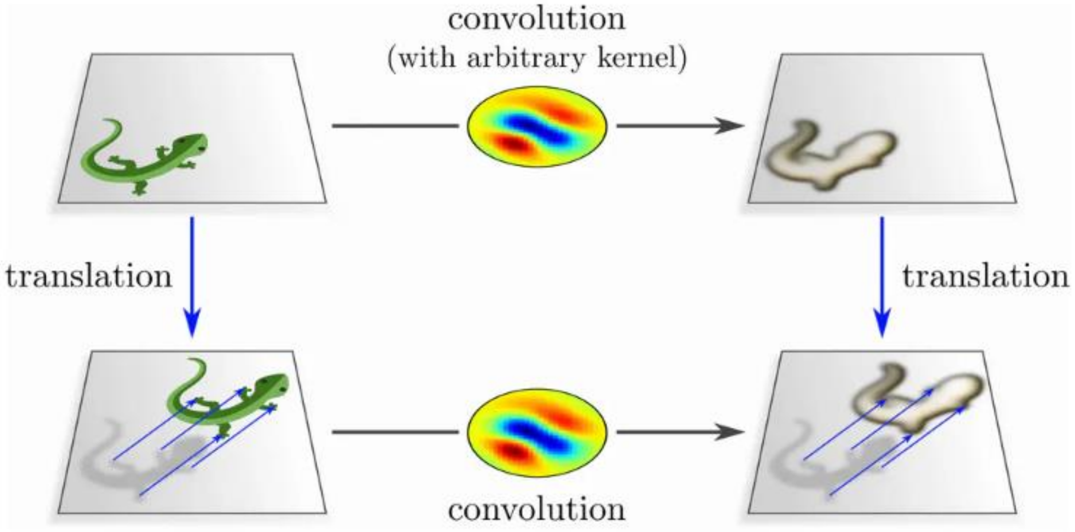
Dankook University

Motivation: Why Study Vision Transformers?

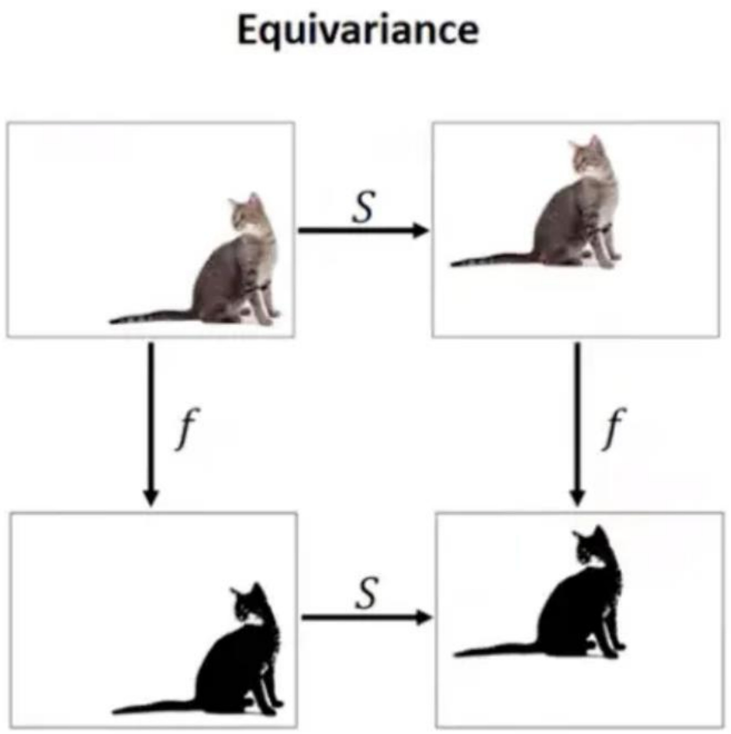
▪ From Inductive Bias in CNNs to Data-Driven Transformers

- **Why Transformers in Vision?**
  - o **CNNs** have been the backbone of computer vision for decades.
  - o Their strength comes from strong ***inductive biases***
    - ✓**Locality**
      - ➢Convolutional filters focus on local neighborhoods.
    - ✓**Translation Equivariance**
      - ➢Features shift predictably when the input image shifts.
  - o These biases allow CNNs to learn effectively from **limited data**, but they can also **restrict scalability** when moving to larger datasets or more complex patterns.





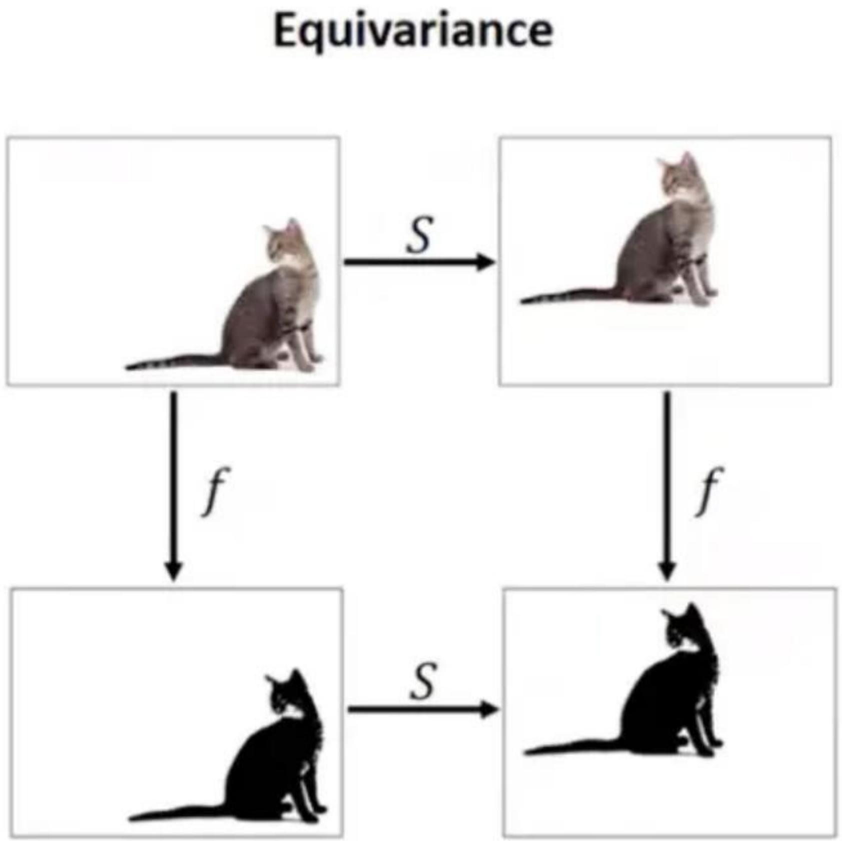
Motivation: Why Study Vision Transformers?



- **From Inductive Bias in CNNs to Data-Driven Transformers**
  - **Recap – What is Translation Equivariance?**
    - **Why CNNs Can Detect the Same Pattern Anywhere in the Image**
      - ✓**Meaning**
        - ➢**Translation equivariance** means that if the **input shifts in space**, the **output feature map also shifts in the same direction**.
        - ➢Example
          - If a cat in an image moves **10 pixels to the right**, the convolution feature map of the cat will also appear **10 pixels to the right**.
      - ✓**Role of Kernels (Filters)**
        - ➢The ***kernel weights do NOT change*** when the input moves.
          - During training, kernels learn to detect specific local patterns (e.g., cat's ear, eye, whisker).
        - ➢The ***same kernel*** is applied across the entire image (sliding operation).
        - ➢As a result, the same feature can be detected **regardless of its position** in the image.

Motivation: Why Study Vision Transformers?

- **From Inductive Bias in CNNs to Data-Driven Transformers**
  - **Recap – What is Translation Equivariance?**
    - **Why CNNs Can Detect the Same Pattern Anywhere in the Image**
      - ✓**Key Takeaway**
        - ➢ **Translation equivariance = input shift → output shift**.
        - ➢ **Kernel weights remain fixed** after training.
        - ➢ **Weight sharing** across the whole image ensures that features are recognized anywhere, independent of position.



Motivation: Why Study Vision Transformers?

- **From Inductive Bias in CNNs to Data-Driven Transformers**
  - **What is Inductive Bias?**
    - ***Inductive Bias = the structural assumptions that a model inherently carries.***
      - ✓It reflects how a model internally views the relationship between **input data** and **output predictions**.
    - Example

Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations

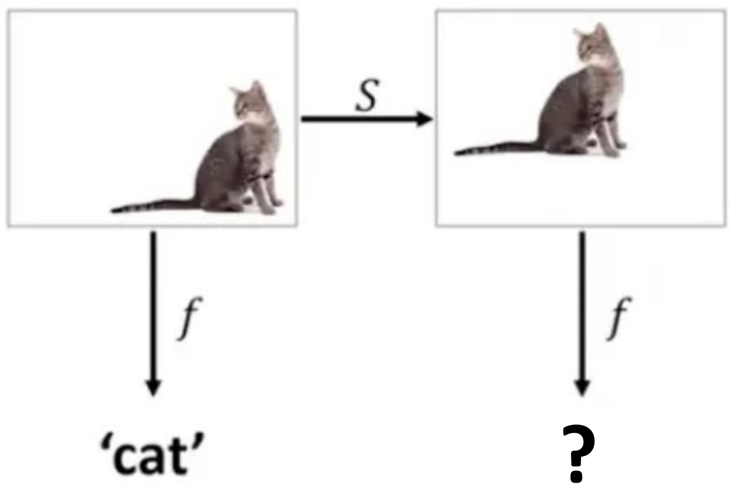
      - ✓**CNNs** assume that local pixel groups contain meaningful features (locality).
      - ✓**RNNs** assume sequential dependencies between timesteps.
      - ✓**Fully connected layers** assume all inputs may interact equally.

    - These built-in assumptions **guide learning**, reducing data requirements but also limiting flexibility.

Motivation: Why Study Vision Transformers?

- **From Inductive Bias in CNNs to Data-Driven Transformers**
  - ***Transformers and Weak Inductive Bias***
    - ***Transformers lack strong inductive biases*** like locality or translation equivariance.
    - Instead, they rely on ***global self-attention*** to learn all relationships directly from data.
    - This makes them
      - ✓**Data-hungry**: need very large datasets for effective training.
      - ✓**Flexible and scalable**: can capture global dependencies more efficiently once enough data is available.
  - **Example**
    - If a cat shifts position in the image (see figure), CNNs still classify it as *"cat"* due to weight sharing and invariance.
    - A Transformer, however, must **learn from data** that "cat on the left" and "cat on the right" are the same class.
    - Without large training data, this generalization may fail.

Invariance



Vision Transformers – Overall Architecture

- **What is a Vision Transformer (ViT)?**
  - A Vision Transformer (ViT) is a model that applies the **Transformer architecture to images** by
    - **Step 1.**
      Splitting the image into fixed-size patches.
    - **Step 2.**
      Flattening and linearly projecting
      each patch into an embedding vector.
    - **Step 3.**
      Adding **positional encodings** to
      preserve spatial information.
    - **Step 4.** Adding a **learnable class token**
      for classification.
    - **Step 5.** Passing the sequence of embeddings
      through **Transformer encoder layers**.

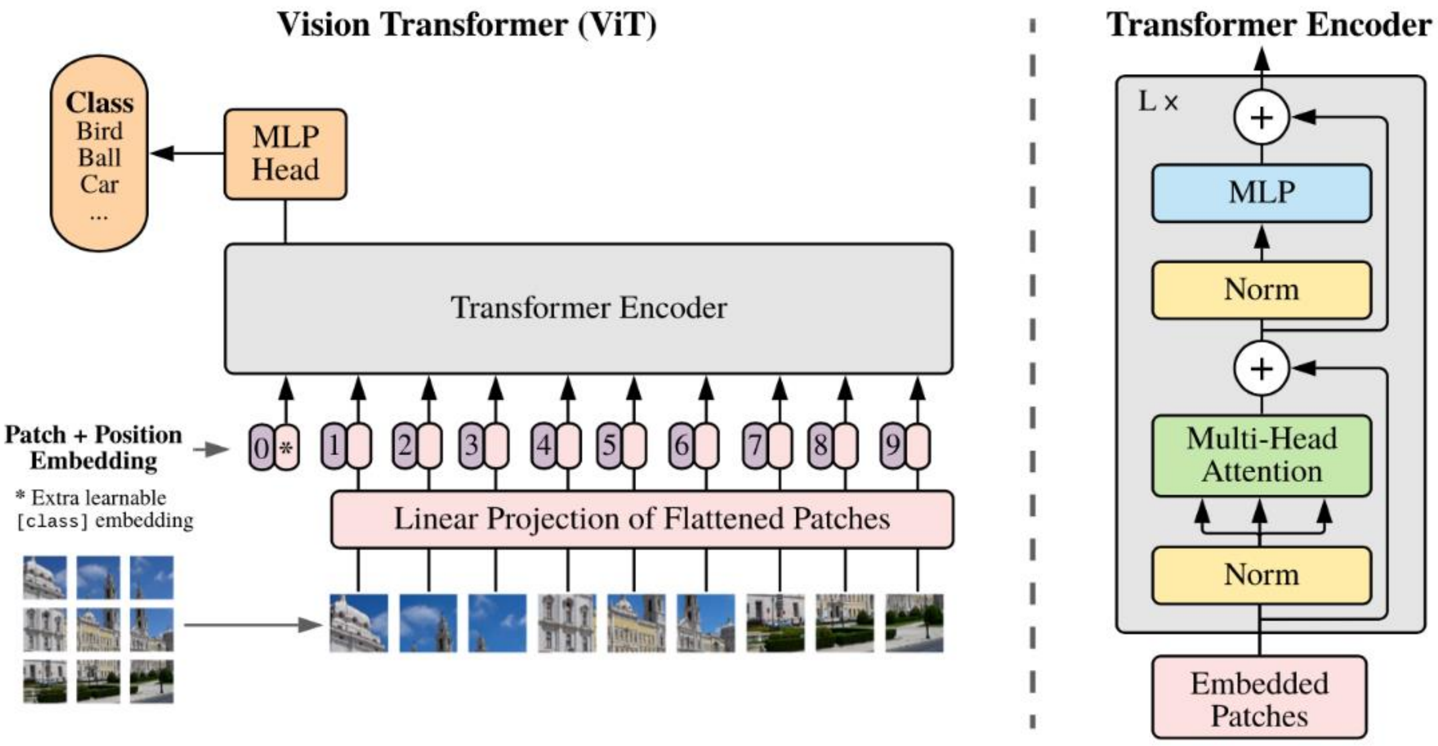
Vision Transformers – Overall Architecture

What is a Vision Transformer (ViT)?

- **One-Sentence Summary**
  - ViT treats an image as a **sequence of patch tokens**, just like words in NLP, and applies a Transformer encoder for classification or transfer learning.

Architecture Overview

- Input = patch embeddings + [CLS] + positional embeddings
- Each of **L blocks**
  - LayerNorm
  - Multi-Head Self-Attention (MSA) + residual
  - LayerNorm
  - MLP (GELU) + residual



- Final [CLS] embedding → image representation → MLP/linear head

Data Preparation for Vision Transformer Input

- **How Images Become Tokens in Vision Transformer**
  - ***From pixels to patch embeddings***
    - Steps
      - ✓**Step 1. Image Patch Creation**
        - ➢Split the image into fixed-size non-overlapping patches.
        - ➢Example: For a 224×224 image and 16×16 patch size → 14×14 = 196 patches.
      - ✓**Step 2. Patch Embedding**
        - ➢Flatten each patch (**P×P×C**) into a vector of length **$P^2 \cdot C$**.
        - ➢Apply a **linear projection** to map it into **D-dimensional embedding space**.
      - ✓**Step 3. Add Class Token**
        - ➢Add a special learnable vector [CLS] to serve as the **image representation** for classification.
      - ✓**Step 4. Positional Embedding**
        - ➢Since Transformers lack spatial structure, add positional encodings so the model knows patch order.

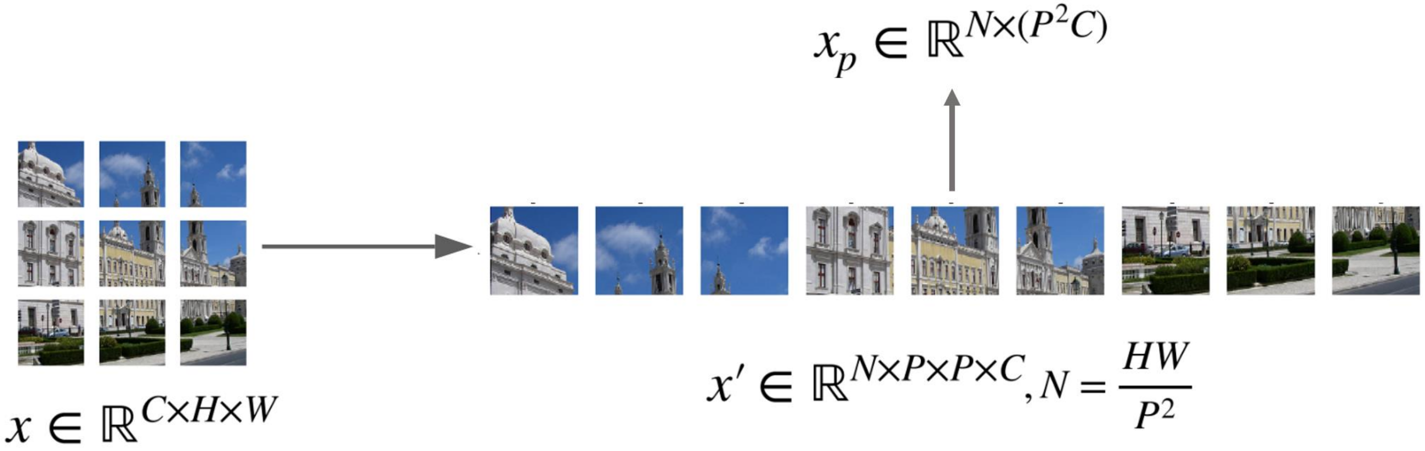
Data Preparation for Vision Transformer Input

- **How Images Become Tokens in Vision Transformer**
  - ***From pixels to patch embeddings***
    - Steps
      - ✓**Step 1. Image Patch Creation**
        - ➢Split the image into fixed-size non-overlapping patches.
        - ➢Example: For a 224×224 image and 16×16 patch size → 14×14 = 196 patches.
      - ✓**Step 2. Patch Embedding**
        - ➢Flatten each patch (**P×P×C**) into a vector of length **$P^2 \cdot C$**.
        - ➢Apply a **linear projection** to map it into **D-dimensional embedding space**.
      - ✓**Step 3. Class Token**
        - ➢Add a special learnable vector [CLS] to serve as the **image representation** for classification.
      - ✓**Step 4. Positional Embedding**
        - ➢Since Transformers lack spatial structure, add positional encodings so the model knows patch order.

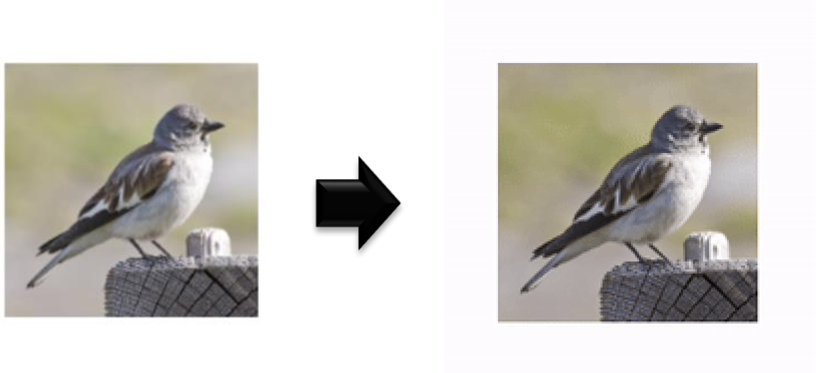
Data Preparation for Vision Transformer Input

Step 1. Image Patch Creation

- From Image to Patches
  - Input image: $\boldsymbol{x} \in \mathbb{R}^{C \times H \times W}$
    - ✓ $\boldsymbol{C}$: number of channels (e.g., 3 for RGB)
    - ✓ $\boldsymbol{H}$: image height
    - ✓ $\boldsymbol{W}$: image width



  - Divide the image into ***non-overlapping patches*** of size $\boldsymbol{P} \times \boldsymbol{P}$.



  - Total number of patches

$$\boldsymbol{N} = \frac{\boldsymbol{H} \cdot \boldsymbol{W}}{\boldsymbol{P}^2}$$

  - After splitting, each patch is represented as

$$\boldsymbol{x}' \in \mathbb{R}^{\boldsymbol{N} \times \boldsymbol{P} \times \boldsymbol{P} \times \boldsymbol{C}}$$

    - ✓ where $\boldsymbol{x}'$ denotes the collection of all image patches before flattening.

Data Preparation for Vision Transformer Input

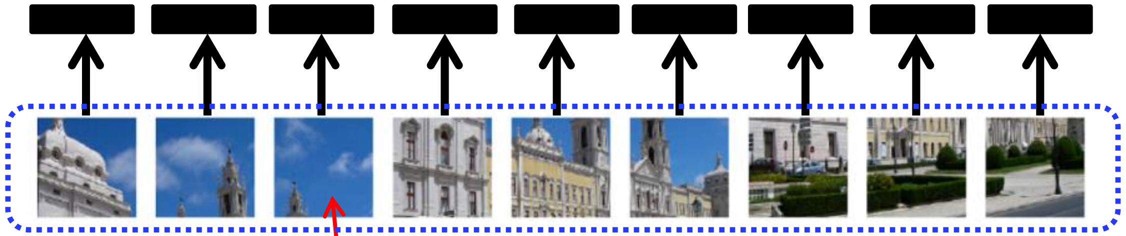
- **Step 2. Patch Embedding**
  - **Step 2-1. Flattening**
    - Each patch x_p^i (the i-th patch) is **flattened** into a vector of length $P^2 \cdot C$

$$x_p^i \in \mathbb{R}^{P^2 C}$$

    - Collecting all patches

$$x_p \in \mathbb{R}^{N \times (P^2 C)}$$

$$x_p \in \mathbb{R}^{\mathrm{N} \times (P^2 C)}; where\ N = 9$$



$$x_p^i \in \mathbb{R}^{P^2 C}$$

Data Preparation for Vision Transformer Input

Key Difference
- **CNN:** “I already know neighbors matter, I’ll focus there.”
- **ViT:** “I don’t assume anything, ***I’ll compare all patches and figure out which relations are important.***”
- Flattening is what makes ViT **closer to NLP tokenization** than to CNN feature maps.

- **Step 2. Patch Embedding**
  - **Step 2-1. Flattening**
    - **Why Flatten? – CNN vs. ViT**
      - ✓ **CNN (Convolutional Neural Networks)**
        - Keep **2D grid structure** of images.
        - Convolution kernel slides locally across height × width.
        - **Built-in inductive biases**
          - **Locality** (neighboring pixels are strongly related)
          - **Translation equivariance** (shifted object still gives similar features)
      - ✓ **Vision Transformer (ViT)**
        - **Flatten patches** → treat them as independent tokens.
        - No spatial grid is preserved explicitly.
        - **Purpose:** remove hard-coded locality → force the model to **learn relationships through self-attention**.
        - Each patch embedding is just a vector — any spatial relation must be discovered by comparing embeddings with others.

Data Preparation for Vision Transformer Input

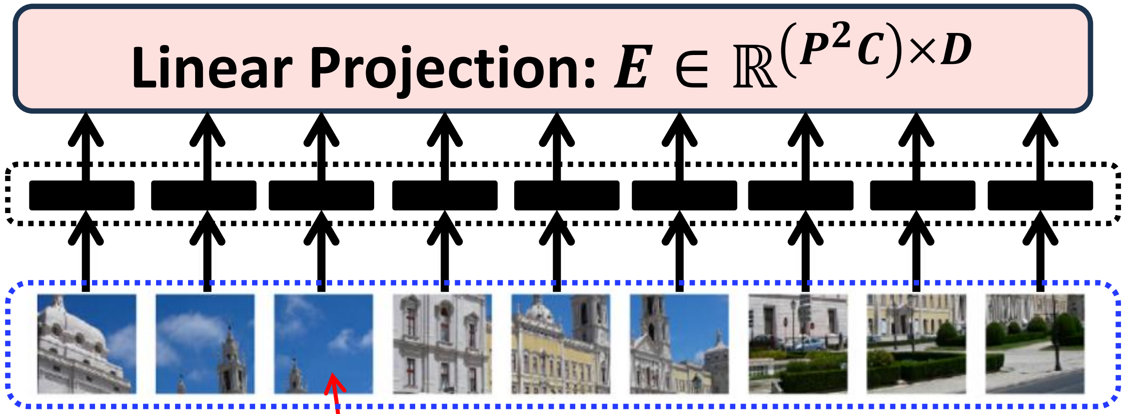
- **Step 2. Patch Embedding**
  - **Step 2-2. Linear Projection**
    - Apply a **linear projection** using matrix $\boldsymbol{E} \in \mathbb{R}^{(P^2C)\times D}$

$$\boldsymbol{x_p} \in \mathbb{R}^{\boldsymbol{N}\times(\boldsymbol{P^2C})} \rightarrow \boldsymbol{E} \in \mathbb{R}^{(\boldsymbol{P^2C})\times\boldsymbol{D}} \rightarrow [\boldsymbol{x_p^1E};\ \boldsymbol{x_p^2E};\ \dots;\ \boldsymbol{x_p^NE};] \in \mathbb{R}^{\boldsymbol{N}\times\boldsymbol{D}}$$

✓ Where …

- $\boldsymbol{D}$: embedding dimension (hidden size for Transformer input)
- $\boldsymbol{E}$: learnable projection matrix

$[\boldsymbol{x_p^1E};\ \boldsymbol{x_p^2E};\ \dots;\ \boldsymbol{x_p^NE};] \in \mathbb{R}^{\boldsymbol{N}\times\boldsymbol{D}}$



Flattened patch →

$\boldsymbol{x_p} \in \mathbb{R}^{\mathrm{N}\times(\boldsymbol{P^2C})}; \boldsymbol{where\ N = 9}$ →

$\boldsymbol{x_p^i} \in \mathbb{R}^{\boldsymbol{P^2C}}$

Data Preparation for Vision Transformer Input

- **Step 2. Patch Embedding**
  - **Step 2-2. Linear Projection**
    - **What is Linear Projection?**
      - ✓ **Definition**
        - ➢ A **Linear Projection** is a simple operation that maps an input vector into another space using a **weight matrix**.
      - ✓ Mathematically

$$\boldsymbol{y} = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$$

        - ➢ where
          - $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{P}^2\boldsymbol{C}}$: flattened patch
          - $\boldsymbol{W} \in \mathbb{R}^{(\boldsymbol{P}^2\boldsymbol{C})\times\boldsymbol{D}}$: projection (weight matrix)
          - $\boldsymbol{b} \in \mathbb{R}^{\boldsymbol{D}}$: bias term
          - $\boldsymbol{y} \in \mathbb{R}^{\boldsymbol{D}}$: patch embedding

Data Preparation for Vision Transformer Input

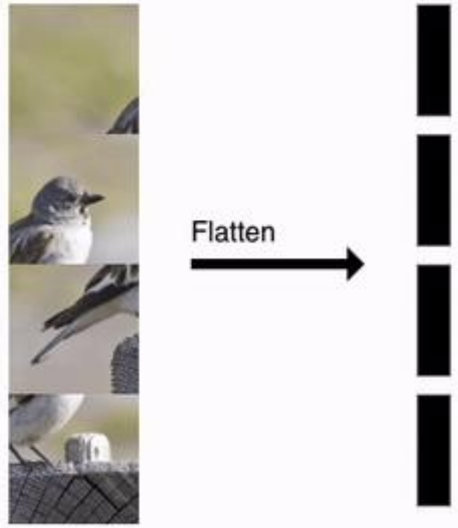
Step 2. Patch Embedding

Step 2-2. Linear Projection

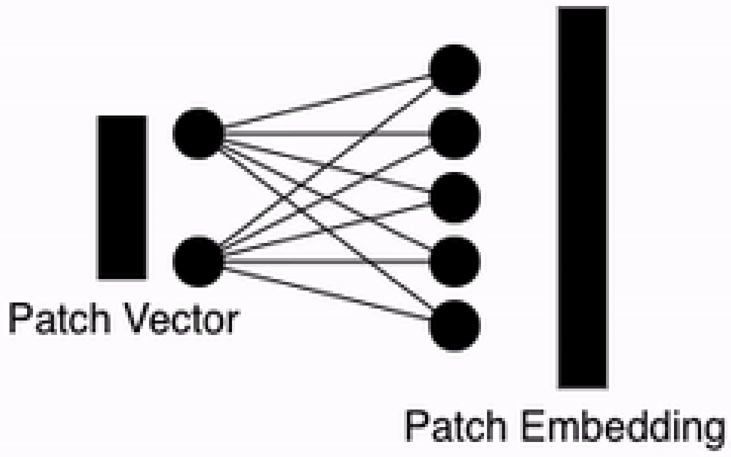
- What is Linear Projection?
  - ✓ **Key Insight**
    - ➢ This operation is **exactly the same as a Linear Layer (Fully Connected Layer)** in neural networks.
  - ✓ **Why do we call it "Projection"?**
    1. Because we are *projecting* a high-dimensional patch vector ($\boldsymbol{P^2C}$) into the embedding dimension D.
    2. "Projection" highlights the idea of ***dimensionality change*** (***compression*** or ***expansion***).
  - ✓ **Connection to Neural Networks**
    - ➢ ***Linear Projection = Linear Layer = Fully Connected (FC) Layer = Dense Layer (Keras)***
    - ➢ No activation function → purely linear transformation
    - ➢ In ViT
      - Each patch $\boldsymbol{x_p^i}$ is passed through the same Linear Layer → produces a D-dimensional embedding
      - All patches share the same projection weights $\boldsymbol{W}$.

Data Preparation for Vision Transformer Input

- **Step 2. Patch Embedding**
  - **Step 2-1. Flattening**
    - Flatten each patch ($\boldsymbol{P} \times \boldsymbol{P} \times \boldsymbol{C}$) into a vector of length $\boldsymbol{P}^2 \cdot \boldsymbol{C}$.
  - **Step 2-2. Linear Projection**
    - Apply a **linear projection** to map it into $\boldsymbol{D}$**-dimensional embedding space**.



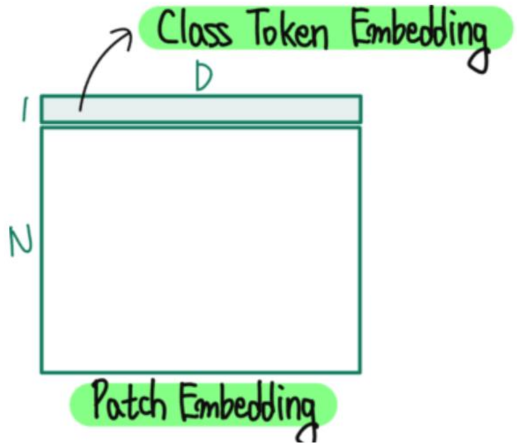
Creating a patch embedding applying a linear projection



Embedding all patches by applying a linear projection to all flattened patches

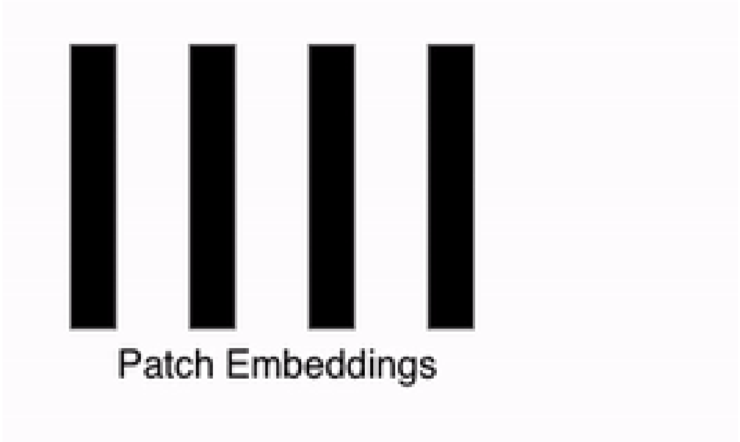
Data Preparation for Vision Transformer Input

- **Step 3. Add Class Token**
  - **Class Token**
    - **What is a Class Token?**
      - ✓A ***learnable vector*** $\boldsymbol{x_{cls}} \in \mathbb{R}^{\boldsymbol{D}}$ that is prepended to the sequence of patch embeddings.
      - ✓After appending, the sequence size changes from $\mathbb{R}^{\boldsymbol{N\times D}} \rightarrow \mathbb{R}^{\boldsymbol{(N+1)\times D}}$.



$$\left[\boldsymbol{x_{cls}};\ \boldsymbol{x_p^1 E};\ \boldsymbol{x_p^2 E}; \ldots;\ \boldsymbol{x_p^N E};\right] \in \mathbb{R}^{\boldsymbol{(N+1)\times D}}$$

      - ✓This token does ***NOT correspond to any image patch*.**
        - ➢It is an **extra parameter** introduced for classification.



Data Preparation for Vision Transformer Input

- **Step 3. Add Class Token**
  - **Motivation: Why Add a Class Token?**
    - **Inspired by NLP (BERT)**
      - ✓In BERT, a [CLS] token is added to summarize a sequence for classification tasks.
      - ✓**ViT adopts the same idea**: let the model ***gather global information*** into one special token.
    - **Avoid Pooling**
      - ✓CNNs often use **global average pooling** before classification.
      - ✓Instead, ViT uses the [CLS] token to ***aggregate information*** **without predefined pooling rules**.
  - **How Does It Work?**
    - During **Multi-Head Self-Attention**,
      - ✓The [CLS] token attends to all patch tokens, **collecting information from the entire image**.
        - ➢*Self-Attention allows* ***all tokens to interact from the first layer.***
      - ✓At the same time, all patches can also attend to the [CLS] token.
    - This bidirectional attention allows [CLS] to act as a **global information hub**.

Data Preparation for Vision Transformer Input

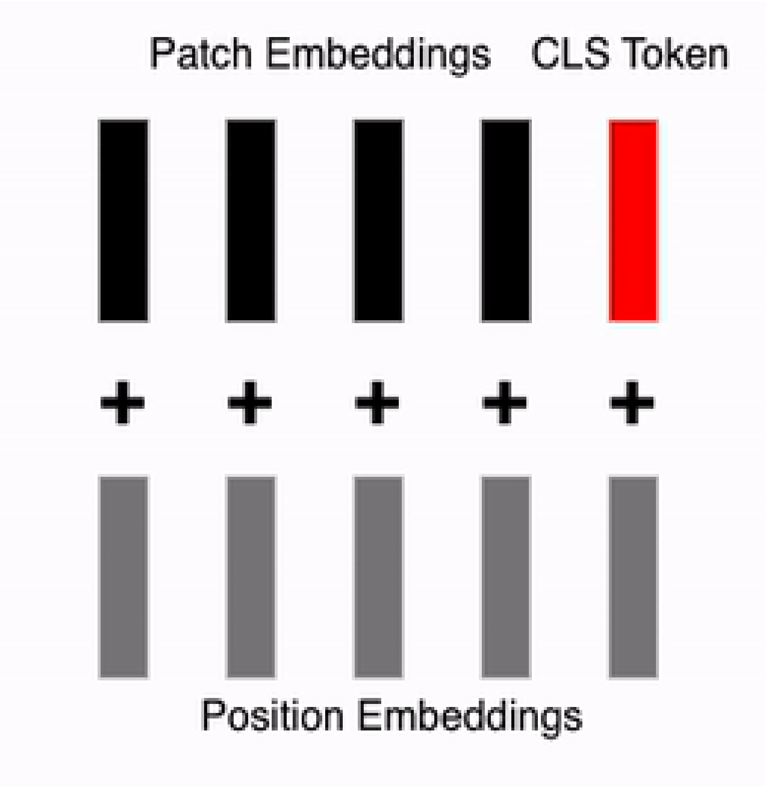
- **Step 3. Add Class Token**
  - **What Information Does It Contain?**
    - After passing through all Transformer encoder layers
      - ✓The [CLS] token embedding $\boldsymbol{z_{cls}}$ becomes ***a compact representation of the entire image***.
      - ✓It has aggregated class-relevant patterns from all patches.
  - **Final Use in Classification**
    - Only the final [CLS] token embedding is passed to the ***classification head (MLP/Linear Layer)***.
      - ✓This single vector is sufficient for classification tasks

$$\boldsymbol{y = softmax(W \cdot z_{cls})}$$

      - ✓In practice, this performs as well (or better) than pooling strategies.
    - ***"By the end of training, it contains enough global information to decide the class label."***

Data Preparation for Vision Transformer Input

- **Step 4. Positional Embedding**
  - **Positional Embedding**
    - A vector that encodes the **position (index or location)** of each token (or patch) in the input sequence, which is added to the token embedding so that the model knows **where** each token comes from.



  - **Why Do We Need Positional Embeddings?**
    - Transformer's **Self-Attention is permutation-invariant**
      - ✓ It only looks at the relationships between tokens.
      - ✓ Without extra information, it cannot distinguish whether a patch comes from the **top-left corner** or the **bottom-right corner**.
    - But in vision, **spatial order matters** (e.g., eyes above the nose, not below).
    - Positional embeddings inject this spatial information into the patch tokens.

Data Preparation for Vision Transformer Input

- **Step 4. Positional Embedding**
  - **How It Works**

1. After adding the **[CLS] token**, we have $(N + 1) \times D$ embeddings.

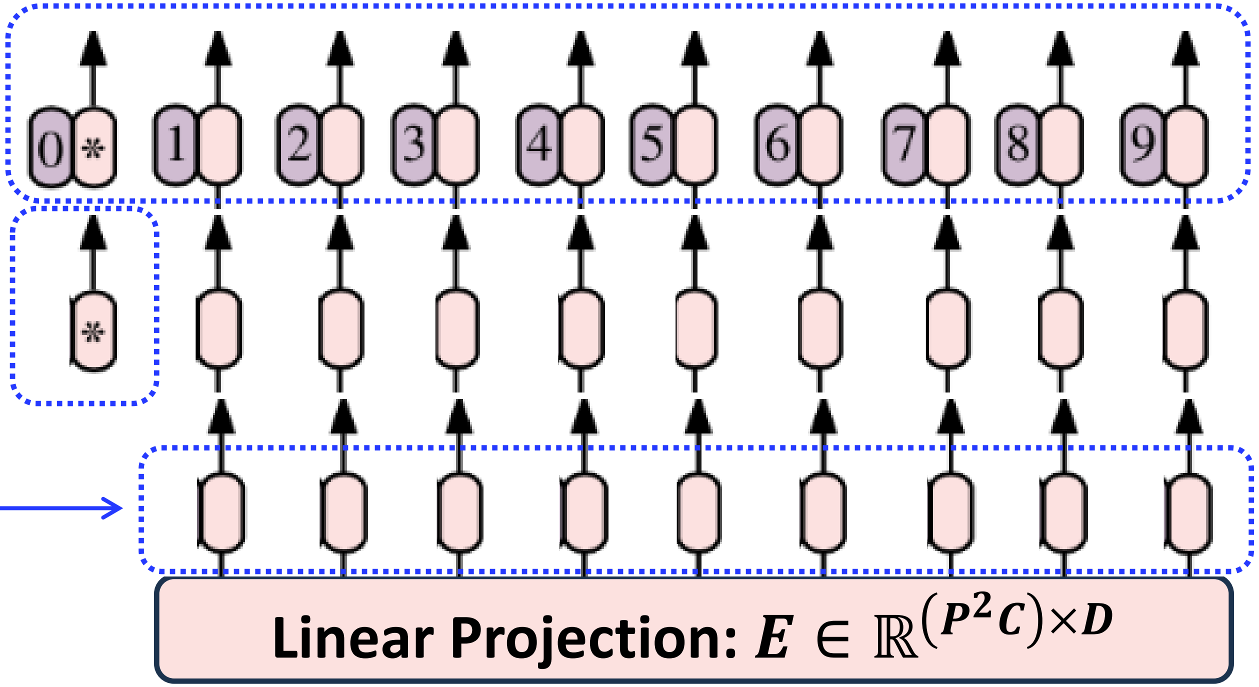
2. A **learnable positional embedding matrix** $E_{pos} \in \mathbb{R}^{(N+1)\times D}$ is added element-wise

$$z_0 = \left[x_{cls};\ ;x_p^1; E;\ x_p^2 E\ \ \dots\ \ x_p^N E\right] + E_{pos} \in \mathbb{R}^{(N+1)\times D}$$

3. Each position (patch index) has
its own embedding vector,
encoding the **absolute location** in the image grid.

$$\left[x_{cls};\ x_p^1 E;\ x_p^2 E;\ \dots;\ x_p^N E;\right] \in \mathbb{R}^{(N+1)\times D}$$

$$\left[x_p^1 E;\ x_p^2 E;\ \dots;\ x_p^N E;\right] \in \mathbb{R}^{N\times D}$$



Data Preparation for Vision Transformer Input

- **Step 4. Positional Embedding**
  - **Variants – Positional Embedding**
    - **Fixed Sinusoidal Embeddings** (NLP Transformer)
      - ✓Use sine and cosine functions with different frequencies.
      - ✓Deterministic and not learned.
    - **Learnable Embeddings (ViT default)**
      - ✓Each position (0 = [CLS], 1 … N = patches) has its own **trainable vector**.
      - ✓These vectors are initialized randomly and **optimized during training**.

  - **Clarification on Learnable Embeddings**
    - They do **not** encode positions by “simply numbering patches sequentially.”
    - Instead, each position i has a dedicated **learnable vector** $\boldsymbol{e}_i \in \mathbb{R}^{\boldsymbol{D}}$.
    - During training, the model **learns the best way to use position info** for image tasks.

Data Preparation for Vision Transformer Input

- **Step 4. Positional Embedding**
  - **Why Learnable?**

1. ***Images are 2D and far more complex*** than word order in NLP.

2. ***A fixed formula*** (like sinusoids) might be ***too rigid to capture subtle spatial patterns***.

3. Learnable embeddings allow the model to ***adaptively learn spatial prior knowledge*** from the dataset.

→ **Prior knowledge = Inductive bias**

4. Empirically shown to ***perform better*** than fixed embeddings for vision.

Data Preparation for Vision Transformer Input

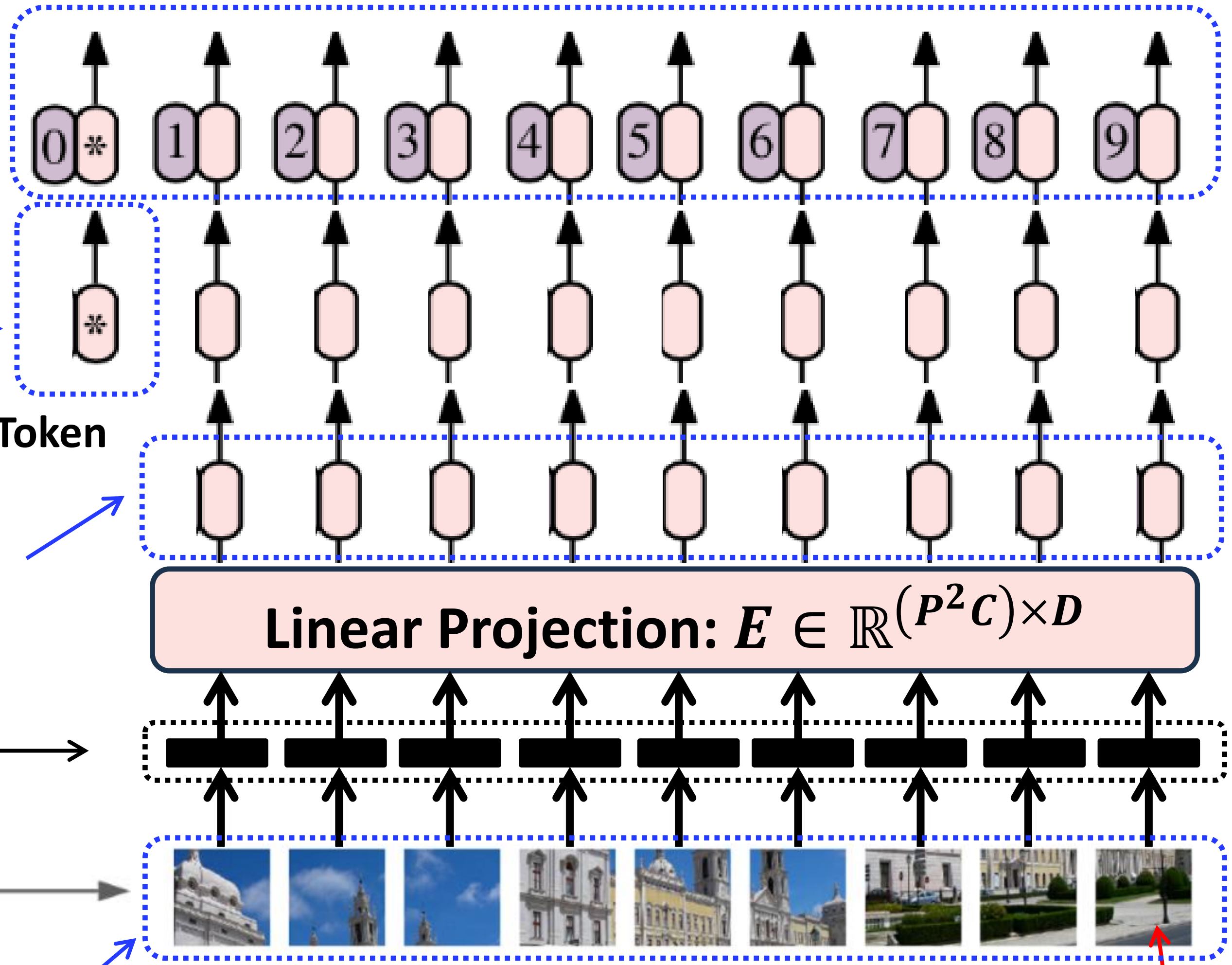
$$z_0 = \left[x_{cls};\ ;x_p^1; E;\ x_p^2 E\ \ldots\ x_p^N E\right] + E_{pos} \in \mathbb{R}^{(N+1)\times D}$$

- **Overall Processes – Data Preparation**

Step 4. Adding Positional Embedding

Step 3. Adding Class Token

$$\left[x_{cls};\ x_p^1 E;\ x_p^2 E; \ldots ;\ x_p^N E;\right] \in \mathbb{R}^{(N+1)\times D}$$



*** Learnable Class Token**

Step 2-2. Patch Embedding: Linear Projection

$$\left[x_p^1 E;\ x_p^2 E; \ldots ;\ x_p^N E;\right] \in \mathbb{R}^{N\times D}$$

Step 2-1. Patch Embedding: Flattening

Flattened patch

Step 1. Image Patch Creation

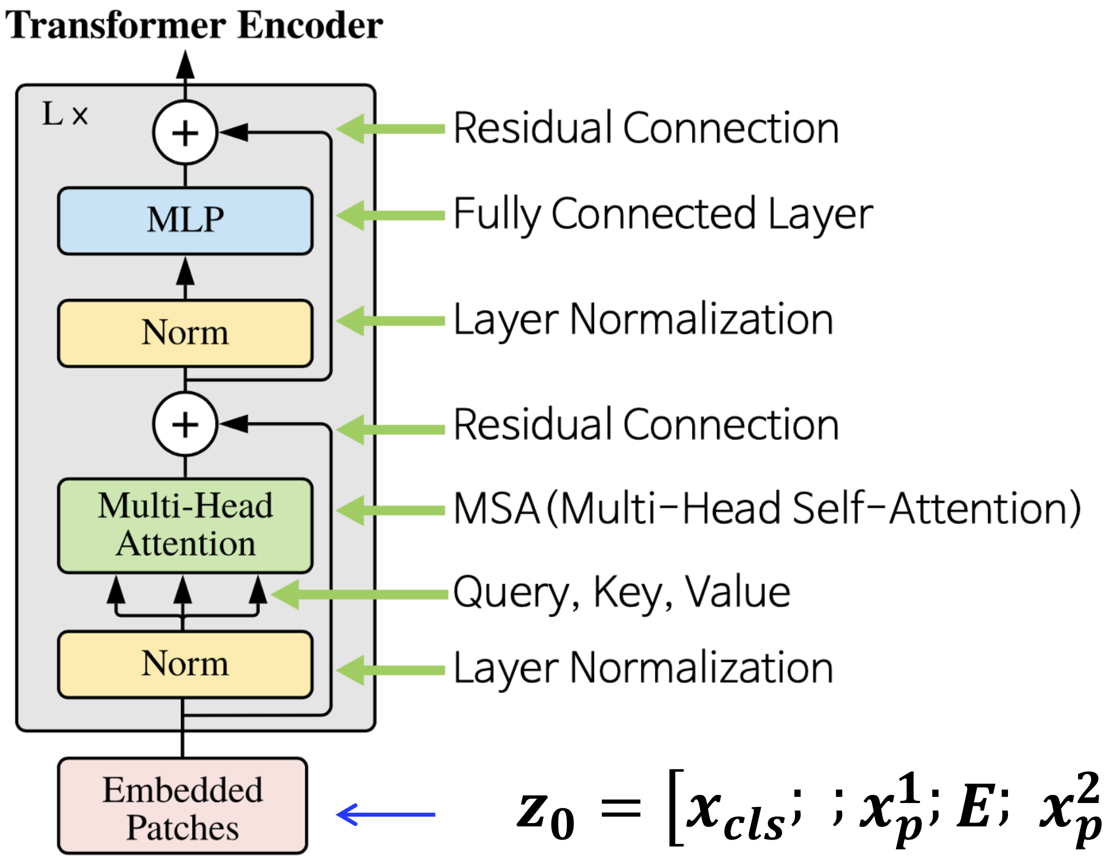
$x_p \in \mathbb{R}^{\mathrm{N}\times(P^2C)}; where\ N = 9$

$x_p^i \in \mathbb{R}^{P^2C}$

Vision Transformer Encoder Overview

▪ Transformer Encoder in Vision Transformer (ViT)

- The Transformer Encoder in ViT is built by stacking **L identical blocks**.

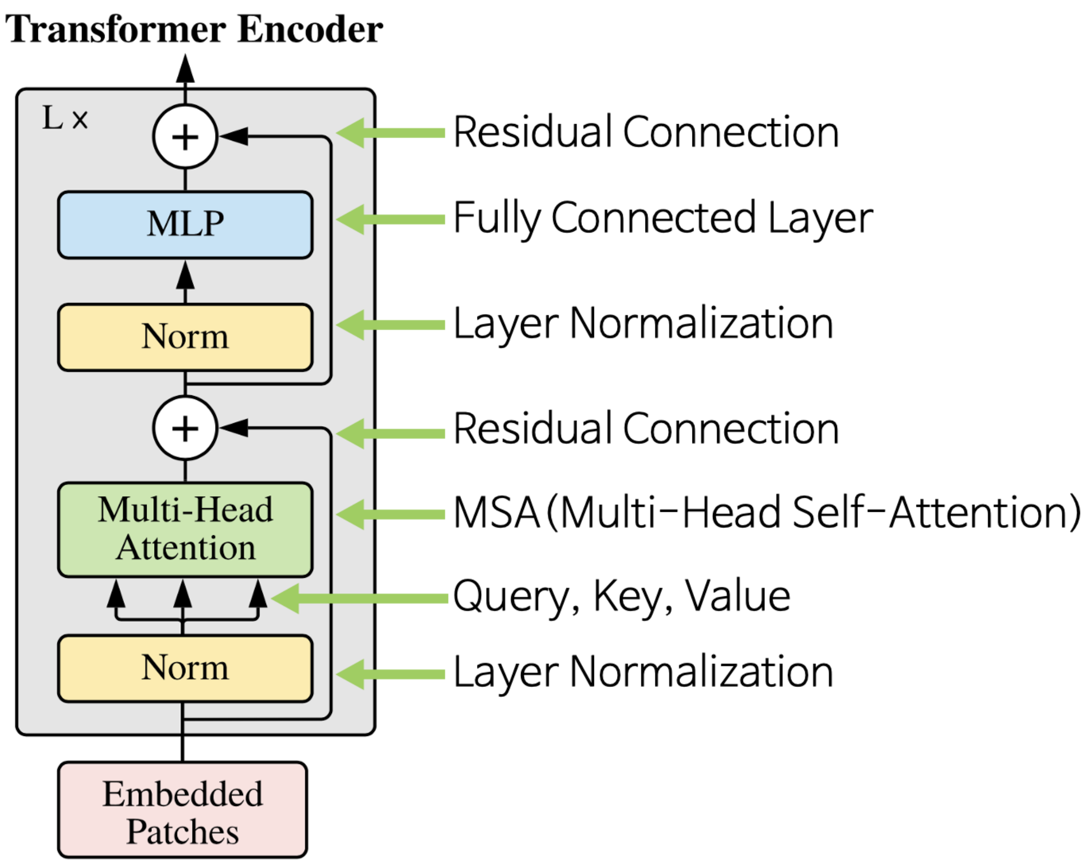


$$z_0 = \left[x_{cls};\ ;x_p^1; E;\ x_p^2 E\ \dots\ x_p^N E\right] + E_{pos} \in \mathbb{R}^{(N+1)\times D}$$

- Each block refines the patch embeddings step by step with **attention and feed-forward processing**, while **residual connections** and **layer normalization** ensure stability.

Vision Transformer Encoder Overview

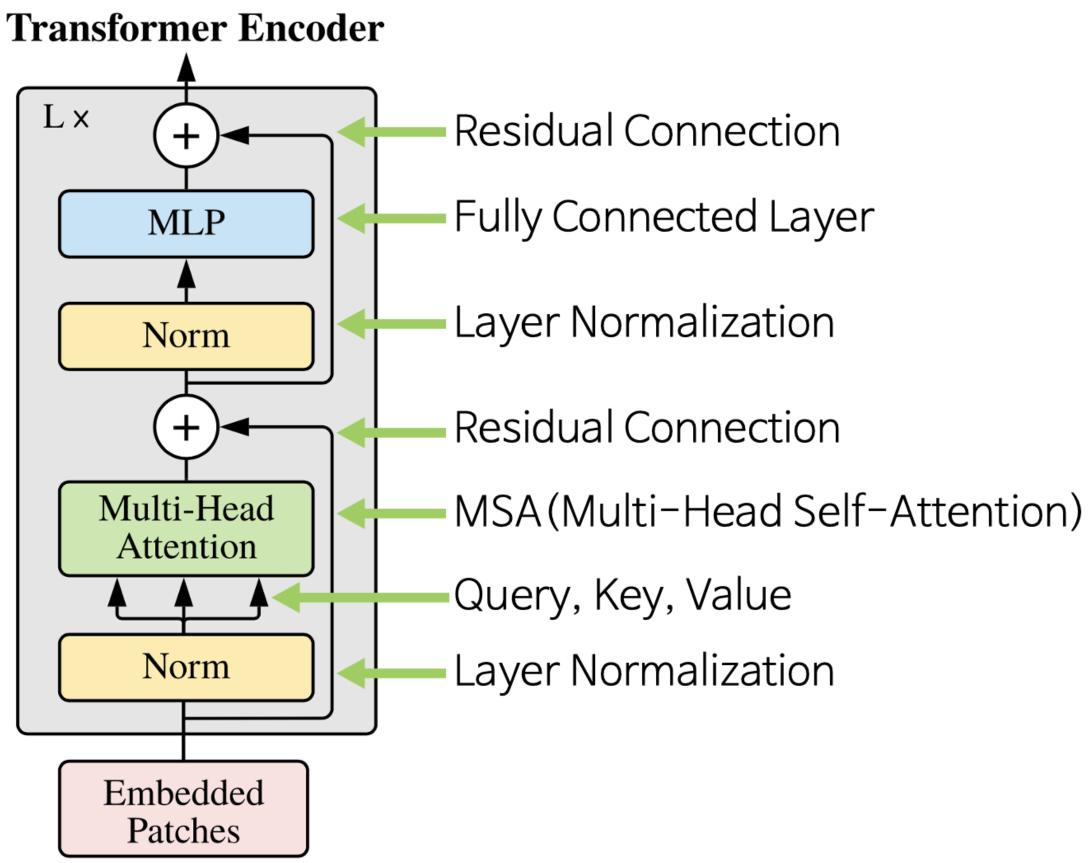
- **Transformer Encoder in Vision Transformer (ViT)**
  - **Flow of Operations (Inside One Encoder Block)**
    - **Input: Embedded Patches**
      - ✓These are the prepared tokens:
        Patch Embeddings + Class Token + Positional Embeddings.
    - **Step 1. Layer Normalization (Norm)**
      - ✓Normalizes inputs across the feature dimension.
      - ✓Prepares stable input for Attention.



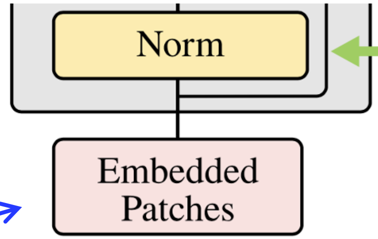
- **Step 2. Q, K, V Projection → Multi-Head Self-Attention (MSA)**
  - ✓Each token embedding is projected into **Query (Q), Key (K), and Value (V)**.
  - ✓Self-Attention lets each token attend to **all others**, including the [CLS] token.
  - ✓Multiple heads allow the model to capture **different types of relations** in parallel.

Vision Transformer Encoder Overview

- **Transformer Encoder in Vision Transformer (ViT)**
  - **Flow of Operations (Inside One Encoder Block)**
    - **Step 3. Residual Connection (+ Skip Connection)**
      - ✓Adds the original input back to the attention output.
      - ✓Ensures stable gradient flow and prevents loss of original information.
    - **Step 4. Second Layer Normalization (Norm again)**
      - ✓Normalizes before the Feed-Forward stage.
    - **Step 5. Feed-Forward Network (MLP Block)**
      - ✓A two-layer fully connected network with GELU activation.
      - ✓Expands representation capacity by applying a non-linear transformation.
    - **Step 6. Residual Connection**
      - ✓Again, adds the input of the MLP back to its output for stability.



Vision Transformer Encoder Overview



Layer Normalization

- **Transformer Encoder in Vision Transformer (ViT)**
  - **Step 1: Layer Normalization (Pre-Norm)**
    - **Equation**

$$LN(z_{l-1}), \quad l = 1 \ldots L$$

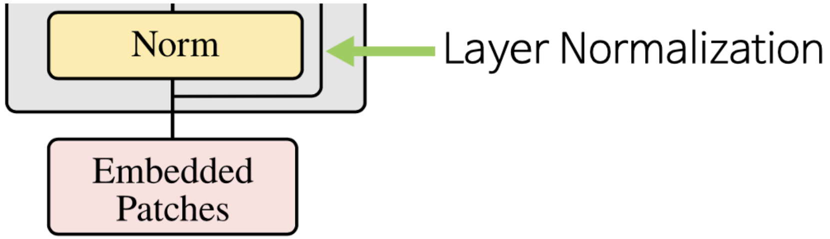
$$\boldsymbol{z_0} = \left[\boldsymbol{x_{cls}};\ ;\boldsymbol{x_p^1};\boldsymbol{E};\ \boldsymbol{x_p^2 E}\ \ldots\ \boldsymbol{x_p^N E}\right] + \boldsymbol{E_{pos}} \in \mathbb{R}^{(N+1)\times D}$$

    - **Role in ViT**
      - ✓Ensures **stable training** before Multi-Head Self-Attention (Pre-Norm).
      - ✓Normalizes input features so that **scale differences** across tokens do not disturb attention.
      - ✓Prevents exploding or vanishing gradients when stacking many Transformer layers.
    - **Motivation**
      - ✓Transformers stack dozens of layers → without normalization, training diverges quickly.
      - ✓Pre-Norm improves **gradient flow** and makes optimization easier compared to Post-Norm.

Key Points
- Layer Norm operates **independently on each token vector**.
- Keeps feature statistics balanced across dimensions.
- A crucial step to stabilize very deep architectures like ViT.

Vision Transformer Encoder Overview

- **Transformer Encoder in Vision Transformer (ViT)**
  - **Step 1: Layer Normalization (Pre-Norm)**
    - **What is Layer Normalization?**



✓ **Definition**

➢ A normalization technique that standardizes **across the hidden dimensions** of each token, not across the batch.

✓ **Equation**

$$LN(x) = \frac{x - \mu}{\sigma} \cdot \gamma + \beta$$

➢ μ, σ: mean and standard deviation computed across the feature dimension.

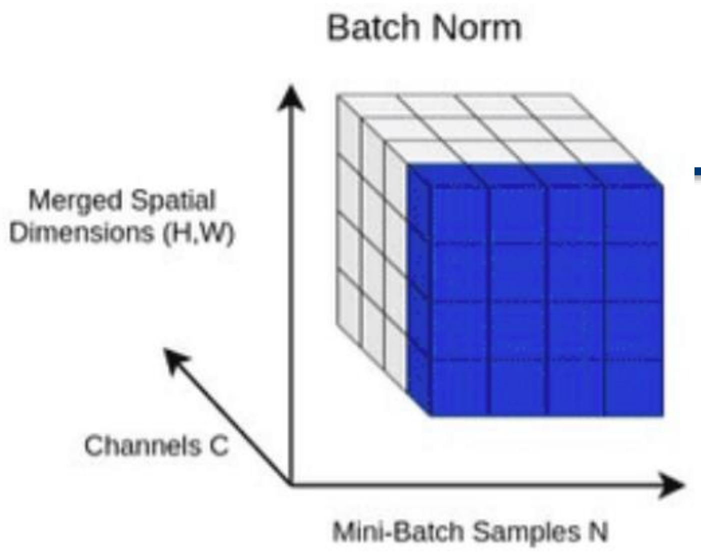
➢ γ, β: learnable scale and shift parameters.

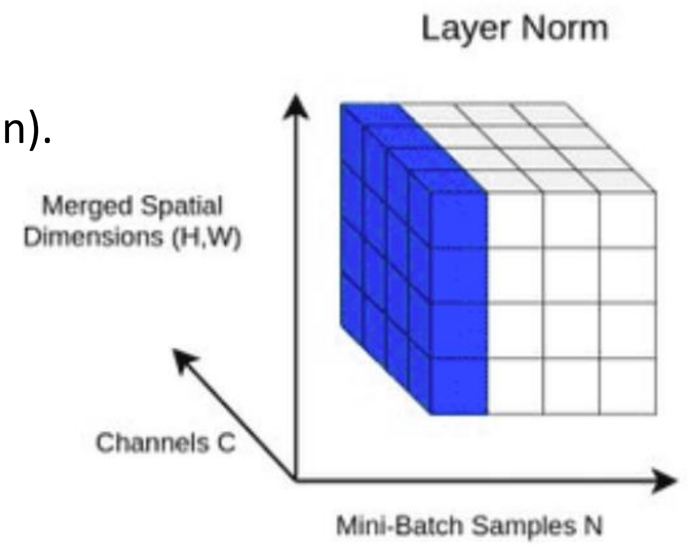
✓ **Key Intuition**

➢ Every token embedding is normalized **individually**, considering its feature vector.

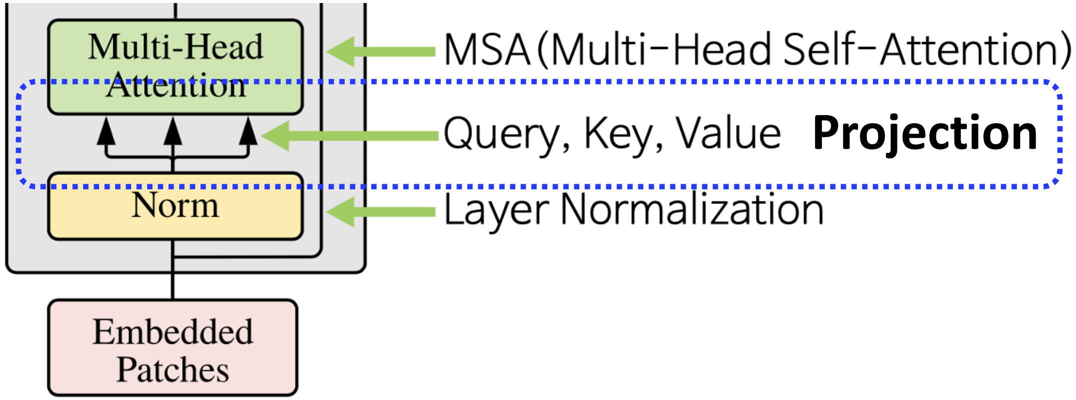
Vision Transformer Encoder Overview

- **Transformer Encoder in Vision Transformer (ViT)**
  - **Step 1: Layer Normalization (Pre-Norm)**
    - **Layer Normalization vs Batch Normalization**
      - ✓ **Batch Normalization (BN)**
        - ➢ **Normalizes _across the batch dimension_.**
          - Computes mean and variance using all samples in the batch, _per feature channel_.
        - ➢ Works well in CNNs, but **less stable in sequence models** (e.g., variable-length inputs) **and small batch-size.**
      - ✓ **Layer Normalization (LN)**
        - ➢ **Normalizes _within each sample_** across the hidden dimensions.
          - Independent of batch size → works well for Transformers (NLP, Vision).
        - ➢ Ensures consistent behavior even when **batch size is small** (important in ViT, where large images can limit batch size).
      - ✓ **Summary**
        - ➢ **BN = normalize across batch** (good for CNNs).
        - ➢ **LN = normalize across features** (good for Transformers).





Vision Transformer Encoder Overview



- **Transformer Encoder in Vision Transformer (ViT)**
  - **Step 2: Q, K, V Projection – How to make Q, K, and V**
    - **Equation**
      - ✓Input

$$LN(Z) = \hat{Z} \in \mathbb{R}^{N \times D}$$

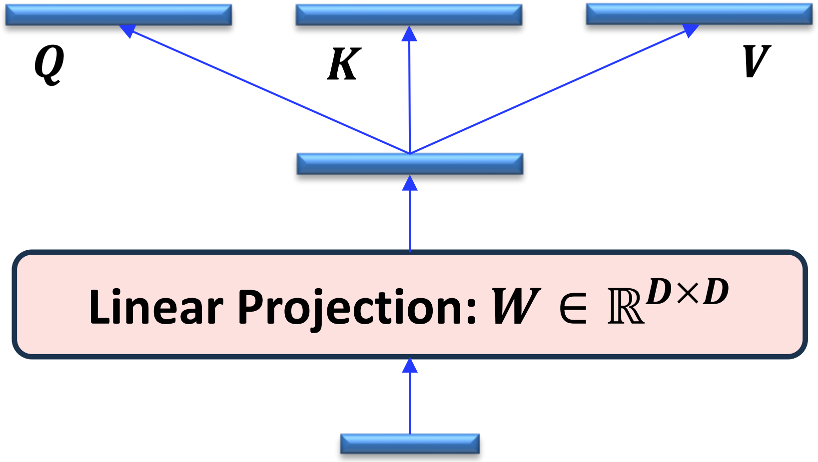
➢ where **N = number of tokens (patches)**, **D = embedding dimension**.

✓**1. Projection**

$$Q = \hat{Z}W_Q, \quad K = \hat{Z}W_K, \quad V = \hat{Z}W_V; \; Q, K, V \in \mathbb{R}^{N \times D}$$

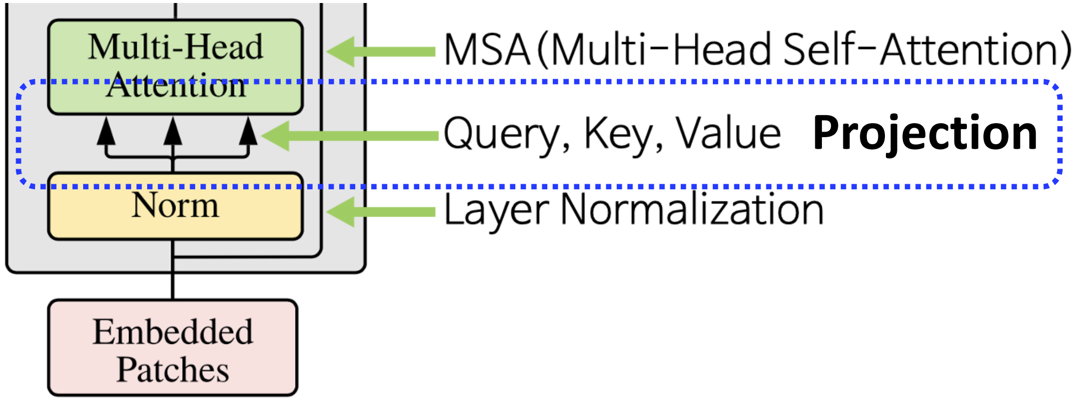
➢where $W_Q, W_K, W_V \in \mathbb{R}^{D \times D}$ are learnable projection matrices.

$$Q, K, V \in \mathbb{R}^{N \times D}$$



$$LN(Z) = \hat{Z} \in \mathbb{R}^{N \times D}$$

Vision Transformer Encoder Overview



- **Transformer Encoder in Vision Transformer (ViT)**
  - **Step 2: Q, K, V Projection – How to make Q, K, and V**
    - **Equation**
      - ✓ **2. Head-wise Split – For multi-head attention with h heads**

$$D_h = \frac{D}{h}$$

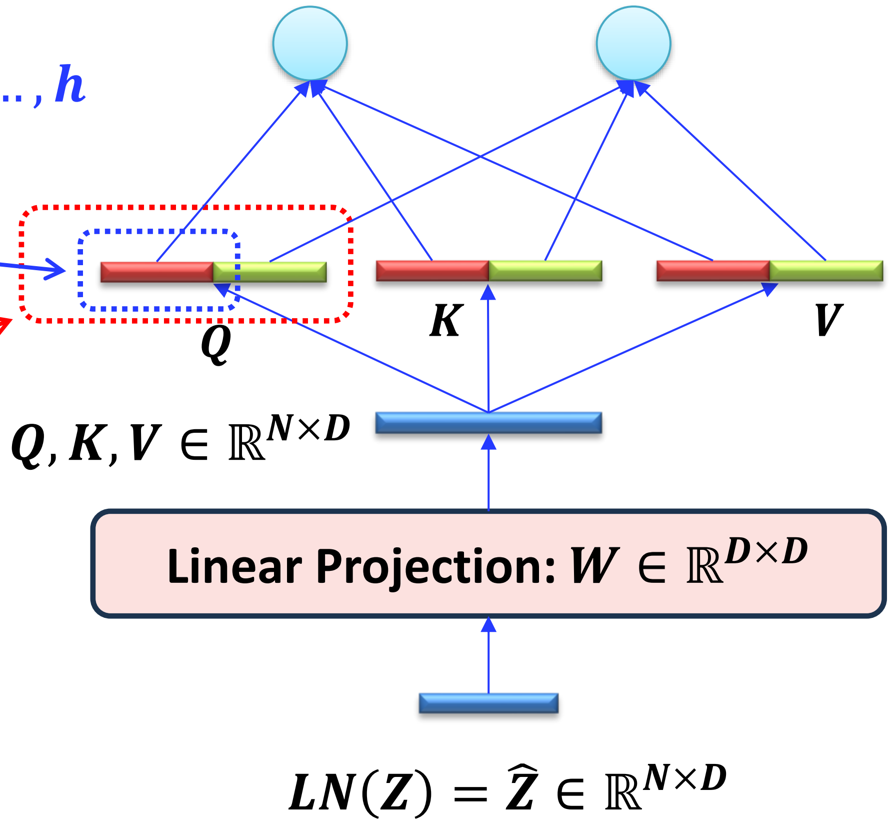
$$Q_i, K_i, V_i \in \mathbb{R}^{N \times D_h}; \quad where\ i = 1, \dots, h$$

➢Each projection is reshaped as

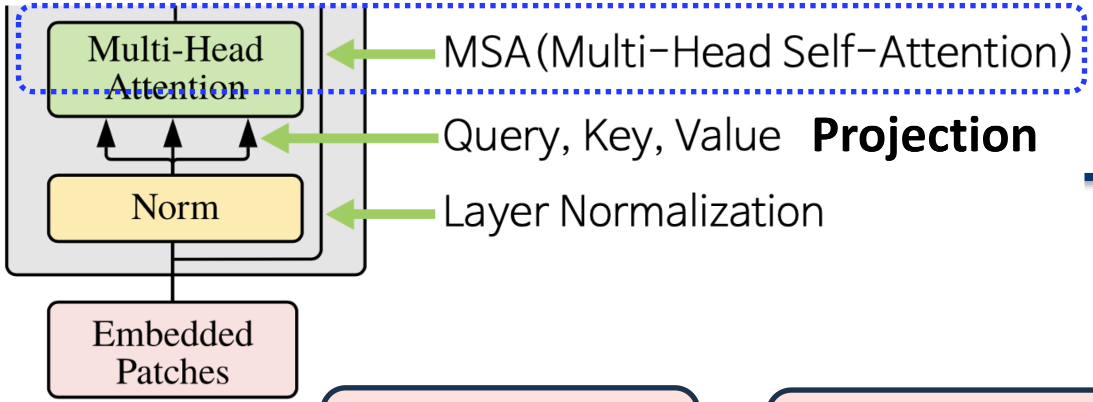
$$Q, K, V \in \mathbb{R}^{N \times D} \quad \longrightarrow \quad [h, N, D_h]$$

$$Q, K, V \in \mathbb{R}^{h \times N \times D_h}$$

$if\ h = 2$



Vision Transformer Encoder Overview



- Transformer Encoder in Vision Transformer (ViT)
  - **Step 2: Multi-Head Self-Attention (MSA)**
    - Equation
      - ✓ **3. Per-head attention – For each head i**

$if\ h = 2$

$$Q_i, K_i, V_i \in \mathbb{R}^{N \times D_h}; \quad where\ i = 1, \dots, h$$

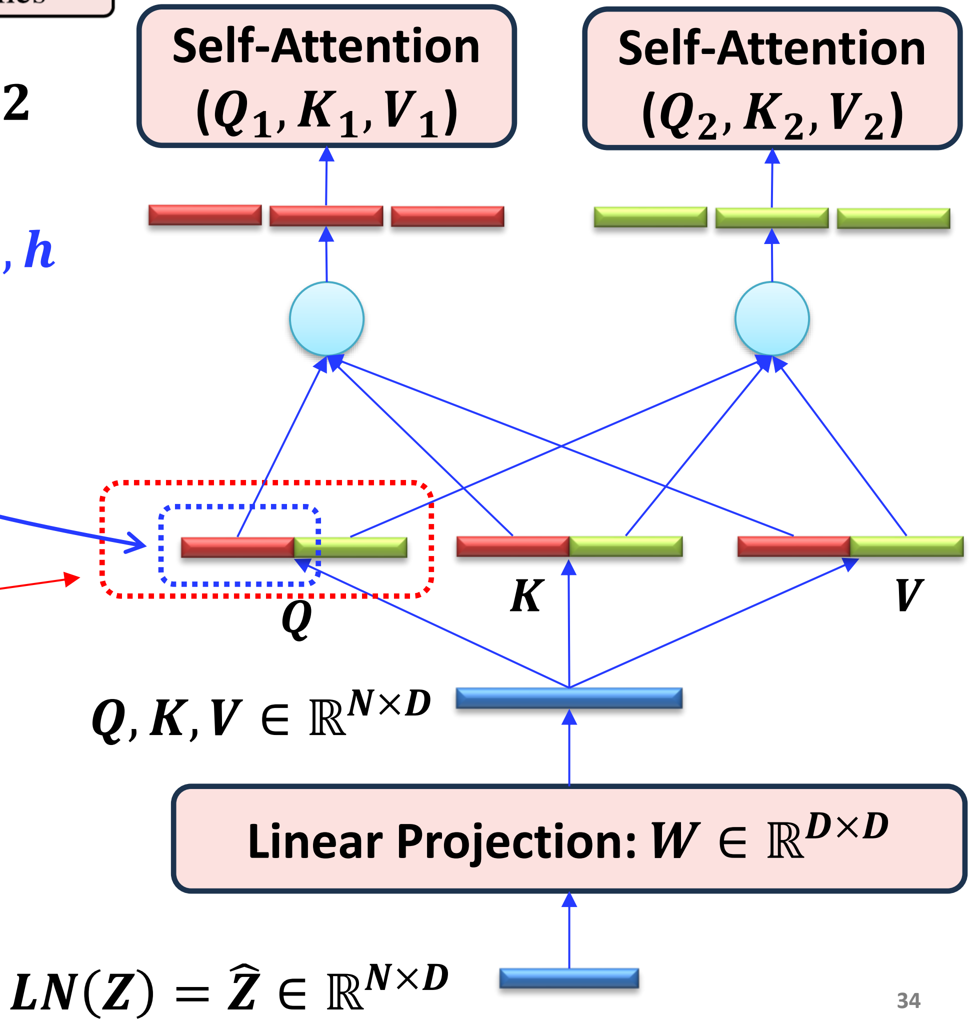
$$Q, K, V \in \mathbb{R}^{h \times N \times D_h}$$

$$Q_i, K_i, V_i \in \mathbb{R}^{N \times D_h}$$

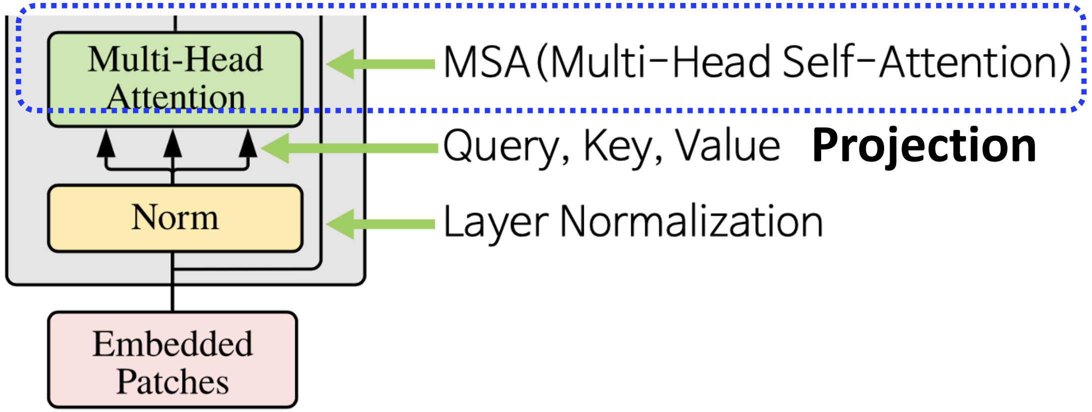
$$A_i = softmax\left(\frac{Q_i K_i^\top}{\sqrt{D_h}}\right)$$

$$O_i = Self\text{-}Attention(Q_i, K_i, V_i) = A_i V_i \in \mathbb{R}^{N \times D_h}$$

$where\quad N: \#tokens, \qquad D: embedding\ dim$



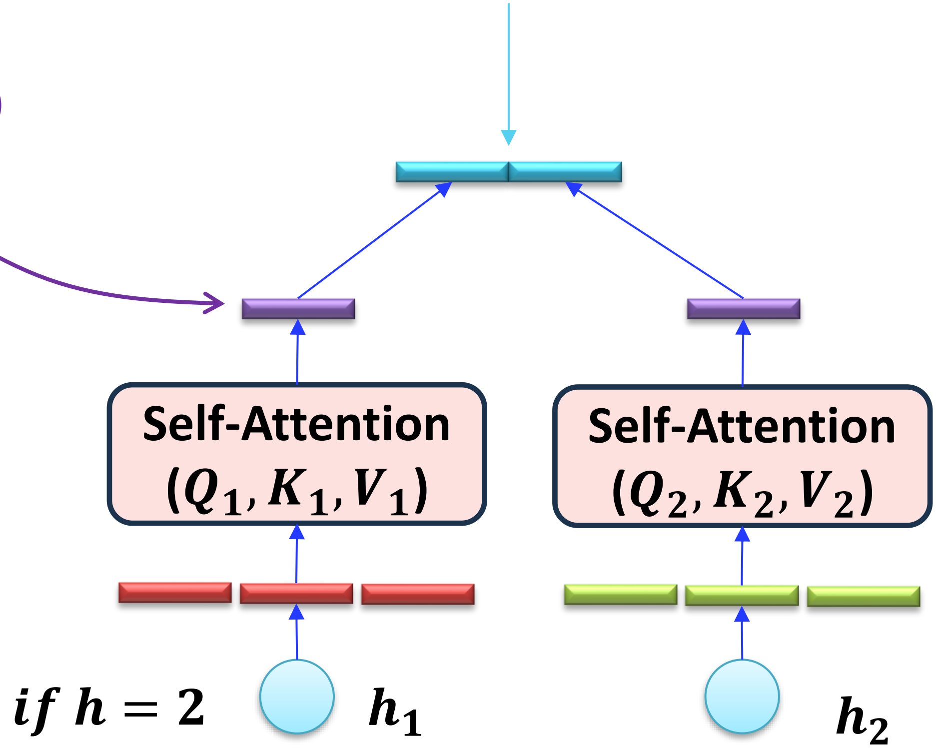
Vision Transformer Encoder Overview



- Transformer Encoder in Vision Transformer (ViT)
  - Step 2: Multi-Head Self-Attention (MSA)
    - Equation
      - ✓4. Concatenate heads (channel-wise)

$$O = Concat(O_1, \dots, O_h) \in \mathbb{R}^{N\times(hD_h)} = \mathbb{R}^{N\times D}$$

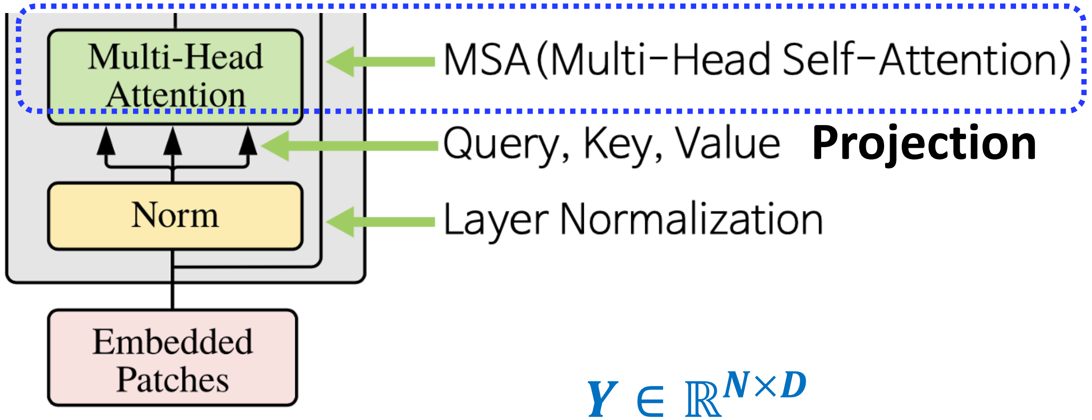
$$O_i = Self-Attention(Q_i, K_i, V_i) = A_i V_i \in \mathbb{R}^{N\times D_h}$$



$$O = Concat(O_1, \dots, O_h) \in \mathbb{R}^{N\times(hD_h)} = \mathbb{R}^{N\times D}$$

➢Each head output is $[N, D_h]$;
after concat you get $[N, h \cdot D_h] = [N, D]$; $where\ D_h = \frac{D}{h}$.

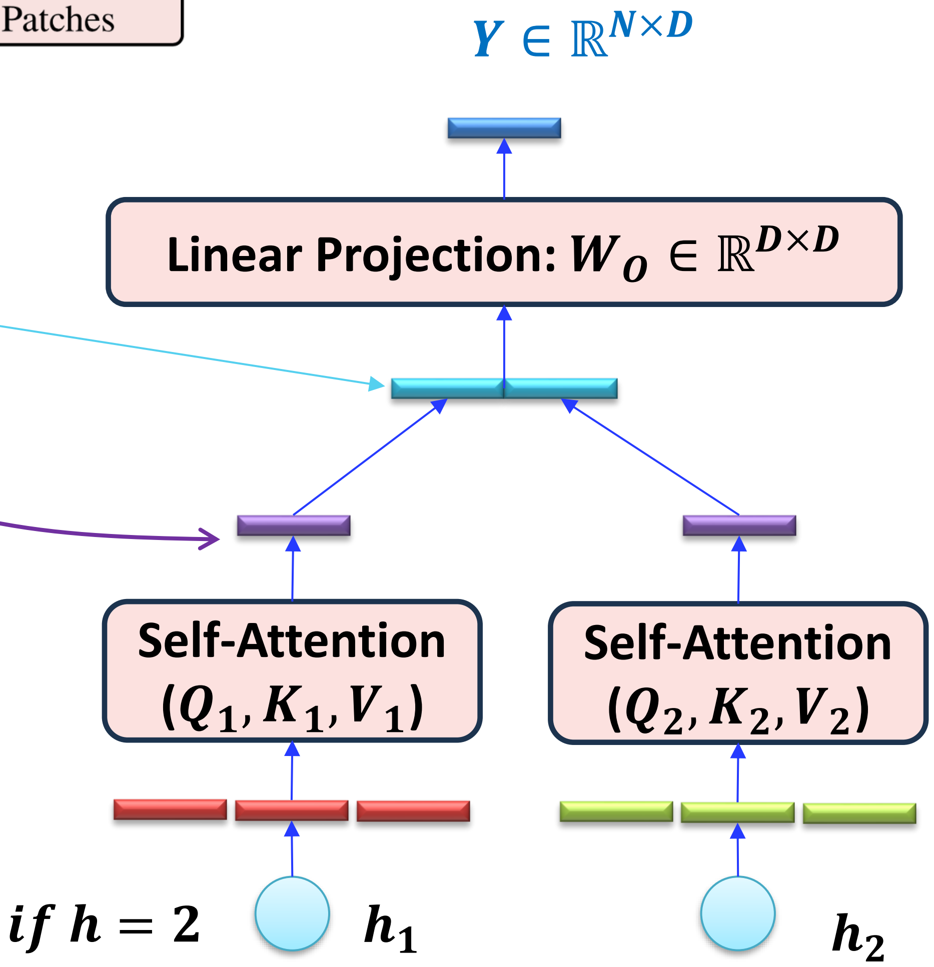
Vision Transformer Encoder Overview



- **Transformer Encoder in Vision Transformer (ViT)**
  - **Step 2: Multi-Head Self-Attention (MSA)**
    - Equation
      - ✓**5. Output projection (mix heads)**

$$O = Concat(O_1, \dots, O_h) \in \mathbb{R}^{N\times(hD_h)} = \mathbb{R}^{N\times D}$$

$$O_i = Self-Attention(Q_i, K_i, V_i) = A_i V_i \in \mathbb{R}^{N\times D_h}$$



$$Y = O\,W_O, \qquad W_O \in \mathbb{R}^{D\times D}, \qquad Y \in \mathbb{R}^{N\times D}$$

- Purpose: **couple the heads** (a learned linear mix) and return to the model width D.
- Without W_O, heads would remain independent concatenations.

Vision Transformer Encoder Overview

- **Transformer Encoder in Vision Transformer (ViT)**
  - **Step 2: Multi-Head Self-Attention (MSA)**
    - **Why Do We Project Twice in Attention?**
      - ✓ **Part 1 — Q, K, V Projection**
        - ➢ **Why?** To create **three different views** of the same embedding.

$$Q = ZW_Q, \qquad K = ZW_K, \qquad V = ZW_V$$

- **Q (Query):** what this token is "asking for"
- **K (Key):** how other tokens "describe themselves"
- **V (Value):** the actual **content information**

➢ **Reason**

- **If Q, K, V were the same**, ***attention would collapse***.
  (**NO distinction** between asking, matching, and answering).
- Projection lets the model learn **different subspaces** for "matching" and "carrying info."
- Each head then focuses on ***different aspects*** **(local edges, global shape, texture, etc.)**.

Vision Transformer Encoder Overview

- **Transformer Encoder in Vision Transformer (ViT)**
  - **Step 2: Multi-Head Self-Attention (MSA)**
    - **Why Do We Project Twice in Attention?**
      - ✓ **Part 2 — Output Projection (After Concatenation)**
        - ➢ **After per-head attention:** Each head produces $\boldsymbol{O_i} \in \mathbb{R}^{N \times D_h}$.

$$\boldsymbol{Concatenation} \rightarrow \boldsymbol{O} = \boldsymbol{Concat}(\boldsymbol{O_1}, \dots, \boldsymbol{O_h}) \in \mathbb{R}^{N \times (hD_h)} = \mathbb{R}^{N \times D}$$

        - ➢ **Why another projection?**

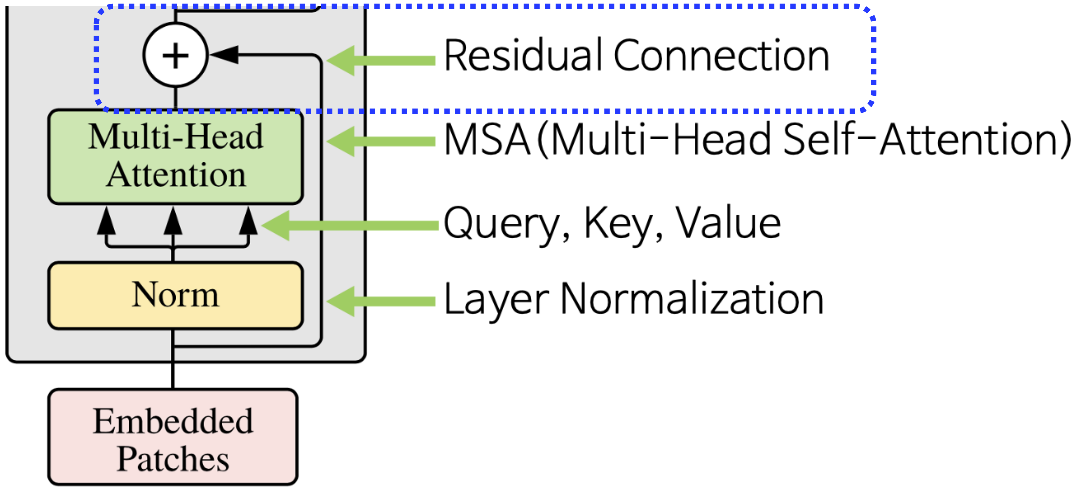
$$\boldsymbol{Y} = \boldsymbol{O}\boldsymbol{W_O}, \qquad \boldsymbol{W_O} \in \mathbb{R}^{D \times D}$$

- To ***mix information across heads*** (otherwise heads remain independent).
- To bring the concatenated result **back into the model's fixed embedding dimension $\boldsymbol{D}$**.
- Ensures the multi-head mechanism acts as ***diverse views + integration***, not just parallel streams.

Vision Transformer Encoder Overview

- **Transformer Encoder in Vision Transformer (ViT)**
  - **Step 2: Multi-Head Self-Attention (MSA)**
    - **Why Do We Project Twice in Attention?**
      - ✓ **Key Points**
        - ➢ Projection **(1st time)** → separates roles into **Q, K, V**.
        - ➢ Multi-head splitting → multiple independent subspaces, each learning a different relation.
        - ➢ Projection **(2nd time)** → mixes those subspaces back together, so the model uses them jointly.
        - ➢ Think of it as: **asking the same question in many ways, then combining all the answers into one clear representation.**

Vision Transformer Encoder Overview

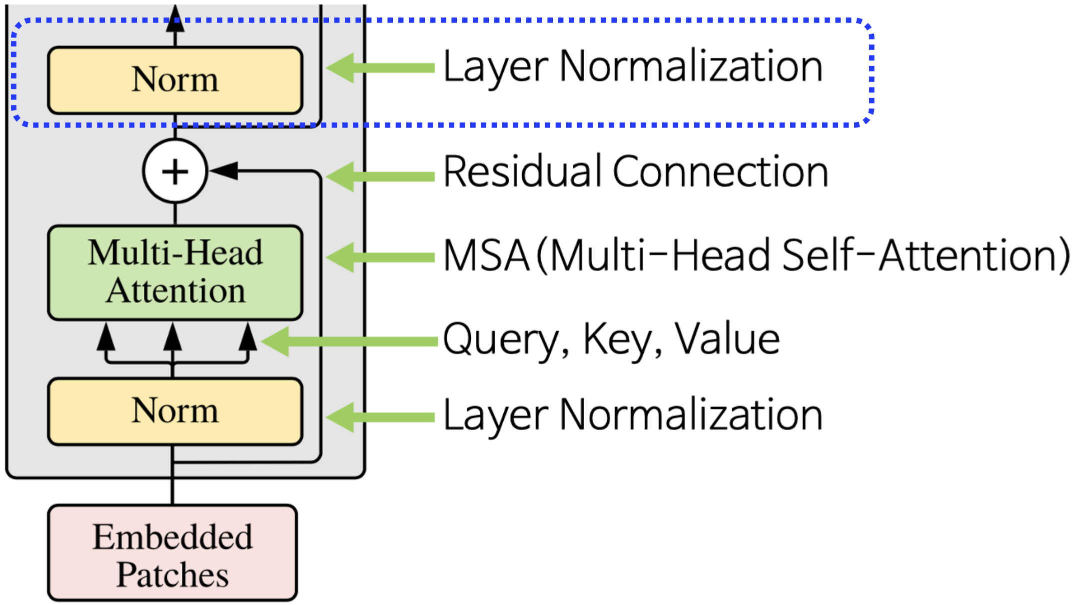


- **Transformer Encoder in Vision Transformer (ViT)**
  - **Step 3: Residual Connection (after MSA)**
    - **Equation**

$$z'_\ell = MSA\big(LN(z_{\ell-1})\big) + z_{\ell-1}, \qquad \ell = 1 \dots L$$

    - **Role & Purpose**
      - ✓Add **shortcut connection** to stabilize training.
      - ✓Prevent vanishing gradients by allowing gradient flow directly.
      - ✓Ensure **information preservation**: even if MSA fails, the model can keep the original representation.
    - **Why Important in ViT?**
      - ✓Vision tasks involve **long sequences (many patches)**, making deep training unstable.
      - ✓Residual paths act as an **“information highway”**, supporting both learning new features and keeping old ones.

Vision Transformer Encoder Overview

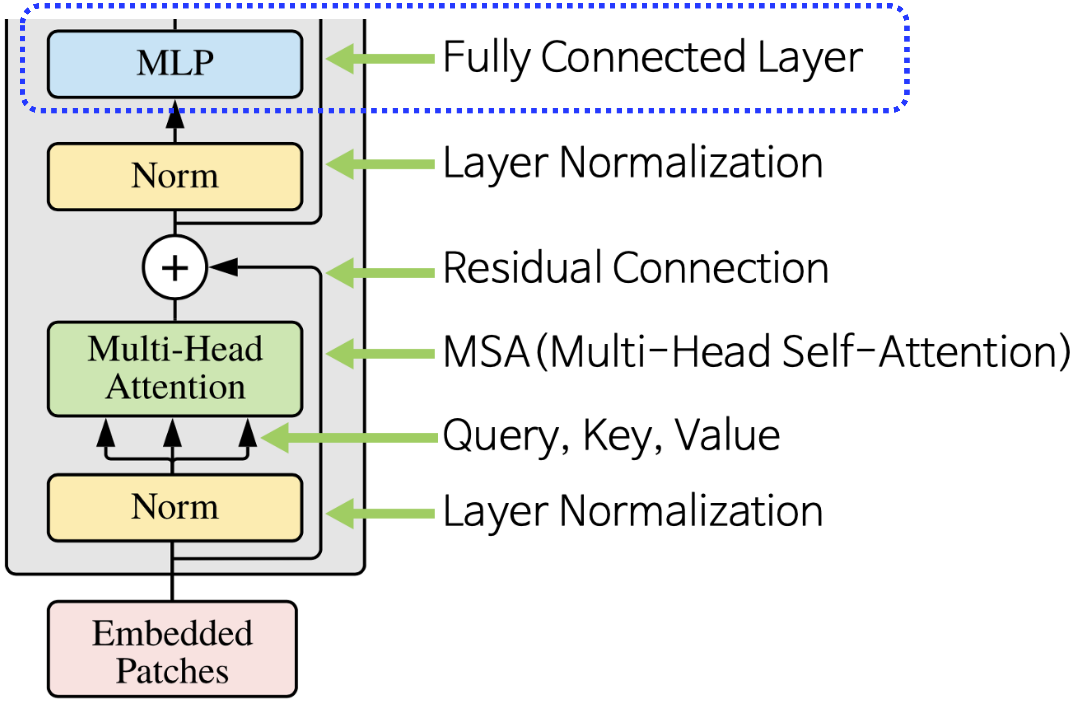


- **Transformer Encoder in Vision Transformer (ViT)**
  - **Step 4: Second Layer Normalization**
    - **Equation**

$$\hat{\mathbf{z}}'_\ell = LN(\mathrm{z}'_\ell), \quad \ell = 1 \dots L$$

    - **Why Another LN?**
      - ✓After residual addition, activations may drift (scale/shift).
      - ✓LN re-centers and re-scales token embeddings before feeding into the **Feed-Forward Network (MLP Block)**.
      - ✓Stabilizes distribution → improves convergence.
    - **Key Point**
      - ✓LN is applied **before every major sub-block** (Pre-Norm design).
      - ✓Without this, deep ViTs would suffer from instability.

Vision Transformer Encoder Overview



- **Transformer Encoder in Vision Transformer (ViT)**
  - **Step 5: Feed-Forward Network (MLP Block)**
    - **Equation**

$$z_\ell = MLP\big(LN(z'_\ell)\big) = MLP(\hat{z}'_\ell), \qquad \ell = 1 \dots L$$

✓Where

$$MLP(x) = W_2 \cdot GELU(W_1 x + b_1) + b_2$$

Why GELU (Gaussian Error Linear Unit)?
- Captures **small variations** effectively (important for vision tasks).
- Empirically improves convergence and accuracy in Transformer models.

    - **Role**

✓***Token-wise transformation***: each patch embedding processed independently.

✓Expands embedding dimension (e.g., $D \rightarrow 4D \rightarrow D$) to ***capture richer features***.

    - **Why Fully-Connected Layers (Not CNN)?**

✓Unlike CNN, ViT has **no spatial inductive bias** — MLP just processes each token vector.

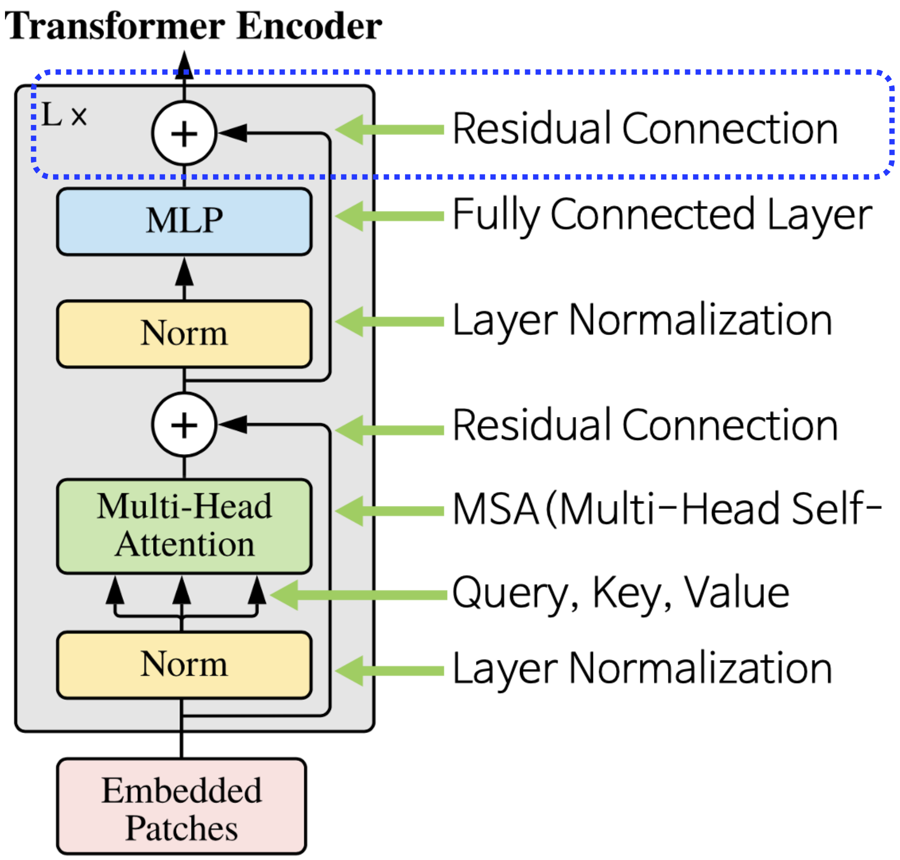
✓Efficient for token-wise non-linear transformation.

Vision Transformer Encoder Overview

- **Transformer Encoder in Vision Transformer (ViT)**
  - **Step 6: Residual Connection (after MLP)**
    - **Equation**

$$z_\ell = MLP\big(LN(z'_\ell)\big) + z'_\ell = MLP(\hat{z}'_\ell) + z'_\ell, \qquad \ell = 1 \ldots L$$

    - **Why Another Residual?**
      - ✓ ***Same motivation***: stabilize deep training, prevent information loss.
      - ✓ After the **MLP transformation**, residual ensures the model can fallback on the input representation.
      - ✓ Enables stacking many Transformer layers (L) without degradation.
    - **Key Point**
      - ✓ Every Transformer block = **(LN → Sub-layer → Residual)** × 2.
        - ➢ First sub-layer = MSA
        - ➢ Second sub-layer = MLP



ViT Output and Classification Head

- **Only the [CLS] token is used for final prediction**
  - **Output of Vision Transformer (ViT)**
    - **Equation**

$$y = LN(z_L^0)$$

✓ z_L^0: the **[CLS] token representation** after passing through L **encoder layers.**

✓ LN: final Layer Normalization applied to stabilize the representation.

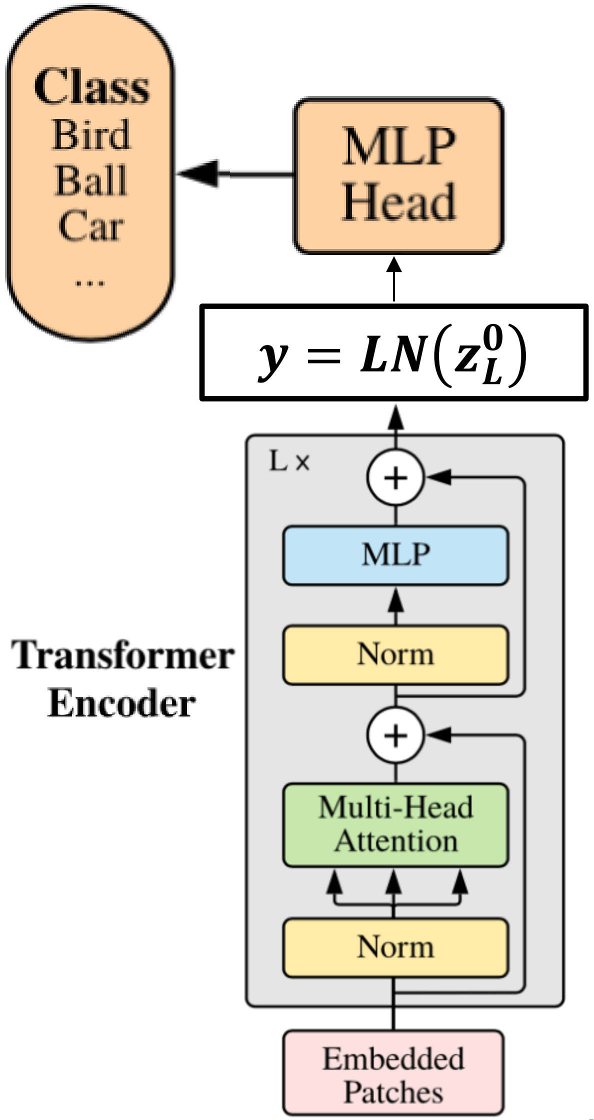
✓ y: serves as the input to the MLP Head for classification.

    - **Classification Head**

✓ After the Transformer Encoder, ***ONLY the [CLS] token is used***.

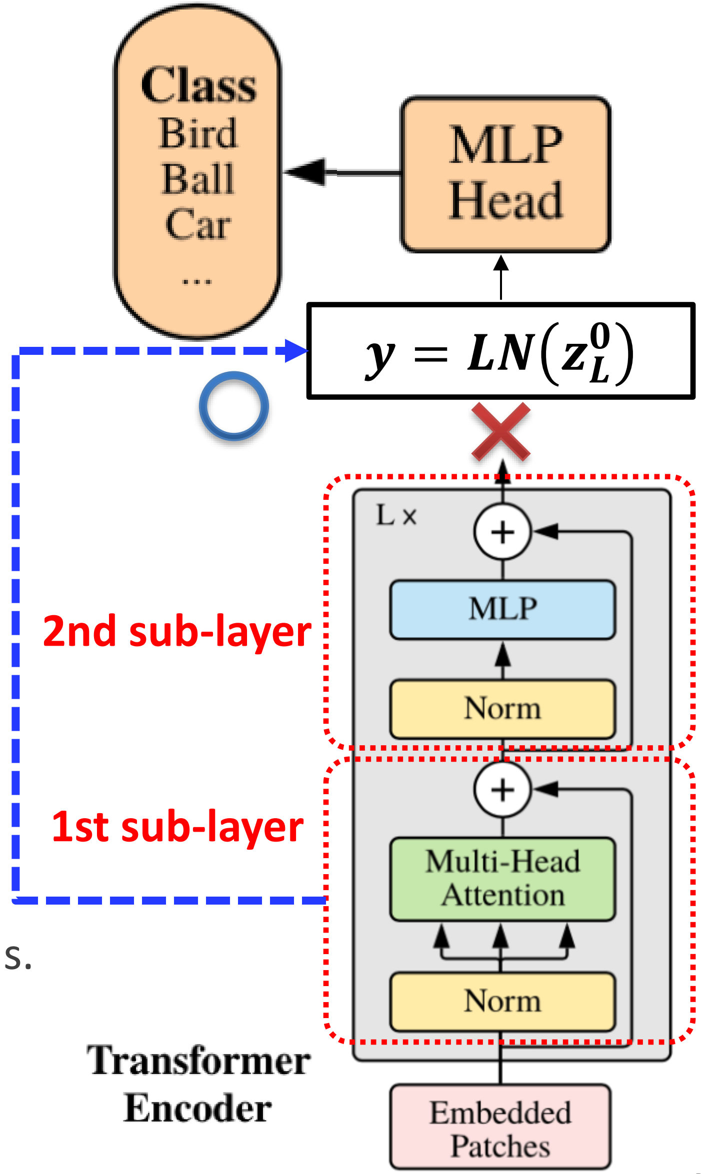
✓ This token has aggregated **global information** from all patches through multiple self-attention layers.

✓ *y is passed through an **MLP Head*** (usually one or two fully connected layers with softmax at the end).



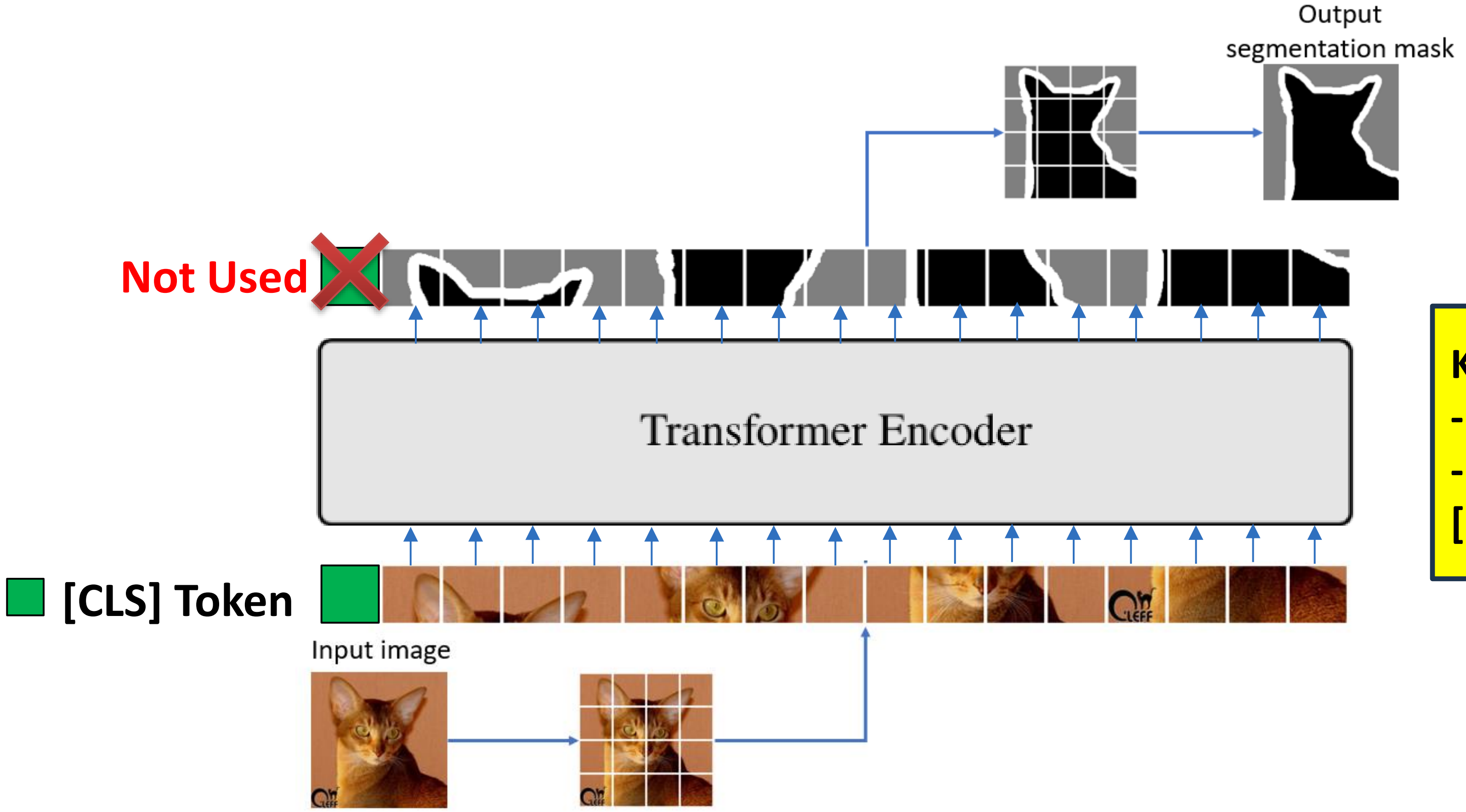
ViT Output and Classification Head

- **Only the [CLS] token is used for final prediction**
  - **Where does z_L^0 from exactly?**
    - Each Transformer block ℓ has ***two sub-layers***
      - ✓***1st sub-layer***: **Multi-Head Self-Attention (MSA) + Residual**
      - ✓***2nd sub-layer***: **MLP (Feed-Forward Network) + Residual**
    - z_L^0
      - ✓The ***intermediate representation*** after the ***first sub-layer (MSA + Residual)*** of the ***last Transformer block (L)***.
  - **Why Only the [CLS] Token?**
    - It acts as a **summary representation** of the entire image.
    - Attention layers allow the [CLS] token to gather information from all other tokens.
    - This makes it sufficient for **classification tasks**, while ignoring other tokens.



Beyond Classification: Other Tasks with ViT

- **Segmentation and Detection need spatial outputs, not just [CLS]**
  - In **Classification**, we only use the **[CLS] token**, because it already summarizes global information.



Key Points
- ViT is **NOT limited** to classification.
- Depending on the task, we either use **[CLS] (global)** or **all patches (spatial)**.

  - But for **other vision tasks** like **Segmentation or Detection**, we need ***spatial details***.
    - Instead of using only the **[CLS] token**, ViT uses ***all patch embeddings***.

Summary: Vision Transformer (ViT)

- **Key Takeaways**
  - **From CNNs to Transformers**
    - **CNNs**: strong inductive bias (locality, translation equivariance).
    - **ViT**: weak inductive bias, relies on **data-driven global self-attention**.
  - **Data Preparation**
    - Split image into patches → Flatten → Linear Projection.
    - Add [CLS] token (global representation).
    - Add positional embeddings (spatial order).
  - **Transformer Encoder**
    - Repeated $\boldsymbol{L}$ blocks
      - ✓LN → QKV Projection → Multi-Head Self-Attention → Residual.
      - ✓LN → MLP (with GELU) → Residual.
    - Residual & $\boldsymbol{LN}$ ensure stability in deep models.

Summary: Vision Transformer (ViT)

- **Key Takeaways**
  - **Output for Classification**
    - Only the **[CLS] token** is used after the final block.
    - Passed through MLP Head → class probabilities.
  - **Beyond Classification**
    - For tasks like **Segmentation, Detection**
      - ✓Use **all patch embeddings** instead of only [CLS].
      - ✓Enables pixel-level or region-level predictions.