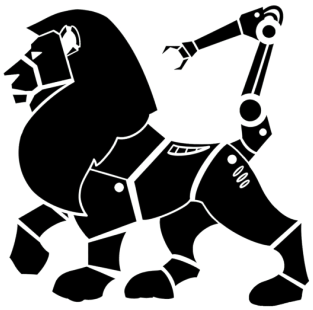


REPORT

# Vision Transformer Implementation for CIFAR-10 Subset

*authors:*
Ahmad HARKOUS
An Vinh Lala
Valentin Dauzere-Peres

Nov, 2024

# Table of contents

# 1 Introduction

This report describes the implementation of a Vision Transformer (ViT) architecture based on the paper "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" by Dosovitskiy et al. (2020).

The primary focus of this project was to understand and implement the core architectural components of ViT, rather than achieving state-of-the-art performance. Simplifications were made to the dataset, architecture, and training procedure to align with computational constraints while retaining the fundamental characteristics of the ViT model.

# 2 Dataset Description

The CIFAR-10 dataset consists of 60,000 32x32 color images across 10 classes (e.g., airplanes, automobiles, birds). For this project:

- **Training Data**: A subset of 20% of the training set was used (10,000 images).
- **Test Data**: The full 10,000-image test set was retained to evaluate the model's generalization.
- No data augmentation was applied, as the focus was on implementing the architecture rather than optimizing performance.

# 3 Selected Architecture and Literature Review

The Vision Transformer architecture treats images as sequences of patches, similar to words in a sentence for NLP transformers. Key components of ViT include:

### A - Patch Embeddings:

The process of patch embedding involves transforming an input image into a sequence of fixed-size patches and projecting them into a high-dimensional vector space:

1. **Dividing the Image into Patches**:

   Given an input image $I$ with dimensions $H \times W \times C$, where:

   - $H$: Height of the image.
   - $W$: Width of the image.
   - $C$: Number of channels (e.g., 3 for RGB).

   The image is divided into non-overlapping patches of size $P \times P$. The total number of patches $N$ is calculated as:

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

2. **Flattening Each Patch**:

   Each patch of size $P \times P \times C$ is flattened into a 1D vector of size:

$$\text{Patch Size} = P^2 \cdot C$$

   For example, if $H = W = 224$, $C = 3$ (RGB), and $P = 16$:

$$N = \frac{224 \cdot 224}{16 \cdot 16} = 196 \quad \text{(number of patches)}$$

   Each patch has:

$$P^2 \cdot C = 16 \cdot 16 \cdot 3 = 768 \quad \text{(features per patch)}.$$

3. **Linear Projection**:

   Each flattened patch is linearly projected into a $D$-dimensional embedding space:

$$\mathbf{e}_p = \mathbf{W} \cdot \mathbf{x}_p + \mathbf{b}$$

   where:

   - $\mathbf{x}_p$: Flattened patch (of size $P^2 \cdot C$).
   - $\mathbf{W}$: Learnable weight matrix of size $D \times (P^2 \cdot C)$.
   - $\mathbf{b}$: Bias term.

   The result is an embedding $\mathbf{e}_p$ of size $D$ for each patch.

$$\mathbf{E} = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_N]$$

4. **Prepending the [CLS] Token:**

   A learnable **[CLS] token** is prepended to the sequence of patch embeddings. This token is used to aggregate global information about the image. This token will eventually be used for the final classification. The sequence becomes:

$$\mathbf{E}_{\text{input}} = [[\text{CLS}], \mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_N]$$

   where $\mathbf{E}_{\text{input}}$ is of size $(N + 1) \times D$.

5. **Positional Encoding:**

   Positional encodings are added to each token in the sequence to inject information about the relative or absolute position of the patches in the original image. This is done element-wise:

$$\mathbf{E}_{\text{input}} = \mathbf{E}_{\text{input}} + \mathbf{P}$$

   where $\mathbf{P}$ is the positional encoding matrix of size $(N + 1) \times D$.

This sequence serves as the input to the transformer encoder.

**B - Transformer Encoder:**

The transformer processes the sequence of patch embeddings augmented with positional encodings. Each transformer layer consists of:

1. **Multi-Head Self-Attention (MHSA)**:

   Each patch embedding interacts with others through self-attention. For each patch:

   - Queries ($Q$), Keys ($K$), and Values ($V$) are computed:
   $$Q = \mathbf{E} \cdot \mathbf{W}_Q, \quad K = \mathbf{E} \cdot \mathbf{W}_K, \quad V = \mathbf{E} \cdot \mathbf{W}_V$$
   where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$ are learnable weight matrices.
   - Scaled Dot-Product Attention:
   $$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{D_k}}\right) \cdot V$$
   where $D_k$ is the dimensionality of $K$.

2. **Feed-Forward Networks (FFN)**:

   Each attention output is passed through a feed-forward network. The FFN uses the **GELU** (Gaussian Error Linear Unit) non-linearity instead of ReLU:
   $$\text{FFN}(x) = \text{GELU}(x \cdot \mathbf{W}_1 + \mathbf{b}_1) \cdot \mathbf{W}_2 + \mathbf{b}_2$$
   where GELU is defined as:
   $$\text{GELU}(x) = x \cdot \Phi(x)$$
   with $\Phi(x)$ being the cumulative distribution function of the standard normal distribution, or equivalently:
   $$\text{GELU}(x) = 0.5x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right)$$

**Layer normalization(LN) is applied before every block, and residual connections after every block of both the MHSA and FFN(MLP).**

**C - Classification Head:**

After passing through the Transformer encoder layers, the **[CLS] token** now contains the aggregated representation of the entire image. The final output corresponding to the **[CLS] token is**:
$$\mathbf{z}_{\text{CLS}} = \text{Transformer Output[CLS]}$$
This is a vector of size $D$, which represents the global features of the image.

The output of the **[CLS] token** is passed through a **classification head**, which is typically a simple **fully connected layer (FC)**:
$$\text{Logits} = \mathbf{z}_{\text{CLS}} \cdot \mathbf{W}_{\text{head}} + \mathbf{b}_{\text{head}}$$
where:

- $\mathbf{W}_{\text{head}}$ is a weight matrix of size num_classes $\times D$,
- $\mathbf{b}_{\text{head}}$ is the bias term.

The final output is a vector of size num_classes, containing the predicted class logits.

**Summary of Vector Representations at Each Step**

1. **Input Image (H x W x C)**: The image is divided into patches and flattened.
2. **Patch Embeddings**: Each patch is flattened and projected to a $D$-dimensional space.
3. **Prepending the [CLS] Token**: The **[CLS] token** is prepended before patch embeddings.
4. **Positional Encoding**: Positional encodings are added to the embeddings.
5. **Transformer Encoder (MHSA + FFN)**: The sequence is passed through the Transformer layers.
6. **Extracting the [CLS] Token**: The **[CLS] token** is extracted after the Transformer layers.
7. **Classification Head**: The final **[CLS] token** is passed through a linear layer to produce class logits.

**Simplifications made:**

- Using fewer Transformer layers (**6** layers instead of the original 12).
- Reducing embedding dimensions to **64** and limiting self-attention heads to **4**.

# 4 Experiments and Results

The experiments aimed to verify the functionality of the ViT implementation. Key aspects of the setup included the following.

- **Optimizer**: Adaptive Moment Estimation with a fixed learning rate of 0.001.
- **Batch Size**: 64.
- **Epochs**: 10, limited due to computational constraints.

**Training Observations:**

- The training loss decreased consistently, reaching approximately 2.3 by the 10th epoch. It shows steady improvement.
- Test accuracy remained constant at ∼10%, confirming the model's inability to generalize. **This highlights underfitting due to reduced training data and simplified architecture**.

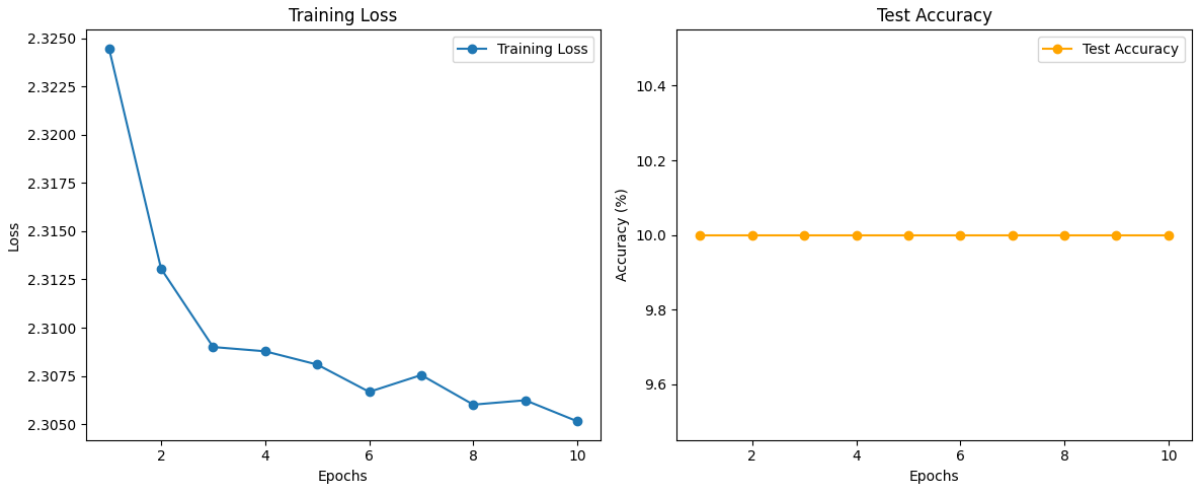


Figure 1:

# 5 Simplifications and Reasoning

Several simplifications were made to balance computational constraints and educational objectives:

- **Architecture Simplifications**:
  - Reduced embedding dimensions, attention heads, and layers to lower memory usage and speed up training.
  - These changes limit the model's capacity, explaining its underperformance but maintaining the structural essence of ViT.
- **Dataset Reduction**:
  - Only 20% of CIFAR-10's training set was used, minimizing computational demand.
  - A larger dataset would improve results but was excluded to keep the implementation lightweight.
- **Hyperparameters**:
  - A fixed learning rate and fewer epochs simplified tuning, emphasizing architecture implementation over optimization.
- **No Augmentation**:
  - Augmentation could improve generalization but was omitted to focus solely on testing the architecture.

# 6 Conclusion

This project successfully implemented a simplified Vision Transformer and demonstrated its functionality using a subset of CIFAR-10. While the results highlight significant underfitting, they align with the project's objective of validating the architecture. Future work could expand the training dataset, include augmentation, and experiment with more expressive hyperparameters to achieve competitive performance.

The article also concluded that ViT transfromers achieved competitive results on image classification tasks, particularly on large datasets like ImageNet.