Vision Transformer: End-Term Evaluation Report

Priyanshu Yadav (240812)

February 8, 2026

1 Introduction

In this project, I undertook the challenge of implementing a Vision Transformer (ViT) from scratch using the PyTorch library. The primary goal was to explore the transition from traditional Convolutional Neural Networks (CNNs) to a Transformer-based architecture for image classification[cite: 6]. Unlike CNNs, which utilize sliding windows and local filters to capture spatial features, the ViT architecture treats an image as a sequence of patches[cite: 7]. This allows the model to leverage self-attention mechanisms, originally developed for sequence modeling in NLP, to understand global relationships within an image right from the initial layers[cite: 7, 79]. For this evaluation, I trained and tested the model on the CIFAR-10 dataset, which served as a benchmark for assessing the model's efficiency on small-scale image data[cite: 6, 11].

2 Data Preprocessing and Augmentation

The CIFAR-10 dataset provides 60,000 color images categorized into 10 classes, with each image having a resolution of 32×32 pixels[cite: 11, 12, 15]. Given the inherent "data-hungry" nature of Transformers, I had to implement a comprehensive preprocessing pipeline to prevent the model from simply memorizing the training samples[cite: 100].

2.1 Transformation Strategy

The following operations were applied to the training pipeline to improve generalization:

- **Random Horizontal Flip:** This transform was applied with a 50% probability to teach the model that object identity remains the same regardless of orientation[cite: 19, 24, 26].
- **Random Crop with Padding:** I added a 4-pixel padding to the images before cropping them back to 32×32. This forces the model to learn features that are invariant to small translational shifts[cite: 21, 27, 29].
- **Image Normalization:** Tensors were normalized using a mean of [0.4914, 0.4822, 0.4465] and a standard deviation of [0.247, 0.243, 0.261][cite: 22, 34]. This centers the data around zero, which I found significantly improved the stability of the training loss[cite: 36, 37].

2.2 Regularization and Optimization Logic

Training deep Transformers is notoriously difficult due to the risk of vanishing or exploding gradients[cite: 48]. To mitigate these issues, I integrated the following techniques:

- **Dropout Layers:** I applied a dropout rate of 0.1 within the attention and MLP blocks to encourage the network to learn redundant and more robust feature representations[cite: 39, 41, 42].
- **AdamW Optimizer:** I chose the AdamW optimizer with a weight decay of 0.05 to penalize large weights and maintain a simpler model hypothesis[cite: 43, 46].
- **Gradient Norm Clipping:** To protect against gradient instability, I capped the global norm at 1.0, ensuring that updates to the model weights stayed within a reasonable range[cite: 47, 50].

3 Architectural Framework

The core logic of the Vision Transformer implementation revolves around the concept of "tokenization"[cite: 61]. The model does not see the image as a grid but as a sequence of flattened visual patches[cite: 61].

3.1 Patch Creation and Linear Embedding

The input 32×32 image is divided into non-overlapping patches of 4×4 pixels[cite: 63, 65]. This results in a sequence of $N = 64$ patches[cite: 66]. Each patch is then flattened and projected into a fixed 384-dimensional embedding space using a linear layer[cite: 68]. This process transforms raw pixel data into a set of feature vectors that the Transformer can process[cite: 69].

3.2 Positional Embeddings and the CLS Token

Since the attention mechanism is permutation-invariant (it doesn't know the order of tokens), I added learnable positional embeddings to each patch vector[cite: 71, 72]. These embeddings provide the model with "spatial awareness" regarding the location of each patch in the original image[cite: 72]. Furthermore, I prepended a learnable [CLS] token to the sequence[cite: 74]. This token aggregates information from all other patches via attention and serves as the final input for the classification head[cite: 75].

3.3 Multi-Head Self-Attention (MHSA) and MLP

The MHSA block allows the model to determine the importance of different patches relative to one another[cite: 78, 79]. Following the attention layer, the tokens pass through a Feed-Forward Network (MLP), which consists of two linear layers with a GELU non-linearity in between[cite: 83, 84, 87].

Listing 1: Multi-Head Self-Attention Implementation

```
# Computing scaled dot-product attention
attn_weights = (q_states @ k_states.transpose(-2, -1)) / math.sqrt(self
    .head_dim)
attn_weights = F.softmax(attn_weights, dim=-1) # Applying softmax for
    probabilities
```

```
attn_weights = F.dropout(attn_weights, p=self.dropout, training=self.
    training)
# Resulting context projection
out = attn_weights @ v_states
```

4 Experimental Setup and Hyperparameter Tuning

Finding the right set of hyperparameters was a critical part of the project. I conducted several runs to observe how the model responded to changes in patch size and embedding depth[cite: 53].

Hyperparameter	Tested Range	Final Selection
Patch Size	2, 4, 8	4
Embedding Dimension	256, 384, 768	384
Number of Attention Heads	4, 8	8
Transformer Layers	4, 6, 8	6
Dropout Rate	0.1, 0.2	0.1
Learning Rate	1e-3, 3e-4, 1e-4	3e-4
Batch Size	64, 128	128

Table 1: Detailed Hyperparameter configuration used for the final training run.

During my experiments, I observed that while a smaller patch size (e.g., 2×2) yielded better accuracy, it significantly increased the computational overhead because the sequence length grew quadratically. A patch size of 4×4 was found to be the most efficient balance for the 32×32 CIFAR images[cite: 53].

5 Results and Performance Analysis

The final model was trained for 50 epochs. Below are the key metrics recorded during the evaluation:

- **Final Training Accuracy:** 95.03% [cite: 99]
- **Best Test Accuracy:** 79.30% [cite: 99]
- **Convergence Rate:** The model showed rapid improvement in the first 20 epochs before stabilizing.

The discrepancy between the training and test accuracy indicates that the model has a very high capacity for learning features but is prone to overfitting on smaller datasets[cite: 100]. This is consistent with existing research stating that Vision Transformers typically require much larger datasets (like ImageNet-21k) to outperform standard CNNs. However, achieving nearly 80% accuracy on CIFAR-10 from scratch demonstrates that the implementation successfully learned how to interpret spatial patterns using purely attention-based mechanisms[cite: 100].

6 Conclusion and Personal Observations

This project provided me with deep insights into the internal workings of Transformer architectures. Moving away from the local receptive fields of convolutions to global self-attention was a significant conceptual

shift. While the current model performs well, I believe that incorporating more advanced data augmentation techniques, such as MixUp or CutMix, could help bridge the gap between training and testing performance. Overall, the implementation confirms that Vision Transformers are a powerful alternative to CNNs, even for small-scale image classification tasks, provided they are regularized correctly.