TinyViT: A Small Vision Transformer

Master's Degree in Computer Science

Lucian Dorin Crainic (1938430)

Academic Year 2024/2025

SAPIENZA
UNIVERSITÀ DI ROMA

This project is based on the well known **Vision Transformer** (ViT) architecture, a deep learning model that applies self attention mechanisms to image processing. We will start by introducing the core principles of ViT, explaining how it differs from traditional CNNs.

Then, we will explore the **TinyViT** a smaller implementation of the ViT architecture. Finally, we will present benchmark results demonstrating its performance across various tasks.

Table of Contents

1 Introduction

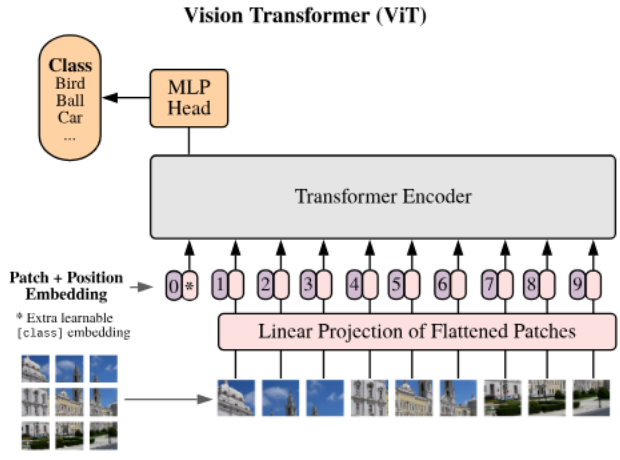
Introduction

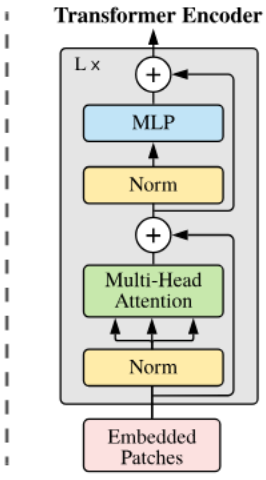
Why Vision Transformers ?

- Vision Transformers (ViTs) have revolutionized image recognition.
- Traditional ViTs require large datasets and heavy computation.
- TinyViT is a simplified alternative, retaining core transformer components while being computationally efficient.

Vision Transformer Architecture

Take a look at the Architecture





Vision Transformer Overview

Follow the steps

- Inspired by NLP transformers (Vaswani et al., 2017).
- Treats images as sequences of patches instead of processing pixels directly.
- Uses a stack of transformer encoder blocks for feature extraction.

Step 1	Step 2	Step 3	Step 4
Patch Embedding	Positional Encoding	Transformer Encoder	Classification Head

Table of Contents

2 Vision Transformer

Vision Transformer Architecture

Patch Embedding and Positional Econding

Patch Embedding:

- Converts images into fixed-size patches.
- Each patch is flattened into a vector and projected into an embedding space.
- Mathematical formulation:

$$X \in \mathbb{R}^{H \times W \times C} \quad \Rightarrow \quad N = \frac{H \times W}{P^2} \tag{1}$$

$$Z = [z_1, z_2, ..., z_N] \in \mathbb{R}^{N \times D} \tag{2}$$

Positional Encoding:

- Adds spatial information to the patches since transformers lack inherent spatial bias.
- Learnable positional embeddings:

$$Z_0 = Z + E_{pos} \tag{3}$$

Vision Transformer Architecture

Transformer Encoder and Classification Head

Transformer Encoder:

- Processes the sequence of patch embeddings.
- Consists of multiple layers of Multi-Head Self-Attention (MHSA) and Feed-Forward Networks (FFN).

$$FFN(z) = \text{GELU}(zW_1 + b_1)W_2 + b_2 \tag{4}$$

$$Q = ZW_Q, \quad K = ZW_K, \quad V = ZW_V \tag{5}$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{6}$$

Classification Head:

- Uses the class token z_{cls} for final classification.
- Softmax layer for output prediction:

$$\hat{y} = \text{softmax}(W_c z_{cls} + b_c) \tag{7}$$

ViT vs. CNN

A brief comparision between the two models

- CNNs use convolutional layers with local receptive fields.
- ViTs process images globally using self-attention mechanisms.
- CNNs have built-in spatial hierarchies, whereas ViTs rely on attention.
- ViTs typically need more data to perform well but can model long-range dependencies.

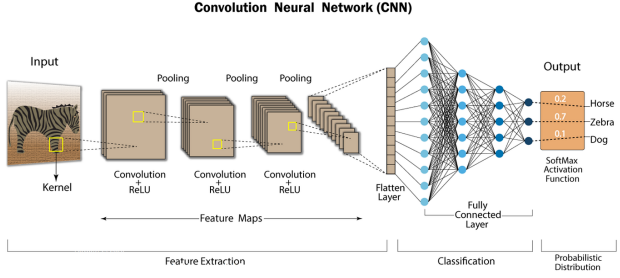


Table of Contents

TinyViT Architecture

Our tiny model

- A minimal Vision Transformer designed for smaller datasets.
- Reduces architectural complexity while preserving key transformer principles.
- Uses fewer layers, smaller embedding dimensions, and computationally efficient components.

```python
import torch
import torch.nn as nn


class PatchEmbedding(nn.Module):
    def __init__(
        self, img_size: int, patch_size: int, in_chans: int, embed_dim: int
    ) -> None:
        super().__init__()
        self.num_patches = (img_size // patch_size) ** 2
        self.patch_size = patch_size
        self.proj = nn.Linear(patch_size * patch_size * in_chans, embed_dim)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        B, C, H, W = x.shape
        x = x.unfold(2, self.patch_size, self.patch_size).unfold(
            3, self.patch_size, self.patch_size
        )
        x = x.contiguous().view(B, C, -1, self.patch_size, self.patch_size)
        x = x.permute(0, 2, 3, 4, 1).contiguous().view(B, self.num_patches, -1)
        x = self.proj(x)
        return x


class TinyViT(nn.Module):
    def __init__(
        self,
        num_classes: int,
        embed_dim: int,
        img_size: int,
        patch_size: int,
        in_chans: int,
        num_heads: int,
        num_layers: int,
        mlp_dim: int,
    ) -> None:
        super().__init__()
        self.patch_embed = PatchEmbedding(img_size, patch_size, in_chans, embed_dim)
        self.cls_token = nn.Parameter(torch.zeros(1, 1, embed_dim))
        self.pos_embed = nn.Parameter(
            torch.zeros(1, self.patch_embed.num_patches + 1, embed_dim)
        )

        nn.init.trunc_normal_(self.cls_token, std=0.02)
        nn.init.trunc_normal_(self.pos_embed, std=0.02)

        encoder_layer = nn.TransformerEncoderLayer(
            d_model=embed_dim,
            nhead=num_heads,
            dim_feedforward=mlp_dim,
            dropout=0.1,
            activation="gelu",
            batch_first=True,
        )

        self.transformer = nn.TransformerEncoder(encoder_layer, num_layers=num_layers)
        self.norm = nn.LayerNorm(embed_dim)
        self.classifier = nn.Linear(embed_dim, num_classes)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        B = x.shape[0]
        x = self.patch_embed(x)
        cls_token = self.cls_token.expand(B, -1, -1)
        x = torch.cat([cls_token, x], dim=1)
        x = x + self.pos_embed
        x = self.transformer(x)
        cls_out = self.norm(x[:, 0])
        return self.classifier(cls_out)
```

TinyViT Parameters

What parameters do we use ?

Table: Parameters of the TinyViT Model for CIFAR-10

Parameter	Value
Number of Classes	10
Embedding Dimension	128
Image Size	32
Patch Size	4
Input Channels	3
Number of Attention Heads	8
Number of Transformer Layers	6
MLP Hidden Dimension	512

Differences between TinyViT and ViT

Some differences between the two models

- TinyViT has significantly fewer transformer layers than standard ViT.
- Smaller embedding dimensions to reduce computational costs.
- Optimized for smaller datasets, unlike ViT, which requires large-scale pretraining.
- Lower memory footprint, making it feasible for resource-constrained environments.

Table of Contents

4 Datasets

CIFAR 10 and 100

Small 32x32 Images

CIFAR-10:

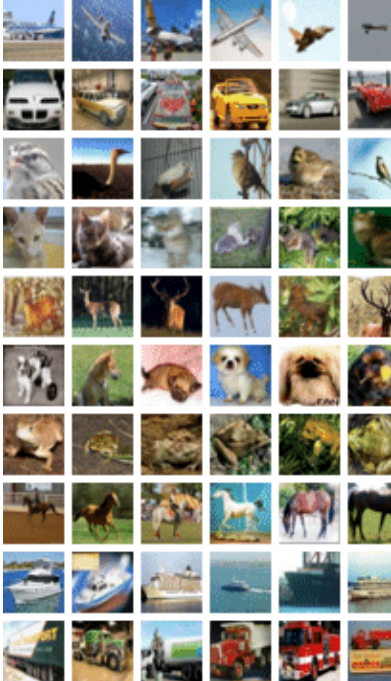
- 60,000 images, 10 classes, 32x32 resolution.

CIFAR-100:

- 60,000 images, 100 classes, 32x32 resolution.

Data Augmentation:

- Random cropping (32x32 with 4-pixel padding).
- Horizontal flipping.
- Normalization using dataset-specific means and standard deviations.



STL 10

96X96 Images more pixels

STL-10:

- 130,000 images, 10 classes, 96x96 resolution.

Data Augmentation:

- Random cropping (12-pixel padding, resized to 96x96).
- Horizontal flipping.
- Color jittering (brightness, contrast, saturation adjustments).
- Random grayscale conversion (10% probability).
- Normalization using dataset-specific statistics.

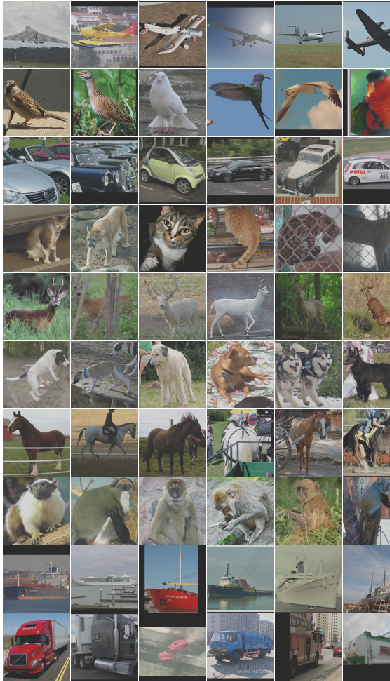


Table of Contents

Models Training Setup

What setup was used ?

Loss Function: Cross-Entropy with Label Smoothing (0.1)

$$L = -\sum_{i} y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \tag{8}$$

Optimizer: AdamW

$$\theta_{t+1} = \theta_t - \eta \left(\frac{m_t}{\sqrt{v_t} + \epsilon} \right) \tag{9}$$

Hyperparameters:

- Learning Rate: $3e^{-4}$
- Weight Decay: 0.05
- Epochs: 100
- Batch Size: 64

Table of Contents

6 Results

Results CIFAR 10 - CIFAR 100

Tiny Model has some strength in him

TinyViT outperforms CNNs on both datasets. Handles complex class distributions better than CNNs.

Dataset	Model	Accuracy	F1	Recall	MCC	Precision
CIFAR-10	**Tiny ViT**	**82.59**	**82.43**	**82.59**	**80.70**	**82.76**
	CNN	80.67	80.55	80.67	78.53	80.56
CIFAR-100	**Tiny ViT**	**59.40**	**59.04**	**59.40**	**59.00**	**60.00**
	CNN	46.13	44.57	46.13	45.61	45.61

Note: All values are percentages (%). Bold indicates best performance in category.

Results STL-10

CNNs outperforms our Tiny Model

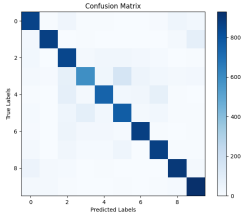
CNN performs better on STL-10, likely due to the higher image resolution. TinyViT may struggle with lower-resolution images in datasets with fewer samples.

Dataset	Model	Accuracy	F1	Recall	MCC	Precision
STL-10	Tiny ViT	64.27	64.30	64.27	60.47	66.13
	CNN	**68.36**	**68.10**	**68.36**	**64.95**	**68.78**

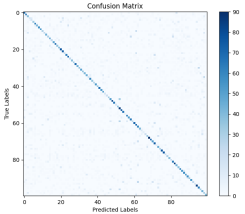
Note: All values are percentages (%). Bold indicates best performance in category.

Confusion Matrices

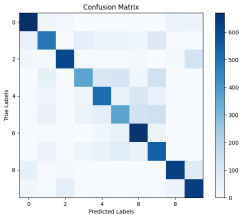
Visualize the results



CIFAR-10 dataset using Tiny ViT



CIFAR-100 dataset using Tiny ViT



STL-10 dataset using CNN

Table of Contents

7 Conlusions and Future Work

Conclusion

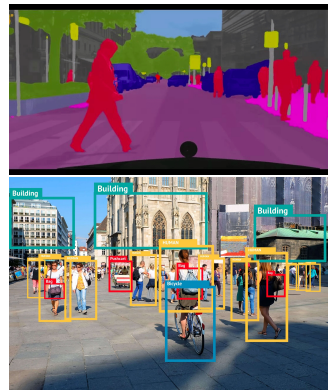
Let's discuss the results

- TinyViT proves that **transformer based models** can be efficient with fewer resources.
- Outperforms CNNs on **smaller datasets** like CIFAR-10 and CIFAR-100.
- **Requires improvements** for larger images like STL-10.

Future Work

How can we improve ?

- Implement TinyViT for **Object Detection** (DETR) and **Segmentation** (Segmenter).
- Experiment with different **hyperparameters** (layers, embedding size, attention heads).
- Explore pretraining on **larger datasets** to improve performance.



TinyViT: A Small Vision Transformer

Thank you for listening!
Any questions?