image

Vision Transformer Implementation

An Implementation of the ICLR 2021 Paper:
"An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale"

Team Members:

Ahsan Ali (22K-4036)
Muneeb ur Rehman (22K-4025)
Ali Suleman (22K-4060)
Moosa Memon (22K-4067)

Artificial Neural Networks
Department of Artificial Intelligence
Spring 2025

Abstract

This report presents our implementation of the Vision Transformer (ViT) model as described in the ICLR 2021 paper "An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale." The Vision Transformer applies the transformer architecture, which has revolutionized natural language processing, to computer vision tasks. Our implementation demonstrates how images can be processed as sequences of patches, enabling transformer models to achieve state-of-the-art performance on image classification tasks without the traditional convolutional neural network architecture.

Introduction

Background

The field of computer vision has been dominated by Convolutional Neural Networks (CNNs) for the past decade. However, the transformer architecture, which relies on self-attention mechanisms rather than convolutions, has shown remarkable success in natural language processing tasks. The Vision Transformer (ViT) model adapts the transformer architecture for image recognition by treating an image as a sequence of patches, similar to how words are treated in NLP.

Motivation

Our motivation for this project was to understand and implement a cutting-edge architecture that challenges the conventional CNN paradigm in computer vision. The ViT model represents a fundamental shift in how we process visual data and has the potential to unify methodologies across different domains of artificial intelligence.

Overview of the Vision Transformer architecture.

Overview of the Vision Transformer architecture.

Theoretical Background

Transformer Architecture

The transformer architecture, introduced by Vaswani et al. in "Attention Is All You Need," relies on multi-head self-attention mechanisms to process sequential data. The key components include:

- Self-attention layers
- Feed-forward networks
- Layer normalization
- Residual connections

The self-attention mechanism allows the model to weigh the importance of different parts of the input sequence when processing each element, enabling it to capture long-range dependencies efficiently.

Vision Transformer

The Vision Transformer adapts the transformer architecture for images by:

1. **Patch Embedding**: Dividing the image into fixed-size patches
2. **Position Embedding**: Adding positional information to maintain spatial relationships
3. **Class Token**: Introducing a learnable token for classification
4. **Transformer Encoder**: Processing the sequence of patch embeddings

Image patching process in Vision Transformer.

Image patching process in Vision Transformer.

Implementation Details

Model Architecture

Our implementation closely follows the original paper, with components structured as follows:

- **Patch Embedding**: Converts image patches to embeddings

- **Position Embedding**: Adds learnable position information
- **Transformer Blocks**: Processes the sequence with attention and feed-forward layers
- **Classification Head**: Produces final class predictions

Key Components

Patch Embedding

The patch embedding module divides the input image into fixed-size patches and projects each patch into an embedding space:

```python
class PatchEmbedding(nn.Module):
    def __init__(self, in_channels=3, patch_size=8, emb_size=128):
        self.patch_size = patch_size
        super().__init__()
        self.projection = nn.Sequential(
            # break-down the image in s1 x s2 patches and flat them
            Rearrange('b c (h p1) (w p2) -> b (h w) (p1 p2 c)', 
p1=patch_size, p2=patch_size),
            nn.Linear(patch_size * patch_size * in_channels, emb_size)
        )

    def forward(self, x: Tensor) -> Tensor:
        x = self.projection(x)
        return x
```

Attention Mechanism

The attention mechanism allows the model to focus on different parts of the input sequence:

```python
class Attention(nn.Module):
    def __init__(self, dim, n_heads, dropout):
        super().__init__()
        self.n_heads = n_heads
        self.att = torch.nn.MultiheadAttention(embed_dim=dim,
                                               num_heads=n_heads,
                                               dropout=dropout)
        self.q = torch.nn.Linear(dim, dim)
        self.k = torch.nn.Linear(dim, dim)
        self.v = torch.nn.Linear(dim, dim)

    def forward(self, x):
        q = self.q(x)
        k = self.k(x)
        v = self.v(x)
        attn_output, attn_output_weights = self.att(x, x, x)
        return attn_output
```

Complete ViT Model

Our full implementation integrates all the components into a cohesive model:

```
class ViT(nn.Module):
    def __init__(self, ch=3, img_size=144, patch_size=4, emb_dim=32,
                n_layers=6, out_dim=37, dropout=0.1, heads=2):
        super(ViT, self).__init__()

        # Attributes
        self.channels = ch
        self.height = img_size
        self.width = img_size
        self.patch_size = patch_size
        self.n_layers = n_layers

        # Patching
        self.patch_embedding = PatchEmbedding(in_channels=ch,
                                              patch_size=patch_size,
                                              emb_size=emb_dim)
        # Learnable params
        num_patches = (img_size // patch_size) ** 2
        self.pos_embedding = nn.Parameter(
            torch.randn(1, num_patches + 1, emb_dim))
        self.cls_token = nn.Parameter(torch.rand(1, 1, emb_dim))

        # Transformer Encoder
        self.layers = nn.ModuleList([])
        for _ in range(n_layers):
            transformer_block = nn.Sequential(
                ResidualAdd(PreNorm(emb_dim, Attention(emb_dim,
n_heads=heads, dropout=dropout))),
                ResidualAdd(PreNorm(emb_dim, FeedForward(emb_dim,
emb_dim, dropout=dropout))))
            self.layers.append(transformer_block)

        # Classification head
        self.head = nn.Sequential(nn.LayerNorm(emb_dim),
                                  nn.Linear(emb_dim, out_dim))
```

Training Procedure

We trained our model on the Oxford-IIIT Pet dataset, which contains images of 37 pet breeds. The training procedure included:

- Adam optimizer with a learning rate of 0.001
- Cross-entropy loss function
- Batch size of 32

- 80/20 train-test split

Training process overview.

Training process overview.

Experimental Results

Dataset

The Oxford-IIIT Pet dataset consists of 7,349 images of cats and dogs, covering 37 different breeds. Each image is annotated with class labels, breed information, and pixel-level segmentation.

Training and Validation

Our model was trained for numerous epochs to ensure convergence. We monitored both training and testing loss during the process.

Results Analysis

The model showed promising results on the test set, with gradually decreasing loss values indicating effective learning. While the training required significant computational resources, the final model demonstrated the capability of transformer architectures to handle image classification tasks effectively.

Predicted vs. actual class comparisons revealed both strengths and limitations of our implementation, particularly in distinguishing between visually similar breeds.

Model performance: Training and testing loss curves.

Model performance: Training and testing loss curves.

Discussion

Comparison with CNNs

Unlike CNNs, which use hierarchical feature extraction through convolutional layers, the ViT processes images as sequences with global self-attention. This approach has several implications:

- ViT can capture global dependencies more efficiently
- ViT requires more data to generalize well
- ViT has higher computational requirements

Limitations

Our implementation has several limitations:

- Higher computational requirements compared to CNNs
- Need for large datasets to avoid overfitting
- Sensitivity to hyperparameter choices

Future Work

Potential directions for future work include:

- Hybrid architectures combining CNNs and Transformers
- Optimizations to reduce computational complexity
- Application to other computer vision tasks (segmentation, detection)

Conclusion

Our implementation of the Vision Transformer demonstrates the viability of applying transformer architectures to image recognition tasks. While there are trade-offs compared to traditional CNNs, the ViT represents an important step toward unifying approaches across different domains of artificial intelligence.

The success of this model suggests that self-attention mechanisms can effectively capture the spatial relationships in images without relying on the inductive biases built into convolutional architectures.

9 Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2021). An image is worth 16×16 words: Transformers for image recognition at scale. *ICLR 2021*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*.

Francesco Zuppichini. (2021). Implementing Visual Transformer in PyTorch. *Towards Data Science*.

Brian Pulfer. (2021). Vision Transformers from Scratch (PyTorch): A Step-by-Step Guide. *Medium*.

Full Implementation Code

Our complete implementation is available in the `vision_transformer.py` file.

Training Logs

Training and validation logs show the progression of loss values and accuracy metrics throughout the training process.