A Comprehensive Comparison of Convolutional Neural Networks and Vision Transformers in Computer Vision

Amchaar Anas, Fardaoui Ilyas
MadeInMorocco.AI Community

February 1, 2025

Contents

1 Introduction **2**

2 CNN's Foundations **2**
2.1 Definition . . . 2
2.2 CNN Components . . . 2
2.3 Why CNNs Are Effective . . . 3
2.4 Advantages of CNNs . . . 3
2.5 Applications of CNNs . . . 3
2.6 Evolution of Convolutional Neural Networks . . . 3
2.7 CNN Architecture in Code . . . 4

3 Vision Transformers (ViTs) **4**
3.1 Overview . . . 4
3.2 Advantages of ViTs . . . 5
3.3 Limitations of ViTs . . . 5
3.4 How it works . . . 5
3.4.1 Image to Patches . . . 5
3.4.2 Patch Embeddings . . . 5
3.4.3 Position Embeddings . . . 5
3.4.4 Class Token . . . 5
3.4.5 Transformer Encoder . . . 6
3.4.6 Classification Head . . . 6
3.5 Vision Transformer Implementation in PyTorch . . . 6
3.5.1 Required Libraries . . . 6
3.5.2 Patch Embedding . . . 6
3.5.3 Multi-Head Self-Attention . . . 7
3.5.4 Transformer Encoder Layer . . . 7
3.5.5 Vision Transformer Model . . . 8

4 Comparative Analysis: CNNs vs ViTs **9**
4.1 Performance . . . 9
4.2 Computational Efficiency . . . 9
4.3 Use Cases . . . 9

5 Hybrid Models **9**

6 Conclusion **9**

Abstract

Convolutional Neural Networks (CNNs) are a revolutionary class of deep learning models designed to handle visual data, achieving groundbreaking performance in tasks such as image classification, object detection, and segmentation. Their hierarchical feature extraction through convolutional layers, pooling mechanisms, and fully connected layers has driven advancements across numerous applications. However, CNNs face limitations, including high computational cost and dependency on large datasets. Alongside CNNs, Vision Transformers (ViTs) have emerged as a novel paradigm, leveraging self-attention mechanisms to model global dependencies in visual data. This paper explores the foundations of CNNs, their evolution from AlexNet to modern architectures like ResNet and EfficientNet, and their comparative strengths and weaknesses against ViTs. Additionally, the potential of hybrid models combining CNNs and ViTs is examined, paving the way for future perspectives in computational efficiency and improved generalization in visual tasks.

1 Introduction

Transformers, initially introduced in the domain of Natural Language Processing (NLP), revolutionized the field with their ability to handle sequential data and model long-range dependencies effectively. In the realm of computer vision, where Convolutional Neural Networks (CNNs) have long been the dominant paradigm, Vision Transformers (ViTs) have emerged as a formidable alternative. ViTs often surpass CNNs in performance on large-scale datasets, provided adequate computational resources and data are available. Despite this, CNNs continue to hold a significant advantage in scenarios characterized by limited data or computational constraints. Moreover, hybrid approaches that combine CNNs and Transformers are increasingly gaining attention, offering a promising pathway to harness the unique strengths of both architectures.

2 CNN's Foundations

2.1 Definition

A CNN is a specialized type of deep learning model that processes data with a grid-like topology, such as images. Unlike traditional fully connected networks, CNNs take advantage of the spatial structure of images by using convolutional operations to extract features.

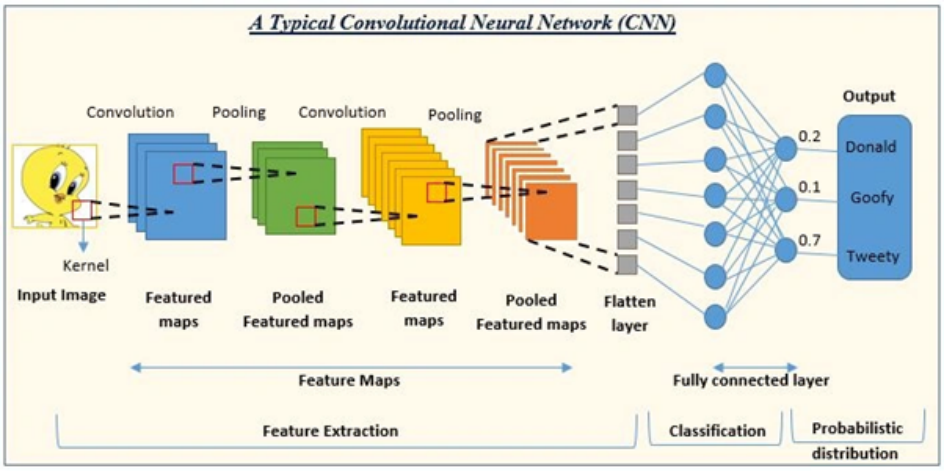

Figure 1: *CNN Model Overview*

2.2 CNN Components

- **Convolutional Layers:** These layers apply filters (kernels) to input images, enabling the extraction of local features such as edges, textures, and patterns.
- **Pooling Layers:** Pooling reduces the spatial dimensions of feature maps, retaining the most important information while improving computational efficiency.

- **Fully Connected Layers:** These layers perform high-level reasoning by connecting all neurons from the previous layer to the output neurons.
- **Activation Functions:** Functions like ReLU (Rectified Linear Unit) introduce non-linearity, enabling the network to learn complex patterns.

2.3 Why CNNs Are Effective

- Exploit spatial hierarchies in images.
- Reduce the number of parameters compared to traditional fully connected networks.
- Provide translation invariance, making them robust to small changes in input.

2.4 Advantages of CNNs

- **High Performance:** Achieve state-of-the-art accuracy in visual tasks.
- **Wide Applicability:** Used in image classification, object detection (e.g., YOLO, R-CNN), and segmentation (e.g., U-Net).
- **Invariance to Small Transformations:** Robust to translations, rotations, and scaling.
- **Scalability:** Architectures like ResNet and EfficientNet allow scaling in depth and complexity.

2.5 Applications of CNNs

- **Image Classification:** Example: AlexNet and ResNet on ImageNet datasets.
- **Object Detection:** Example: YOLO (You Only Look Once) for real-time detection.
- **Image Segmentation:** Example: U-Net for biomedical image analysis.
- **Video Analysis:** Action recognition and surveillance applications.
- **Other Domains:** Autonomous vehicles, medical imaging, facial recognition, and agriculture.

2.6 Evolution of Convolutional Neural Networks

From the first CNN architecture to now, there has been a huge development of ConvNets. The main milestones in this evolution are:

- **LeNet-5 (1989):** First CNN architecture with convolutional layers and pooling, designed for digit recognition.

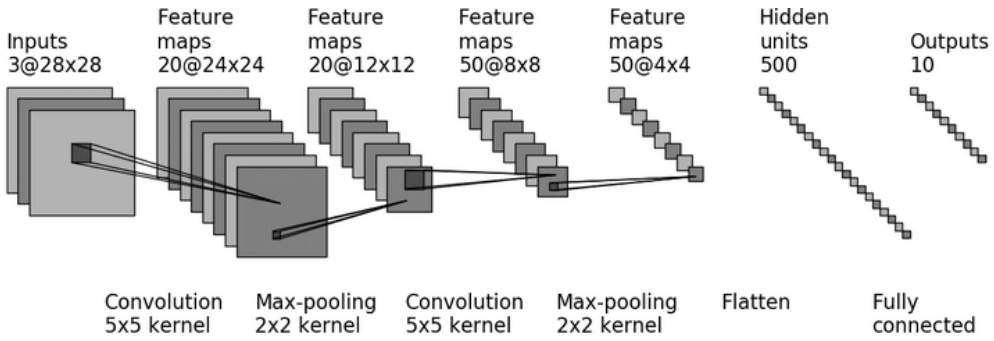

Figure 2: *LeNet-5 Architecture*

- **AlexNet (2012):** Popularized CNNs and demonstrated their potential by:
  - Winning the ImageNet challenge
  - Introducing ReLU activation and dropout
  - Using GPU acceleration for deep learning
  - Achieving 15.3% error rate on ImageNet
- **VGGNet (2014):** Introduced deep networks with important innovations:
  - Using small 3×3 filters consistently
  - Increasing depth to 16-19 layers
  - Demonstrating the power of network depth
  - Standardizing CNN architecture design

- **ResNet (2015):** Solved the vanishing gradient problem through:
  - Introducing residual connections
  - Enabling training of very deep networks (152+ layers)
  - Achieving 3.57% error rate on ImageNet
  - Setting new standards for CNN architecture
- **EfficientNet (2019):** Achieved better accuracy with fewer parameters by:
  - Introducing compound scaling method
  - Balancing network depth, width, and resolution
  - Using Neural Architecture Search
  - Setting new efficiency benchmarks

2.7 CNN Architecture in Code

It is important to know theoretical information about CNNs, but it is also Important to know how do we implement it in Codes:

```python
import tensorflow as tf
from tensorflow.keras import layers, models

# Define the CNN model
model = models.Sequential()

# First convolutional layer
model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(64, 64, 3)))
model.add(layers.MaxPooling2D((2, 2)))

# Second convolutional layer
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Third convolutional layer
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Flatten the output and feed it into a dense layer
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Print the model summary
model.summary()
```

Figure 3: *CNN Architecture in Codes.*

3 Vision Transformers (ViTs)

3.1 Overview

The Vision Transformer (ViT) architecture divides an image into fixed-size patches, flattens them, and embeds them into vectors. These patch embeddings, combined with positional encodings to retain spatial information, are passed through standard Transformer layers. Each layer uses self-attention mechanisms and feedforward networks to model global dependencies across the image.

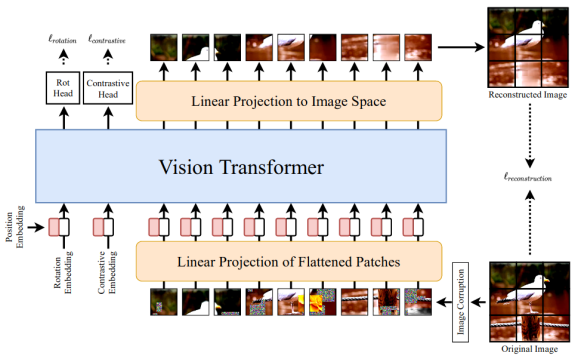

Figure 4: *ViT Model Overview.*

3.2 Advantages of ViTs

- **Global Context Modeling:** Self-attention mechanisms capture long-range dependencies across the entire image.
- **Scalability:** ViTs can handle larger datasets and higher resolutions effectively.
- **Flexibility:** Can be adapted to various vision tasks with minimal architectural changes.

3.3 Limitations of ViTs

- **Computational Cost:** Requires significant computational resources for training.
- **Data Dependency:** Performs best with large-scale datasets.
- **Complexity:** More challenging to interpret compared to CNNs.

3.4 How it works

3.4.1 Image to Patches

An input image $\mathbf{x} \in R^{H \times W \times C}$ is divided into fixed-size patches, where:

- $H \times W$ is the image resolution,
- C is the number of channels (e.g., 3 for RGB images),
- Each patch has a resolution of $P \times P$.

The image is reshaped into a sequence of flattened patches $\mathbf{x}_p \in R^{N \times (P^2 \cdot C)}$, where $N = \frac{HW}{P^2}$ is the number of patches.

3.4.2 Patch Embeddings

Each patch is linearly projected into a D-dimensional embedding space using a trainable linear layer. These are called **patch embeddings**.

3.4.3 Position Embeddings

To retain spatial information, learnable 1D position embeddings are added to the patch embeddings. This helps the model understand the relative positions of patches in the image.

3.4.4 Class Token

Similar to BERT's `[CLS]` token, a learnable class embedding is prepended to the sequence of patch embeddings. The final state of this token after passing through the Transformer encoder is used as the image representation for classification.

3.4.5 Transformer Encoder

The sequence of patch embeddings (with position embeddings and the class token) is fed into a standard Transformer encoder. The encoder consists of alternating layers of:

- **Multi-Head Self-Attention (MSA):** Captures relationships between patches.
- **Multi-Layer Perceptron (MLP):** Processes the features.

Layer Normalization (LN) is applied before each block, and residual connections are added after each block.

3.4.6 Classification Head

During both pre-training and fine-tuning, a classification head is attached to the class token's final output. The head is implemented as:

- An MLP with one hidden layer during pre-training,
- A single linear layer during fine-tuning.

3.5 Vision Transformer Implementation in PyTorch

After understanding the theoretical foundations of Vision Transformers (ViT), we will now examine its practical implementation using PyTorch. This section presents a detailed breakdown of the key components that make up the ViT architecture.

3.5.1 Required Libraries

First, we import the necessary PyTorch libraries and dependencies:

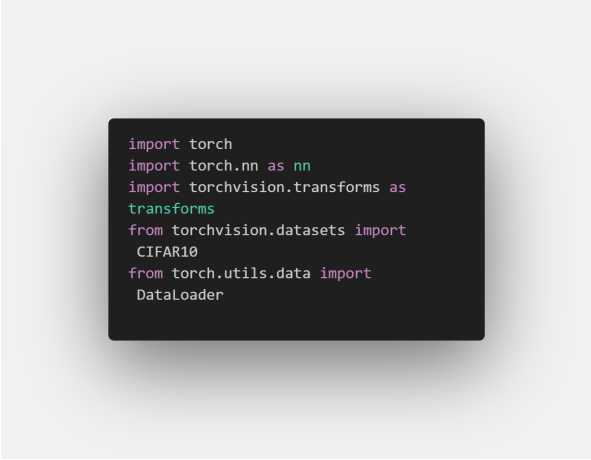

Figure 5: *Required PyTorch imports for ViT implementation*

3.5.2 Patch Embedding

The PatchEmbedding class is responsible for dividing the input image into fixed-size patches and projecting them into a lower-dimensional embedding space:

```python
class PatchEmbedding(nn.Module):
    def __init__(self, in_channels=3, patch_size=16,
emb_size=768, img_size=224):
        super().__init__()
        self.patch_size = patch_size
        self.proj = nn.Conv2d(in_channels, emb_size,
kernel_size=patch_size, stride=patch_size)
        self.cls_token = nn.Parameter(torch.randn(1, 1,
emb_size))
        self.pos_embed = nn.Parameter(torch.randn((img_size
 // patch_size) ** 2 + 1, emb_size))

    def forward(self, x):
        B, C, H, W = x.shape
        x = self.proj(x).flatten(2).transpose(1, 2)
        cls_tokens = self.cls_token.expand(B, -1, -1)
        x = torch.cat((cls_tokens, x), dim=1)
        x = x + self.pos_embed
        return x
```

Figure 6: *PatchEmbedding class implementation*

3.5.3 Multi-Head Self-Attention

The MultiHeadAttention class implements the core attention mechanism of the transformer, allowing the model to focus on different parts of the input simultaneously:

```python
class MultiHeadAttention(nn.Module):
    def __init__(self, emb_size=768, num_heads=8, dropout=0.1):
        super().__init__()
        self.emb_size = emb_size
        self.num_heads = num_heads
        self.qkv = nn.Linear(emb_size, emb_size * 3)
        self.att_drop = nn.Dropout(dropout)
        self.proj = nn.Linear(emb_size, emb_size)

    def forward(self, x):
        B, N, C = x.shape
        qkv = self.qkv(x).reshape(B, N, 3, self.num_heads, C //
self.num_heads)
        qkv = qkv.permute(2, 0, 3, 1, 4)
        q, k, v = qkv[0], qkv[1], qkv[2]
        att = (q @ k.transpose(-2, -1)) * (1.0 / (k.size(-1) **
0.5))
        att = att.softmax(dim=-1)
        att = self.att_drop(att)
        x = (att @ v).transpose(1, 2).reshape(B, N, C)
        x = self.proj(x)
        return x
```

Figure 7: *MultiHeadAttention class implementation*

3.5.4 Transformer Encoder Layer

The TransformerEncoderLayer class combines multi-head attention with feed-forward networks:

```python
class TransformerEncoderLayer(nn.Module):
    def __init__(self, emb_size=768, num_heads=8, dropout=0.1,
forward_expansion=4):
        super().__init__()
        self.norm1 = nn.LayerNorm(emb_size)
        self.norm2 = nn.LayerNorm(emb_size)
        self.attn = MultiHeadAttention(emb_size, num_heads,
dropout)
        self.ff = nn.Sequential(
            nn.Linear(emb_size, forward_expansion * emb_size),
            nn.GELU(),
            nn.Linear(forward_expansion * emb_size, emb_size),
            nn.Dropout(dropout)
        )

    def forward(self, x):
        x = x + self.attn(self.norm1(x))
        x = x + self.ff(self.norm2(x))
        return x
```

Figure 8: *TransformerEncoderLayer implementation with attention and feed-forward networks*

3.5.5 Vision Transformer Model

Finally, the VisionTransformer class brings all components together into a complete architecture:

```python
class VisionTransformer(nn.Module):
    def __init__(self, img_size=224, patch_size=16, in_channels=3
, num_classes=10, emb_size=768, depth=12, num_heads=8, dropout=0.1
):
        super().__init__()
        self.patch_embed = PatchEmbedding(in_channels, patch_size
, emb_size, img_size)
        self.encoder = nn.Sequential(*[TransformerEncoderLayer(
emb_size, num_heads, dropout) for _ in range(depth)])
        self.norm = nn.LayerNorm(emb_size)
        self.head = nn.Linear(emb_size, num_classes)

    def forward(self, x):
        x = self.patch_embed(x)
        x = self.encoder(x)
        x = self.norm(x[:, 0])
        x = self.head(x)
        return x
```

Figure 9: *Complete Vision Transformer model implementation*

This implementation follows the original ViT architecture while maintaining a clean and modular structure. Each component is designed to be easily customizable through various hyperparameters such as embedding size, number of heads, and dropout rate.

4 Comparative Analysis: CNNs vs ViTs

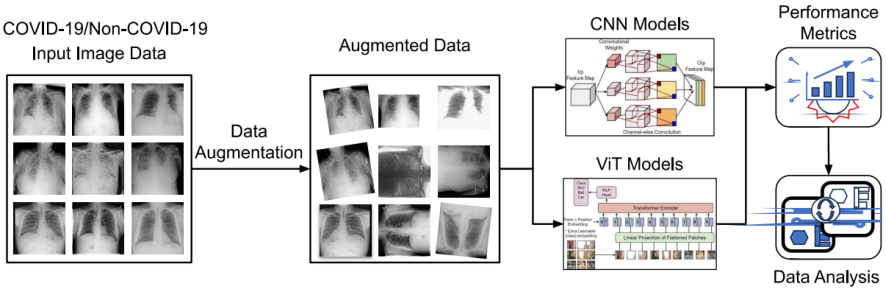

Figure 10: *CNN and ViT from [5]*

4.1 Performance

- CNNs excel in scenarios with limited data and computational resources.
- ViTs outperform CNNs on large-scale datasets with sufficient computational power.

4.2 Computational Efficiency

- CNNs are more computationally efficient for smaller datasets and real-time applications.
- ViTs require more resources but scale better with larger datasets.

4.3 Use Cases

- CNNs are preferred for tasks like real-time object detection and edge computing.
- ViTs are ideal for tasks requiring global context understanding, such as high-resolution image classification.

5 Hybrid Models

Hybrid models combine the strengths of CNNs and ViTs, leveraging the local feature extraction capabilities of CNNs and the global context modeling of ViTs. These models are gaining traction in applications requiring both efficiency and high performance.

6 Conclusion

This paper has explored the foundations, advantages, and limitations of CNNs and ViTs, highlighting their respective strengths and use cases. Hybrid models offer a promising direction for future research, combining the best of both architectures to achieve improved performance and efficiency in visual tasks.

This paper references several works on CNNs and ViTs. For instance, the study by Dosovitskiy et al. [1] introduced ViTs as a powerful alternative to CNNs, demonstrating their effectiveness in large-scale image recognition. Similarly, Cuenat and Couturier [6] analyzed CNNs and ViTs in the context of digital holography, providing insights into their comparative performance. Another relevant study by Koresha et al. [2] investigates hybrid architectures that unify CNNs and Transformers, shedding light on their underlying learning mechanisms. Furthermore, the pioneering work of Krizhevsky et al. [3] on deep CNNs set the stage for modern computer vision advancements. Finally, O'Shea and Nash [4] provide an introduction to CNNs, offering foundational knowledge for understanding their role in deep learning.

Future work could explore optimizing hybrid models to leverage the complementary strengths of CNNs and ViTs while addressing their computational challenges. As research progresses, these architectures will continue to shape the landscape of deep learning in computer vision.

References

[1] Alexander Kolesnikov Dirk Weissenborn Xiaohua Zhai Thomas Unterthiner Mostafa Dehghani Matthias Minderer Georg Heigold Sylvain Gelly Jakob Uszkoreit Neil Houlsby Alexey Dosovitskiy, Lucas Beyer. An image is worth 16x16 words: Transformers for image recognition at scale. 2021.

[2] Yuval Meir1 Yarden Tzacha Tal Halevia Ella Koresha, Ronit D. Grossa and Ido Kantera. Unified cnns and transformers underlying learning mechanism reveals multi-head attention modus vivendi. 2022.

[3] Sutskever I. Hinton G. E. Krizhevsky, A. Imagenet classification with deep convolutional neural networks. advances in neural information processing systems. 2012.

[4] Keiron O'Shea1 and Ryan Nash2. An introduction to convolutional neural networks. 2015.

[5] Kanishk Arya Shreyas Bangalore Vijayakumar, Krishna Teja Chitty-Venkata and Arun K. Somani. Convision benchmark: A contemporary framework to benchmark cnn and vit models. 2024.

[6] Raphael Couturier Stephane Cuenat. Convolutional neural network (cnn) vs vision transformer (vit) for digital holography. 2022.