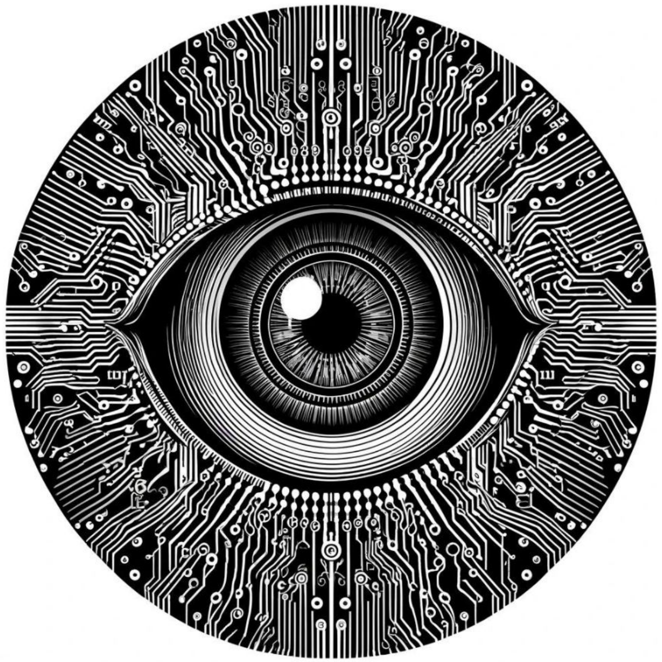
Vision Transformers



Antonio Rueda-Toicen

SPONSORED BY THE
Federal Ministry
of Education
and Research

Learning goals

- Gain an overview of the Vision Transformer (ViT) architecture and the self-attention mechanism
- Understand tradeoffs between vision transformers and convolutional networks

The Vision Transformer (ViT) architecture

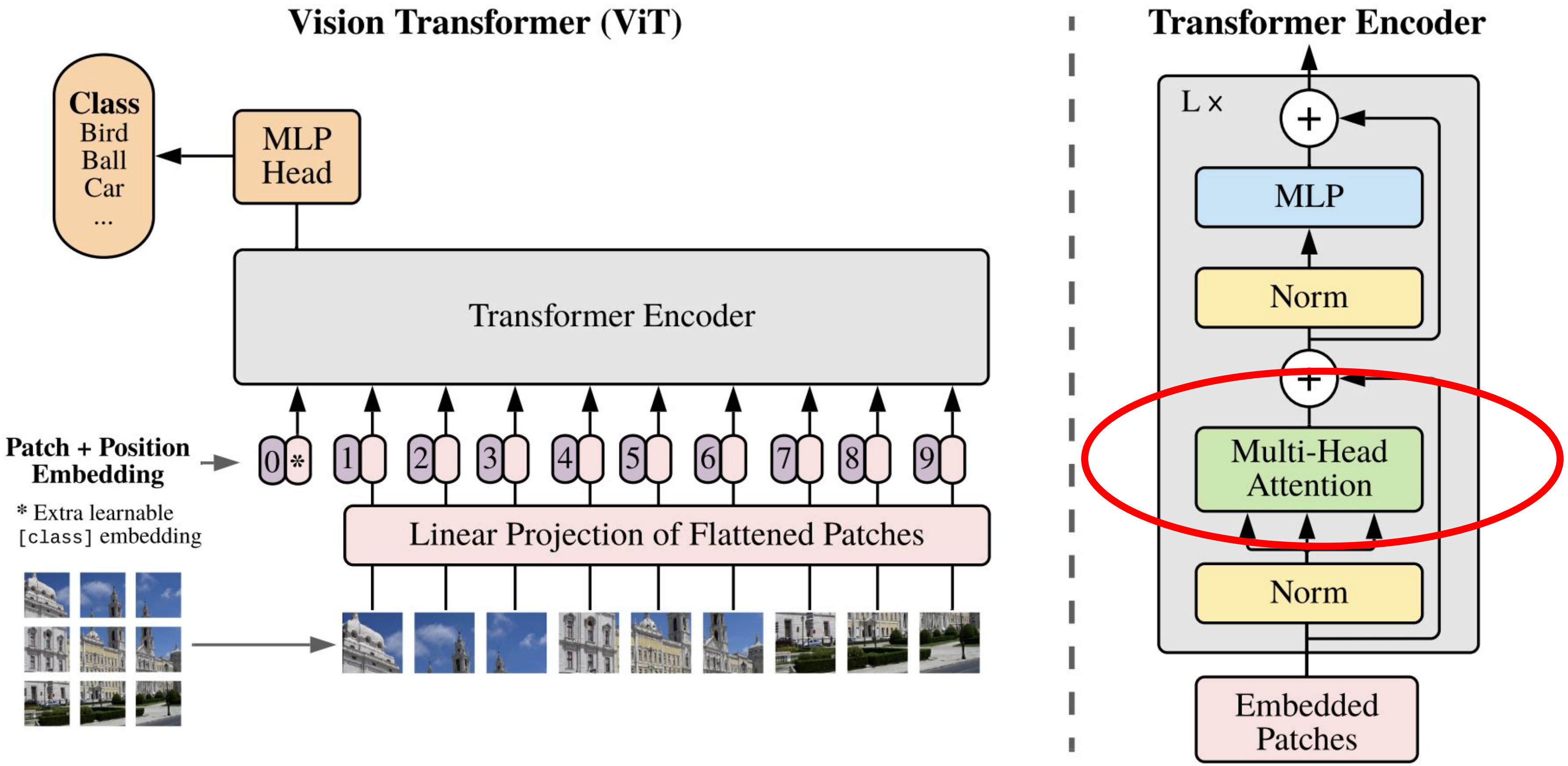


Image from [An Image is Worth 16x16 Words - Transformers for Image Recognition at Scale]()

The Vision Transformer (ViT) architecture

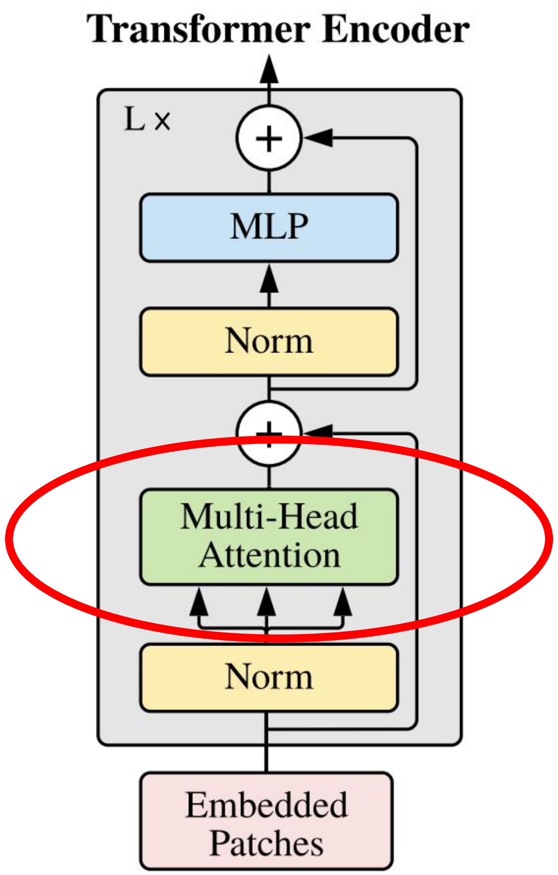
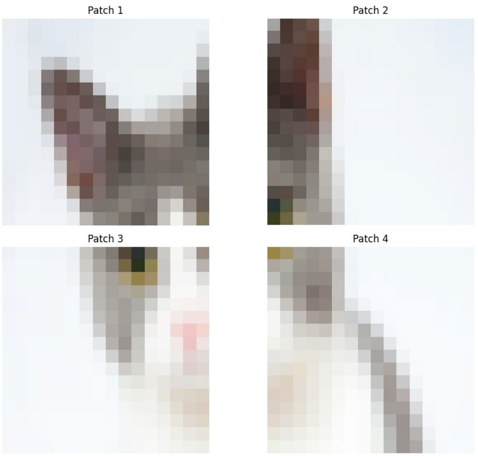


Image from [Transformers for Image Recognition at Scale]

Projection of flattened patches and adding positional embeddings

```
import torch
import torch.nn as nn

# Example CIFAR-10-like data: (batch_size, 3, 32, 32)
data = torch.randn(4, 3, 32, 32)

# Hyperparameters
patch_size = 16
channels = 3
embed_dim = 768  # Standard ViT-Base embedding dimension

# Calculate number of patches
num_patches = (32 // patch_size) * (32 // patch_size)  # 4 = 2*2
patch_dim = patch_size * patch_size * channels  # 768 = 16*16*3

# Linear projection and positional embedding
patch_embed = nn.Linear(patch_dim, embed_dim)
pos_embed = nn.Parameter(torch.zeros(1, num_patches, embed_dim))

# Reshape data into patches
# From (batch_size, channels, height, width) to (batch_size, channels, h_patches, patch_size, w_patches, patch_size)
patches = data.unfold(2, patch_size, patch_size).unfold(3, patch_size, patch_size)

# Reshape to (batch_size, num_patches, patch_dim)
patches = patches.permute(0, 2, 4, 1, 3, 5).reshape(data.size(0), num_patches, -1)

# Linear projection of patches
x = patch_embed(patches)  # Shape: (batch_size, num_patches, embed_dim)

# Add positional embeddings
x = x + pos_embed

print(f"Input shape: {data.shape}")  # [4, 3, 32, 32]
print(f"Patches shape: {patches.shape}")  # [4, 4, 768]
print(f"Output shape: {x.shape}")  # [4, 4, 768]
```



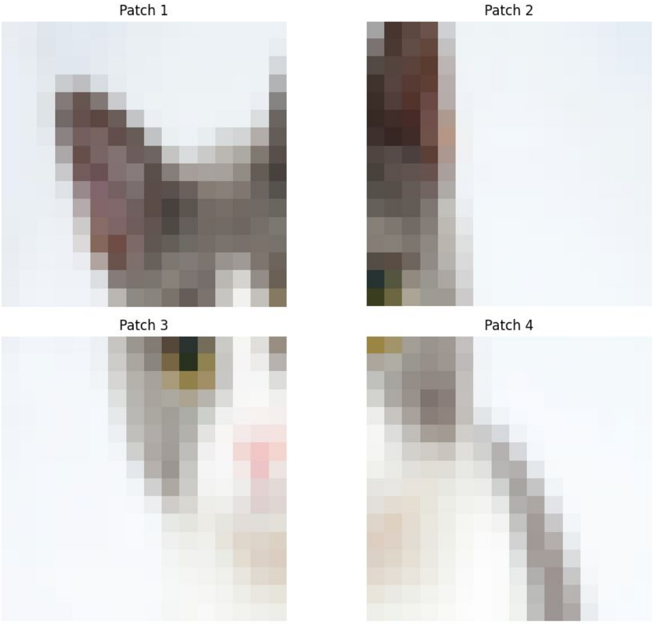
Notice that every pixel of the original image has its own positional embedding value

Colab notebook

The CLS patch / token

CLS
[0.2, 0.1]

CLS
[0.1, 0.3]



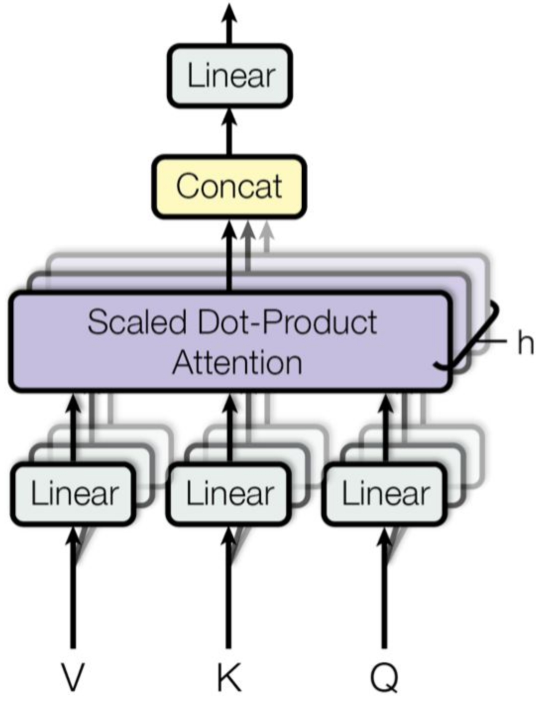

```
cls_token = nn.Parameter(torch.randn(1, 1, embed_dim) * 0.02)
```

Similar class images end up with similar CLS embeddings only after training

The Vision Transformer (ViT) architecture

$$\mathrm{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\mathrm{head}_1, \ldots, \mathrm{head}_h]\mathbf{W}_0$$

$$\text{where } \mathrm{head}_i = \mathrm{Attention}\left(\mathbf{QW}_i^Q, \mathbf{KW}_i^K, \mathbf{VW}_i^V\right)$$



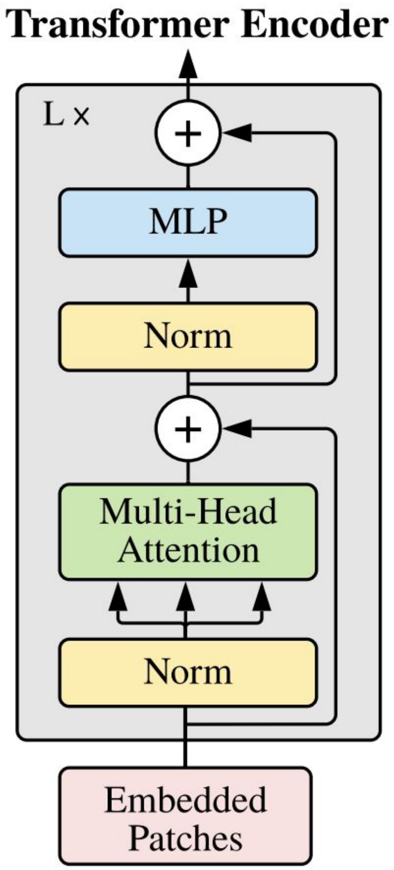
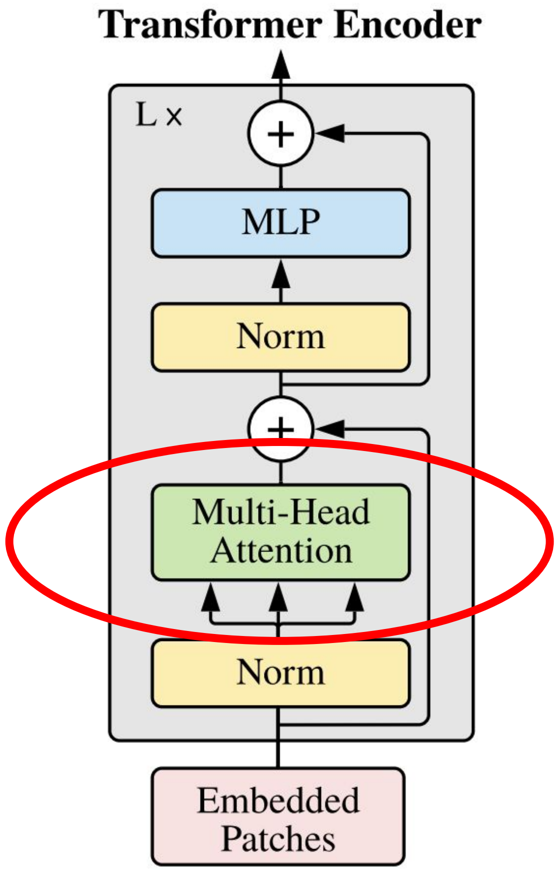


Image from Multi-Head Attention Explained | Papers With Code

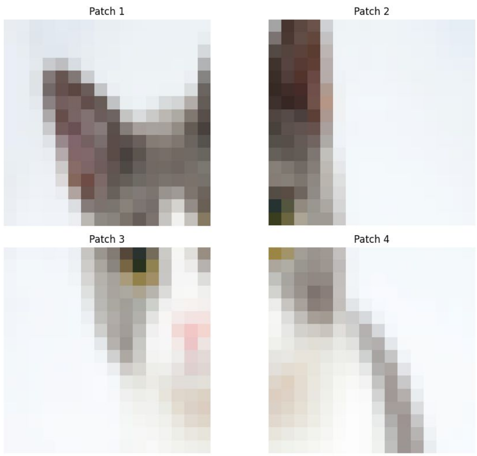
The attention component

$$\mathrm{head}_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



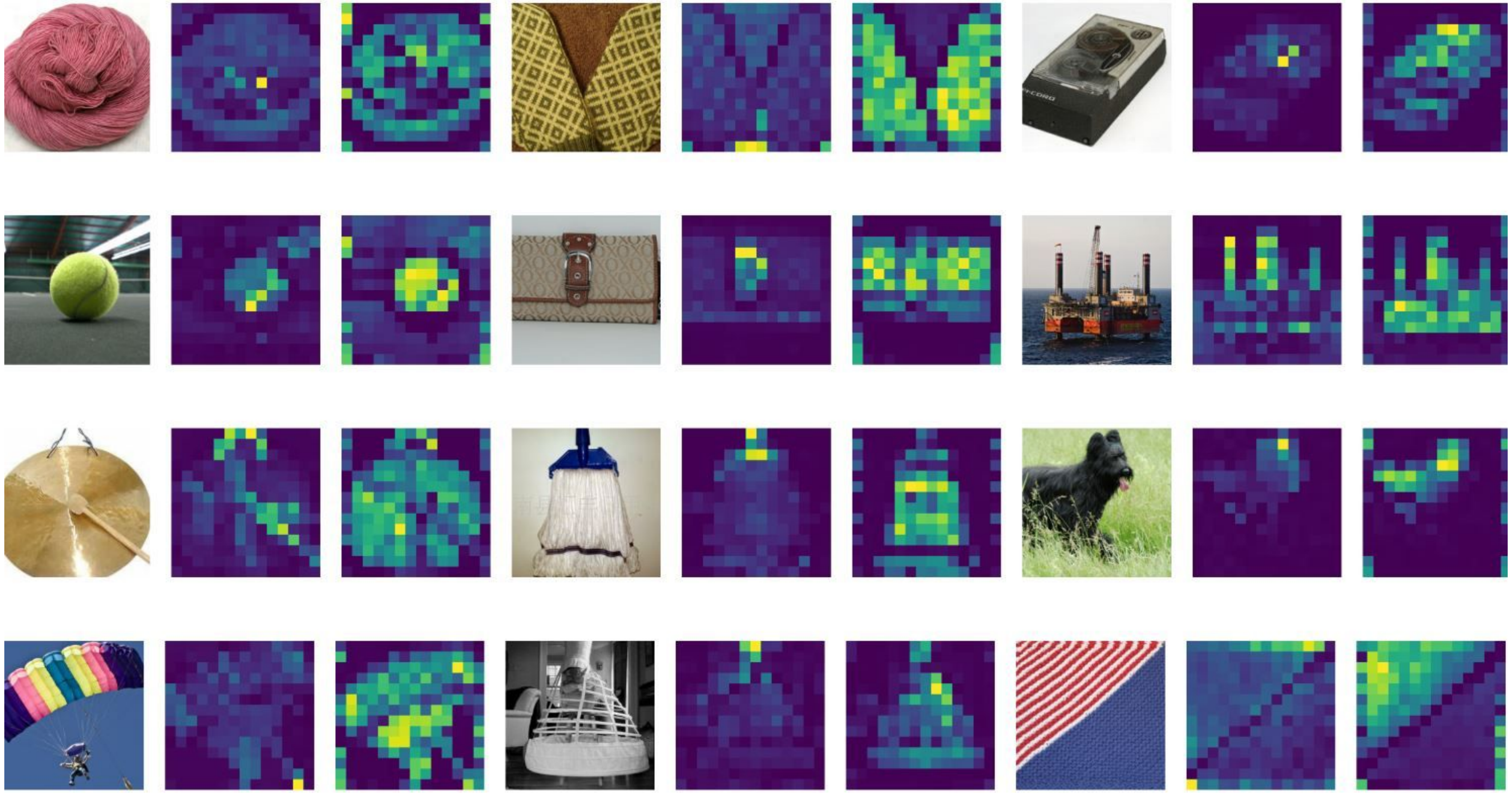
Attention mechanism simplified




```python
def attention_mechanism(patch):
    # Transform patch into query, key, value representations
    query = linear_layer(patch)  # "What am I looking for?"
    key = linear_layer(patch)    # "What do I contain?"
    value = linear_layer(patch)  # "Actual information"

    # Calculate attention scores
    attention_scores = softmax((query @ key.transpose()) / sqrt(d_k))

    # Get weighted sum of values
    output = attention_scores @ value
```

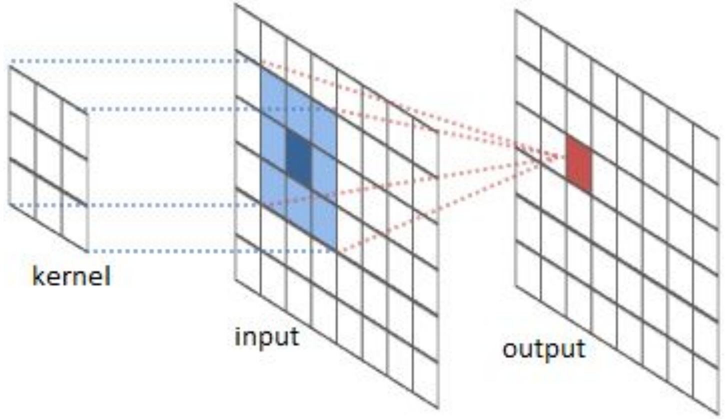
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

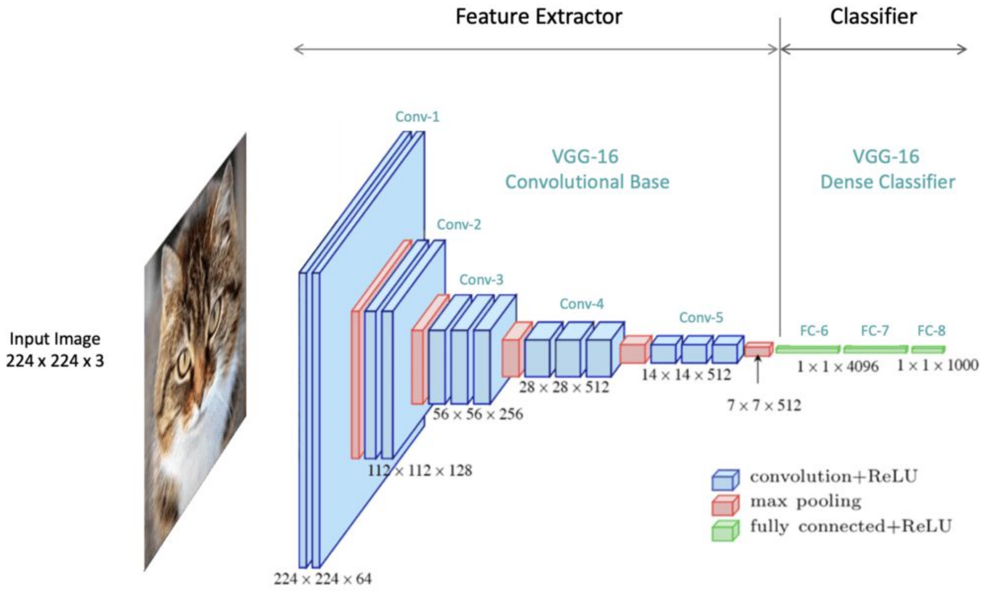
Interpretability of attention maps



Raw images (left) with attention maps of the ViT-S/16 model (right) Source

The locality bias of convolutional networks

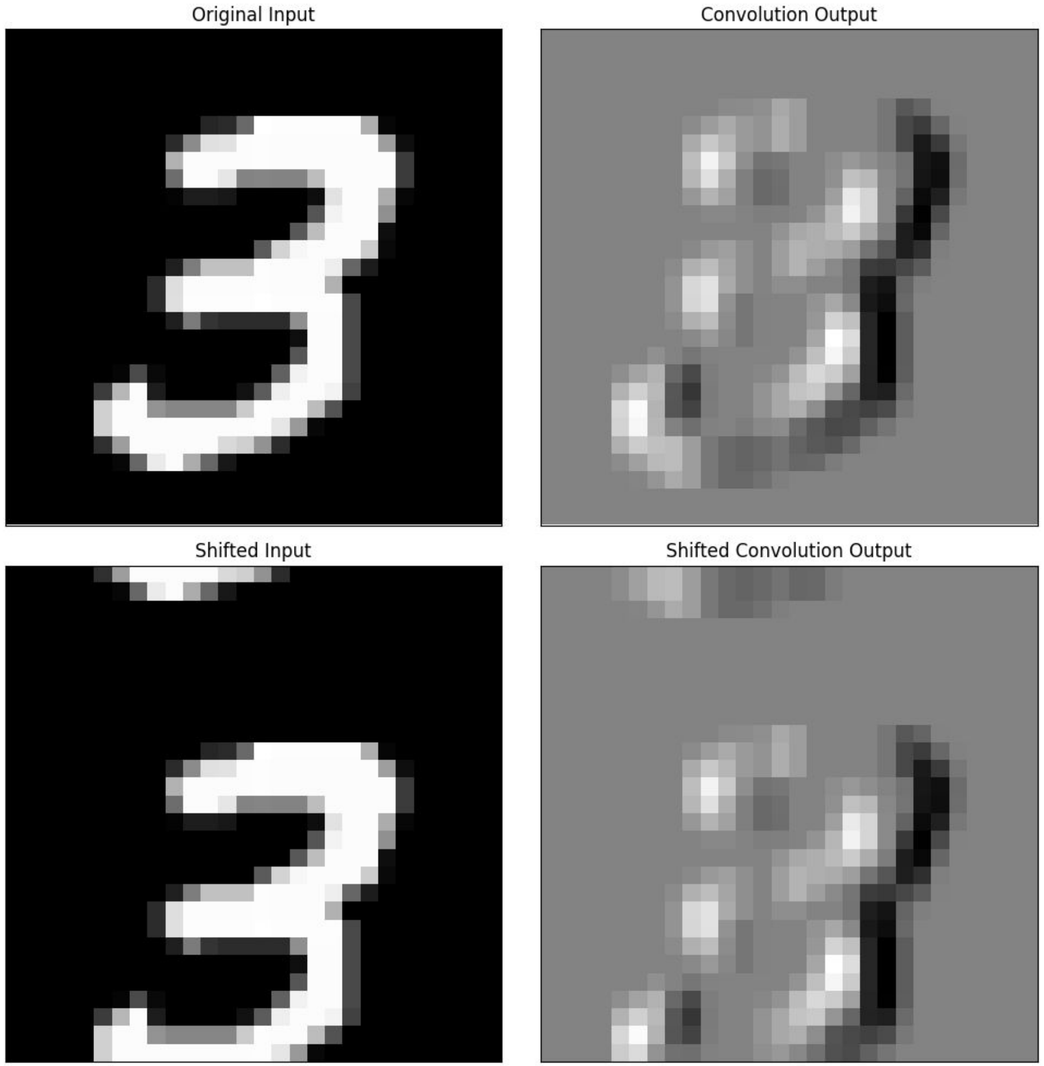




Convolutional networks have the inductive biases of the convolution: pixels in a neighborhood activate together (**locality bias**). The whole image receives the same set of weights in a convolution which creates

We get a “global view” of the image the deeper we go into a convolutional network (aggregating on aggregations) after pooling or strided convolutions.

The ‘translational equivariance’ bias of convolutional networks



$$W = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

If the inputs gets shifted **n** pixels, so does the output.

Weights remain the same while capturing this desirable effect

Tradeoffs of ViT with convolutional networks (CNNs)

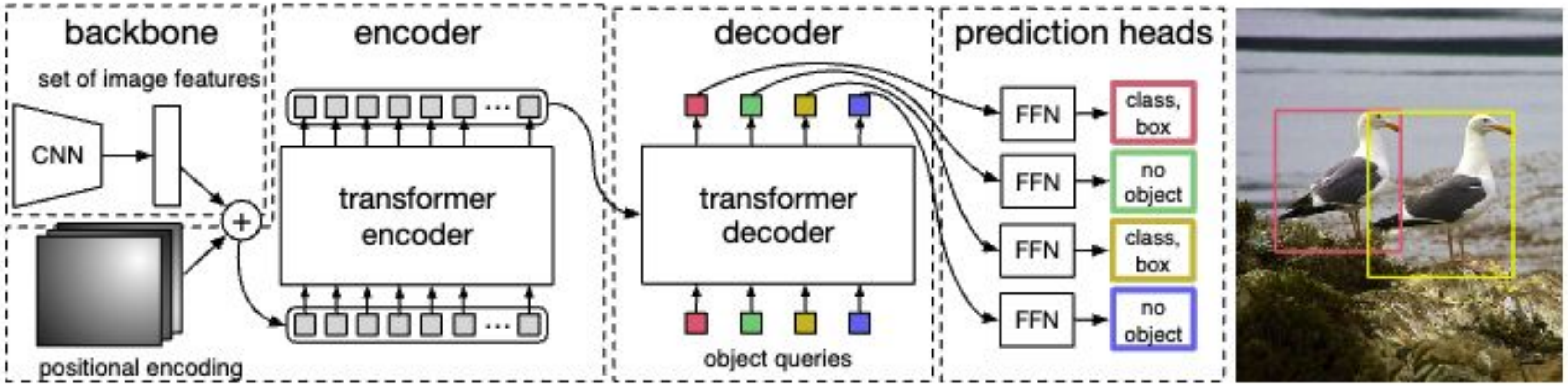
In general transformers use more parameters than CNNs and require:

- More computing power to train and run inference (quadratic time per patch due to self-attention)

- More data to train (lack the translational equivariance of convolution), ViT was trained on a Google-internal dataset containing **300 million labeled images**

- DeIT is a vision transformer variant that relies on distillation and data augmentation and compute to reduce the size of the training data corpus

In practice, many architectures (like DETR) **combine convolutions with transformers**



Overview of the DETR architecture (source)

Summary

The Vision Transformer (ViT) architecture

- ViT splits images into patches and process them through self-attention
- The attention mechanism computes relationships between all image regions
- Position embeddings maintain spatial information of the patches
- The attention mechanism uses queries, keys, and values to compute weighted relationships between patches
- Multiple attention heads capture different features

Tradeoffs with convolutional neural networks (CNNs)

- ViTs excel at capturing global relationships
- ViTs require more compute and data than CNNs
- CNNs offer built-in translation equivariance
- Hybrid architectures like DETR combine the benefits of both approaches

SPONSORED BY THE
Federal Ministry of Education and Research

Further reading and references

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

- https://arxiv.org/abs/2010.11929

End-to-end object detection with transformers

- https://arxiv.org/abs/2005.12872

Training data-efficient image transformers & distillation through attention

- https://arxiv.org/abs/2012.12877

SPONSORED BY THE

Federal Ministry
of Education
and Research