CV / VLMs

Unit 5: State-of-the-Art Object Detection Techniques

5.2.3

Anchor-Free Object Detection

Vision Transformers (ViT) for Object Detection

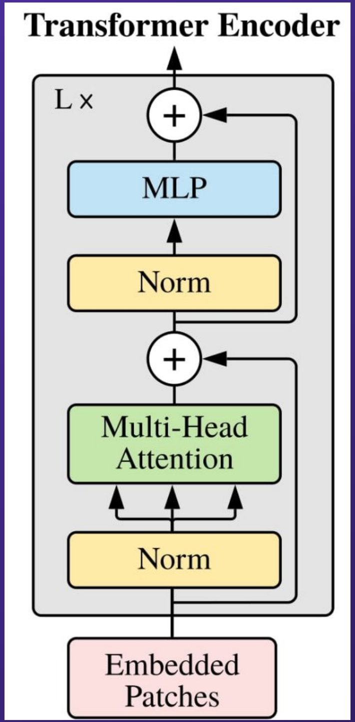
Vision Transformers (ViT)

Overview

- Transformer architecture is the de-facto standard for NLP tasks and is also applicable to computer vision.

- Unlike CNNs, it does not have a locally restricted receptive field (kernels); instead, it converts everything to sequences and processes it all together.

- It can match or surpass state-of-the-art CNNs when trained on datasets larger than 14 million samples. However, for smaller datasets, ResNets or EfficientNets are more effective.

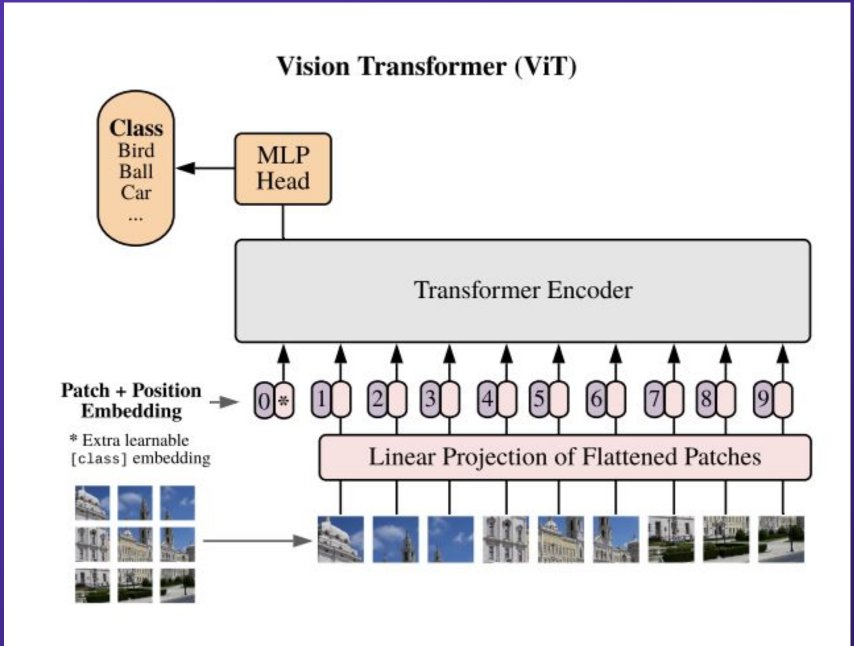


- [Vision Transformer (ViT) (huggingface.co)]
- [Vision Transformer Explained | Papers With Code]
- [How the Vision Transformer (ViT) works in 10 minutes: an image is worth 16x16 words | AI Summer (theaisummer.com)]

TIL-AI
TODAY I LEARNED-AI

Vision Transformers (ViT)
Procedure

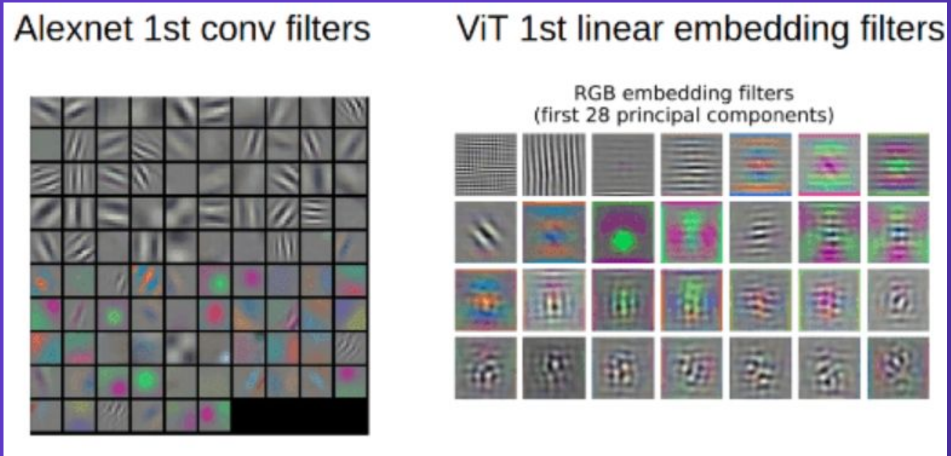
1. Split an image into patches
2. Flatten the patches
3. Produce lower-dimensional linear embeddings from the patches
4. Add positional embeddings
5. Feed the sequence as an input to a standard transformer encoder.
6. Pretrain the model with image labels (fully supervised on a large dataset).
7. Finetune on downstream datasets for image classification



Vision Transformer (ViT) (huggingface.co)

TIL-AI
TODAY I LEARNED-AI

Vision Transformers (ViT)
Visualizing early layers



(top) Early layers of AlexNet and ViT are picking up similar edges and patterns.

As it states in Stanford's Course CS231n: Convolutional Neural Networks for Visual Recognition:

"Notice that the first-layer weights are very nice and smooth, indicating a nicely converged network. The color/grayscale features are clustered because the AlexNet contains two separate streams of processing, and an apparent consequence of this architecture is that one stream develops high-frequency grayscale features and the other low-frequency color features."

TIL-AI
TODAY I LEARNED-AI

Vision Transformers (ViT)

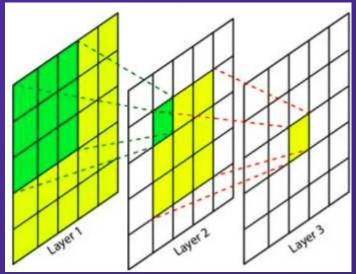
Comparison with CNNs

CNNs are not naturally adept at handling geometric transformations such as rotations, scaling transformations Additionally, their receptive field is restricted.

To overcome these limitations, , data augmentation techniques are employed to make CNNs more robust and adaptable to various transformations.

Transformers lack the inductive **biases of Convolutional Neural Networks (CNNs).**

(top) Image transformations (rotation)
The pug would not be recognizable in a CNN if no specific augmentations are included in the training.



(top) Receptive field of filter kernels
convolution is a linear local operator. Only the neighbor pixel values are considered as indicated by the kernel.

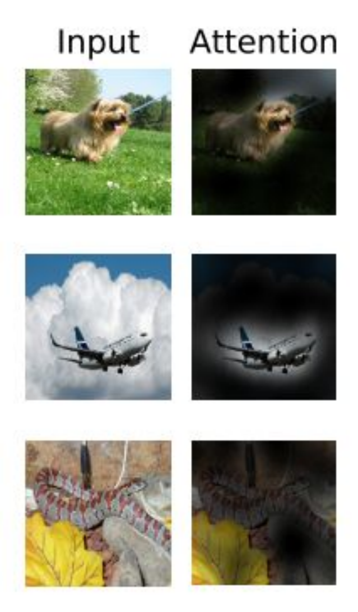
TIL-AI
TODAY I LEARNED-AI

Vision Transformers (ViT)
Comparison with CNNs

The unique mechanism of transformers - **Self-Attention** has some unique advantages:

i) **Embedding**
Transformers has unique ability to relate the combination of image linear embedding with its position embedding.

ii) **Receptive field**
Unlike CNN, self-attention also allows ViTs to integrate information across the entire image (whereas CNN requires layers of pooling and convolution).

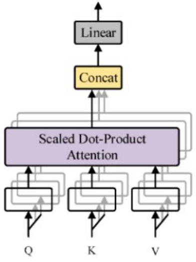


(top) Representative examples of attention from the output token to the input space.

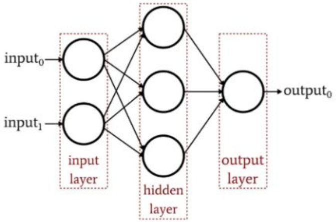
The "Attention" array of image illustrated ViT's ability to focus on regions of images to extract meaningful information.

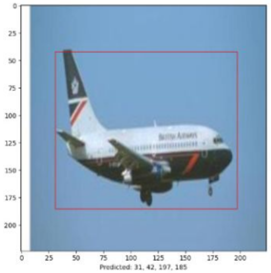
Object detection with Vision Transformers

Image → Linear Transformation → Encoded Image Patch → Multi Head Attention (Self-attention) → Normalization → Fully Connected Layer (FC) → Output (4 dimension) Bounding Box Coordinate)



Multi-Head self-Attention





Patch encoding layer
The PatchEncoder layer linearly transforms a patch by projecting it into a vector of size projection_dim. It also adds a learnable position embedding to the projected vector.

```
class PatchEncoder(layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super().__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(units=projection_dim)
        self.position_embedding = layers.Embedding(
            input_dim=num_patches, output_dim=projection_dim
        )
```

Vit with Bounding Box
The ViT model has multiple Transformer blocks:

- **MultiHeadAttention** layer is used for self-attention, applied to the sequence of image patches.
- **Encoded patches** (skip/resnet connection)
- **Batch normalized** layer
- **Fully connected** (FC) Layer.
- The model outputs four dimensions representing the bounding box coordinates of an object.

```
    # Create a multi-head attention layer.
    attention_output = layers.MultiHeadAttention(
        num_heads=num_heads, key_dim=projection_dim, dropout=0.1
    )(x1, x1)
    # Skip connection 1.
    x2 = layers.Add()([attention_output, encoded_patches])
    # Layer normalization 2.
    x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
    # MLP
    x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)
    # Skip connection 2.
    encoded_patches = layers.Add()([x3, x2])

# Create a [batch_size, projection_dim] tensor.
representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
representation = layers.Flatten()(representation)
representation = layers.Dropout(0.3)(representation)
# Add MLP.
features = mlp(representation, hidden_units=mlp_head_units, dropout_rate=0.3)

bounding_box = layers.Dense(4)(
    features
)  # Final four neurons that output bounding box
```

(left) Vit can also be used for Object Detection via the following layers and process.

Object detection with Vision Transformers (keras.io)