Vision Transformer

[paper (https://arxiv.org/pdf/2010.11929.pdf)](https://arxiv.org/pdf/2010.11929.pdf)

We have been conditioned to make different architectural choices based on the type of data

- Image: Convolutional Neural Network (CNN)
- Text: Transformer

The *Vision Transformer (ViT)* is a proof of concept that Transformers can replace CNN's for Vision tasks.

We have some experience with encoding images as Tokens

- VQ-VAE
    - "tokenizes" an Image
    - creating a spatial grid of Token Ids (indices into a codebook)
    - flattened into a sequence of Token Ids
- CLIP
    - flattened feature maps of a deep layer of a CNN trained for a Vision task
    - *Single* vector encodes an Image
        - **not** a sequence of tokens (i.e., not similar to Text as a sequence of Text Token Ids)
    - can project the flattened vector to a common space
        - shared with the single vectors representing a sequence of Text Tokens

What's wrong with a CNN ?

Transformers scale better than CNN

In our module on the [Transformer (Transformer.ipynb#Complexity:-summary)](Transformer.ipynb#Complexity:-summary) we compared the computational burden of various architectures

- given a sequence $\mathbf{x}$ of length T
- each element of sequence of dimension d
- CNN kernel size k

Type	Parameters	Operations	Sequential steps	Path length
CNN	$\mathcal{O}\left(k * d^2\right)$	$\mathcal{O}\left(T * k * d^2\right)$	$\mathcal{O}\left(T\right)$	$\mathcal{O}\left(T\right)$
RNN	$\mathcal{O}\left(d^2\right)$	$\mathcal{O}\left(T * d^2\right)$	$\mathcal{O}\left(T\right)$	$\mathcal{O}\left(T\right)$
Self-attention	$\mathcal{O}\left(T * d^2\right)$	$\mathcal{O}\left(T^2 * d\right)$	$\mathcal{O}\left(1\right)$	$\mathcal{O}\left(1\right)$

Reference:

- [Table 1 of Attention paper (https://arxiv.org/pdf/1706.03762.pdf#page=6)](https://arxiv.org/pdf/1706.03762.pdf#page=6)
- See [Stack overflow (https://stackoverflow.com/questions/65703260/computational-complexity-of-self-attention-in-the-transformer-model)](https://stackoverflow.com/questions/65703260/computational-complexity-of-self-attention-in-the-transformer-model) for correction of the number Operations calculated in paper

One clear advantage of the Transformer (Self-attention) over the CNN is time

- constant Path Length, versus T for the CNN

The sequence length T (typical: 64-128) is usually less than d (typical: 256).

- So the Transformer entails fewer Operations than the CNN

Inductive Bias

There are a number of *inductive biases* associated with the CNN

- Translation invariance
    - the CNN can recognize a sub-object in the Image, even if its position is shifted
    - this is because the same kernel is slid over the entire Image
- Locality
    - The Convolution operation is local
        - Pixels within the scope of the kernel

Experiments by the authors show that these biases

- give a performance (Accuracy) benefit to the CNN versus the Transformer
- **when training datasets are not very large**

However, once the Transformer is trained on a sufficiently large dataset

- it can transfer these skills to other tasks (transfer learning)
- and surpass CNN performance on these tasks

When training datasets are very large, the relative advantage shifts to the Transformer.

This is not completely surprising

- More training data facilitates larger models (more parameters)
- Larger models are able to create more complex function approximations
    - Recall our lecture on [Neural Network is a Universal Function Approximator (Universal_Function_Approximator.ipynb)](Universal_Function_Approximator.ipynb)

With more training examples,

- the Transformer learns to attend to distant pixels
    - global view versus local view of CNN
- Transfer Learning more successful
    - The ViT seems to learn more general properties of Images when given a very large number
    - Allowing transfer to Target tasks different than training
        - by creating a Target-specific Classification head

Details

The ViT is the Encoder side of the original Encoder/Decoder Transformer.

- Multi-head **non-masked** attention to all inputs
    - Can attend to the entire input
    - No causal ordering restrictions

The Transformer is augmented with a Classification Head (labelled "MLP Head") and trained on a Vision Classification task.

So how exactly do we go from a spatially organized grid $(w \times h \times c)$ of pixel intensities (RGB) to the sequences that a Transformer can process ?

The obvious answer

- Flatten the non-feature (spatial) dimensions into a sequence ("raster order": left to right, top to bottom)
- Sequences would be too long !

Instead: the spatial grid of $(w \times h)$ pixels is decomposed into

- (16×16) groups (called *patches*) of adjacent pixels
- Total number of patches $T = \frac{w*h}{16*16}$

The patches are analogous to tokens of text.

Just like text tokens, the patches are processed through an Embedding layer

- are encoded as vectors using a Linear Projection
    - maps a flattened vector of length $(16 * 16)$ into a vector of length d
    - mapping is a **learned embedding** (just like word embeddings)

The result of converting to patches followed by embedding is a sequence of image embeddings

- of length T (number of patches)
- each element of length d

An optional token `<CLS>` may be prepended to the sequence

- pictured as * in the diagram

As we have seen for other sequence summary/classification tasks

- the `<CLS>` token is taken as a proxy for the summary of the sequence
- that is: the latent state associated with processing the `<CLS>` token is interpretted as a summary of the entire sequence

Alternative to patches

Rather than using raw pixel grids as patches, the authors also suggest using feature maps of a CNN.

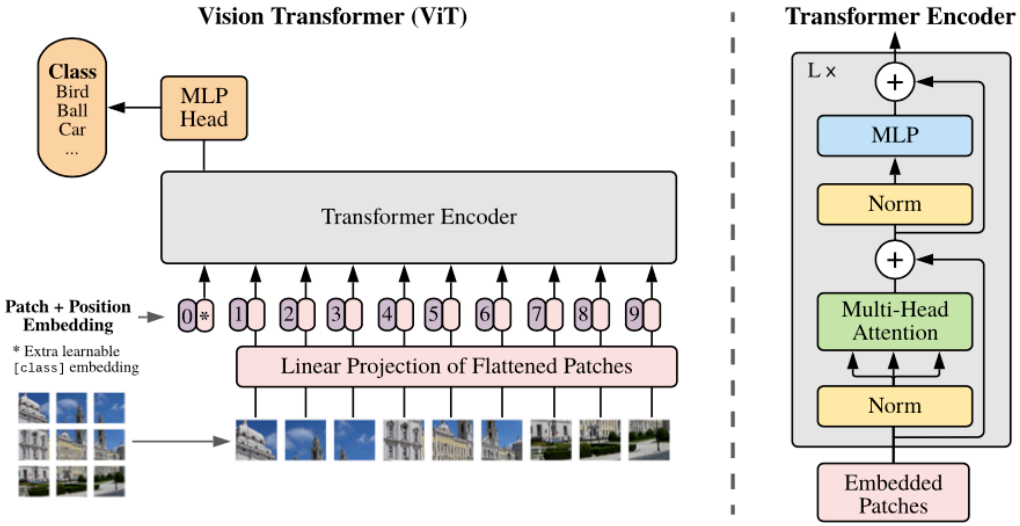
We have seen this approach used before in the [VQ-VAE (VQ_VAE_Generative.ipynb)](VQ_VAE_Generative.ipynb).

The CNN preserves the number of non-feature (spatial) dimensions

- perhaps reducing the length of each, i.e., *down-sampling*
- to retain a spatially oriented grid of *feature* vectors (i.e., the vector formed by the features/channels at each spatial location)
- each vector is analogous to a "patch" of pixels
    - but may have semantics (rather than just syntactic) content

Since the Transformer can attend to any element in the sequence at any time

- a learned Position Embedding is paired with each element of the sequence
- so that the Transformer can infer an ordering of patches
    - eventually will "learn" spatial relationships between patches ?


Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0
*
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches
Transformer Encoder
L x
+
MLP
Norm
+
Multi-Head Attention
Norm
Embedded Patches

Analysis

The authors run a number of experiments to better understand the model.

Position embeddings

The authors experiment with several ways of describing the position of a patch

- no position embedding
- as a one dimensional index into the linearized order of patches
- as a (row, column) pair into the downsized spatial grid of patches

The experiment was run against a single task

- No position embedding performed the worst
- Little difference in performance with the 2D versus 1D position embedding.

The authors speculate

- Since there are a small number of patches
    - the difference between 1D and 2D is less important than it would be
    - if we were required to encode position of individual pixels, which are far greater in number
    - with large number of examples, the ViT can *learn* the 2D spatial relationship

Attention Distance

Recall the Attention lookup mechanism

- There is a *soft lookup* table
    - (key, value) pairs
    - where $\text{key} = \text{value}$
    - and there is one key that is equal to each element of the input sequence
- A *query* is run against the table
    - returning a weighted sum (across keys) of the values associated with the key
    - the weights are the *attention weights*
        - measure of how closely the query matches the key (e.g., cosine similarity)

Given a single example sequence and a single head in the Transformer at some layer l

- run the Transformer against the sequence
- measure the distance (in spatial grid dimensions ?) from the query pixel to each key
- weight the distances by the associated attention weight

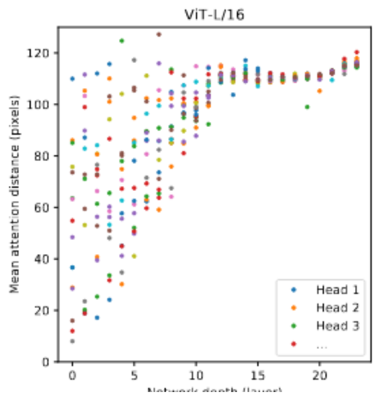
The *Attention Distance* is the average examples of the weighted distance between the query pixel and the other pixels.

The authors provide a diagram of the Attention Distance

- for each layer $1 \leq l \leq L$
- each of the 16 attention heads at each layer
- for examples of spatial dimensions (224×224)

This is analogous to the *Receptive Field* of a CNN.

ViT Attention Distances



In shallow layers: heads attend to pixels at all distances

In deep layers: heads attend to far away ($120 \approx 50\%$ of width) pixels

- For a CNN with kernel size 3 and stride 1 to have a receptive field of 120 pixels
    - would need more than 58 layers

In [1]:
```
print("Done")
```

```
Done
```