

Visual Transformers (ViT)

Aayush Bajaj

July 10, 2026

Contents

1 patchify: an image as a sequence	1
2 the embedding stack	1
3 inductive bias: vit vs cnn	2
4 closing the data gap: DeiT	3
5 hierarchical and hybrid variants	3
6 attention as interpretability	3
7 patchify and embedding shapes, from scratch	3
8 see also	4

the vision transformer asks a blunt question: if attention replaced recurrence for text, can it replace convolution for images? the answer (dosovitskiy et al. 2020, *an image is worth 16x16 words*) is yes — chop the image into patches, treat each patch as a token, and feed the sequence to a standard **transformer** encoder with almost no vision-specific machinery.

the title is literal: a 16x16 pixel patch is one 'word' in the image's sentence

1 patchify: an image as a sequence

a transformer eats a sequence of vectors; an image is a grid of pixels. the bridge is to cut the image into a grid of non-overlapping square patches and flatten each into a vector. for an $H \times W$ image with C channels and patch size $P \times P$, the number of patches is

$$N = \frac{HW}{P^2}, \quad (1)$$

and each patch flattens to a vector of length P^2C . the canonical vit-base setup: 224×224 rgb, $P = 16$, giving $N = 224^2/16^2 = 196$ patches, each a $16^2 \cdot 3 = 768$ -dimensional vector. this is the *entire* act of turning a picture into text-shaped data.

2 the embedding stack

three additions turn flat patches into transformer input:

- **linear patch embedding.** project each flattened patch through a single learned matrix $E \in \mathbb{R}^{(P^2C) \times D}$ to the model width D (768 for vit-base). that one matrix is the only “vision” layer in the whole model.
- **[CLS] token.** prepend a single learnable vector, exactly as bert does. it attends to every patch through the stack, and its *final* state is the image representation fed to the classification head — a token that belongs to no patch, whose job is to aggregate.
- **learned position embeddings.** patches carry no inherent order once flattened, so add a learned position vector to each (including the [CLS] slot). vit uses learned 1d positions and finds 2d-aware variants barely help — the model recovers spatial structure on its own.

the sequence length into the transformer is therefore $N + 1$: the patches plus the [CLS] token.

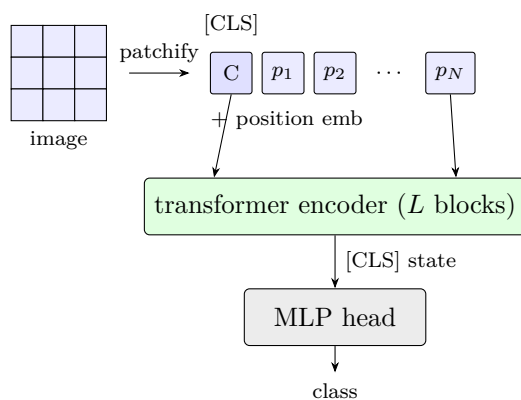


Figure 1: the vit pipeline: an image is gridded into patches, each linearly embedded and tagged with a learned position; a [CLS] token is prepended; the sequence goes through a standard transformer encoder, and the [CLS] output is classified.

after this front-end, the model is a bog-standard transformer encoder — the block, the multi-head attention, the layernorm placement, all as in **transformers**. that reuse is the entire point.

3 inductive bias: vit vs cnn

a **cnn** hard-codes two assumptions about images: **locality** (nearby pixels relate) via small kernels, and **translation equivariance** (a cat is a cat wherever it sits) via weight sharing. these are strong, correct priors — they let a cnn learn from modest data.

vit throws them away. self-attention is global from layer one (every patch sees every patch) and permutation-agnostic (order comes only from the learned positions). the consequence is a sharp **data-dependence**:

- on imagenet-1k alone (~1.3M images), vit *underperforms* a comparable resnet — with no locality prior it must learn from data what convolution assumes for free, and there is not enough of it.
- pretrained on jft-300m (300M images), vit **overtakes** the best cnns and transfers down to imagenet at state-of-the-art accuracy. given enough data, learning the right bias beats being handed a possibly-suboptimal one.

honest statement of the trade: vit is not “better than cnns”, it is **less biased and more data-hungry**. the crossover point is roughly tens of millions of labelled images (dosovitskiy et al. 2020).

*the bitter-lesson pattern again:
general methods that scale with
data and compute eventually
beat hand-built priors*

4 closing the data gap: DeiT

the data-hunger looked fatal for anyone without google’s dataset — until **deit** (touvron et al. 2020, *training data-efficient image transformers*) trained a competitive vit on imagenet-1k *only*, with heavy augmentation and **distillation**. the twist is a **distillation token**: a second special token (alongside [CLS]) that learns to predict the output of a cnn teacher, letting the transformer absorb the teacher’s convolutional inductive bias through its labels rather than its architecture. it made vits trainable without an internet-scale corpus.

5 hierarchical and hybrid variants

plain vit keeps one resolution throughout and pays $O(N^2)$ attention over all patches — fine at $N = 196$, ruinous for dense prediction at high resolution.

- **hybrid**. replace raw-patch embedding with a few conv layers (a cnn stem) that produce the tokens — reintroducing a little locality prior where it is cheapest, often improving low-data behaviour.
- **swin** (liu et al. 2021, *swin transformer*) rebuilds the cnn’s pyramid inside attention. it computes attention only within local **windows** (linear in image size, not quadratic), and **shifts** the window partition every other layer so information crosses window boundaries. merging patches between stages gives a multi-scale feature hierarchy, making swin a drop-in backbone for detection and segmentation where plain vit is awkward.

6 attention as interpretability

because every layer’s attention is an explicit $N \times N$ matrix, you can *see* what a patch looks at. aggregating attention across layers (**attention rollout**) or using self-supervised training (dino) yields maps that segment the salient object with no pixel labels at all — the [CLS] token’s attention lands on the foreground. it is a genuinely useful, if not fully faithful, window into the model — with the standard caveat that attention weights are not a complete causal explanation of the output.

7 patchify and embedding shapes, from scratch

the whole front-end — patchify, linear embed, [CLS], positions — in numpy, tracking shapes at each step.

```
1 import numpy as np
2
3 rng = np.random.default_rng(0)
4
5 # --- a toy image batch: (batch, channels, height, width)
6 B, C, H, W = 2, 3, 224, 224
7 P = 16                                     # patch size (16x16 "words")
8 D = 768                                    # embedding dim (ViT-Base)
9
10 img = rng.normal(size=(B, C, H, W))
```

```

11 n_patches = (H // P) * (W // P)          # HW / P^2
12 print(f"image {C}x{H}x{W}, patch {P}x{P} -> n_patches = HW/P^2 =
13       ↪ {n_patches}")
14
15 # --- patchify: reshape into non-overlapping PxP blocks, flatten each
16 img = img.reshape(B, C, H // P, P, W // P, P)          # split H and W
17 img = img.transpose(0, 2, 4, 1, 3, 5)                    # (B, nH, nW, C, P, P)
18 patches = img.reshape(B, n_patches, C * P * P)          # (B, N, C*P*P)
19 print("flattened patches:", patches.shape, " (each patch is", C*P*P,
20       ↪ "numbers)")
21
22 # --- linear patch embedding: project each flat patch to D dims
23 W_emb = rng.normal(size=(C * P * P, D)) / np.sqrt(C * P * P)
24 tokens = patches @ W_emb                                # (B, N, D)
25
26 # --- prepend a learnable [CLS] token, then add learned position embeddings
27 cls = rng.normal(size=(1, 1, D)) / np.sqrt(D)
28 cls = np.broadcast_to(cls, (B, 1, D))
29 tokens = np.concatenate([cls, tokens], axis=1)          # (B, N+1, D)
30 pos = rng.normal(size=(1, n_patches + 1, D)) / np.sqrt(D)
31 tokens = tokens + pos                                    # broadcast over batch
32
33 print("sequence into transformer:", tokens.shape, " = (batch, N+1 tokens, D)")
34 print("the +1 is the [CLS] token whose final state is the image
35       ↪ representation")

```

image 3x224x224, patch 16x16 -> n_patches = HW/P^2 = 196
 flattened patches: (2, 196, 768) (each patch is 768 numbers)
 sequence into transformer: (2, 197, 768) = (batch, N+1 tokens, D)
 the +1 is the [CLS] token whose final state is the image representation

the shapes tell the whole story: a 224×224 image becomes 196 patch tokens, each a 768-vector; prepend [CLS] and you hand the transformer a length-197 sequence — indistinguishable, from the encoder's point of view, from a short paragraph of text.

8 see also

- [transformers](#) — the encoder block vit reuses unchanged
- [cnn from scratch](#) — the inductive biases vit trades away
- [stable diffusion](#) — where vit-style attention conditions image generation
- [tokenisers](#) — the text analogue of patchify