Vision Transformers | 17

Chapter Abstract. If patching erases a tiny visual cue before the model sees it, attention cannot recover that evidence later. This chapter explains how Transformer ideas are used for image classification and other vision tasks. The running experiment is a modest Food-101 image-classification workflow, where accuracy, speed, and memory must be measured together. The chapter uses image-tensor shapes and Transformer self-attention only where they affect ViT decisions, then develops the image-specific pieces: patches, patch embeddings, class tokens, position information, efficient attention patterns, self-supervised visual pretraining, and practical reporting. The emphasis is applied: a pretrained Vision Transformer is useful only when the data split, image preprocessing, validation protocol, training recipe, cost measurements, and error analysis make the result interpretable.

17.1 Running Example and Applied Goal

An image model does not begin with objects, digits, or diagrams. It begins with a decision about what evidence the model is allowed to see. A Vision Transformer makes that decision by turning image regions into patch tokens and then applying Transformer layers to the resulting sequence. This section sets up the running task and the experimental claim. The next section does the patch arithmetic.

The running task in this chapter is Food-101 image classification. The model must map a photograph to a dish label, but the chapter's real object of study is the ViT recipe that makes that prediction: image size, patch grid, position information, pretrained checkpoint, trainable layers, and measured cost. A single accuracy number is not enough. The report must also say what was fed to the model, which parameters moved during training, and how much time and memory the recipe used.

The first design question is qualitative before it is mathematical: did the patch grid preserve the evidence needed for the label? A coarse grid may blur a crisp edge, garnish, sauce boundary, or small printed mark before attention ever has a chance to connect it with the rest of the image. A finer grid may keep more detail, but it gives the Transformer more tokens to process. That tradeoff is the chapter's starting point.

Patch tokens set the evidence budget. Attention can connect tokens; it cannot restore detail that never became a token.

Food-101 keeps the workflow concrete because the labels are ordinary dish names, while the cues are varied enough to expose the ViT tradeoff. Some classes depend on local texture, such as crisp edges or sauce patterns. Others depend on broader layout, such as container shape, garnish placement, or the relationship between several food regions. A CNN baseline may handle repeated local textures efficiently. A pretrained ViT may help when the useful evidence is distributed across the image. The validation examples, not the architecture name, decide which explanation is credible.

Detail-sensitive images such as charts, screenshots, and rendered expressions appear in this chapter only as stress tests for the same design choice. A thin axis label, decimal point, or operator can vanish under an aggressive resize or fall awkwardly across patch boundaries. These examples are reminders to inspect what the patch grid preserves before attributing a failure to the Transformer layers.

The applied goal in this chapter is to design and report a complete pretrained Vision Transformer run: data source, split rule, image preprocessing, patch size, model checkpoint, trainable layers, optimizer, learning rate, batch size, precision, hardware, stopping rule, and evaluation metric. A recipe that improves validation accuracy by one point but takes four times longer and exceeds common memory limits may be a poor classroom recipe. A faster recipe that is slightly less accurate may be better when it allows controlled ablations and honest error analysis. The chapter therefore treats accuracy, elapsed time, throughput, and peak memory as measurements of the same experiment, not as separate afterthoughts.

Pause and predict

Before computing any token counts, what validation mistakes would suggest that image size or patch size should be the next knob to investigate? What cost measurements would you record beside accuracy?

17.2 From Pixels to Patch Tokens

The central move in a Vision Transformer is to treat an image as a sequence. The original Vision Transformer paper showed that an image can be divided into patches, embedded as vectors, and processed by a Transformer encoder [48]. This does not mean that a patch is a word. A word token is a discrete linguistic unit. A patch token is a learned vector computed from a small rectangular block of pixels. The analogy is useful because both become sequences, but the source of the sequence is different.

Figure 17.1 shows the whole conversion path: pixels become patches, patches become vectors, and position information is added before the Transformer sees the sequence.

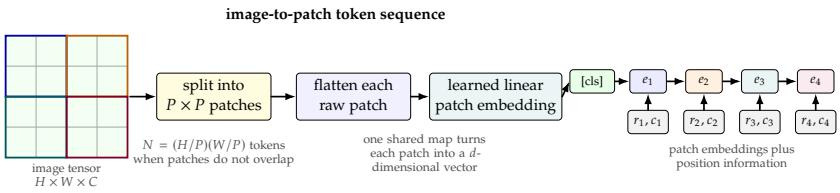

Figure 17.1: A Vision Transformer first converts an image into a sequence: split the tensor into patches, flatten each patch, map it into an embedding vector, and add position information before Transformer layers process the tokens.

Definition 17.2.1 (Patch) *Given an image tensor $x \in \mathbb{R}^{H\times W\times C}$, a patch is a smaller spatial block of the image. A non-overlapping $P \times P$ patch contains P^2C numerical values.*

Definition 17.2.2 (Patchification) *Patchification is the operation that splits an image into patches and orders those patches as a sequence.*

Assume first that P divides both H and W. Splitting the image into non-overlapping $P \times P$ patches gives

$$N = \frac{H}{P}\frac{W}{P}$$

patches. A 224×224 RGB image with 16×16 patches has

$$\frac{224}{16}\frac{224}{16} = 14 \cdot 14 = 196$$

patches. Each raw patch contains $16^2 \cdot 3 = 768$ numbers. In a food image, one such patch might contain part of a garnish, a sauce boundary, a plate edge, or only background.

Definition 17.2.3 (Patch embedding) *A patch embedding is the vector representation obtained after a flattened image patch is passed through a learned map.*

If the flattened patch is $p_i \in \mathbb{R}^{P^2C}$, a common patch embedding is a linear map

$$e_i = p_i W_E + b_E, \qquad e_i \in \mathbb{R}^d,$$

where $W_E \in \mathbb{R}^{P^2C\times d}$ and $b_E \in \mathbb{R}^d$ are learned parameters. For the 224×224 example with $P = 16$, the flattened patch length is 768. If $d = 768$, then W_E has $768 \cdot 768 = 589{,}824$ weights before counting the bias. This is where patch size becomes an information choice: a tiny mark, decimal point, or thin operator can be averaged into a patch before the Transformer ever receives a token.

A tiny grayscale example makes the arithmetic explicit. Consider

$$\begin{bmatrix} 1 & 0 & 2 & 2 \\ 0 & 1 & 2 & 3 \\ 4 & 4 & 0 & 0 \\ 5 & 4 & 1 & 0 \end{bmatrix}.$$

With 2×2 patches, the upper-left patch flattens to $p_1 = [1, 0, 0, 1]$. Suppose a two-dimensional embedding uses

$$W_E = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ -1 & 0 \end{bmatrix}, \qquad b_E = [0.5, -0.5].$$

Then

$$p_1 W_E + b_E = [1, 0, 0, 1]W_E + [0.5, -0.5] = [0, 0] + [0.5, -0.5] = [0.5, -0.5].$$

This toy calculation is not a toy idea. A real Vision Transformer performs the same kind of learned conversion, just with more patches and larger vectors.

The attention chapter used token sequences for text. In a ViT, the token sequence is an ordered list of patch vectors processed by a Transformer.

Definition 17.2.4 (Class token) *A class token is a learned vector prepended to the patch-token sequence in some classification Vision Transformers. After the Transformer layers, the classifier can use that token's final state as an image summary.*

The class token is one summary mechanism, not a requirement for every vision Transformer architecture. In the common classification recipe, however, it is the token whose final representation feeds the classifier head.

The positional-information issue is also inherited from Transformers, but the meaning is visual here. A positional embedding or related position signal tells the model which patch row and column produced a token. Patch order alone is not enough. Without position information, the model would not know whether a patch came from the top-left corner or the bottom-right corner. This is critical for ordinary images and for detail-sensitive diagrams. Moving a garnish, a label, or a chart axis can change the interpretation. Vision Transformers therefore add position information to patch embeddings before the Transformer layers process them.

Patch size is an accuracy-speed knob. Smaller patches preserve detail but increase the token count.

Pause and predict

A 384×384 RGB image is split into 16×16 patches. Compute the number of patch tokens and the flattened length of each raw patch.

The token count is $24 \cdot 24 = 576$. The flattened patch length is still $16^2 \cdot 3 = 768$. The raw size of each patch did not change, but the number of tokens increased from 196 to 576. This is why increasing image resolution is not a free accuracy improvement. It may preserve small visual evidence, but it also gives the Transformer many more tokens to process.

17.3 Self-Attention Over Image Patches

Once an image has become a sequence of patch embeddings, the Transformer machinery from Chapter 10 can be applied, but the tokens still come from pieces of a picture. In a rendered expression, one patch may hold the lower half of a 7, another the multiplication sign, and another a table cell boundary. The model must relate those patches rather than treat each crop as a complete symbol. In a food photograph, a patch of browned texture is weak evidence by itself; it becomes more convincing when nearby patches supply the shape of a crust, plate, or background.

The attention chapter already derived query, key, value, and scaled dot-product self-attention. For ViTs, the new shape to watch is the patch-token count N. An image-token matrix $H \in \mathbb{R}^{N \times d}$ produces an $N \times N$ attention-score matrix for each head, so every patch can compare itself with every other patch in that layer.

For one head, the calculation is

$$Q = HW_Q, \qquad K = HW_K, \quad V = HW_V,$$

$$A = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_h}}\right),$$

$$\text{Attention}(H) = AV.$$

The rows of the softmax matrix say which patch tokens each patch reads. This is the same formula as text self-attention, but the token count N now comes from the image resolution and patch size.

Figure 17.2 makes the key shape issue visible: global attention forms a row for every reading patch and a column for every patch being read.

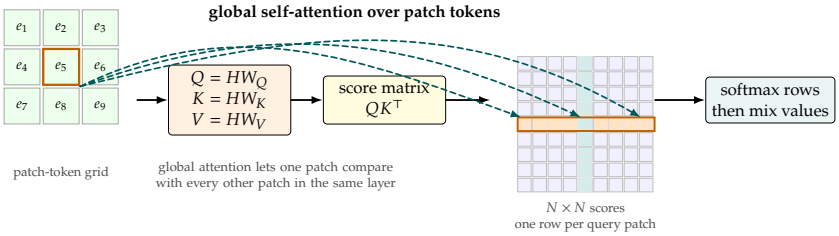

Figure 17.2: For global image self-attention, each query patch forms a score against every source patch. The attention matrix therefore grows quadratically with the number of patch tokens.

The quadratic cost from Chapter 10 now depends on image-derived token count. For global self-attention over N tokens, the attention-score matrix has N^2 entries per head, which is a useful first approximation to the memory and compute pressure of attention. A 224×224 image with 16×16 patches gives $N = 196$, so one attention head forms $196^2 = 38{,}416$ scores in one layer. A 384×384 image with the same patch size gives $N = 576$, so the score count becomes $576^2 = 331{,}776$. The image width and height increased by a factor of about 1.7, but the attention-score count increased by a factor of about 8.6.

The multi-head machinery is the same as in text Transformers. Multiple heads do not remove the cost issue; they multiply the work by a constant factor. This is why image size, patch size, batch size, and number of trainable layers must be chosen together. If a run fails because it exceeds accelerator memory, reducing image resolution or increasing patch size can be more direct than changing the optimizer.

Cost follows tokens. Doubling image side length roughly quadruples tokens and can make global attention much more expensive.

Pause and predict

For a 224×224 RGB image, compare 16×16 patches with 8×8 patches. How many patch tokens does each setting produce, and what happens to the attention-score count?

With 16×16 patches, the token count is 196. With 8×8 patches, the token count is $28 \cdot 28 = 784$. The token count is four times larger, and the attention-score count is sixteen times larger: $784^2/196^2 = 16$. Smaller patches may help with small evidence, but they can also make training much slower.

17.4 Vision Transformers Versus CNNs

Vision Transformers and the convolutional networks from Chapter 4 and Chapter 6 both learn visual representations, but they begin with different assumptions. A convolutional layer applies the same small filter across many locations. This strongly encourages local pattern detection and parameter sharing. A Vision Transformer turns an image into tokens and uses attention to compare those tokens. This allows broad interactions, but the model has less built-in knowledge that nearby pixels are usually related.

Figure 17.3 contrasts the two starting assumptions: local shared filters for CNNs and token-to-token comparisons for ViTs.

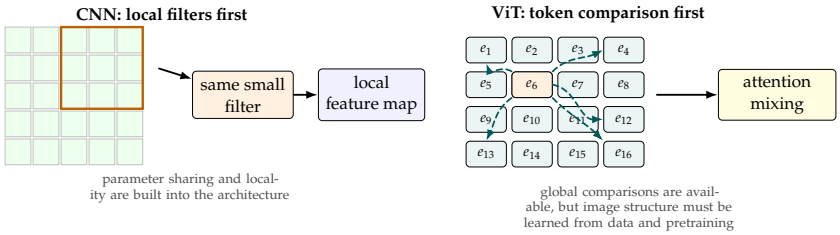

Figure 17.3: A convolutional layer strongly favors local repeated patterns, while a Vision Transformer represents the image as tokens that can attend globally. This difference explains why CNNs can be strong small-data baselines and why ViTs often depend on pretrained representations.

The CNN basics chapter defined inductive bias. In this comparison, the relevant bias is concrete: convolution makes local, repeated visual patterns easy to reuse, while a plain ViT relies more heavily on patch tokens, attention, data, and pretraining to learn comparable visual habits.

Chapter 4 also defined receptive field. The contrast here is that a small convolutional layer has an immediate local receptive field, while global self-attention lets a token directly read every other token in the same layer.

This difference shows up early in an applied project. If the data are a few thousand classroom images, a CNN baseline may win simply because locality and shared filters are already the right bet. A plain ViT trained from scratch has to learn those habits from the data, so it often needs a strong checkpoint before it becomes a fair competitor. The original ViT result emphasized that pure Transformers can perform very well at sufficient scale [48]. DeiT then showed that augmentation, distillation, and careful training can make Image Transformers practical with less data [279].

Using the transfer-learning vocabulary from Chapter 7, a pretrained baseline starts from source-trained parameters and then evaluates or fine-tunes on the target task before more complex changes are attempted. The word baseline should not mean weak. In an applied project, the first baseline should be a serious competitor. For a small image-classification dataset,

a pretrained ResNet, modern ConvNet, DeiT, or ViT can be a better first experiment than a randomly initialized custom model. The goal is to learn what the data and metric require before inventing architecture changes.

Starting point	Useful when	Applied warning
CNN baseline	data is small, speed matters, local texture is important	may miss global layout unless deep or well designed
Pretrained ViT	transfer checkpoint is available and global evidence matters	token count can make high resolution expensive
Hybrid or modern ConvNet	fast training and Transformer-era recipes are both desired	still needs the same validation discipline
Hierarchical transformer	high-resolution images or multi-scale evidence matter	cost must be measured with the chosen input size

A mistake gallery is the image-classification version of the error-inspection habit used in earlier chapters: collect a small set of validation examples where the model is wrong and show the input, true label, predicted label, and confidence when available. This gallery often decides the next experiment more honestly than the architecture name. If the model confuses visually similar food classes, a larger pretrained checkpoint or stronger augmentation may help. If it fails on tiny printed labels in a chart, image resolution or a document-oriented model may matter more. If it makes mistakes on corrupted labels, model changes will not solve the problem. A good applied workflow starts with measurements and examples, not with the assumption that the newest architecture must be best.

Compare recipes. Architecture, preprocessing, and training choices form one experimental recipe.

Pause and predict

A pretrained CNN and a pretrained ViT have similar validation accuracy, but the ViT run takes three times longer. What evidence would justify keeping the ViT as the preferred recipe?

17.5 Training Vision Transformers in Practice

Training a large Vision Transformer from random weights is rarely the right first experiment in an applied project. The realistic starting point is a pretrained checkpoint. The practical work is then to choose the checkpoint, decide how much of it to train, set image preprocessing, and measure whether the result is worth its cost.

As in Chapter 7, fine-tuning starts from a saved pretrained checkpoint instead of random weights, and frozen layers keep selected parameters fixed during training. The ViT-specific choice is which parts of the encoder, head, or adapters should move for the available data and hardware.

A good first run is boring on purpose. Load a small pretrained ViT or DeiT-style checkpoint, resize images to the checkpoint's expected size, replace the classification head, and start with little or none of the encoder changing. If validation accuracy is poor, inspect the wrong examples before touching five knobs. Blurry labels point toward resolution or preprocessing; repeated confusions between similar food classes point toward data, augmentation, or a stronger checkpoint. Only after that diagnosis is a broader fine-tune easy to interpret.

The trainable-parameter count should be reported for each run because freezing a ViT encoder changes both optimization behavior and memory use.

Suppose a pretrained encoder has 85 million parameters and the new classification head has 77,669 parameters. If the encoder is frozen, the optimizer updates only the head, less than 0.1% of the model. This can be fast and stable, but it may underfit if the target images differ substantially from the pretraining images. Unfreezing more layers increases adaptation capacity and memory use at the same time.

LoRA was defined in Chapter 13 as a small trainable low-rank update attached to selected linear layers while the original pretrained weights remain frozen [107]. In a Vision Transformer classifier, the ViT-specific question is where to attach it: common targets are attention projection layers such as query and value projections, while the new classification head is also trained. LoRA gives a middle path between a head-only run and a broader fine-tune. It usually trains more parameters than a new classifier head alone, but far fewer than updating the full ViT body. For a Food-101 homework run, this makes the comparison concrete: keep the same split, checkpoint, image size, and epoch budget, then compare a frozen-body head-only baseline with a LoRA-adapted attention stack plus the classifier head.

Position Embeddings and Resolution Transfer

Changing image resolution after loading a pretrained ViT is not the same as changing the size of an ordinary input tensor. The patch embedding weights can usually be reused when the patch size and channel count stay fixed, but the number of patch positions changes. A checkpoint trained at 224×224 with 16×16 patches has a 14×14 grid, or 196 patch positions. The same patch size at 384×384 gives a 24×24 grid, or 576 patch positions. The model now needs position information for many more patch tokens.

Definition 17.5.1 (Position-embedding interpolation) *Position-embedding interpolation resizes a pretrained grid of position vectors so it can be used with a different image resolution or patch-token grid.*

For a classification ViT with a class token, the class-token position vector is usually kept separate while the two-dimensional patch-position grid is resized. Conceptually, the 14×14 learned position grid is treated like

a small feature map and interpolated to 24×24, then flattened back into the patch-token order. This lets the model start from the old spatial prior instead of learning every new location from scratch.

Interpolation is useful, but it does not make resolution transfer free. A higher-resolution run may preserve small text, decimal points, and chart labels, yet it also increases token count, activation memory, and attention cost. It can also expose a pretraining mismatch: a checkpoint trained mostly on natural photos may not have learned the right spatial habits for documents or visual math, even after its position embeddings are resized. The validation report should therefore record the checkpoint's original image size, the fine-tuning image size, patch size, whether position embeddings were interpolated, and the mistakes that justify the extra cost.

Resolution transfer has a shape cost. Reusing a checkpoint at a larger image size usually requires resized position information and a new cost measurement.

The training knobs are inherited from earlier chapters, but the ViT token count makes their memory consequences sharper. Chapter 5 introduced mini-batches, AdamW, weight decay, and augmentation as controlled recipe choices; Chapter 10 and Chapter 13 used precision, device, batch size, and gradient accumulation as part of applied Transformer training reports. The ViT-specific point is that image size and patch size multiply the number of token activations kept during a forward–backward pass, so familiar knobs can change memory and throughput dramatically [172].

Chapter 16 introduced gradient checkpointing as a general training tool that saves memory by recomputing selected activations during backpropagation instead of storing all of them. In this chapter, the same memory tradeoff becomes especially visible because image size, patch size, and Transformer depth control how many activations must be kept during training.

Training choices must respect task semantics. Random crops, MixUp, CutMix, and strong augmentation can improve classification generalization [315, 312, 279]. They can also damage labels. A horizontal flip is harmless for many food photographs, but it can change a diagram, a chart, or a handwritten expression. A crop that removes a small discriminative region can turn a clear image into an ambiguous one.

Choice	Typical effect	Applied warning
larger image size	may preserve small details	increases tokens or preprocessing cost
smaller patch size	may preserve finer evidence	increases attention cost sharply
larger batch size	can improve throughput and stability	may exceed memory
mixed precision	often speeds training and reduces memory	must be checked for numerical issues
freezing layers	reduces memory and overfitting risk	may limit adaptation
strong augmentation	can improve generalization	can damage labels if the transformation changes meaning

This chapter reports throughput and peak memory beside accuracy. Throughput is the number of examples processed per unit time, often examples per second. Peak memory is the largest amount of accelerator memory used during a run. Suppose two runs reach the same validation accuracy. Run A uses 224×224 images, finishes in 18 minutes, and peaks at 7 GB of memory. Run B uses 384×384 images, finishes in 63 minutes, and peaks at 15 GB. If their mistakes are similar, Run A is the better applied recipe. If Run B correctly reads small labels that Run A misses, then the additional cost may be justified. The conclusion depends on measured errors, not on image resolution alone.

Report cost with quality. Accuracy without time and memory is an incomplete applied result.

Pause and predict

A fine-tuning run reaches high training accuracy quickly but validation accuracy stalls. The mistake gallery shows ordinary images, not tiny unreadable details. Which first change is more defensible: increasing image resolution, freezing more of the pretrained body, or changing several settings at once?

17.6 Efficient Vision Transformers: The Swin Transformer

Global attention is simple to explain, but large images make it expensive. The patch-token count grows quickly when image resolution increases or patch size decreases, and full attention pays for every token-to-token comparison. An efficient vision Transformer changes the attention pattern, the token hierarchy, or both so useful visual evidence can be processed at a practical cost. This section stays at the backbone level; Chapter 20 and Chapter 21 introduce their own task definitions, output formats, and metrics later.

Definition 17.6.1 (Window attention) *Window attention is self-attention applied separately inside local groups of tokens. Tokens attend within their own window instead of attending globally to all image tokens in that layer.*

Swin Transformer uses local windows, shifted windows, and hierarchical feature maps to make Transformer-based vision more efficient and useful for dense tasks [167]. The arithmetic shows why this matters. If an image produces $N = 1024$ patch tokens, full attention forms $1024^2 = 1{,}048{,}576$ scores per head. If those tokens are divided into 16 windows of 64 tokens each, window attention forms

$$16 \cdot 64^2 = 65{,}536$$

scores per head. That is sixteen times fewer scores for that layer. The trade off is that a token cannot immediately read every token in the image; shifted windows and multiple layers help information move across windows.

Swin also changes the shape of the representation as depth increases. Neighboring patch groups are merged, so later stages operate on fewer tokens with wider feature vectors, much like a CNN becomes coarser and more semantic in deeper layers. This hierarchy can expose feature maps at several spatial scales instead of only one final class-token summary. The point for applied work is not that every task needs Swin specifically. It is that efficient attention and multi-scale features are architectural choices that must match the image resolution, compute budget, and evidence the task depends on.

Figure 17.4 shows the architectural idea: the model reduces score-matrix cost while carrying a spatial feature map forward for later layers or task heads.

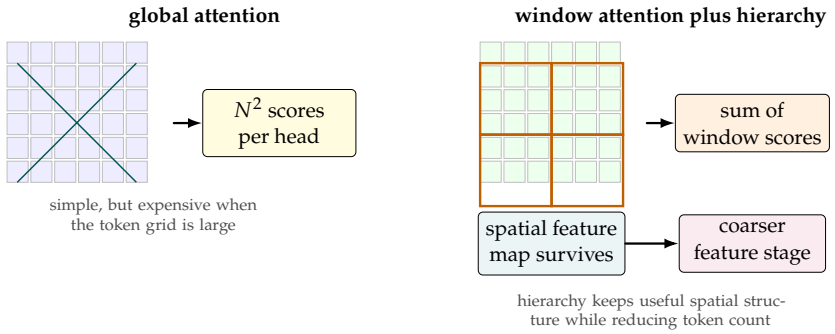

Figure 17.4: Efficient vision Transformers reduce attention cost by restricting some layers to local windows, while hierarchical feature stages preserve spatial structure at lower token counts.

Definition 17.6.2 (Hierarchical representation) *A hierarchical representation stores features at multiple spatial resolutions. Early stages keep many fine-grained tokens, while later stages use fewer tokens with wider feature vectors.*

For applied work, efficient attention should be justified by measurement rather than by architecture fashion. If a higher-resolution ViT improves validation accuracy but doubles memory and cuts throughput in half, the report should show that tradeoff. If the mistake gallery shows missed tiny details, a smaller patch size or larger input may be worth the cost. If the errors are ordinary class confusions, better augmentation, fine-tuning, or a stronger checkpoint may matter more than changing the attention pattern. Efficient ViTs are tools for making the desired image evidence affordable; they do not remove the need for a baseline, ablation, and cost report.

Measure the recipe. Efficient attention matters when it changes the accuracy–cost tradeoff on the target validation split.

Pause and predict

Why can window attention greatly reduce the number of attention scores while still needing shifted windows or multiple layers to move information across the whole image?

17.7 Self-Supervised Visual Pretraining and Transfer

Large labeled image datasets are expensive. Chapter 8 already used self-supervised learning for text: the target is constructed from the input data rather than manually supplied labels. In vision, the same idea often appears as a *pretext task*, a constructed task whose main purpose is to learn representations for later use. This matters even when an applied project never pretrains a ViT from scratch, because the downloaded checkpoint may owe much of its usefulness to self-supervised, weakly supervised, or image-text pretraining. The transfer-learning vocabulary from Chapter 7 still applies: representation transfer means using features learned on one task or dataset as the starting point for another task or dataset.

Masked image modeling is one important family. A masked autoencoder hides many image patches, encodes the visible patches, and trains a decoder to reconstruct the missing content [88]. If an image has 196 patches and the method masks 75 percent of them, then only 49 patches are visible and 147 patches are hidden. The model must learn visual regularities because it cannot solve the task by copying every patch directly.

Definition 17.7.1 (Masked image modeling) *Masked image modeling is a self-supervised training method in which part of an image is hidden and the model is trained to predict missing pixels, representations, or patch content from the visible part.*

If M is the set of masked patch indices, a pixel-reconstruction version can use

$$\mathcal{L}_{\text{MIM}} = \frac{1}{|M|} \sum_{j \in M} \left\| x_j - \hat{x}_j \right\|_2^2 .$$

Other variants reconstruct latent features or discrete targets, but the common principle is the same: only hidden patch content supplies the pretext loss.

Figure 17.5 summarizes the masked-image pretraining loop in patch terms.

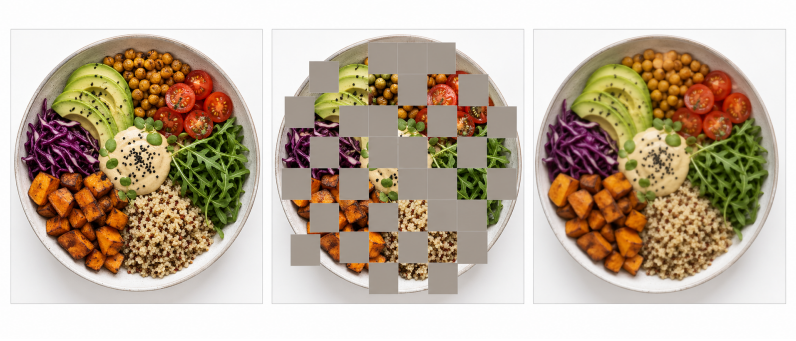

Figure 17.5: Masked image modeling can be read as a patch-level pretext task: the model sees only part of an image, predicts the hidden content or representation, and later transfers the learned visual structure to the target task.

The analogy to masked language modeling is useful but imperfect. Missing words are discrete tokens. Missing image patches are continuous regions with strong local redundancy. In a natural image, nearby patches often share

color, texture, and structure. In a detail-sensitive diagram or document image, hiding a small mark can remove crucial information. The usefulness of a pretext task depends on the structure of the target data.

Chapter 16 introduced distillation as training one model to match teacher outputs or representations. DINO uses that idea in a self-supervised visual setting, but the teacher is not necessarily a separate supervised model: it may be a slow-moving copy of the learner, for example updated by averaging learner parameters over time. Two differently augmented views of the same image are passed through related networks: a learner network and a teacher network. The learner is trained to match the teacher's output distribution for another view, while the teacher is updated slowly from the learner's weights rather than from ordinary label loss. No human class label is required; the training signal is that different views of the same image should lead to stable visual representations [20].

DINOv2 pushed that idea into stronger general-purpose visual checkpoints [201]. For an applied ViT workflow, a DINO-style checkpoint is a representation candidate: it may transfer well when labels are scarce or when a good visual backbone matters more than a closed-set classifier head. It is not a guarantee that the target task is solved. This DINO family is about representation learning; later chapters use the same name for a separate detector family with a different training recipe.

CLIP-style contrastive pretraining learns from paired images and text [218]. The differences matter in use. An image-text checkpoint may handle a new food label written in ordinary language better than a closed-set classifier, while a masked-image checkpoint may preserve spatial detail but still miss the meaning of a tiny exponent. The pretraining method gives a guess about transfer, not a guarantee.

For a batch of B image-text pairs, a CLIP-style image-to-text term can be written as

$$\mathcal{L}_{\text{i2t}} = -\frac{1}{B}\sum_{i=1}^{B}\log\frac{\exp(z_i^\top t_i/\tau)}{\sum_{j=1}^{B}\exp(z_i^\top t_j/\tau)},$$

with a symmetric text-to-image term. The matched caption sits on the diagonal; the other captions in the batch become contrastive alternatives.

Pretraining type	Target signal	Transfer intuition
supervised classification	human class labels	learn features useful for named categories
contrastive image-text	matching image and text pairs	align visual and language representations
masked image modeling	hidden patch content or representations	learn structure from incomplete images
self-distillation	teacher outputs across views	learn stable features without manual labels

CLIP-Style Checkpoints in a ViT Workflow

The detailed CLIP vector-space story belongs in Chapter 8: two encoders produce normalized vectors, and image-text matching is scored by cosine similarity. In this chapter, the relevant question is more architectural and experimental. A CLIP-style model may use a ViT image encoder internally, so the image side still begins with patch tokens, position information, and Transformer blocks. What changes is the pretraining signal: the image representation has been shaped to align with text prompts, not only to predict a fixed supervised class list.

This makes a CLIP-style checkpoint a useful zero-shot or few-shot baseline when class names are naturally expressible as prompts. For a Food-101 task, prompts such as `a photo of ramen` or `a photo of chocolate cake` can be embedded and compared with the image embedding before any classifier head is trained. That is a baseline, not an oracle. Prompt wording, cultural food names, fine-grained visual differences, and domain shift can all change the ranking. The right report therefore treats prompt templates as part of the recipe and validates them on the same validation split used for other model choices.

CLIP-style alignment also does not turn a classifier into a detector, OCR system, or reasoning model. If the target output is a box, a mask, a text span, or a generated answer, later chapters use models whose training objectives and interfaces match those contracts. For the ViT chapter, the practical use is narrower: try image-text pretrained encoders when open-vocabulary labels or retrieval-like scoring are central, and compare them against supervised or self-supervised visual checkpoints under the same data split, image preprocessing, and error-analysis discipline.

CLIP is still a recipe. Prompt templates, image encoder, preprocessing, and validation split are all part of the comparison.

For applied work, the main decision is usually not which pretraining algorithm to invent. It is which pretrained checkpoint to trust enough to try first, and what evidence would make you abandon it. A checkpoint pretrained on natural photographs may transfer well to food recognition and poorly to tiny chart text. A checkpoint trained with image-text pairs may follow open-vocabulary labels better than a closed-set classifier. Validation examples decide this. Pretraining explains why a checkpoint may work; it does not remove the need for target-task measurement.

Transfer must earn trust. A strong checkpoint is a hypothesis; target validation examples decide whether it transfers.

Pause and predict

If a masked autoencoder hides 75 percent of a 16-patch toy image, how many patches remain visible? Why might this pretext task still fail to teach the model to read tiny arithmetic symbols?

17.8 Evaluation, Reporting, and Debugging

Chapter 5 supplies the general supervised-training checklist: controlled comparisons, validation-based selection, one final test evaluation, and enough metadata to rerun the experiment. The ViT-specific addition is

that image tokenization and pretrained checkpoints make cost and failure evidence visible in new places. A validation ablation should change one planned ViT factor, such as image size, patch size, trainable layers, or LoRA targets, while keeping the rest of the recipe fixed as much as practical.

A compact ViT report can therefore start from the earlier checklist and add the fields that decide whether this visual Transformer recipe is interpretable: checkpoint name, image size, patch size if visible, trainable parameter count, adapter or full-fine-tuning choice, gradient accumulation, precision, throughput, peak memory, and several inspected mistakes. These details are not bureaucracy. They tell another reader whether the gain came from the model, the resolution, the adaptation method, or a different compute budget. The table is boring on purpose; without it, a prettier validation number can hide a changed experiment.

ViT evidence. Report checkpoint, tokenization, trainable parts, cost, and inspected mistakes beside accuracy.

For the Food-101 run, this checklist becomes concrete. If a larger image size helps ramen but not gnocchi, the report should show whether the improvement came from visible texture, larger object area, or a checkpoint that already knew more about food photographs. Without that callback, the run table can look precise while hiding the reason the ViT change mattered.

Error analysis keeps the next ViT change tied to observed failures. Consider a validation set of 600 images. If a pretrained ViT correctly classifies 510 of them, the validation accuracy is $510/600 = 0.85$, or 85 percent. That number is useful, but it is not enough. If most of the 90 mistakes are low-resolution images with small text, increasing image size may be defensible. If many mistakes are between visually similar classes, data augmentation or a stronger checkpoint may be more relevant. If several mistakes are mislabeled examples, changing the model is the wrong first response.

A useful mistake gallery does not need to be large. Ten inspected failures are often enough to change the next experiment. For example, if several wrong predictions are crop-like images where the object fills only a corner, larger input resolution may help. If the errors are all fine-grained class pairs with clean labels, a stronger checkpoint or more class-balanced examples is more defensible. If the gallery contains labels that a careful human would dispute, the split deserves attention before the model does.

Figure 17.6 shows the reporting loop that keeps those decisions grounded in measured quality, resource cost, and inspected examples.

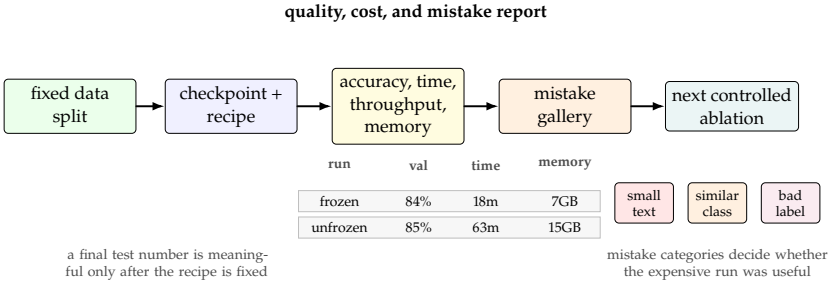

Figure 17.6: A useful ViT report connects the split, recipe, quality metric, resource cost, and inspected errors. The next experiment should follow the observed failure mode rather than changing several settings at once.

A clear report pairs quality and cost because ViT improvements often create an accuracy-speed tradeoff: a change improves a quality metric such as accuracy but also increases elapsed time, memory, or latency. A bounded

Food-101 subset run on a single accelerator makes the tradeoff concrete. Both rows used `google/vit-base-patch16-224-in21k`, 224×224 images, the same training subset, and the same 600-example validation split. The LoRA row improved validation accuracy by seven percentage points, but it also roughly doubled elapsed time, halved training throughput, and used more memory:

Run	Validation accuracy	Time	Examples/sec.	Peak memory
Frozen body	72.5%	33.7 s	510.6	0.433 GB
LoRA adapters	79.5%	63.1 s	250.0	1.387 GB

The LoRA run trained 667,493 parameters instead of only the 77,669-parameter classifier head. Its mistake table still confused fine-grained food pairs such as tuna tartare versus beef tartare and miso soup versus ramen, so the next experiment should inspect whether those errors are label ambiguity, visual similarity, or insufficient adaptation. The point is comparability: use the same split, image size, batch size, throughput definition, and memory measurement rule for both rows before deciding whether the extra trainable adapters were worth the cost.

Chapter 8 defined data leakage. In ViT image work, leakage can be subtle. Duplicate images may appear across splits. The same source image may be cropped into several examples and divided across training and test. The test set may be inspected repeatedly while hyperparameters are chosen. These are not small administrative errors; they change what the reported number means.

Pause and predict

A run with larger images improves validation accuracy from 84 percent to 85 percent but doubles training time and memory. What additional evidence would you inspect before deciding whether the change is worthwhile?

17.9 Summary

A Vision Transformer treats an image as a sequence of patch tokens. The image tensor is divided into patches, each patch is flattened and embedded, position information is added, and Transformer layers mix information across the token sequence. This view makes the connection to text Transformers clear, but it also exposes a practical cost: image resolution and patch size determine the number of tokens, and global attention forms a score matrix whose size grows quadratically with the token count. Before choosing a larger model, ask whether the patch grid preserved the evidence the task needs.

Do not treat the architecture name as the result. Pretrained checkpoints, image size, trainable layers, batch size, mixed precision, augmentation, elapsed time, throughput, and peak memory all affect whether a run is useful. When a pretrained ViT is used at a new resolution, position embeddings may need to be interpolated and the larger token grid must be measured as a new recipe. CNNs, Vision Transformers, and hybrid or hierarchical transformer models are starting points for experiments, not answers by themselves. Swin Transformer illustrates this applied lesson: local windows, shifted windows, and hierarchical features are useful because they reduce attention cost while keeping spatial structure available inside the visual backbone.

Self-supervised and image-text pretraining explain why modern visual checkpoints are worth trying, but target validation examples decide whether the borrowed representation actually helps. DINO-style self-distillation, masked image modeling, supervised classification, and CLIP-style image-text training give different reasons a checkpoint might transfer. CLIP-style models remain embedding systems when they score image-text similarity, and they become ViT-relevant here when their image encoders and pretraining signals affect checkpoint choice. A credible final report names the split and recipe, reports quality and cost together, shows what the model gets wrong, and keeps the test set for the final claim.

Chapter 18 is the adjacent comparison: it asks what convolution still buys after the Transformer view, using the same accuracy, speed, memory, and failure evidence. Chapter 19 later uses visual encoders and image tokens to connect this patch-based view of images to language generation. Chapter 20 and Chapter 21 introduce their own spatial-output definitions and metrics while reusing some of the same backbone ideas.

17.10 Exercises

Exercise 1. A 224×224 RGB image is split into 16×16 non-overlapping patches. Compute the number of patch tokens and the flattened length of each raw patch.

Exercise 2. A 384×384 RGB image is split into 16×16 patches. Compute the number of patch tokens. Compare the global attention-score count with the 224×224 case from the previous exercise.

Exercise 3. The grayscale image

$$\begin{bmatrix} 2 & 0 & 1 & 1 \\ 3 & 2 & 0 & 1 \\ 4 & 4 & 5 & 0 \\ 4 & 3 & 5 & 1 \end{bmatrix}$$

is split into 2×2 patches in row-major order. Write the four flattened patch vectors.

Exercise 4. Let $p = [1, 2, 0, 1]$,

$$W = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \\ -1 & 1 \end{bmatrix}, \qquad b = [0.5, -0.5].$$

Compute the patch embedding $pW + b$.

Exercise 5. Suppose an image produces $N = 6$ patch tokens. How many entries are in one full attention-score matrix for one head? If the image resolution doubles in both height and width while patch size stays fixed, what happens to N and to the score-matrix size?

Exercise 6. An image produces 1024 patch tokens. Compare the number of attention scores in full attention with the number in 16 local windows of 64 tokens each. By what factor is the score count reduced?

Exercise 7. Explain why a CNN baseline may be preferable to a ViT baseline for a small image dataset. Then give one situation where the ViT baseline may be the more defensible first choice.

Exercise 8. A ViT fine-tuning run has high training accuracy but poor validation accuracy. The mistake gallery shows ordinary images rather than tiny unreadable details. Which change would you try first: increasing image resolution, freezing more layers, or changing several settings at once? Explain the reason.

Exercise 9. Match each pretraining method to its target signal: supervised classification, contrastive image-text pretraining, masked image modeling, and self-distillation. For each method, name one reason transfer might fail on a Food-101 fine-tuning task. Then explain why the CLIP similarity calculation belongs in the embeddings chapter while CLIP-style checkpoint choice also matters for a ViT workflow.

Exercise 10. Design a minimal report table for a pretrained ViT fine-tuning experiment. Include fields for data, model, image preprocessing, trainable parameters, optimizer, validation metric, final test metric, speed, memory, and error analysis.

17.11 Homework: Fine-Tune and Audit a LoRA-Adapted Vision Transformer

The homework is an audit of a small transfer-learning experiment, not a contest for the highest Food-101 number. Fine-tune a pretrained Vision Transformer, compare a head-only baseline with one LoRA adaptation, and make the result auditable: fixed split, visible recipe, measured cost, inspected mistakes, and one final test measurement after the recipe is chosen.

The companion implementation lives in the chapter code directory. Use the notebook route, `pretrained_vit_walkthrough.ipynb`. For repeatable command-line runs and artifact generation, use `pretrained_vit_experiment.py`.

Required Work

Use Food-101 as the default dataset. The official training split should be shuffled with a fixed seed and divided into training and validation examples; the official test split must remain untouched until the final selected recipe is evaluated. If hardware cannot support a full Food-101 run, use a controlled Food-101 subset or the CIFAR-10 fallback in the companion script, and state exactly what was reduced. Run `google/vit-base-patch16-224-in21k` as the main checkpoint unless the instructor approves a smaller DeiT checkpoint for hardware reasons. Use a fixed image size, record the preprocessing, replace the classification head with a 101-class Food-101 head, and train a head-only baseline with the pretrained body frozen. Then run one controlled ablation: LoRA fine-tuning with the same dataset, split, checkpoint, image size, and epoch budget. A sensible default is LoRA rank 16,

alpha 16, dropout 0.1, attention target modules `query` and `value`, and `classifier` saved with the adapter. The ablation should change the adaptation method, not several unrelated settings at once.

Notebook Walkthrough

Use `pretrained_vit_walkthrough.ipynb` for executable cells. The in-book contract is to freeze the Food-101 split, train the head-only baseline, run one LoRA ablation with the same data and checkpoint, inspect validation mistakes, report cost, and run the final test once after selecting the recipe.

Required Run Table

Run	Model and data	Training recipe	Cost	Validation result
Head-only baseline	Checkpoint, Food-101 split, image size, trainable parameters	Optimizer, learning rate, batch size, precision	Time, throughput, peak memory	Accuracy and mistake summary
LoRA ablation	Same fields, with LoRA rank, alpha, dropout, targets, and saved modules	Same fields, with dependent changes noted	Same cost fields	Accuracy and mistake summary

Evaluation

Report the data source, split rule, model, preprocessing, trainable parameters, optimizer settings, hardware, cost, validation result, final-test result, and mistake analysis. The error analysis should say whether the LoRA corrections justify the added cost or whether the baseline is the simpler applied choice.