

Vision Transformer

Vision Transformers (ViT)

Objectifs

- Comprendre le **mécanisme d'attention** et la structure d'un **Transformer Encoder**.
 - Comprendre comment adapter un Transformer au **traitement d'images** (patchification, embeddings, tokens).
 - Comparer **CNN vs Transformers** (biais inductifs, besoin en données, coût, performances).
 - Situer ViT dans l'écosystème : **DeiT, Swin, TrOCR** et modèles hybrides.
-

1. Vision Transformer (ViT) : vue d'ensemble

1.1 Pipeline général

Pour une image \times de taille $H \times W \times C$:

1. **Découpage en patches** (patchify) de taille $P \times P$.
2. Chaque patch est aplati en vecteur et passé dans une **projection linéaire** : Patch Embedding .
3. On ajoute un **token de classification** [CLS] (optionnel selon les variantes).
4. On ajoute un **encodage de position** (position embedding).
5. La séquence de tokens passe à travers L blocs **Transformer Encoder**.
6. Une **tête** (classification, régression, segmentation, etc.) produit la sortie.

Forme des tenseurs (classification) :

- Nombre de patches : $N = (H/P) \times (W/P)$
- Dimension d'embedding : D
- Séquence en entrée de l'encodeur : $(N+1) \times D$ si [CLS] , sinon $N \times D$.

1.2 Pourquoi ça marche en vision ?

- **Self-attention** : capture des dépendances longues distances (objets, contexte global).
- **Scalabilité** : performance augmente fortement avec plus de données / compute.

- **Transfert** : pré-entraînement sur gros dataset, puis fine-tuning sur tâches spécialisées.

Limites :

- Moins de **biais inductif** qu'un CNN (pas de localité/translation equivariance "gratuite").
 - En pratique, ViT "pur" demande souvent **beaucoup de données** ou un pré-entraînement solide.
 - Coût attention $O(N^2)$ (sur le nombre de tokens/patches).
-

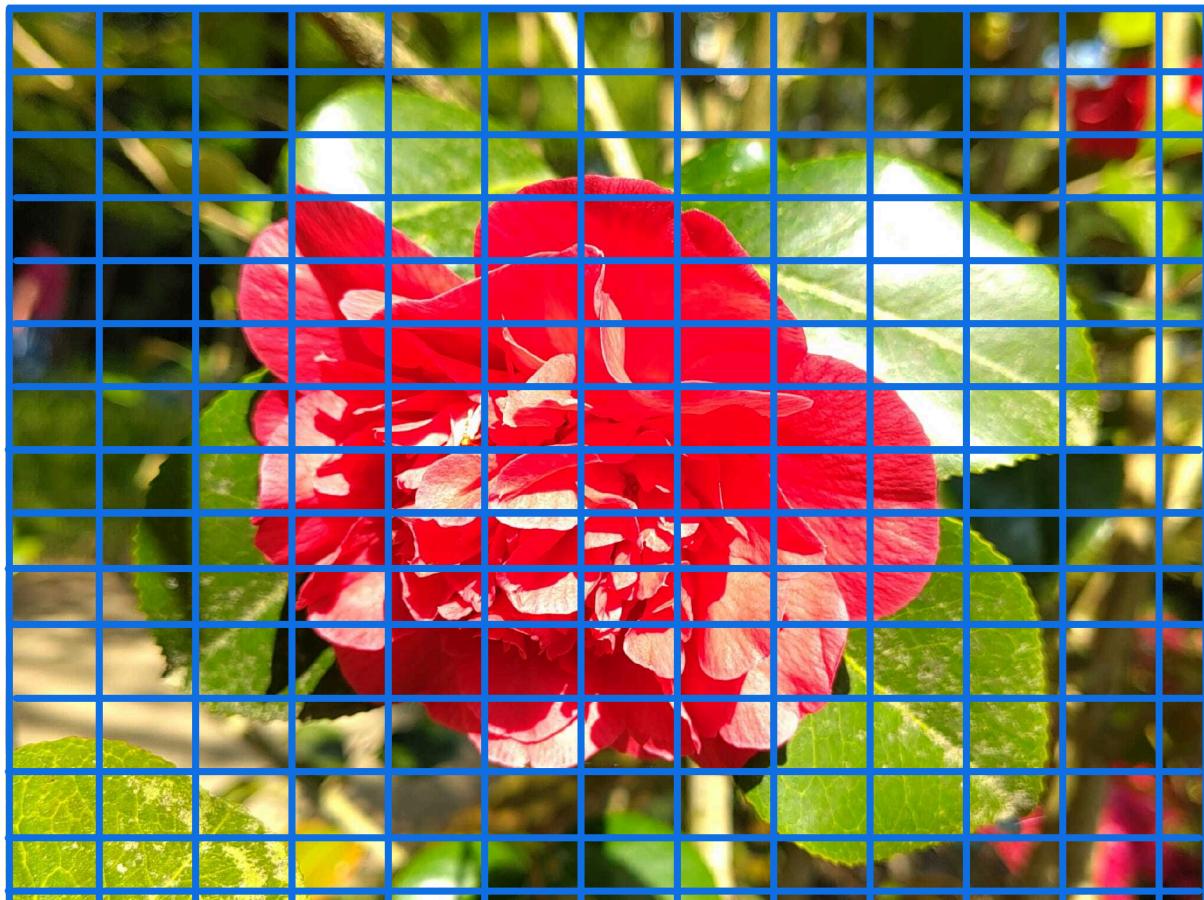
2. Détail des couches et concepts (Transformer Encoder)

Un ViT standard est essentiellement un empilement de **Transformer Encoder blocks**.

2.1 Patchify + Patch Embedding

a. Patchify

- On découpe l'image en patches $P \times P$.
- Exemple : 224×224 avec $P=16 \rightarrow 14 \times 14 = 196$ patches.

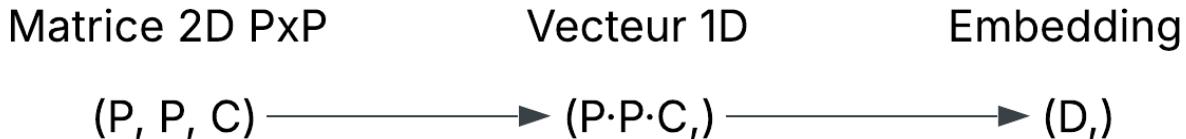


b. Projection linéaire (Patch Embedding)

Chaque patch aplati a une dimension $P \times P \times C$. On applique une couche linéaire :

$$E = X_{\text{patch}} \cdot W + b, \text{ avec } W \in \mathbb{R}^{(P \cdot P \cdot C) \times D}$$

Remarque d'implémentation : c'est souvent réalisé par une **convolution** `kernel=P, stride=P` (équivalent mathématique à une projection par patch, mais plus efficace).



2.2 Token [CLS]

Origine

Le token `[CLS]` (pour **Classification**) vient du modèle **BERT** (NLP, 2018). Dans BERT, on ajoutait un token spécial au début de chaque phrase pour obtenir une représentation globale de la séquence entière. ViT reprend exactement cette idée pour les images.

Fonctionnement

- C'est un **vecteur appris** de dimension D , initialisé aléatoirement et mis à jour pendant l'entraînement.
- Il est **concaténé** au début de la séquence de patch embeddings : `[CLS, patch_1, patch_2, ..., patch_N]`.
- Il **n'est associé à aucun patch** de l'image : c'est un token "virtuel".
- Grâce au mécanisme de **self-attention**, le token `[CLS]` interagit avec **tous les patches** à chaque couche de l'encodeur. Il agrège progressivement l'information de toute l'image.
- En sortie du dernier bloc Transformer, le vecteur `[CLS]` contient une **représentation globale** de l'image, utilisée par la tête de classification.

Pourquoi ne pas utiliser directement les patches ?

Le `[CLS]` joue le rôle de **résumé neutre** : il n'est biaisé vers aucune région spatiale de l'image. Si on utilisait un patch spécifique, la représentation serait biaisée vers cette zone.

Alternative : **global average pooling** sur tous les tokens de sortie (utilisé dans certaines variantes comme DeiT). Les deux approches donnent des performances comparables.

2.3 Encodage de position (Positional Embedding)

Le Transformer n'a pas d'ordre "naturel" : il faut injecter la position.

Dans ViT, on utilise souvent des **embeddings de position appris** :

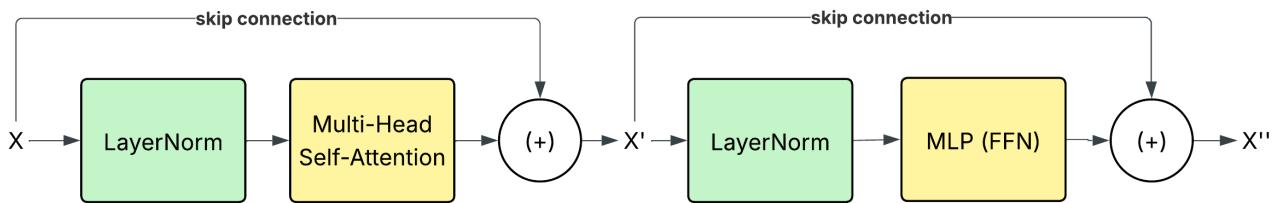
$$z_0 = [x_{cls}; x_1; \dots; x_N] + E_{pos}$$

où $E_{pos} \in \mathbb{R}^{(N+1) \times D}$.

Points importants :

- Si on change la résolution (donc N), on **interpole** souvent E_{pos} (ex. interpolation bicubique sur la grille 2D).
- Variantes : encodage sinusoïdal, encodages relatifs, biais d'attention relatifs (surtout dans Swin).

2.4 Bloc Transformer Encoder : structure générale



Un bloc (pré-norm, courant en vision) :

1. $X \leftarrow X + \text{MSA}(\text{LN}(X))$
2. $X'' \leftarrow X' + \text{MLP}(\text{LN}(X'))$

Composants clés :

- **LayerNorm (LN)**
- **Multi-Head Self-Attention (MSA)**
- **MLP / Feed-Forward Network (FFN)**
- **Connexions résiduelles** (skip connections)
- (Souvent) **Dropout / Stochastic Depth (DropPath)**

a. Connexions résiduelles

- Favorisent la propagation du gradient.
- Stabilisent l'entraînement de réseaux profonds.

b. Layer Normalization (LayerNorm)

- Normalise chaque token sur sa dimension D .
- Diffère de BatchNorm (qui dépend du batch) : LN est très utilisé en Transformers.

2.5 Self-Attention

Analogie : une salle de classe

Imaginez que chaque patch de l'image est un **élève** dans une salle de classe. Le mécanisme d'attention, c'est le moment où chaque élève **pose une question à tous les autres** pour décider à qui il doit prêter attention.

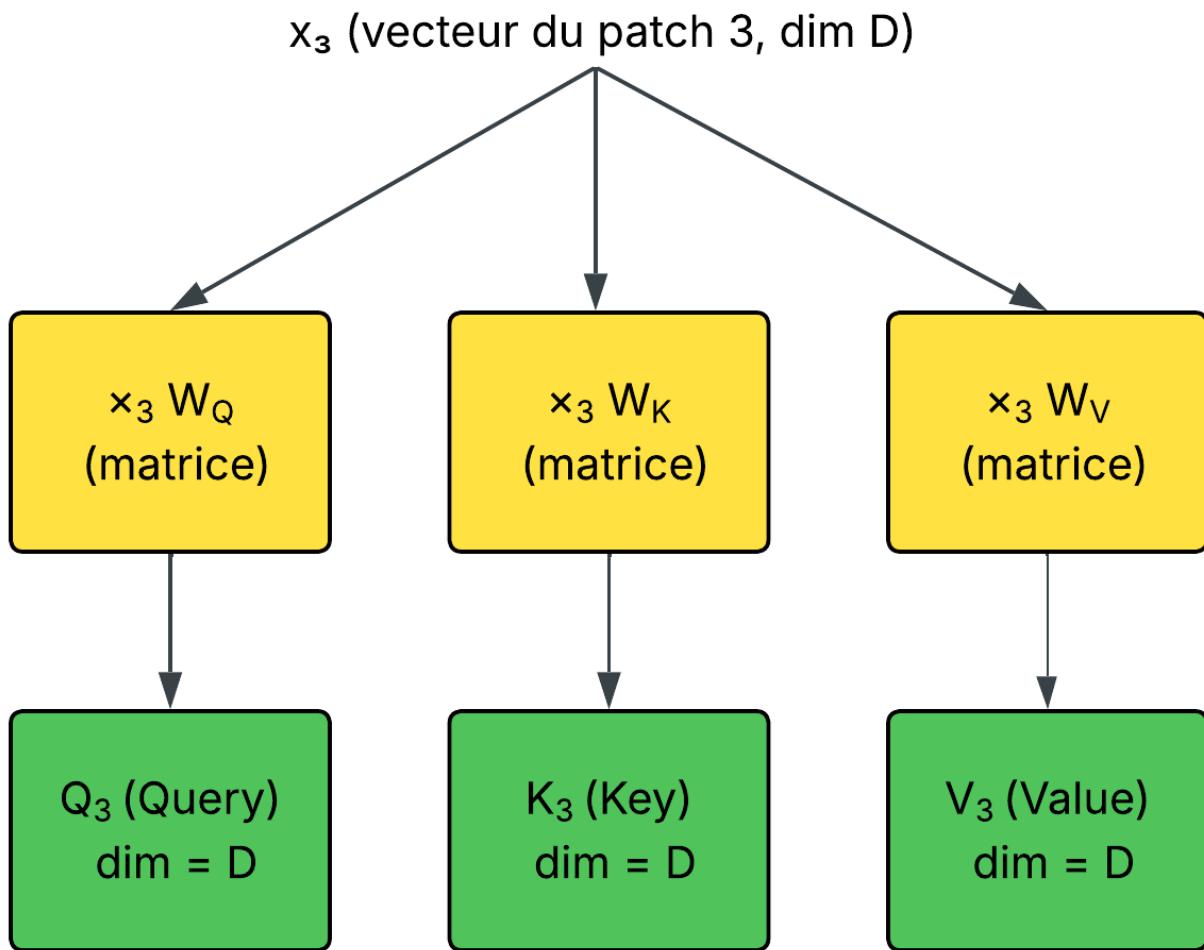
Pour une matrice d'entrées $X \in \mathbb{R}^{\{T \times D\}}$ (ici $T=N+1$ tokens) :

Étape 1 — Chaque token se prépare 3 "fiches" (Q, K, V) :

- $Q = X W_Q \rightarrow \text{Query (Question)}$: "Qu'est-ce que je cherche ?"
- $K = X W_K \rightarrow \text{Key (Étiquette)}$: "Voilà ce que je propose / ce que je suis"
- $V = X W_V \rightarrow \text{Value (Contenu)}$: "Voilà l'information que je porte"

$W_Q, W_K, W_V \in \mathbb{R}^{\{D \times d\}}$ sont des matrices de poids appris — elles transforment chaque token en ses 3 rôles.

chaque élève écrit une question (Q), une étiquette décrivant son expertise (K), et une prédiction de sa réponses (V).



Étape 2 — On calcule "qui est pertinent pour qui" :

```
A = softmax(Q K^T / sqrt(d))
```

Concrètement :

- $Q K^T$: on compare la question de chaque token avec l'étiquette de tous les autres → on obtient un **score de similarité** (une grande valeur = "ces deux tokens sont très liés").
- $/ \sqrt{d}$: on divise par \sqrt{d} pour **éviter que les scores soient trop grands** (sinon le softmax donnerait des poids extrêmes : tout ou rien).
- $\text{softmax}(\dots)$: on transforme les scores en **pourcentages** (entre 0 et 1, somme = 100%) → ce sont les **poids d'attention**.

chacun regarde les étiquettes de tous les autres et décide **à quel point** chacun est pertinent pour sa question (Q). L'élève qui a l'étiquette (K) la plus proche de ma question reçoit le plus d'attention.

Étape 3 — On récupère l'information utile :

$$Y = A \cdot V$$

On fait une **moyenne pondérée** des contenus (V) de tous les tokens, en utilisant les poids d'attention (A).

chaque élève lit les prédictions de tous les autres, mais accorde **plus d'importance** aux fiches des élèves qu'il juge pertinents. Le résultat est un **réponse pondérées** pour chaque élève.

En résumé : chaque patch "regarde" tous les autres patches, décide lesquels sont importants, et construit une nouvelle représentation en mélangeant leurs informations proportionnellement à leur pertinence.

Multi-Head Attention

Au lieu d'une seule attention, on utilise h têtes en parallèle :

- Chaque tête opère dans un sous-espace $d_{\text{head}} = D/h$.
- On concatène les sorties puis projection finale :

$$\text{MSA}(X) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W_0$$

imaginez que dans la salle de classe, au lieu d'un seul tour de questions, on organise **plusieurs groupes de discussion en parallèle**. Chaque groupe (= une tête) se concentre sur un **aspect différent** :

- Groupe 1 : "Qui sont mes **voisins proches** ?" (relations locales)
- Groupe 2 : "Qui a la même **texture/couleur** que moi ?" (apparence)
- Groupe 3 : "Qui fait partie du **même objet** que moi, même s'il est loin ?" (contexte global)
- Groupe h : d'autres relations encore...

À la fin, chaque élève **combine les résumés** de tous les groupes pour avoir une vision complète. C'est la concaténation + projection W_0 .

Coût

- Mémoire/temps dominés par QK^T : $O(T^2)$.
- En vision, T dépend du nombre de patches, donc de la résolution et de P .

si la classe a **T élèves**, chaque élève doit poser sa question à **tous les T autres**. Ça fait $T \times T$ interactions. Avec 196 patches, c'est gérable ($196^2 \approx 38\ 000$). Mais si on double la résolution de l'image, on a **4x plus de patches**, donc **16x plus d'interactions** — la classe devient ingérable !

2.6 MLP / Feed-Forward Network (FFN)

Après l'attention, un MLP par token (identique pour tous les tokens) :

$$\text{MLP}(x) = W2 \cdot \sigma(W1 \cdot x + b1) + b2$$

- $W1 : D \rightarrow D_{\text{hidden}}$ (souvent $D_{\text{hidden}} = 4D$)
- Activation σ : historiquement GELU .
- Regularisation : dropout.

après les groupes de discussion (attention), chaque élève retourne **seul à son bureau** pour **réfléchir et digérer** ce qu'il a appris. Le MLP, c'est ce moment de réflexion individuelle : chaque élève transforme et enrichit ses notes personnelles. Tous les élèves utilisent le **même processus de réflexion** (mêmes poids $W1, W2$), mais comme leurs notes sont différentes, le résultat est unique pour chacun.

Rôle :

- Transformation non-linéaire et enrichissement de la représentation par token.

3. Têtes (heads) selon la tâche

3.1 Classification

- On utilise le token [CLS] final : $y = \text{Head}(X_L[\text{CLS}])$.
- Head est souvent une couche linéaire (parfois précédée d'une LN).

3.2 Détection / segmentation

ViT "pur" n'est pas directement une architecture de détection/segmentation complète.

Approches courantes :

- Ajouter un **decoder** (type U-Net, ou Transformer decoder) sur les tokens.
- Utiliser des architectures spécialisées (ex. **DETR** pour détection, **SegFormer**, etc.).

3.3 OCR et texte (pont Vision \leftrightarrow NLP)

Pour l'OCR moderne, on couple souvent :

- Un **encodeur visuel** (CNN ou ViT/Swin)
- Un **décodeur texte** (Transformer autoregressif) pour générer la séquence de caractères/tokens

4. Entraînement : pré-entraînement et fine-tuning

4.1 Pourquoi le pré-entraînement est crucial

Biais inductif : ce sont les **hypothèses intégrées dans l'architecture** d'un modèle, avant même qu'il voie la moindre donnée. Ce sont des "raccourcis" que l'architecte impose pour guider l'apprentissage.

Imaginez deux élèves qui doivent apprendre à reconnaître des objets dans des photos :

- **Élève CNN** : on lui donne des **lunettes spéciales** qui ne lui permettent de regarder qu'une petite zone à la fois (localité), et qui voient la même chose peu importe où dans l'image (invariance par translation). Ces lunettes sont un **biais inductif fort** — elles l'aident à apprendre vite, même avec peu d'exemples.
- **Élève ViT** : on ne lui donne **aucune lunette**. Il peut regarder partout en même temps (attention globale), mais il doit **tout découvrir par lui-même** — que les voisins sont souvent liés, que les motifs se répètent, etc. Il a besoin de **beaucoup plus d'exemples** pour apprendre ce que l'élève CNN savait déjà grâce à ses lunettes.

ViT a moins de biais inductifs que les CNN :

- Un CNN impose **localité** (chaque filtre ne regarde qu'un voisinage) et **invariance par translation** (le même filtre est appliqué partout) via la convolution → biais inductif fort.
- ViT ne suppose rien de tout cela → il doit les apprendre à partir des données, d'où le besoin de **beaucoup plus de données**.

D'où l'intérêt :

- Pré-entraînement sur gros dataset (ImageNet-21k, JFT, LAION, etc.).
- Fine-tuning sur la tâche cible.

4.2 Stratégies de fine-tuning

- **Fine-tuning complet** : on met à jour tout le modèle.
- **Linear probing** : on gèle l'encodeur et on n'entraîne que la tête.
- **Partial fine-tuning** : geler les premiers blocs, entraîner les derniers.

4.3 Data augmentation et régularisation

Couramment utilisées pour ViT :

- **RandAugment / AutoAugment**
- **Mixup / CutMix**
- **Label smoothing**
- **Dropout et Stochastic Depth**

Objectif : améliorer la généralisation et compenser le faible biais inductif.

5. Comparaison CNN vs ViT (points d'analyse)

5.1 Biais inductifs

- **CNN :**
 - Localité (réceptif local)
 - Partage de poids
 - Équivariance translation (approximative)
- **ViT :**
 - Peu de biais inductifs “hardcodés”
 - Apprend des relations globales par attention

5.2 Données et performances

- À faible volume de données, un CNN peut être plus simple à entraîner.
- Avec un bon pré-entraînement, ViT devient très compétitif, souvent supérieur.

5.3 Coût

- CNN : coût souvent proportionnel à la taille de l'image et aux canaux.
 - ViT : coût dominé par l'attention $O(N^2)$; N augmente vite avec la résolution.
-

6. Variantes importantes

6.1 DeiT (Data-efficient Image Transformers)

Objectif : rendre ViT plus efficace en données sur ImageNet.

Idées clés :

- **Distillation token** : un token supplémentaire apprend via un teacher (souvent CNN).
- Recettes d'entraînement/augmentation adaptées.

6.2 Swin Transformer (Shifted Window)

Objectif : réduire le coût et introduire une hiérarchie (comme un CNN).

Idées :

a. Window Attention (attention par fenêtres)

Au lieu de calculer l'attention entre **tous les N patches** (coût $O(N^2)$), on découpe l'image en **fenêtres locales** de taille $M \times M$ (typiquement $M=7$) et on calcule l'attention **uniquement à l'intérieur de chaque fenêtre**.

- Chaque fenêtre contient $M^2 = 49$ tokens → attention $O(M^4)$ par fenêtre, **indépendant de la taille de l'image**.
- Coût total : $O(N \times M^2)$ au lieu de $O(N^2)$ → **quasi linéaire** en N.

Analogie : au lieu que tous les élèves de l'école discutent entre eux (ViT classique = chaos), on les répartit en **petits groupes de 49** dans des salles séparées. Chaque groupe discute efficacement, mais les groupes ne communiquent pas entre eux... d'où l'idée suivante.

b. Shifted Windows (fenêtres décalées)

Le problème de la window attention : les fenêtres sont **isolées**, aucune information ne circule entre elles.

Solution : à chaque couche paire, on utilise les fenêtres normales. À chaque couche impaire, on **décale les fenêtres de $M/2$ pixels** (horizontalement et verticalement). Ainsi, les tokens qui étaient aux **bords** de fenêtres différentes se retrouvent **dans la même fenêtre** à la couche suivante.

Exemple sur une grille 4×4 tokens avec des fenêtres 2×2 :

1 COUCHE L – Fenêtres normales (2×2) :

2

3 Les 4 fenêtres sont fixes :

5	W1	W2
6	a1 a2	b1 b2
7	a3 a4	b3 b4
<hr/>		
9	W3	W4
10	c1 c2	d1 d2
11	c3 c4	d3 d4

12

13

```

14   → a2 et b1 sont voisins mais dans des fenêtres DIFFÉRENTES (W1 vs W2)
15   → Ils ne peuvent PAS communiquer !
16
17
18 COUCHE L+1 – Fenêtres décalées de 1 (= M/2) :
19
20 On décale la grille d'un cran vers le bas et la droite :
21
22
23 | d4 | c3  c4 | d3 |
24 |-----|
25
26 | b2 | a1  a2 | b1 |
27 | b4 | a3  a4 | b3 |
28 |-----|
29 | d2 | c1  c2 | d1 |
30
31
32 → Maintenant a2 et b1 sont dans la MÊME fenêtre (celle du centre) !
33 → Ils peuvent communiquer via l'attention.

```

En alternant fenêtres normales et décalées à chaque couche, **tous les tokens finissent par pouvoir communiquer** avec tous les autres, indirectement.

Analogie : à chaque pause, on **réorganise les groupes** en décalant les places. Les élèves qui étaient au bord du groupe A se retrouvent avec ceux du groupe B. En alternant, l'information finit par circuler dans toute l'école.

Implémentation : Swin utilise un **masque d'attention** astucieux avec un décalage cyclique (`torch.roll`) pour éviter de créer des fenêtres de tailles différentes aux bords, ce qui permet de garder des opérations de taille fixe (efficace sur GPU).

c. Construction pyramidale (Patch Merging)

Comme un CNN qui réduit la résolution spatiale et augmente les canaux à chaque étage, Swin construit une **hiérarchie de résolutions** :

Stage	Résolution tokens	Dimension	Nb fenêtres
Stage 1	56×56	96	64 fenêtres 7×7
Stage 2	28×28 (÷2)	192 (×2)	16 fenêtres 7×7
Stage 3	14×14 (÷2)	384 (×2)	4 fenêtres 7×7
Stage 4	7×7 (÷2)	768 (×2)	1 fenêtre 7×7

Le passage d'un stage à l'autre se fait par **Patch Merging** : on regroupe 2×2 tokens voisins en un seul (concaténation + projection linéaire), divisant la résolution par 2 et doublant la dimension.

Analogie : c'est comme un **tournoi** — à chaque tour, on fusionne les groupes voisins en groupes plus grands mais moins nombreux. Au dernier tour, il ne reste qu'un seul grand groupe qui a une vision d'ensemble.

Cette structure pyramidale rend Swin compatible avec les architectures de détection/segmentation (FPN, U-Net) qui attendent des feature maps multi-échelles, contrairement au ViT classique qui garde la même résolution partout.

6.3 Hybrides CNN-Transformer

- CNN en “stem” (extraction locale) + Transformer ensuite.
- Combine localité (CNN) et globalité (attention).

6.4 TrOCR (pour OCR)

- Encodeur image (souvent type ViT)
- Décodeur texte (Transformer autoregressif)
- Entraîné en séquence-à-séquence : image → tokens texte

7. Lecture « couche par couche » (résumé opérationnel)

Pour un ViT de classification :

1. **Input** : image (H, W, C)
2. **Patchify** : N patches de taille $P \times P \rightarrow$ matrice $N \times (P \cdot P \cdot C)$
3. **Linear projection / Conv patch embedding** : $N \times D$
4. **Add [CLS]** : $(N+1) \times D$
5. **Add positional embeddings** : $(N+1) \times D$
6. Répéter L fois :
 - LN → MSA → Residual Add
 - LN → MLP(GELU) → Residual Add
7. **Head** : Linear (sur [CLS]) → logits/classes