

Vision Transformer

☑ Trắc nghiệm 7 ⬇ PDF bài

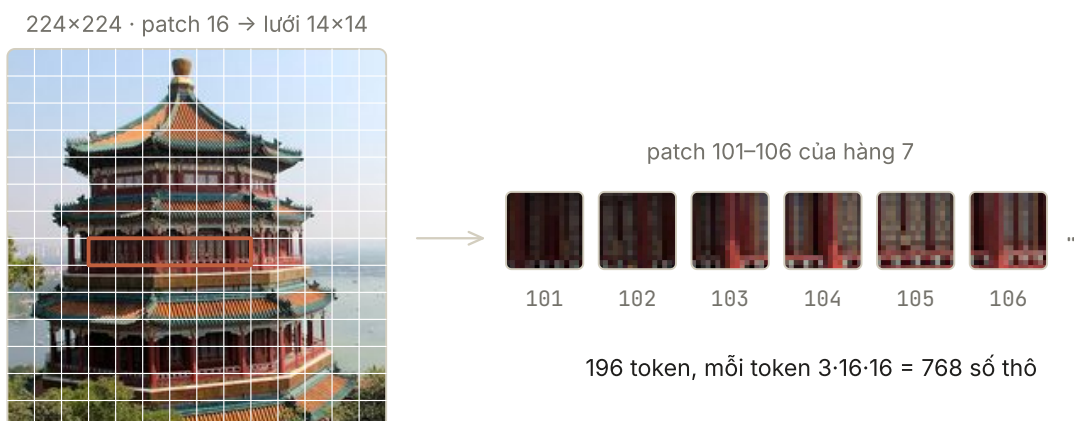
Vision Transformer (ViT) đưa Transformer từ văn bản sang ảnh: cắt ảnh thành các mảnh vuông nhỏ (patch), biến mỗi mảnh thành một vector, rồi coi các vector đó như các token trong câu. Nhờ attention, mỗi mảnh lấy được thông tin từ mọi mảnh khác ngay ở lớp đầu, trong khi CNN phải qua nhiều lớp mới nối được hai vùng xa nhau. Bài này đưa một ảnh nhỏ qua từng bước của ViT, và học cách mạng biết vị trí của mỗi mảnh.

MỤC TIÊU

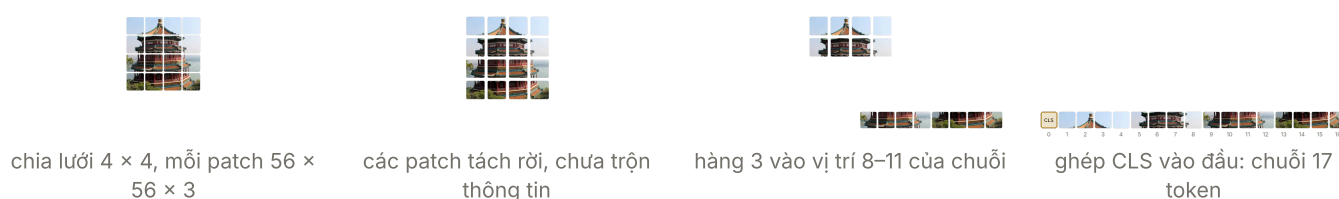
- Chia ảnh thành patch, đếm số token và chi phí attention.
- Viết patch embedding.
- Hiểu position embedding và class token.
- So sánh ViT với CNN.

Patch là một đơn vị đọc ảnh

Patch là khối điểm ảnh không chồng lấn. Với ảnh $H \times W$ có C kênh, patch $P \times P$, nếu hai chiều chia hết cho P thì số patch là $L = (H/P)(W/P)$, mỗi patch có CP^2 số. Ta trải mỗi patch thành vector rồi nhân cùng một ma trận E để có vector D chiều. D là chiều biểu diễn được chọn, không phải số pixel.



Ảnh 224×224 chia patch 16×16 thành 196 token.



Ảnh chia lưới 4×4 ; các hàng patch trải thành chuỗi token, rồi ghép CLS ở đầu.

Với ảnh RGB 224×224 , patch 16 cho $14 \cdot 14 = 196$ token ảnh, mỗi patch thô có $3 \cdot 16^2 = 768$ số. Bản ViT-Base chọn $D = 768$; việc trùng với 768 số thô chỉ là tình cờ của cấu hình, vì ViT-Large cùng patch 16 dùng $D = 1024$. Tên ViT-B/16 đọc là bản Base với patch 16×16 .

[Bài báo ViT](#) (Dosovitskiy và cộng sự, 2020) đưa chuỗi patch embedding vào một Transformer encoder gần như giữ nguyên từ NLP; tên bài báo nói đúng ý đó: một ảnh đáng giá 16×16 từ. Các mục sau dựng từng bước trên một ảnh 4×4 , đủ nhỏ để kiểm bằng tay.

HỎI NHANH

Ảnh RGB 32×32 được chia thành các patch 8×8 . Có bao nhiêu patch, và mỗi patch thô dài bao nhiêu số?

- A 64 patch, mỗi patch 192 số
- B 16 patch, mỗi patch 192 số
- C 16 patch, mỗi patch 64 số
- D 4 patch, mỗi patch 768 số

Chia ảnh thành patch bằng tensor

Ta đánh số ảnh xám từ 0 đến 15 theo hàng. Patch trên trái gồm 0, 1, 4, 5; trên phải gồm 2, 3, 6, 7. Nếu flatten toàn ảnh rồi cứ cắt mỗi bốn số, ta nhận 0, 1, 2, 3 là một patch — đó là một hàng, không phải ô vuông.

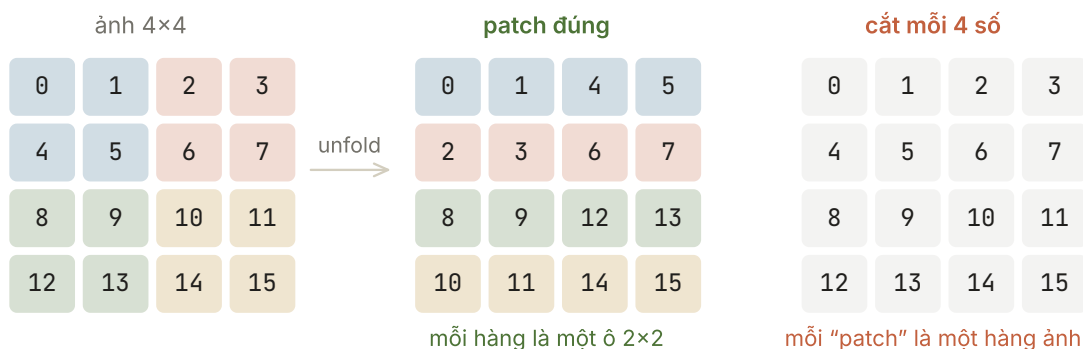
unfold tạo cửa sổ theo chiều; permute đặt thứ tự batch, hàng patch, cột patch, kênh, hàng trong patch, cột trong patch. reshape chỉ dùng sau khi thứ tự trục đã đúng.

Python

```
import torch
from torch import nn

torch.set_num_threads(1)
torch.manual_seed(54)
images = torch.arange(16, dtype=torch.float32).reshape(1, 1, 4, 4)
patch_size = 2
windows = images.unfold(2, patch_size, patch_size)
windows = windows.unfold(3, patch_size, patch_size)
patches = windows.permute(0, 2, 3, 1, 4, 5).reshape(1, 4, 4)
print(patches[0].to(torch.int64).numpy())
print(tuple(patches.shape))
```

Đọc dòng đầu để đối chiếu bốn ô tay. Shape $(1, 4, 4)$ có hai chiều cùng bằng 4 nhưng ý nghĩa khác: bốn patch và bốn số trong mỗi patch. Khi tăng số kênh, chiều cuối tăng, số patch giữ nguyên.



unfold rồi permute cho patch đúng; flatten rồi cắt chỉ lấy một hàng ảnh.

HỎI NHANH

Ảnh RGB 4×4 , patch cỡ 2, nên windows có shape $(1, 3, 2, 2, 2)$. Một bạn bỏ bước permute và gọi thẳng `windows.reshape(1, 4, 12)`. Hàng đầu của kết quả chứa gì?

- A Ba kênh của patch trên trái, nên permute là thừa
- B Ba kênh của bốn điểm ảnh ở hàng ảnh đầu tiên
- C Kênh R của ba patch đầu tiên
- D Không có gì, vì không reshape được tensor sáu chiều

Patch embedding là phép tuyến tính dùng chung

Với vector patch $\mathbf{p}_i$, ta tính $\mathbf{z}_i = \mathbf{p}_i E + \mathbf{b}$, trong đó E có shape (CP^2, D) . Mọi patch dùng chung E . Cách khác là conv có kernel P , stride P , không padding; mỗi kênh đầu ra là một chiều embedding. Hai cách tương đương nếu xếp trọng số cùng thứ tự.

Ta đặt trọng số cụ thể để kết quả dễ kiểm, rồi đối chiếu:

Python

```
projection = nn.Linear(4, 3)
with torch.no_grad():
    projection.weight.copy_(torch.tensor([[1., 0., 0., 0.],
                                           [0., 1., 0., 0.],
                                           [0.25, 0.25, 0.25, 0.25]]))

    projection.bias.zero_()
tokens = projection(patches)

patch_conv = nn.Conv2d(1, 3, kernel_size=2, stride=2)
with torch.no_grad():
    patch_conv.weight.copy_(projection.weight.reshape(3, 1, 2, 2))
    patch_conv.bias.copy_(projection.bias)
    conv_tokens = patch_conv(images).flatten(2).transpose(1, 2)
print(tokens[0].detach().tolist())
print(torch.allclose(tokens, conv_tokens))
```

Token đầu là (0, 1, 2.5): hai chiều đầu đọc hai ô đầu, chiều thứ ba lấy trung bình $(0 + 1 + 4 + 5)/4$. True xác nhận hai cách xếp ô nhất quán. Trọng số ở đây đặt tay để dễ đọc; trong ViT thật, E được học cùng phần còn lại của mạng.

HỎI NHANH

Tăng D từ 3 lên 12, giữ nguyên ảnh và patch size. Đại lượng nào không đổi?

- A Số patch
- B Số tham số của phép chiếu
- C Chiều của mỗi vector token
- D Shape trọng số của Conv2d tương đương

HỎI NHANH

Patch embedding của ViT-B/16 là `nn.Conv2d(3, 768, kernel_size=16, stride=16)`, chạy trên ảnh 224×224 . Lớp này có bao nhiêu tham số?

A 150 528

B 115 756 032

C 589 824

D 590 592

Attention nối các mảnh bằng nội dung

Trong attention, từ token ta tạo query Q , key K và value V . Điểm $QK^\top / \sqrt{d_k}$ đo mức phù hợp, softmax theo từng hàng tạo trọng số cộng thành 1, rồi lấy tổng có trọng số của V :

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^\top}{\sqrt{d_k}} \right) V.$$

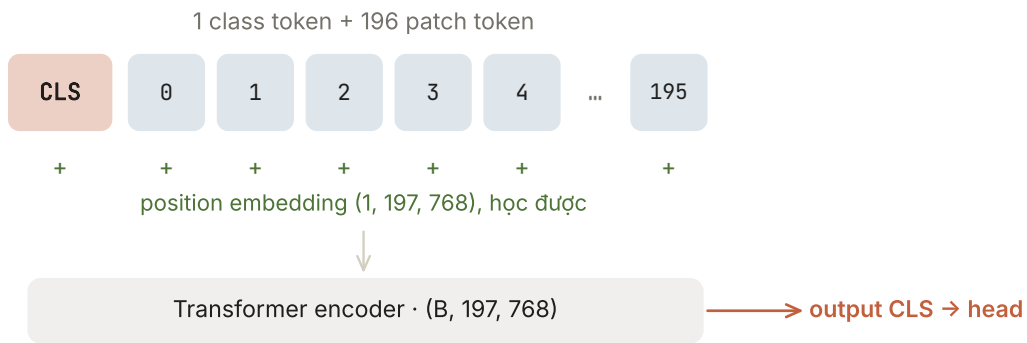
d_k là số chiều query/key mỗi head, không phải số patch. Ô (i, j) trong ma trận attention nói token i đọc token j bao nhiêu ở head này. Nó không tự là xác suất vật thể nằm ở patch j , cũng không chứng minh vùng ấy gây ra quyết định.

Điểm khác CNN nằm ở chỗ ngay từ lớp đầu, mọi patch đã có đường đọc trực tiếp mọi patch khác: quyền sách và cái đèn ở hai góc bàn trao đổi trong một bước. CNN phải xếp chồng nhiều lớp thì receptive field mới phủ được hai góc (bài 5.1).

Mỗi block của ViT ghép attention và MLP; trước mỗi phần có LayerNorm, sau mỗi phần có residual connection.

Class token (CLS) là một vector học được, ghép vào đầu chuỗi nên chuỗi dài $L + 1$. Nó không ứng với vùng ảnh nào, chỉ đọc các patch qua attention, và output của nó ở lớp

cuối đi vào head phân loại. Với ViT-B/16 trên ảnh 224, chuỗi vào encoder có shape $(B, 197, 768)$; mỗi lớp có 12 head, mỗi head một ma trận 197×197 .

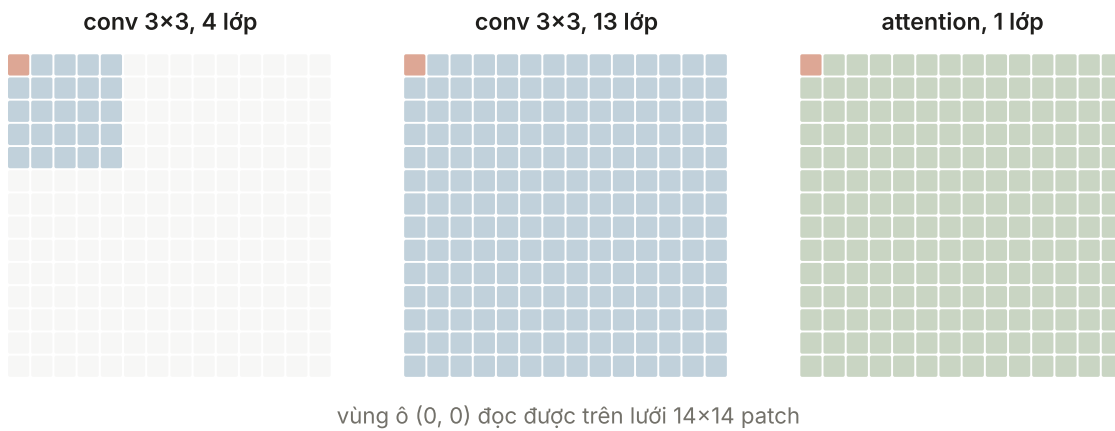


Chuỗi vào ViT: token CLS và 196 patch token, cộng mã vị trí, shape $(B, 197, 768)$.

HỎI NHANH

Lưới có 14×14 patch. Sau **một** lớp attention, token của patch $(0, 0)$ đã nhận được thông tin trực tiếp từ patch $(13, 13)$ chưa? Nếu thay bằng CNN chỉ gồm conv 3×3 stride 1, không pooling, thì cần ít nhất bao nhiêu lớp để receptive field của ô $(0, 0)$ chứa ô $(13, 13)$?

- A Rồi, và cần 7 lớp conv
- B Rồi, và cần 13 lớp conv
- C Chưa, cần nhiều lớp attention, và cần 7 lớp conv
- D Chưa, cần nhiều lớp attention, và cần 13 lớp conv



Conv 3×3 cần 13 lớp mới nối hai góc ảnh; một lớp attention đã nối mọi patch.

Vì sao cần thông tin vị trí?

Attention dựa vào nội dung không biết token này từ góc trái hay góc phải. Nếu đảo thứ tự token và đảo đầu ra tương ứng, self-attention thông thường không có position vẫn cho cùng quan hệ nội dung. Mean pooling trên đầu ra vì vậy không phân biệt cùng tập mảnh bị đổi chỗ.



Thiếu mã vị trí, hai ảnh cùng tập patch khác bố cục cho cùng kết quả.

Position embedding thêm thông tin vị trí vào token trước encoder. Nếu đổi ảnh nhưng giữ các mã tại vị trí của chúng, model có cơ sở phân biệt bố cục. Đổi cả token và mã đi cùng nó chỉ là đổi cách sắp danh sách, chưa thay quan hệ vị trí được biểu diễn.

ViT gốc dùng bảng vị trí 1D học được: patch đánh số theo hàng từ 0 đến $L - 1$, mỗi chỉ số và CLS có một vector riêng, cả bảng có shape $(1, L + 1, D)$ và được cộng vào token.

Bảng không ghi sẵn rằng trên lưới 14×14 , patch 0 và patch 14 kề nhau theo chiều dọc; model tự học điều đó từ dữ liệu. Bài báo cho thấy sau huấn luyện, vector vị trí của các patch cùng hàng hoặc cùng cột giống nhau hơn, và các mã vị trí 2D thiết kế sẵn không cho lợi ích đáng kể.

Ta kiểm tính đối xứng bằng một block nhỏ khởi tạo ngẫu nhiên, vì câu hỏi ở đây là về cấu trúc chứ chưa phải về điều model học được. Dropout tắt để khác biệt không do ngẫu nhiên.

Python

```
encoder = nn.TransformerEncoderLayer(
    d_model=3, nhead=1, dim_feedforward=6,
    dropout=0.0, batch_first=True,
)
encoder.eval()
permutation = torch.tensor([2, 0, 3, 1])
positions = torch.tensor([[[0.4, 0.0, 0.0],
                           [0.0, 0.4, 0.0],
                           [0.0, 0.0, 0.4],
                           [0.2, -0.3, 0.1]]])

with torch.no_grad():
    pooled = encoder(tokens).mean(dim=1)
    shuffled = encoder(tokens[:, permutation]).mean(dim=1)
    with_position = encoder(tokens + positions).mean(dim=1)
    shuffled_content = encoder(tokens[:, permutation] + positions).mean(dim=1)
print("without position:", torch.allclose(pooled, shuffled, atol=1e-5))
print("with fixed positions:", torch.allclose(with_position, shuffled_content, atol=1e-5))
```

Dòng True xác nhận đối xứng khi chưa có vị trí: đảo các mảnh thì kết quả sau pooling không đổi, nên sách ở góc trái hay góc phải là như nhau với block này. Dòng False cho thấy cộng mã vị trí cố định đã phá đối xứng đó: cùng tập mảnh ở bố cục khác cho output khác. Dùng thông tin vị trí ấy thế nào là phần model tự học từ dữ liệu.

HỎI NHANH

Ta đảo thứ tự các token ảnh, và position embedding cũng đi theo từng token. Sau đó lấy mean pooling. Vì sao kết quả có thể giữ nguyên?

- A Vì position embedding chưa được train nên chưa có tác dụng
- B Vì mean pooling xóa hết thông tin vị trí trong token
- C Vì chỉ thứ tự danh sách thay đổi, mỗi patch vẫn giữ mã vị trí cũ
- D Vì encoder ở chế độ eval() nên bỏ qua position embedding

CNN và ViT: so giả định trước khi so điểm

CNN có thiên hướng cục bộ và chia sẻ trọng số theo vị trí. ViT với attention toàn cục có đường trao đổi trực tiếp giữa patch xa, nhưng mất thông tin bên trong patch nếu projection nén quá mạnh, và phải học quan hệ không gian qua biểu diễn vị trí. “Toàn cục” không đồng nghĩa tự hiểu mọi quan hệ.

Ít giả định có sẵn thì phải học nhiều hơn từ dữ liệu. Bài báo ViT báo cáo rằng khi chỉ huấn luyện trên ImageNet (khoảng 1.3 triệu ảnh) mà không dùng regularization mạnh, ViT kém ResNet cùng cỡ vài điểm phần trăm; khi pretrain trên JFT-300M (303 triệu ảnh), ViT vượt ResNet và tốn ít tài nguyên pretrain hơn, rồi fine-tune tốt sang các tập nhỏ.

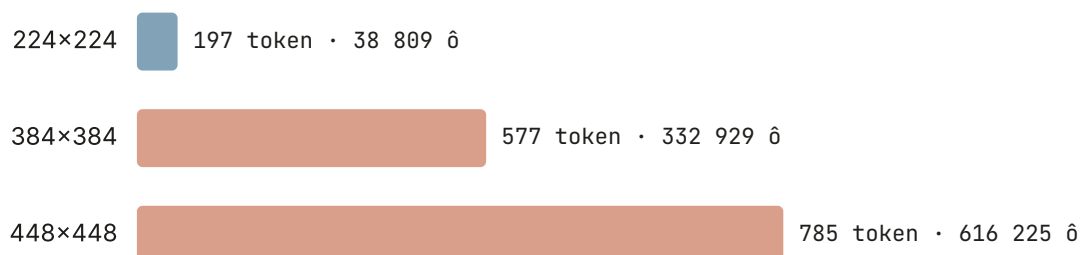
HỎI NHANH

Train từ đầu trên 3000 ảnh. Một CNN nhỏ đạt 85% trên validation. Một ViT nhỏ có số tham số tương đương đạt 100% trên train nhưng chỉ 70% trên validation. Lý do khả dĩ nhất là gì?

- A ViT thiếu giả định cục bộ nên overfit khi có ít ảnh
- B ViT underfit, cần thêm lớp để tăng sức chứa
- C Code ViT có lỗi, vì attention toàn cục phải tốt hơn
- D Validation của ViT bị rò dữ liệu từ tập train

Với attention đầy đủ, ma trận điểm có L^2 ô cho mỗi head, hoặc $(L + 1)^2$ khi có CLS. Giữ patch size, tăng gấp đôi cả hai chiều ảnh làm số patch gấp 4 và số ô attention gấp 16. Convolution có kernel cố định thường tăng công việc theo diện tích, khoảng gấp 4.

Phần MLP và các phép chiếu của ViT chỉ tăng theo L , tức gấp 4, nên tổng thời gian tăng ít hơn 16 lần. Thời gian thật còn phụ thuộc cách cài và bộ nhớ activation, vì vậy ước lượng trước rồi đo.



ô attention mỗi head, mỗi lớp = $(L + 1)^2$; cạnh ảnh gấp đôi thì gấp khoảng 16

Gấp đôi cạnh ảnh thì ma trận attention gấp khoảng 16 lần.

ViT pretrained có thể là baseline mạnh khi có checkpoint và preprocessing phù hợp. Train ViT lớn từ đầu trên 60 ảnh bàn học khó kiểm soát hơn một baseline nhỏ. Patch quá

lớn có thể làm chữ rất nhỏ bị gộp; patch quá nhỏ tăng chi phí. Chọn theo đối tượng, dữ liệu và ngân sách, rồi so trên cùng split.

Đổi độ phân giải khi fine-tune

Bài báo ViT fine-tune ở độ phân giải cao hơn lúc pretrain, giữ nguyên patch size, và nội suy 2D bằng vị trí theo vị trí của patch trong ảnh gốc. Các bước:

1. Tách vector CLS.
2. Đưa L vector còn lại về lưới $h \times w \times D$.
3. Nội suy lưới sang kích thước mới.
4. Trải lại rồi ghép CLS vào đầu.

Để thấy hình học được giữ, ta dùng một bảng vị trí tự đặt, trong đó vector của mỗi patch chính là (hàng, cột) của nó, còn CLS là $(-1, -1)$:

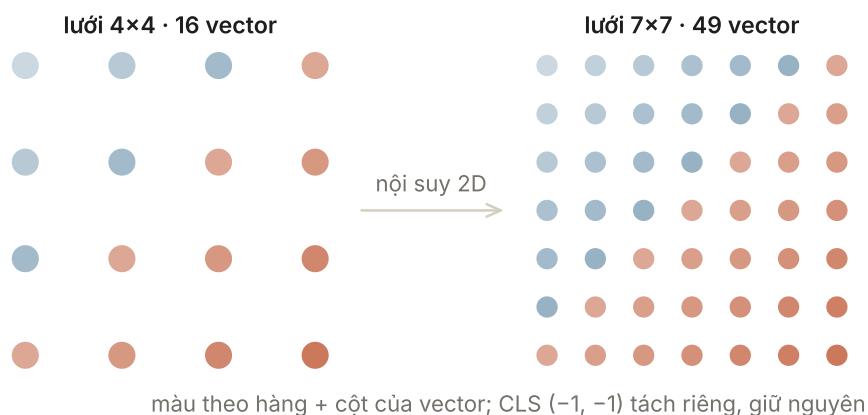
Python

```
import torch.nn.functional as F

def resize_position_table(table, old_grid, new_grid):
    cls_part, grid_part = table[:, :1], table[:, 1:]
    dim = table.shape[-1]
    grid = grid_part.reshape(1, old_grid, old_grid, dim).permute(0, 3, 1, 2)
    grid = F.interpolate(grid, size=(new_grid, new_grid),
                        mode="bilinear", align_corners=True)
    grid_part = grid.permute(0, 2, 3, 1).reshape(1, new_grid * new_grid, dim)
    return torch.cat([cls_part, grid_part], dim=1)

rows, cols = torch.meshgrid(torch.arange(4.0), torch.arange(4.0), indexing="ij")
coords = torch.stack([rows, cols], dim=-1).reshape(1, 16, 2)
table = torch.cat([torch.full((1, 1, 2), -1.0), coords], dim=1)
resized = resize_position_table(table, old_grid=4, new_grid=7)
print(tuple(table.shape), "->", tuple(resized.shape))
print("CLS:", resized[0, 0].tolist())
print("row of first column:", resized[0, 1:, 0].reshape(7, 7)[: , 0].tolist())
```

Bảng đi từ 17 lên 50 vector, tức CLS cộng lưới 7×7 . CLS giữ nguyên $(-1, -1)$, còn tọa độ hàng ở cột đầu chạy đều từ 0 đến 3: mỗi vị trí mới nhận vector của đúng chỗ tương ứng trên lưới cũ. Ví dụ dùng bilinear để số dễ đọc; thư viện thường dùng bicubic. Sau khi nội suy vẫn cần fine-tune ở độ phân giải mới, vì bảng chỉ là điểm xuất phát hợp lý.



Nội suy bảng vị trí từ lưới 4×4 sang 7×7 ; CLS tách ra rồi ghép lại.

HỎI NHANH

Checkpoint ViT-B/16 train ở 224×224 có bảng vị trí shape $(1, 197, 768)$. Bây giờ fine-tune ở 384×384 , vẫn giữ patch 16. Bảng mới cần bao nhiêu vector?

A 197

B 337

C 576

D 577

Khi đi thi

- Đếm token $L = (H/P)(W/P)$ trước khi tăng độ phân giải: attention tốn L^2 ô mỗi head, mỗi lớp.
- Đọc cấu hình checkpoint: patch size, độ phân giải lúc train, mean/std của processor.

- Ít dữ liệu và không có pretrained thì thử CNN nhỏ trước ViT.

Tóm tắt

- ViT cắt ảnh thành patch, biến mỗi patch thành token bằng một phép chiếu dùng chung, thêm vị trí rồi đưa qua Transformer.
- Thiếu thông tin vị trí, model không phân biệt được các bố cục của cùng một tập patch; đổi độ phân giải thì bảng vị trí phải nội suy 2D.
- Class token là một cách tổng hợp cả ảnh; mean pooling là cách khác.
- Attention nối mọi patch ngay từ lớp đầu, nhưng chi phí tăng bậc hai theo số token, và ViT cần nhiều dữ liệu hơn CNN.

Kiểm tra cuối bài

CÂU 1

Một đề vật lý cho mỗi mẫu một tensor shape $(64, 64, 2)$, là hai thành phần của một trường vector đo trên lưới 64×64 . Một đội dùng ViT patch 8×8 có class token. Chuỗi vào encoder dài bao nhiêu, và mỗi patch thô dài bao nhiêu số?

- A 64 token, mỗi patch 128 số
- B 65 token, mỗi patch 128 số
- C 64 token, mỗi patch 192 số
- D 65 token, mỗi patch 192 số

PHÒNG THEO [USA-NA-AIO 2026 · Vòng 2 · Ngày 1 · Problem 2](#)

CÂU 2

ViT đọc ảnh như một chuỗi patch. Một đội muốn giữ mọi chi tiết, nên cho mỗi pixel của ảnh 224×224 thành một token. So với patch 16×16 , tức 196 token, ma trận attention của mỗi head ở mỗi lớp có số ô tăng bao nhiêu lần?

A 16

B Không đổi, vì ma trận attention có kích thước $D \times D$

C 256

D 65 536

PHÒNG THEO NOAI 2026 Singapore · Vòng sơ loại · Câu 260

CÂU 3

Ảnh 230×230 được đưa thẳng vào `nn.Conv2d(3, 768, kernel_size=16, stride=16)`, không padding. Kết quả ra sao?

A Báo lỗi, vì 230 không chia hết cho 16

B Tự đệm thêm để ra lưới 15×15 , tức 225 token

C Ra lưới 14×14 , còn 6 hàng và 6 cột pixel cuối không được đọc

D Ra lưới 14×14 , phần dư được gộp vào hàng và cột patch cuối

PHÒNG THEO CMU 10-601 · Exam 3 Practice · Q1.2(b)

CÂU 4

Encoder ViT-B/16 chạy trên ảnh 224 trả về out shape (1, 197, 768), trong đó token 0 là CLS. Head segmentation cần feature map shape (1, 768, 14, 14). Dòng nào đúng?

- A `out.reshape(1, 768, 14, 14)`
- B `out[:, 1:].reshape(1, 768, 14, 14)`
- C `out[:, 1:].transpose(1, 2).reshape(1, 768, 14, 14)`
- D `out[:, :196].transpose(1, 2).reshape(1, 768, 14, 14)`

CÂU 5

Một ViT patch 16 trên ảnh 224×224 bị quên cộng position embedding. Head đọc output của CLS, model chạy ở `eval()`. Ta thử hai phép tráo. Phép (i) cắt tại pixel 112 theo cả hai chiều, rồi đổi chỗ khối trên trái với khối dưới phải. Phép (ii) đổi chỗ dải dọc cột 0–99 với dải cột 100–199. Logits sau mỗi phép so với ảnh gốc thế nào?

- A (i) giống ảnh gốc, (ii) có thể khác
- B (i) có thể khác, (ii) giống ảnh gốc
- C Cả hai đều giống ảnh gốc
- D Cả hai đều có thể khác

CÂU 6

Checkpoint có bảng vị trí (1, 197, 768), gồm CLS rồi lưới 14×14 đánh số theo hàng. Để chạy ảnh 384, một bạn nội suy tuyến tính 1D cả chuỗi 197 vector thành 577 vector. Cách này sai ở đâu?

- A Không sai, vì nội suy 1D vẫn giữ thứ tự các token
- B CLS bị trộn vào lưới, và cuối hàng trộn với đầu hàng sau
- C Không sai nếu fine-tune, vì model sẽ học lại mọi vị trí
- D Chỉ sai ở viền ảnh, phần giữa giống nội suy 2D

CÂU 7

Ảnh RGB 224×224 . Để giữ nét nhỏ, ta đổi patch từ 16 xuống 8 và giữ nguyên mọi thứ khác. Điều nào đúng?

- A 392 token, ô attention gấp 4, patch thô 384 số
- B 784 token, ô attention gấp 16, patch thô 192 số
- C 784 token, ô attention gấp 4, patch thô 192 số
- D 392 token, ô attention gấp 16, patch thô 384 số

PHÒNG THEO CMU 10-601 · Exam 3 Practice · Q1.1(b)

CÂU 8

Ảnh gốc 2048×2048 có vết nứt rộng khoảng 3 pixel. Pipeline resize ảnh về 224×224 rồi đưa vào ViT-B/16 pretrained, và recall với vết nứt thấp. Thay đổi nào nhằm đúng nguyên nhân?

- A Tăng D để mỗi token chứa nhiều thông tin hơn
- B Đổi sang patch 32 để mỗi token nhìn vùng rộng hơn
- C Thêm head attention để model chú ý vết nứt hơn
- D Cắt ảnh gốc thành ô nhỏ, đưa từng ô vào model

CÂU 9

Trên cùng validation, CNN pretrained đạt macro-F1 0.91 và chạy 40 ms mỗi ảnh. ViT pretrained đạt 0.92 và chạy 160 ms mỗi ảnh, cùng CPU và đã chọn batch tốt nhất. Tập test có 20 000 ảnh, giới hạn chạy dự đoán là 30 phút, quá giờ thì 0 điểm. Nên chọn model nào?

- A CNN, vì ViT cần khoảng 53 phút cho tập test
- B ViT, vì F1 trên cùng validation cao hơn
- C Ensemble cả hai, để F1 cao hơn từng model
- D ViT, chạy một nửa test rồi nộp phần đã xong

CÂU 10

Một đội giữ tỉ lệ khung hình của ảnh. Ảnh A 224×224 cho 196 patch, ảnh B 224×320 cho 280 patch. Để chạy chung một batch qua encoder, đội nối thêm 84 vector 0 vào cuối chuỗi token của A. Còn thiếu gì?

- A Không thiếu gì, vì vector 0 cho điểm attention bằng 0
- B Cắt chuỗi của B về 196 token để hai chuỗi bằng nhau
- C Key padding mask, để token thật không đọc token đệm
- D Đặt các token đệm lên đầu chuỗi, ngay trước CLS

PHÒNG THEO CMU 10-601 · Exam 3 Practice · Q2.2

Đọc thêm

- Dosovitskiy và cộng sự — [An Image is Worth 16×16 Words](#): xem kiến trúc patch, class token và điều kiện pretraining của nghiên cứu.
- [PyTorch — TransformerEncoderLayer](#): đối chiếu batch_first, nhead và dropout.
- [Dive into Deep Learning — Transformers for Vision](#): đọc đường từ ảnh đến encoder sau khi đã kiểm patch bằng tay.