

Augmentation và transfer learning

☑ Trắc nghiệm 29 📧 Thực hành 5 🔗 Chạy trên Kaggle ⬇ PDF bài

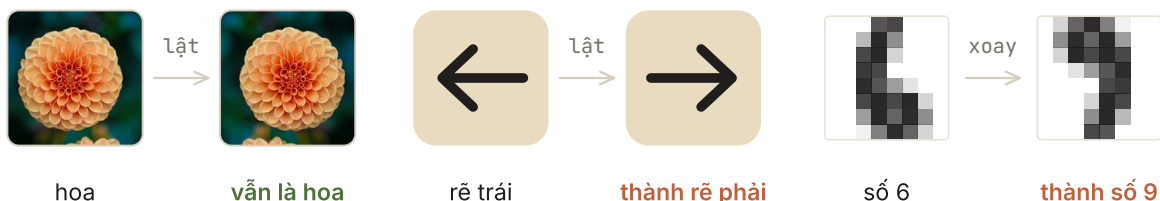
Khi chỉ có ít ảnh, ví dụ 60 ảnh chai nước, huấn luyện CNN từ đầu sẽ học thuộc. Có hai cách bù: augmentation (tạo thêm ảnh bằng cách lật, cắt, đổi độ sáng ảnh có sẵn) và transfer learning (dùng lại model đã học trên dữ liệu lớn). Cả hai đều có điều kiện: phép biến đổi không được làm đổi nhãn, ví dụ xoay ảnh số 6 thành số 9; và model cũ chỉ giúp được khi dữ liệu cũ đủ giống dữ liệu mới.

MỤC TIÊU

- Chọn phép augmentation không làm đổi nhãn.
- Biến đổi ảnh cùng với box và mask.
- Đo tác dụng của augmentation trên validation.
- Chọn giữa feature extraction và fine-tuning.

Dữ liệu nhiều hơn theo nghĩa nào?

Augmentation tạo một phiên bản biến đổi mỗi khi đọc mẫu: lật, crop, đổi sáng hoặc thêm nhiễu hợp lý. Nó không mang thêm thông tin độc lập như một buổi chụp mới; các phiên bản vẫn chung một nguồn. Vì vậy không được chia các bản biến đổi của cùng một ảnh sang cả train lẫn validation rồi coi chúng là các mẫu độc lập.



nhân giữ hay đổi tùy điều mà nhãn mô tả

Lật hoa vẫn là hoa; lật biển rẽ trái thành rẽ phải; xoay số 6 thành số 9.

Gọi T là phép biến đổi ảnh. Với classification, ta thường muốn $y(T(x)) = y(x)$: nhãn không đổi. Với detection hay segmentation, đầu ra có tọa độ nên cần $y(T(x)) = T(y(x))$: nhãn **cũng phải biến đổi**. Hai quan hệ này khác nhau. Lật ảnh chó vẫn là chó; lật biển chỉ hướng có thể đổi trái thành phải; xoay chữ số 6 có thể thành 9.

Crop còn tinh vi hơn. Nếu crop bỏ vật thể mà vẫn giữ nhãn “có chai”, ta tạo nhãn sai. Với nhãn cả ảnh như “phong cảnh”, một crop nhỏ có thể mất phần quyết định. Vì vậy mỗi augmentation là một giả định về điều kiện ứng dụng: phải xem ảnh sau biến đổi và hỏi đáp án còn đúng không.

HỎI NHANH

Bài toán là phân biệt chữ “b” với chữ “d”. Lật ngang ảnh mà vẫn giữ nhãn cũ thì có ổn không?

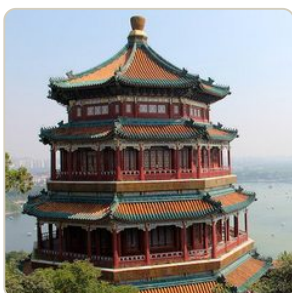
- A Ổn, vì lật ngang an toàn với mọi bài phân loại ảnh
- B Ổn, nếu mỗi epoch chỉ lật một nửa số ảnh
- C Không ổn, vì chữ “b” lật ngang trông như chữ “d”
- D Không ổn, vì phép lật chỉ dùng được cho ảnh màu

Các phép thường dùng chia thành ba nhóm:

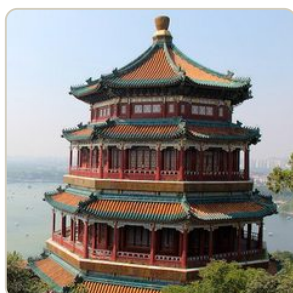
- **Biến đổi hình học** (lật, crop, xoay, co giãn, dời) đổi vị trí điểm ảnh, nên với box, mask hay điểm mốc phải áp cùng phép lên nhãn. Lật ngang ảnh có điểm mốc thì ngoài đổi tọa độ còn phải đổi tên: điểm “tay trái” thành “tay phải”.
- **Biến đổi cường độ** (độ sáng, tương phản, màu, làm mờ, nhiễu) giữ nguyên vị trí, nhưng có thể đụng vào chính tín hiệu của nhãn, như màu của quả chín hay độ sáng của vùng tổn thương trên phim X-quang. Thêm nhiễu Gaussian nhỏ dạy model bền với nhiễu cảm biến.
- **Che và trộn mẫu**: Cutout (random erasing) xóa một ô chữ nhật và giữ nhãn, buộc model dùng nhiều phần của vật thay vì bám vào một chi tiết. Mixup lấy $\tilde{x} = \lambda x_i + (1 - \lambda)x_j$ và trộn hai nhãn one-hot thành $\tilde{y} = \lambda y_i + (1 - \lambda)y_j$, với $\lambda \in (0, 1)$ rút ngẫu

nhiên. CutMix dán một mảnh của ảnh này lên ảnh kia và trộn nhẵn theo tỉ lệ diện tích.

Với nhãn mềm $\tilde{y}$, cross-entropy vẫn là $-\sum_k \tilde{y}_k \log p_k$. Ví dụ dán vùng 16×16 của ảnh chó vào ảnh mèo 32×32 : mảnh chiếm $1/4$ diện tích nên nhãn là 0.75 mèo, 0.25 chó.



gốc



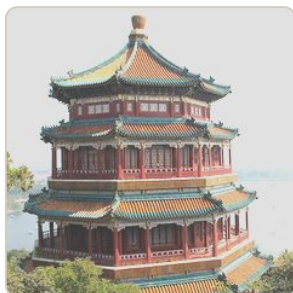
lật ngang



crop rồi phóng



xoay 15°



sáng hơn, nhạt hơn



cutout

Năm phép augmentation trên một ảnh tháp; cả năm đều giữ nhãn.



ảnh gốc



epoch 2: crop 48% diện tích,
xoay $+15^\circ$, lật ngang, sáng
 $\times 1.05$



epoch 5: crop 60% diện tích,
xoay -4° , lật ngang, sáng $\times 1.24$



epoch 8: crop 46% diện tích,
xoay $+3^\circ$, sáng $\times 1.01$



Mỗi epoch, cùng một ảnh hoa được crop, xoay, lật và đổi sáng ngẫu nhiên; nhãn vẫn là hoa.

HỎI NHANH

Model dự đoán tọa độ mắt trái và mắt phải trên ảnh chân dung rộng W . Khi lật ngang ảnh, ta đã đổi mọi tọa độ x thành $W - x$. Nhấn còn thiếu bước nào?

- A Đổi chỗ hai điểm, mắt trái thành mắt phải
- B Không thiếu gì, vì tọa độ đã biến đổi theo ảnh
- C Đổi thêm y thành $H - y$ cho đối xứng
- D Chia tọa độ cho W để đưa về khoảng $[0, 1]$

Lật ảnh thì lật cả box và mask

Ta dùng tọa độ biên liên tục $xyxy$: $[x_1, y_1, x_2, y_2]$, trong ảnh rộng W . Sau lật ngang, khoảng $[x_1, x_2]$ thành $[W - x_2, W - x_1]$. Với box $[1, 2, 4, 6]$ và $W = 10$, box mới là $[6, 2, 9, 6]$. Độ rộng vẫn là 3. Quy ước điểm ảnh đóng ở hai đầu sẽ có công thức khác; phải xác nhận format trước khi dùng.

Ta kiểm với box và mask nhỏ. NumPy lật chiều cột; `.copy()` tạo mảng có bộ nhớ thông thường để chuyển sang tensor khi cần.

Python

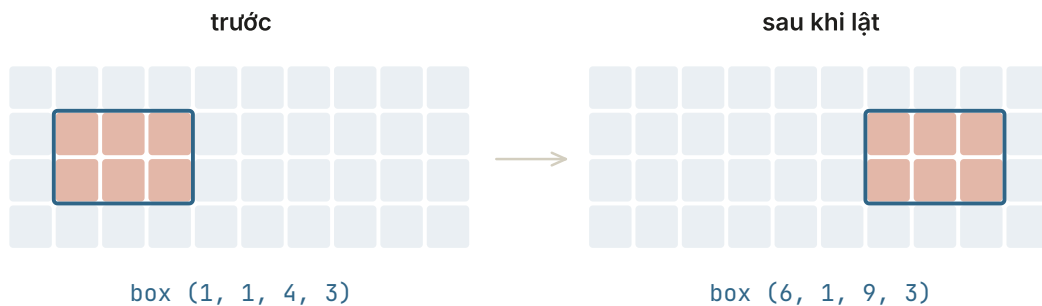
```
import numpy as np

def flip_box_xyxy(box, width):
    x1, y1, x2, y2 = box
    return np.array([width - x2, y1, width - x1, y2])

mask = np.zeros((4, 10), dtype=np.uint8)
mask[1:3, 1:4] = 1
box = np.array([1, 1, 4, 3])
flipped_mask = np.flip(mask, axis=1).copy()
flipped_box = flip_box_xyxy(box, mask.shape[1])
print(flipped_box.tolist())
```

```
print(np.flatnonzero(flipped_mask[1]).tolist())
print(int(mask.sum()), int(flipped_mask.sum()))
```

Mask mới sáng ở cột 6, 7, 8 và box chứa khoảng biên [6, 9]. Số ô foreground không đổi. Nếu chỉ lật ảnh mà không lật mask, model sẽ bị yêu cầu tô vị trí không còn chứa vật thể.



Lật ảnh thì phải lật cả mask và box theo.

Resize ảnh thường dùng bilinear; resize mask chứa ID lớp dùng nearest neighbor để không sinh ID phân số. Crop và xoay phải dùng cùng tham số hình học cho ảnh và mask.

[TorchVision transforms v2](#) nhận ảnh, box và mask trong cùng một lời gọi và áp cùng tham số ngẫu nhiên cho tất cả; phép nào hợp lệ về nghĩa thì người dùng vẫn phải tự quyết. Với tensor (C, H, W), lật ngang là `img.flip(-1)`.

Xoay 90° đổi chỗ hai cạnh: ảnh rộng 640, cao 480 thành rộng 480, cao 640. Khi crop cắt gần hết một vật, như box chỉ còn 10% diện tích trong vùng cắt, thường bỏ hẳn box ấy thay vì giữ một box gần như không còn vật.

HỎI NHANH

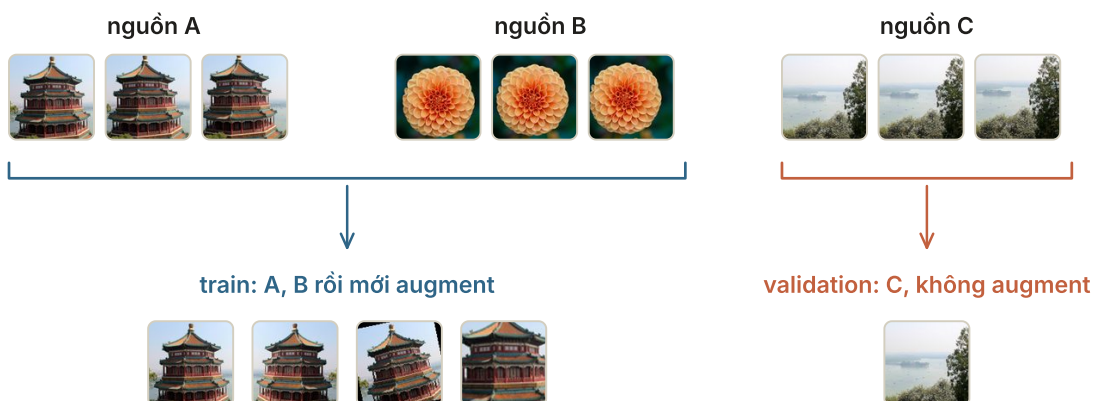
Mask có ID lớp 0, 1 và 2. Resize bằng bilinear tạo ra giá trị 1.4 ở biên giữa vùng lớp 0 và vùng lớp 2. Như vậy có sao không?

- A Không sao, ép về long thì 1.4 thành lớp 1 hợp lệ
- B Không sao, biên đã có thêm một lớp trung gian thật
- C Có, phải chuẩn hóa mask về $[0, 1]$ trước khi resize
- D Có, ID lớp không nội suy được, nên phải dùng nearest

Validation là ảnh ta thực sự sẽ gặp

Train dùng biến đổi ngẫu nhiên để model học bền hơn. Validation dùng preprocessing xác định để các thí nghiệm được so trong cùng điều kiện.

Test-time augmentation (TTA) chạy model trên vài phiên bản của cùng một ảnh test, ví dụ ảnh gốc và ảnh lật, rồi lấy trung bình xác suất. TTA chỉ hợp lệ khi mọi phiên bản giữ nhãn; nó phải được thử trên validation, và thời gian suy luận nhân lên theo số phiên bản.



các frame của một nguồn gần như trùng nhau: chia theo nguồn, không chia theo ảnh

Chia theo nguồn trước khi augment; augmentation chỉ sinh từ ảnh train.

Với ảnh lấy từ video, theo người chụp hoặc theo thiết bị, ta chia theo nhóm nguồn. Random split theo frame gần nhau cho validation gần như bản sao của train, và augmentation không sửa được leakage này.

HỎI NHANH

Model phân loại biển báo “rẽ trái” và “rẽ phải”. Một đội thêm TTA bằng cách lấy trung bình xác suất của ảnh gốc và ảnh lật ngang. Điểm sẽ thay đổi ra sao?

- A Điểm tăng, vì TTA luôn giảm dao động của dự đoán
- B Xác suất hai lớp bị kéo về gần nhau, điểm giảm
- C Điểm không đổi, vì lúc train model đã gặp ảnh lật
- D Điểm giữ nguyên, chỉ thời gian suy luận tăng gấp đôi

Muốn biết một phép augmentation có ích không, làm thí nghiệm bật/tắt từng phần (ablation) với bốn cấu hình: baseline; baseline thêm phép A; baseline thêm phép B; cả hai. Giữ split, seed, số bước và metric; chỉ augmentation thay đổi.

Validation nên có cả phần giống điều kiện train lẫn phần mô phỏng điều kiện mới mà model phải chịu được, vì một phép biến đổi có thể không đổi điểm ở phần này mà đổi mạnh ở phần kia. Mục sau làm đúng thí nghiệm ấy. Validation ở điều kiện mới cũng chỉ ra cần augmentation nào: ảnh train toàn ban ngày mà dùng thật cả ban đêm thì thêm ảnh đêm thật nếu có, hoặc mô phỏng tối, nhiều, đổi màu cho train, rồi chấm trên ảnh đêm thật.

Thí nghiệm: dời ảnh và lật ảnh chữ số

Bộ chữ số viết tay 8×8 có sẵn trong scikit-learn đủ nhỏ để chạy bốn cấu hình trên CPU. Ta lấy 300 ảnh làm train, phần còn lại làm validation. Validation có hai bản: ảnh gốc, và chính các ảnh đó bị dời 1 pixel theo một hướng ngẫu nhiên, như chữ viết lệch khỏi khung hay camera đặt lệch. Hai phép augmentation:

- dời 1 pixel: giữ nhãn, vì chữ 2 dời sang phải vẫn là chữ 2;
- lật ngang: vẫn giữ nhãn cũ, dù phần lớn chữ số lật xong không còn là chính nó.

Bộ dữ liệu không kèm ID người viết, nên split ở đây ngẫu nhiên theo ảnh; cả bốn cấu hình dùng chung một split nên vẫn so được với nhau.

Python

```
import numpy as np
import torch
from torch import nn
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

torch.set_num_threads(1)
digits = load_digits()
images = torch.tensor(digits.images / 16.0, dtype=torch.float32).unsqueeze(1)
labels = torch.tensor(digits.target)
train_ids, valid_ids = train_test_split(
    np.arange(len(labels)), train_size=300, stratify=digits.target, random_state=53)
train_x, train_y = images[train_ids], labels[train_ids]
valid_x, valid_y = images[valid_ids], labels[valid_ids]

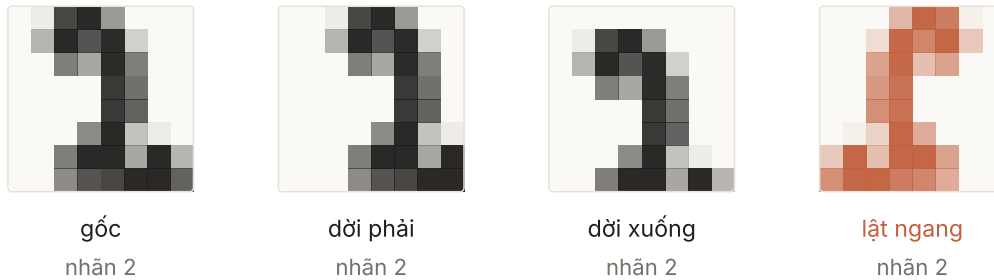
MOVES = [(0, 1), (0, -1), (1, 0), (-1, 0)] # (hàng, cột): phải, trái, xuống, lên

def shift(batch, dy, dx):
    """Dời ảnh dy hàng, dx cột; phần lộ ra điền 0, không cuộn vòng."""
    out = torch.zeros_like(batch)
    h, w = batch.shape[-2:]
    out[..., max(dy, 0):h + min(dy, 0), max(dx, 0):w + min(dx, 0)] = \
        batch[..., max(-dy, 0):h + min(-dy, 0), max(-dx, 0):w + min(-dx, 0)]
    return out

gen = torch.Generator().manual_seed(0)
move_of = torch.randint(0, 4, (len(valid_x),), generator=gen).tolist()
valid_moved = torch.stack([shift(x, *MOVES[k]) for x, k in zip(valid_x, move_of)])
versions = torch.stack([train_x] + [shift(train_x, *move) for move in MOVES])
print("train/valid:", len(train_x), len(valid_x))
print("versions:", tuple(versions.shape))
```

versions giữ sẵn năm phiên bản của mỗi ảnh train: nguyên bản và bốn hướng dời. Mỗi batch chọn ngẫu nhiên một phiên bản cho từng ảnh, nên mỗi epoch model gặp một tổ

hợp khác. Cách này tương đương biến đổi lúc đọc mẫu, chỉ rẻ hơn vì phép dời ở đây có đúng năm khả năng.



Dời một pixel vẫn là chữ 2; lật ngang thì không còn là chữ 2.

Mỗi cấu hình train cùng một CNN nhỏ trong 60 epoch; mỗi dòng in trung bình accuracy của ba seed.

Python

```
def augment(rows, use_shift, use_flip, gen):
    if use_shift:      # mỗi ảnh: giữ nguyên hoặc một trong bốn hướng dời
        pick = torch.randint(0, 5, (len(rows),), generator=gen)
    else:
        pick = torch.zeros(len(rows), dtype=torch.long)
    x = versions[pick, rows]
    if use_flip:      # lật ngang khoảng một nửa số ảnh, nhãn giữ nguyên
        flip = torch.rand(len(rows), generator=gen) < 0.5
        x = torch.where(flip[:, None, None, None], x.flip(-1), x)
    return x

def accuracy(model, x, y):
    with torch.no_grad():
        return (model(x).argmax(dim=1) == y).float().mean().item()

def run(use_shift, use_flip, seed, epochs=60):
    torch.manual_seed(seed)
    model = nn.Sequential(
        nn.Conv2d(1, 16, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
        nn.Conv2d(16, 32, 3, padding=1), nn.ReLU(),
        nn.AdaptiveAvgPool2d(1), nn.Flatten(1), nn.Linear(32, 10))
    optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
    gen = torch.Generator().manual_seed(seed)
    for _ in range(epochs):
```

```

model.train()
order = torch.randperm(len(train_x), generator=gen)
for start in range(0, len(order), 50):
    rows = order[start:start + 50]
    loss = nn.functional.cross_entropy(
        model(augment(rows, use_shift, use_flip, gen)), train_y[rows])
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
model.eval()
return accuracy(model, valid_x, valid_y), accuracy(model, valid_moved, valid_y)

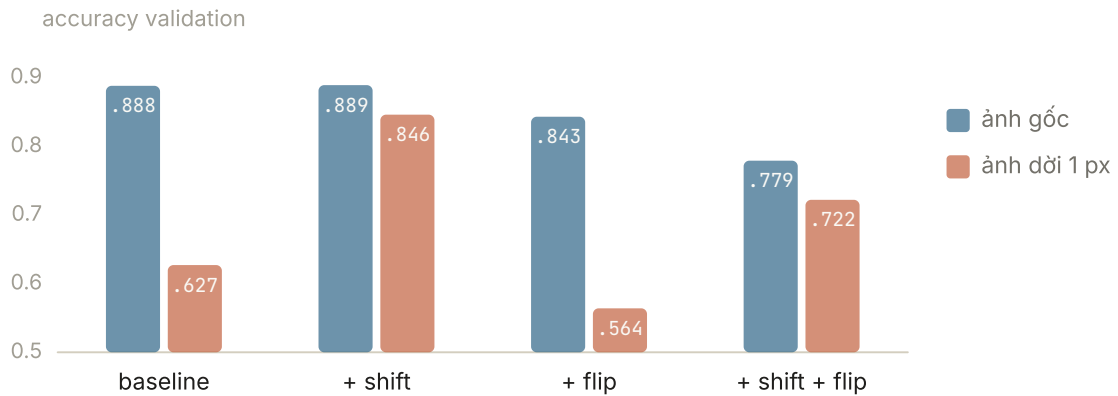
print("config          original  shifted")
for name, use_shift, use_flip in [("baseline", False, False), ("+ shift", True, False),
                                  ("+ flip", False, True), ("+ shift + flip", True, True)]:
    scores = np.array([run(use_shift, use_flip, seed) for seed in (1, 2, 3)])
    original, shifted = scores.mean(axis=0)
    print(f"{name:14s} {original:8.3f} {shifted:7.3f}")

```

Đọc từng cột:

- **Cột original:** baseline và thêm phép dời gần như bằng nhau, 0.888 và 0.889. Chênh lệch này nhỏ hơn nhiều so với dao động khi đổi seed, nên cột ảnh gốc không cho biết phép dời giúp hay hại. Nếu validation chỉ có cột này, ta dễ kết luận phép dời vô ích.
- **Cột shifted** cho thấy điều ngược lại: 0.627 lên 0.846. Phép dời dạy model một bất biến mà validation gốc không hỏi tới.
- **Lật ngang** làm giảm cả hai cột (0.843 và 0.564), vì các chữ 2, 3, 4, 5, 6, 7, 9 sau khi lật không còn là chính chúng mà vẫn mang nhãn cũ.
- **Ghép hai phép**, phép lật ăn bớt phần lợi của phép dời: 0.722 so với 0.846 ở cột shifted.

Augmentation không tự cộng dồn lợi ích; mỗi phép phải được thử riêng.



Phép dời giúp model đoán đúng ảnh bị dời; phép lật làm hại cả hai.

HỎI NHANH

Một đội chỉ có validation gồm ảnh gốc. Baseline được 0.888, thêm phép dời được 0.889. Họ bỏ phép dời vì cho rằng nó không có tác dụng. Bạn thấy kết luận ấy thế nào?

- A Hợp lý, phép dời không đổi điểm thì chỉ tốn thời gian train
- B Hợp lý, nên thay bằng lật ngang cho điểm thay đổi rõ hơn
- C Chưa đủ, cần thêm seed tới khi cột ảnh gốc tăng rõ rệt
- D Chưa đủ, validation không có ảnh dời để đo đúng tác dụng ấy

Transfer learning chuyển cái gì?

Encoder chuyển ảnh thành vector hoặc feature map; head chuyển biểu diễn đó thành nhãn của bài hiện tại. Transfer learning bắt đầu từ encoder đã học ở một nhiệm vụ nguồn, với giả định một phần đặc trưng có ích cho nhiệm vụ đích: cạnh và vân thường dùng lại được, còn ý nghĩa lớp và miền ảnh có thể khác.

Có ba mức thực hành:

- **Feature extraction** đóng băng encoder, chỉ train head mới.

- **Fine-tuning** mở một số block cuối hoặc toàn bộ encoder, thường với learning rate nhỏ hơn cho phần đã học.
- **Train từ đầu** bỏ lợi thế tri thức nguồn, nhưng là lựa chọn duy nhất khi đề cấm pretrained, và có thể hợp khi miền quá khác.

Cơ chế đóng băng, learning rate riêng, BatchNorm và optimizer đã có ở bài 4.3; mục này bàn phần riêng của ảnh.

Hai câu hỏi định hướng lựa chọn: có bao nhiêu ảnh có nhãn, và ảnh đích giống ảnh nguồn đến đâu.

	Miền gần nguồn (đồ vật chụp bằng điện thoại)	Miền xa nguồn (X-quang, siêu âm, ảnh vệ tinh)
Ít ảnh	Feature extraction làm mốc; mở thêm để học thuộc	Lấy đặc trưng ở block giữa, hoặc mở dần kèm augmentation
Nhiều ảnh	Mở block cuối hoặc toàn mạng, learning rate nhỏ	Mở nhiều block; so thêm với train từ đầu

Bảng cho điểm xuất phát; validation quyết định. Lý do của cột bên phải: các block đầu học cạnh và vân nên vẫn dùng lại được, còn các block sau gắn với vật thể của tập nguồn nên ít hợp với miền mới.

Pretrained không tự động tốt hơn. Nếu nguồn là ảnh RGB tự nhiên, còn đích là ảnh xám rất nhỏ mà chi tiết một pixel quan trọng, resize và preprocessing có thể phá thông tin.

Augmentation và transfer bổ sung cho nhau: khi fine-tune trên ít ảnh, augmentation hợp lệ giảm học thuộc; khi chỉ train head trên đặc trưng đã lưu, augmentation ngẫu nhiên mất tác dụng vì đặc trưng không được tính lại (bài 4.2).

HỎI NHANH

Có 600 ảnh siêu âm có nhãn và một ResNet ImageNet. Lấy đặc trưng ở block cuối rồi train head thì validation thấp hơn nhiều so với kỳ vọng. Nên thử gì tiếp?

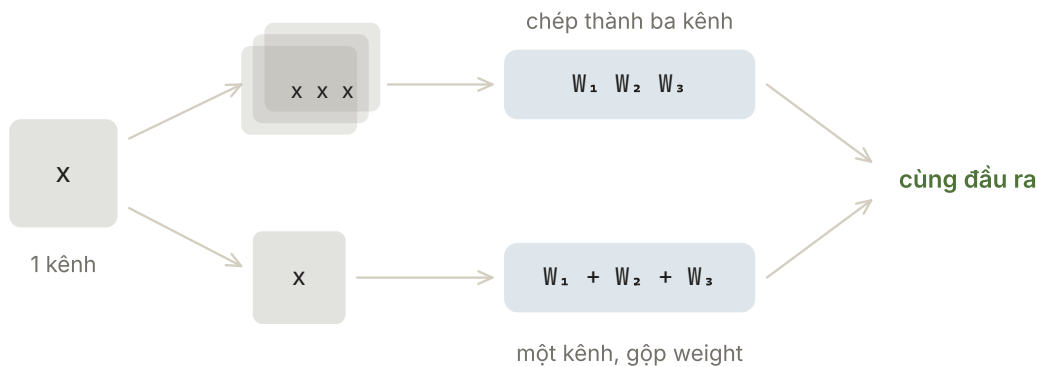
- A Dùng đặc trưng của block giữa, hoặc mở thêm block encoder
- B Bỏ pretrained, train ResNet-50 từ đầu trên đúng 600 ảnh này
- C Giữ nguyên encoder, train head lâu gấp mười lần
- D Đổi sang encoder ImageNet lớn hơn, vẫn chỉ lấy block cuối

Ảnh xám vào encoder ba kênh

Encoder ImageNet chờ ảnh RGB: conv đầu tiên có weight shape $(C_{\text{out}}, 3, k, k)$. Ảnh X-quang, ảnh nhiệt hay chữ số chỉ có một kênh. Cách đơn giản là chép kênh xám thành ba kênh giống nhau. Vì convolution tuyến tính theo đầu vào, khi cả ba kênh đều là x thì

$$\sum_{c=1}^3 W_c * x = \left(\sum_{c=1}^3 W_c \right) * x,$$

tức một conv một kênh có weight bằng **tổng** ba lát weight theo kênh cho đúng cùng đầu ra. Với conv đầu của ResNet-18, weight giảm từ $64 \times 3 \times 7 \times 7 = 9408$ xuống $64 \times 1 \times 7 \times 7 = 3136$ số, và phép tính ở lớp đầu giảm ba lần. Đoạn sau kiểm đẳng thức trên bốn ảnh chữ số; conv_rgb khởi tạo ngẫu nhiên đóng vai conv đầu đã học, vì đẳng thức đúng với mọi giá trị weight.



Ảnh xám chép thành ba kênh cho cùng đầu ra với conv một kênh có weight là tổng ba lát.

Python

```
torch.manual_seed(53)
conv_rgb = nn.Conv2d(3, 8, 3, padding=1) # đóng vai conv đầu của encoder RGB
conv_gray = nn.Conv2d(1, 8, 3, padding=1)
with torch.no_grad():
    conv_gray.weight.copy_(conv_rgb.weight.sum(dim=1, keepdim=True))
    conv_gray.bias.copy_(conv_rgb.bias)

gray = valid_x[:4] # bốn ảnh chữ số, shape (4, 1, 8, 8)
with torch.no_grad():
    out_repeat = conv_rgb(gray.repeat(1, 3, 1, 1)) # chép kênh xám thành ba kênh
    out_folded = conv_gray(gray) # một kênh, weight đã gộp
print("weight:", tuple(conv_rgb.weight.shape), "->", tuple(conv_gray.weight.shape))
print("output:", tuple(out_repeat.shape), tuple(out_folded.shape))
print("same output:", torch.allclose(out_repeat, out_folded, atol=1e-6))
```

Hai đầu ra cùng shape và trùng nhau đến sai số làm tròn.

Ngoài số kênh, đầu vào còn phải đúng chuẩn hóa: encoder ImageNet học trên ảnh đã trừ mean (0.485, 0.456, 0.406) và chia std (0.229, 0.224, 0.225) theo từng kênh. Chép kênh xám rồi chuẩn hóa từng kênh như lúc pretrain là cách đơn giản và an toàn; muốn gộp weight như trên thì phải xử lý thêm cả mean và std.

Kích thước cũng phải hợp: ResNet giảm cạnh ảnh 32 lần, nên ảnh 28×28 chỉ còn bản đồ 1×1 ở block cuối. Phóng lên 224×224 cho block ấy bản đồ 7×7 nhưng không thêm chi tiết nào, và tốn gấp 64 lần số điểm ảnh.

HỎI NHANH

Để đưa ảnh xám vào ResNet ImageNet, một đội thay conv đầu bằng conv một kênh. Weight của nó là **trung bình** ba lát weight RGB. Mọi tầng sau giữ nguyên, và model được dùng ngay để lấy đặc trưng. So với cách chép kênh xám thành ba kênh, đầu ra của conv đầu khác thế nào?

- A Giống hệt, vì trung bình và tổng chỉ khác tên gọi
- B Nhỏ đi 3 lần, nhưng ReLU phía sau tự bù lại thang đo
- C Phần trước bias nhỏ đi 3 lần, các tầng sau nhận sai thang
- D Lớn lên 3 lần, vì bias cũng bị chia đều cho ba kênh

Thực hành với encoder ảnh có sẵn

Script của chương nạp ResNet-18 từ một state dict ImageNet có sẵn trên máy, không tải từ Internet. Nó cần ba thứ: tệp checkpoint, tệp metadata ghi sha256, loại kiến trúc và 1000 tên lớp theo đúng thứ tự output, và một ảnh. Chạy không có `--run` chỉ kiểm điều kiện; có `--run` thì in lớp ImageNet có xác suất cao nhất cùng xác suất.

Kết quả ấy là phép kiểm đường nạp: ảnh một đồ vật quen thuộc được gọi đúng tên cho thấy trọng số và preprocessing khớp nhau. Nó chưa phải model cho nhãn của bài; bước tiếp theo là thay fc bằng head đúng số lớp. Đừng dùng `weights=None` rồi gọi model là pretrained: đó là mạng ngẫu nhiên cho đến khi đã nạp trọng số.

Trình tự hợp lý là kiểm preprocessing trên vài ảnh, chạy feature extraction làm mốc, rồi mở block cuối kèm augmentation hợp lệ. Khi mở encoder, lưu cả model, head và mapping lớp; nạp lại ở tiến trình mới để kiểm. Ghi cả score và thời gian, vì fine-tuning hay TTA có thể vượt ngân sách thi.

HỎI NHANH

Chạy script với một ảnh chai nước của câu lạc bộ, model in ra `water bottle 0.62`. Bài cần phân ba lớp là chai, cốc và hộp. Ta kết luận được gì?

- A Model đã sẵn sàng, chỉ cần đọc ba lớp ImageNet gần nhất
- B Đường nạp có vẻ đúng, nhưng vẫn cần head ba lớp và validation
- C Trọng số chưa được nạp, vì model đã học sẽ cho xác suất trên 0.9
- D Preprocessing sai, vì ảnh câu lạc bộ khác phân phối ImageNet

Khi đi thi

- Đọc luật về model pretrained trước khi chọn hướng.
- Lưu một lưới ảnh sau augmentation kèm nhãn, box hoặc mask, và nhìn trước khi train lâu.
- Chia validation theo nguồn trước khi augment; validation không augment ngẫu nhiên.

Tóm tắt

- Augmentation chỉ dùng những biến đổi mà đáp án vẫn đúng; với box hay mask, nhãn phải biến đổi theo; Mixup và CutMix trộn cả nhãn.
- Lợi ích của một phép chỉ đo được trên validation chứa đúng điều kiện nó nhắm tới: phép dời nâng điểm ảnh dời từ 0.627 lên 0.846, còn phép lật làm sai nhãn.
- Chia train/validation trước, augment sau; mọi bản biến đổi của một ảnh nằm cùng phía.
- Transfer learning: chọn feature extraction hay fine-tuning theo lượng ảnh và độ gần miền; đưa ảnh vào đúng số kênh, chuẩn hóa và kích thước mà encoder đã học.

Kiểm tra cuối bài

CÂU 1

Bài toán phân loại X-quang ngực có hay không có tràn dịch. Dấu hiệu bệnh là vùng phổi sáng hơn bình thường. Ảnh test chụp thẳng, lệch nhiều nhất khoảng 10°. Encoder là ResNet ImageNet. Pipeline train dưới đây có một bước có thể làm sai nhãn.

```
train_tf = transforms.Compose([
    transforms.Grayscale(num_output_channels=3),
    transforms.Resize((224, 224)),
    transforms.RandomRotation(degrees=10),
    transforms.ColorJitter(brightness=0.3, contrast=0.3),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
])
```

Đó là bước nào?

- A Grayscale(3), vì chép một kênh thành ba kênh làm sai thông tin
- B RandomRotation(10), vì nó tạo tư thế phổi không có thật
- C ColorJitter, vì đổi độ sáng có thể che đi hoặc tạo ra dấu hiệu bệnh
- D Normalize theo ImageNet, vì phải tính mean và std riêng cho X-quang

PHÒNG THEO NOAI 2026 Singapore · Chung kết · Câu 12

CÂU 2

CNN phân loại ảnh X-quang xám 512×512 , pixel từ 0 đến 255. Có ba cách tiền xử lý. Cách A chia cho 255. Cách B trừ 127.5 rồi chia cho std tính trên toàn tập train. Cách C lấy mỗi ảnh trừ mean rồi chia std **của chính ảnh đó**. Trên validation, A được 82.3%, B được 84.7%, còn C được 79.1% và loss dao động. Vì sao C kém nhất?

- A C cho đầu vào không có trung bình 0, nên BatchNorm mất ổn định
- B C xóa khác biệt độ sáng giữa các phim, mà đó là thông tin chẩn đoán
- C C nén dài giá trị khi ảnh không phủ hết 0–255, nên mất tương phản
- D C dùng thống kê của toàn tập, nên rò thông tin validation vào train

PHÒNG THEO NOAI 2026 Singapore · Vòng sơ loại · Câu 307

CÂU 3

Đề cho ảnh RGB 112×112 nền trắng. Mỗi ảnh có đúng một hình, là hình tròn, tam giác hoặc chữ nhật, với vị trí, kích thước, góc xoay và màu bất kỳ. Tam giác không nhất thiết là tam giác đều. Chỉ có 200 ảnh có nhãn, và model phải tự dựng. Bộ augmentation nào giữ đúng nhãn?

- A Xoay góc bất kỳ, lật, dời vị trí, đổi màu hình
- B Co giãn riêng trục ngang theo tỉ lệ 2:1, đổi màu hình
- C Random crop 40×40 từ ảnh 112×112 , giữ nhãn cũ
- D Cutout che một ô 56×56 ở vị trí ngẫu nhiên, giữ nhãn cũ

PHÒNG THEO USA-NA-AIO 2026 · Vòng 2 · Ngày 2 · Problem 4

CÂU 4

Một CNN train trên 500 ảnh đạt 99% accuracy trên train và 71% trên validation. Thêm augmentation hợp lệ, gồm crop nhẹ và đổi sáng nhẹ, thì train còn 93%, còn validation lên 78%. Augmentation đã giúp chủ yếu ở chỗ nào?

- A Giảm overfit, vì dữ liệu train đa dạng hơn
- B Train nhanh hơn, vì mỗi epoch model gặp ảnh dễ hơn
- C Model nhỏ đi, vì có ít tham số phải ghi nhớ hơn
- D Suy luận nhanh hơn, vì ảnh test được biến đổi trước

PHÒNG THEO [NOAI 2026 Singapore · Vòng sơ loại · Câu 299](#)

CÂU 5

Ba đội cùng dùng kiến trúc ResNet-18 cho 2000 ảnh lá cây. Đội A khởi tạo ngẫu nhiên rồi train toàn mạng. Đội B nạp trọng số ImageNet rồi train toàn mạng với learning rate nhỏ. Đội C nạp trọng số ImageNet, đóng băng encoder và chỉ train head. Đội nào đang dùng transfer learning?

- A Chỉ C, vì B đã thay đổi hết trọng số ban đầu
- B Chỉ B, vì C không cập nhật encoder
- C B và C, vì cả hai đều bắt đầu từ trọng số đã học
- D Cả ba, vì cùng dùng một kiến trúc đã công bố

PHÒNG THEO [NOAI 2026 Singapore · Vòng sơ loại · Câu 205](#)

CÂU 6

Mask rộng 8 pixel, các cột được đánh số từ 0 đến 7. Foreground nằm ở các cột 1, 2 và 3. Lật ngang thì foreground chuyển sang những cột nào?

A {4, 5, 6, 7}

B {4, 5, 6}

C {5, 6, 7}

D {1, 2, 3}

CÂU 7

Một đội sinh 10 bản augmentation cho mỗi ảnh, lưu chung một thư mục, rồi chia ngẫu nhiên 80/20 theo tệp. Validation đạt 0.99, nhưng trên ảnh mới chỉ được 0.71. Lý do khả dĩ nhất là gì?

A Các bản biến đổi của cùng một ảnh gốc nằm ở cả hai phía

B Model underfit, cần train thêm nhiều epoch nữa

C Augmentation quá yếu nên chưa chịu được ảnh mới

D Ảnh test của đề có nhiều nhãn bị gán sai

CÂU 8

Cấu hình không augmentation đạt 0.84 trên split tạo bằng seed 1. Cấu hình có augmentation đạt 0.88 trên split tạo bằng seed 2. Ta chắc chắn được điều gì?

- A Augmentation tăng đúng 0.04 điểm trên mọi split
- B Chưa tách được tác dụng của augmentation khỏi độ khó của split
- C Augmentation tăng ít nhất 0.04 điểm, vì split 2 khó hơn
- D Augmentation vô ích, vì chênh lệch nhỏ hơn 0.05

CÂU 9

Bài có 3 lớp. Mixup trộn ảnh mèo, lớp 0, với trọng số $\lambda = 0.7$ và ảnh chó, lớp 2, với trọng số 0.3. Model dự đoán $p = (0.5, 0.1, 0.4)$ cho ảnh trộn. Cross-entropy của mẫu này bằng bao nhiêu?

- A 0.485
- B 0.693
- C 0.755
- D 0.760

CÂU 10

Notebook nộp bài được chạy tối đa 20 phút. Đọc và resize toàn bộ ảnh test mất 5 phút, chỉ làm một lần rồi dùng chung cho mọi phiên bản. Mỗi lượt forward trên toàn bộ test mất 3 phút. Trên validation, TTA với 2, 4 và 8 phiên bản tăng điểm lần lượt 0.6, 0.8 và 0.9 so với không dùng TTA. Nên chọn bao nhiêu phiên bản?

A Không dùng TTA

B 2 phiên bản

C 4 phiên bản

D 8 phiên bản

Đọc thêm

- [PyTorch — Transfer Learning for Computer Vision](#): phân biệt train head cố định với fine-tuning; kiểm lại preprocessing của weights đang dùng.
- [TorchVision — Transforms](#): xem hỗ trợ ảnh, box, mask và cách dùng chung phép biến đổi hình học.
- [Dive into Deep Learning — Image Augmentation](#): đọc động cơ tăng biến thiên train, rồi đối chiếu với điều kiện giữ nhãn của bài.
- [Zhang và cộng sự — mixup: Beyond Empirical Risk Minimization \(2018\)](#): bài gốc của Mixup; đọc phần định nghĩa nhãn trộn và thí nghiệm với nhãn nhiễu.

[Tải gói recipe của chương](#) — gồm các module phụ và hướng dẫn chuẩn bị asset.