

AI VIET NAM – AI COURSE 2025

# Exercise: Advanced CNN Architectures

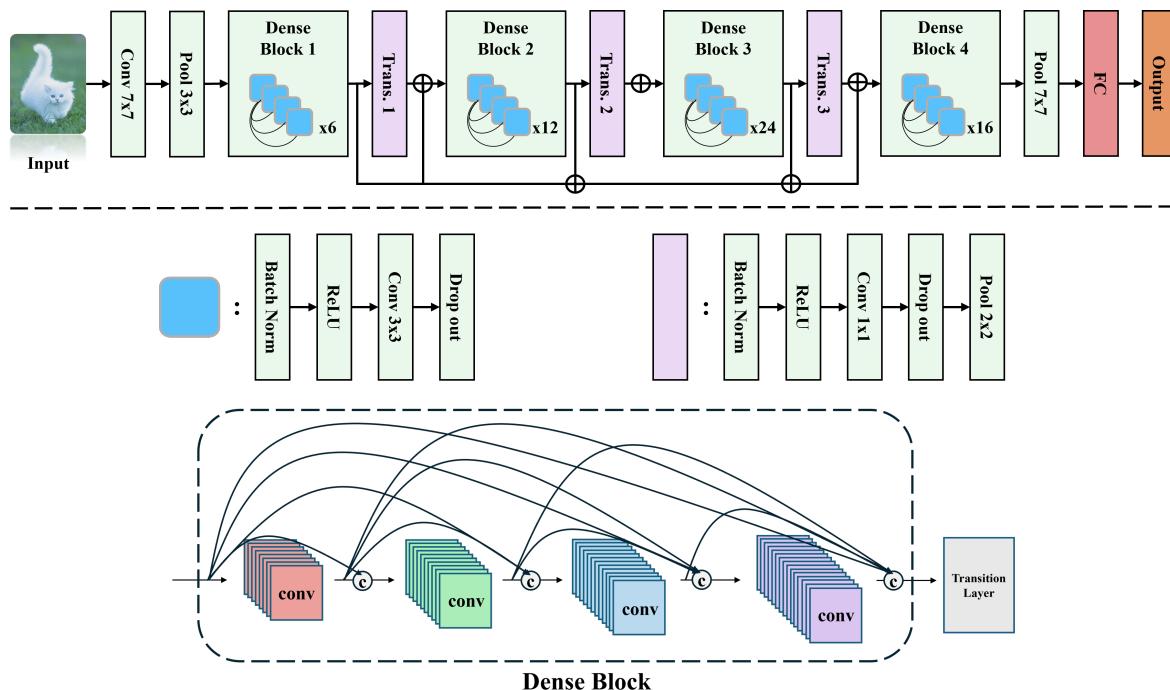
Nguyễn Quốc Thái

Hồ Quang Hiển

Đinh Quang Vinh

## I. Giới thiệu

Sau khi các kiến trúc CNN cổ điển như LeNet, AlexNet hay VGG chứng minh được hiệu quả trên nhiều bài toán thị giác, cộng đồng nghiên cứu nhanh chóng nhận ra rằng việc đơn giản tăng số lớp lên hàng chục, hàng trăm không phải lúc nào cũng cải thiện chất lượng mô hình. Khi mạng càng sâu, ta đối mặt với các vấn đề như gradient biến mất, gradient bùng nổ và hiện tượng degradation: độ sâu tăng nhưng độ chính xác trên tập kiểm tra lại giảm. Những hạn chế này thúc đẩy sự ra đời của các kiến trúc CNN nâng cao với thiết kế kết nối mới nhằm cải thiện dòng chảy thông tin và gradient trong mạng.

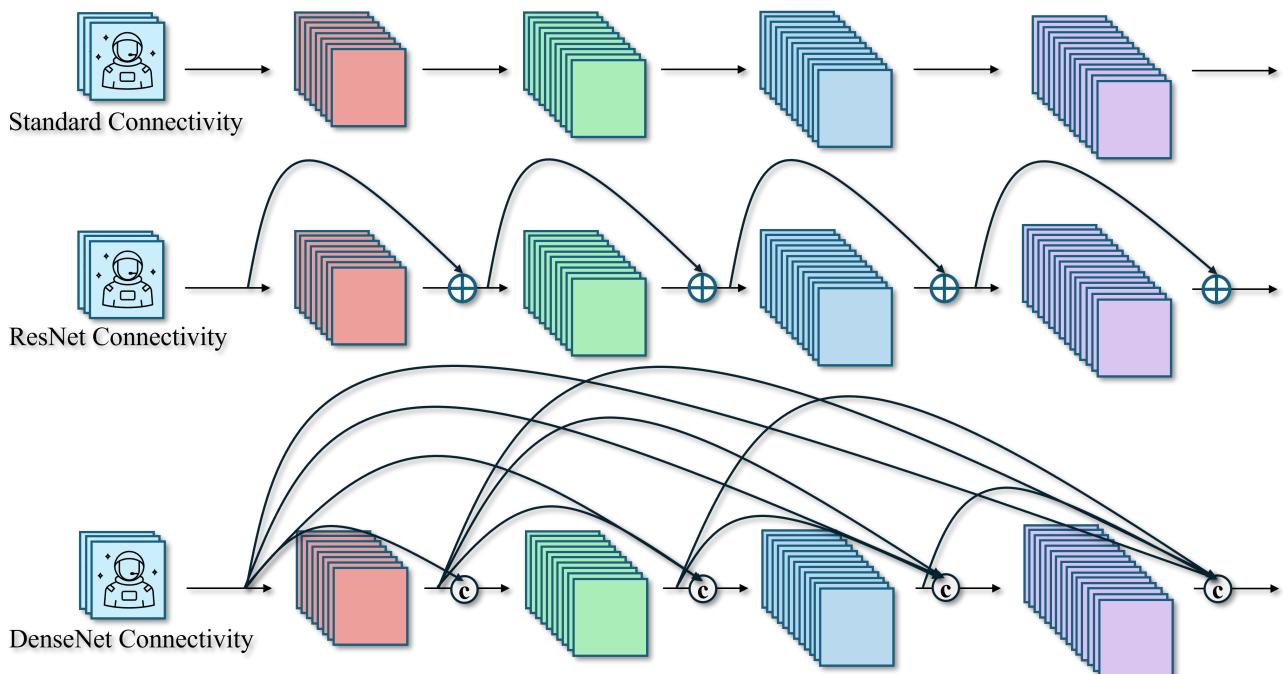


Hình 1: Mô phỏng kiến trúc mô hình DenseNet.

Trong số đó, ResNet là một mốc quan trọng với ý tưởng residual learning và các kết nối tắt (skip connection). Thay vì học trực tiếp ánh xạ đầu vào–đầu ra, mỗi khối chỉ học phần sai lệch  $F(x)$

rồi cộng lại thành  $F(x) + x$ , giúp gradient lan truyền ổn định hơn qua nhiều tầng và cho phép huấn luyện các mạng rất sâu (ResNet-50, ResNet-101, ...) mà không bị degradation nghiêm trọng.

DenseNet đi xa hơn khi cho mỗi lớp trong một dense block nhận đầu vào là *toàn bộ* feature map của các lớp trước đó trong cùng khối. Cách kết nối dày đặc này tăng cường tái sử dụng đặc trưng, giúp gradient truyền tốt hơn và đạt hiệu quả cao với số tham số ít hơn so với nhiều kiến trúc CNN sâu truyền thống.



Hình 2: So sánh cơ chế kết nối giữa CNN truyền thống, ResNet và DenseNet..

Trong bài tập này, ở phần lập trình, chúng ta sẽ thực hành xây dựng hai mô hình phân loại ảnh sử dụng kiến trúc ResNet và DenseNet, áp dụng vào hai bài toán phân loại ảnh đa lớp: *Weather Image Classification* và *Scenes Classification*. Thông qua đó, người học không chỉ ôn tập lại các khái niệm nền tảng của CNN mà còn hiểu rõ hơn cách các kiến trúc nâng cao giải quyết những vấn đề khi mạng ngày càng sâu. Phần bài tập trắc nghiệm ở cuối sẽ giúp củng cố lại lý thuyết về CNN, ResNet và DenseNet một cách hệ thống.

# Mục lục

I.	Giới thiệu . . . . .	1
II.	Kiến trúc ResNet . . . . .	4
II.1.	Residual learning và residual block . . . . .	4
II.2.	Skip connection và các dạng block trong ResNet . . . . .	4
III.	Kiến trúc DenseNet . . . . .	5
III.1.	Dense block và dense connectivity . . . . .	5
III.2.	Bottleneck layer và transition layer . . . . .	6
III.3.	Ưu điểm của DenseNet . . . . .	7
IV.	Bài toán Weather Image Classification với mô hình ResNet . . . . .	8
V.	Bài toán Scenes Classification với mô hình DenseNet . . . . .	17
VI.	Câu hỏi trắc nghiệm . . . . .	25
	Phụ lục . . . . .	29

## II. Kiến trúc ResNet

Khi tăng độ sâu của CNN lên hàng chục hoặc hàng trăm tầng, ta thường gặp hiện tượng suy giảm hiệu năng (*degradation*): thêm tầng nhưng độ chính xác trên tập kiểm tra không tăng, thậm chí giảm. Nguyên nhân chính đến từ tối ưu hoá khó hơn và gradient suy yếu (*vanishing gradient*) khi lan truyền qua nhiều tầng. ResNet (Residual Network) được đề xuất để giảm vấn đề này bằng cách thay đổi dạng ánh xạ mà mỗi khối phải học và bổ sung các *skip connection*.

### II.1. Residual learning và residual block

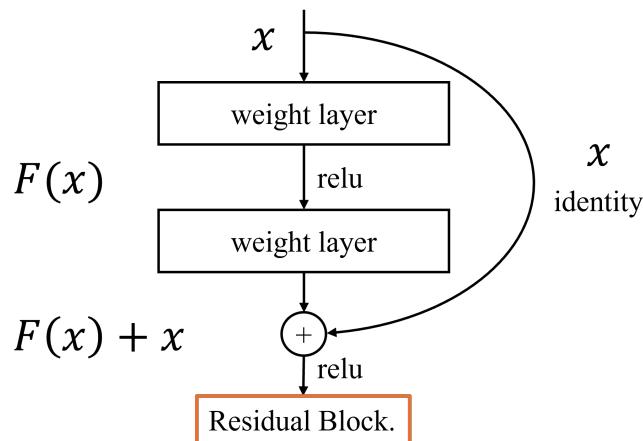
Ý tưởng trung tâm của ResNet là *residual learning*: thay vì để một khối học trực tiếp ánh xạ  $H(x)$ , ta để khối học *phản dư* (*residual*)  $F(x)$  so với đầu vào  $x$ :

$$H(x) = F(x) + x.$$

Với một *residual block*, đầu ra thường được viết:

$$y = F(x, \mathbf{W}) + x,$$

trong đó  $x$  là input feature map,  $F(x, \mathbf{W})$  là kết quả của một chuỗi Conv–BN–ReLU tham số hoá bởi  $\mathbf{W}$ , và phép cộng là cộng element-wise. Khi  $F(x) \approx 0$ , khối tiến gần *identity mapping*, nên việc thêm nhiều residual block sẽ ít làm mô hình suy giảm hiệu năng, đồng thời giúp bài toán tối ưu ổn định hơn.



Hình 3: Mô phỏng cơ chế hoạt động của Residual Block

### II.2. Skip connection và các dạng block trong ResNet

Để thực hiện  $F(x) + x$ , ResNet sử dụng *skip connection* (hoặc *shortcut connection*). Có hai dạng chính:

- **Identity shortcut:** dùng khi kích thước không gian (height, width) và số kênh của đầu vào và đầu ra block trùng nhau. Lúc này  $x$  được cộng trực tiếp với  $F(x)$ .

- **Projection shortcut:** dùng khi kích thước không trùng khớp. ResNet áp dụng một Conv  $1 \times 1$  (kèm BN) trên nhánh tắt để biến đổi  $x$  thành  $W_s x$  cùng kích thước với  $F(x)$ :

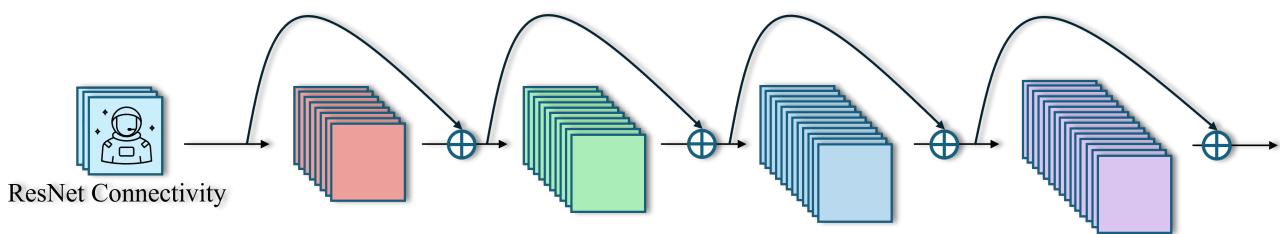
$$y = F(x, \mathbf{W}) + W_s x.$$

Trong các mô hình nông hơn như ResNet-18 và ResNet-34, mỗi residual block thường là *basic block* gồm hai Conv  $3 \times 3$  (mỗi Conv đi kèm BN và ReLU). Với các mô hình sâu hơn như ResNet-50, ResNet-101, ResNet-152, ResNet sử dụng *bottleneck block* để giảm số tham số và chi phí tính toán. Bottleneck block có dạng:

$$1 \times 1 \rightarrow 3 \times 3 \rightarrow 1 \times 1,$$

trong đó Conv  $1 \times 1$  đầu giảm số kênh (bottleneck), Conv  $3 \times 3$  trích xuất đặc trưng chính, và Conv  $1 \times 1$  cuối khôi phục lại số kênh ban đầu trước khi cộng với nhánh tắt.

Nhờ residual learning và skip connection, ResNet cho phép huấn luyện mạng rất sâu mà vẫn duy trì gradient tốt, trở thành một kiến trúc backbone chuẩn cho nhiều bài toán thị giác hiện đại như phân loại ảnh, phát hiện đối tượng và phân đoạn ảnh.



Hình 4: Mô phỏng đơn giản hóa cơ chế hoạt động của ResNet

### III. Kiến trúc DenseNet

DenseNet (Densely Connected Convolutional Network) cũng được thiết kế nhằm cải thiện khả năng huấn luyện mạng CNN sâu, nhưng theo một hướng khác so với ResNet. Thay vì chỉ thêm *skip connection* giữa đầu vào và đầu ra của một block, DenseNet sử dụng *dense connectivity*: mỗi lớp trong một khối sẽ nhận đầu vào là *tất cả* feature map của các lớp trước đó trong cùng khối. Cách kết nối này tăng cường *feature reuse* và cải thiện *gradient flow* trong mạng.

#### III.1. Dense block và dense connectivity

Giả sử một *dense block* có  $L$  lớp. Với DenseNet, đầu vào của lớp thứ  $l$  ( $1 \leq l \leq L$ ) được định nghĩa:

$$x_l = H_l([x_0, x_1, x_2, \dots, x_{l-1}]),$$

trong đó:

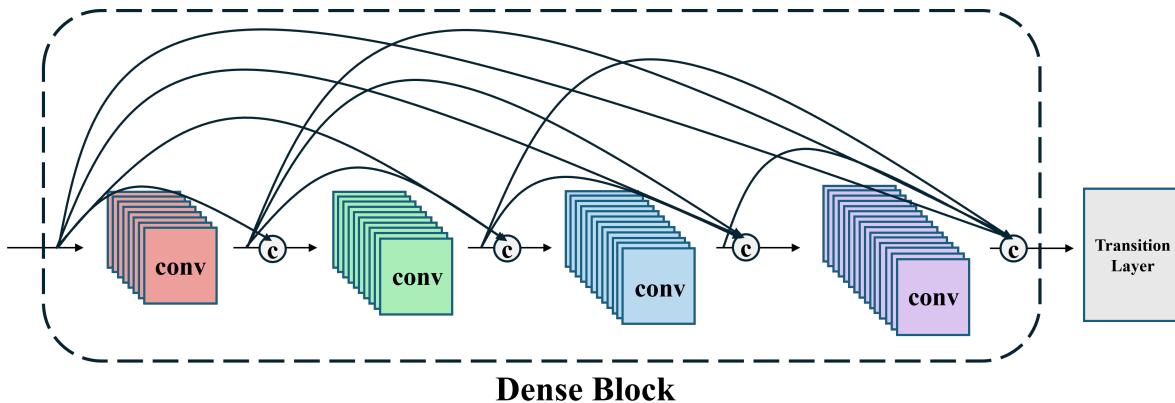
- $x_0$  là đầu vào ban đầu của dense block,

- $x_i$  là đầu ra của lớp thứ  $i$  trong block,
- $[ \cdot ]$  là phép nối theo kênh (channel-wise concatenation),
- $H_l(\cdot)$  là ánh xạ phi tuyến của lớp thứ  $l$  (thường BN-ReLU-Conv).

Mỗi lớp trong dense block thường sinh thêm một số kênh cố định gọi là *growth rate*  $k$ . Nếu đầu vào block có  $k_0$  kênh và block có  $L$  lớp, thì đầu ra block có:

$$k_0 + L \cdot k$$

kênh. Do đặc trưng của tất cả các lớp trước đều được giữ lại bằng phép nối, các lớp sau có thể trực tiếp khai thác cả đặc trưng thấp (low-level) và đặc trưng cao (high-level) mà không cần “học lại từ đầu”.



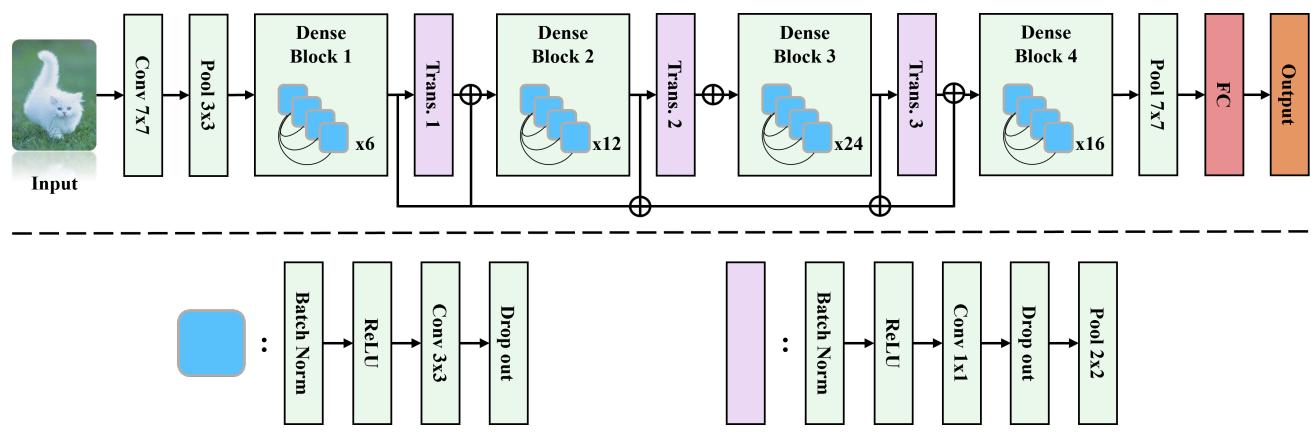
Hình 5: Mô phỏng đơn giản Dense block

### III.2. Bottleneck layer và transition layer

Để kiểm soát số kênh tăng nhanh và chi phí tính toán, DenseNet sử dụng hai thành phần quan trọng:

- **Bottleneck layer:** mỗi lớp trong dense block thường có cấu trúc BN-ReLU-Conv  $1 \times 1$ -BN-ReLU-Conv  $3 \times 3$ . Lớp Conv  $1 \times 1$  đóng vai trò *bottleneck*, giảm số kênh trước khi áp dụng Conv  $3 \times 3$ , giúp giảm số phép tính và bộ nhớ.
- **Transition layer:** được đặt giữa hai dense block liên tiếp, gồm BN-ReLU-Conv  $1 \times 1$ -Pooling. Lớp này vừa giảm kích thước không gian (qua pooling), vừa có thể giảm số kênh theo một *compression factor* nhỏ hơn 1, tránh để số kênh phình to sau nhiều block.

Nhờ kết hợp dense block, bottleneck layer và transition layer, DenseNet duy trì được độ sâu mạng lớn nhưng vẫn kiểm soát tốt số tham số và chi phí tính toán.



Hình 6: Kiến trúc mô hình DenseNet

### III.3. Ưu điểm của DenseNet

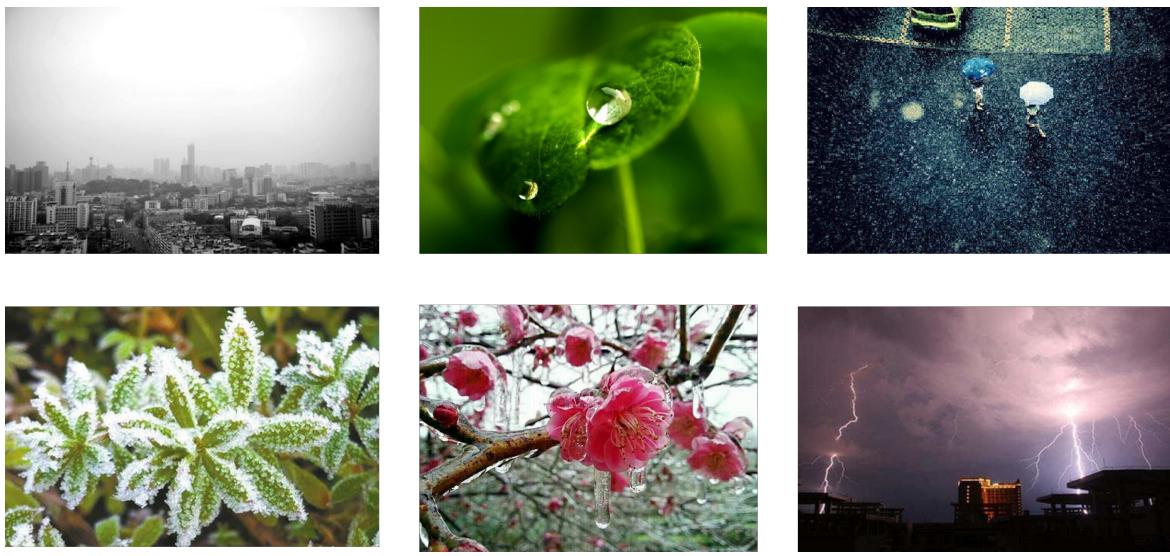
Cấu trúc kết nối dày đặc của DenseNet mang lại một số ưu điểm:

- **Tái sử dụng đặc trưng (feature reuse):** mỗi lớp truy cập trực tiếp các feature map từ nhiều mức trừu tượng khác nhau, giúp tận dụng lại thông tin đã học và giảm trùng lặp trong biểu diễn.
- **Dòng chảy gradient tốt (improved gradient flow):** do có nhiều đường kết nối ngắn từ các lớp gần đầu ra về gần đầu vào, gradient dễ lan truyền ngược qua toàn bộ block, giảm hiện tượng *vanishing gradient*.
- **Implicit deep supervision:** vì nhiều lớp nằm “gần” hàm mất mát hơn về mặt đường đi gradient, quá trình tối ưu giống như được giám sát ở nhiều độ sâu khác nhau, giúp việc huấn luyện mạng sâu ổn định hơn.

Nhờ những đặc điểm này, DenseNet đạt hiệu năng cao với số tham số tương đối ít, và thường được sử dụng như một kiến trúc CNN nâng cao hiệu quả trong các bài toán phân loại ảnh, phân đoạn và nhiều tác vụ thị giác khác.

## IV. Bài toán Weather Image Classification với mô hình ResNet

Trong phần này, chúng ta áp dụng kiến trúc ResNet cho bài toán *Weather Image Classification*: phân loại ảnh theo các lớp thời tiết khác nhau (ví dụ: sunny, cloudy, rainy, foggy, ...).



Hình 7: Vài mẫu dữ liệu trong bài toán Weather Image Classification.

### Bước 1: Import thư viện và tải dữ liệu

```

1 import torch
2 import torch.nn as nn
3 import os
4 import random
5 import numpy as np
6 import pandas as pd
7 import matplotlib.pyplot as plt
8 from tqdm import tqdm
9
10 from PIL import Image
11 from torch.utils.data import Dataset, DataLoader
12 from sklearn.model_selection import train_test_split
13
14 root_dir = "weather-dataset/dataset"
15 classes = {
16     label_idx: class_name \
17         for label_idx, class_name in enumerate(
18             sorted(os.listdir(root_dir))
19         )

```

```

20 }
21
22 img_paths = []
23 labels = []
24 for label_idx, class_name in classes.items():
25     class_dir = os.path.join(root_dir, class_name)
26     for img_filename in os.listdir(class_dir):
27         img_path = os.path.join(class_dir, img_filename)
28         img_paths.append(img_path)
29         labels.append(label_idx)

```

Ở bước này, ta import các thư viện cần thiết cho bài toán *Weather Image Classification*, bao gồm PyTorch (mô hình và huấn luyện), NumPy và pandas (xử lý dữ liệu), Matplotlib (trực quan hoá), tqdm (thanh tiến trình), PIL (đọc ảnh), torch.utils.data (Dataset, DataLoader) và train\_test\_split từ sklearn. Biến root\_dir trả về thư mục chứa ảnh thời tiết, được tổ chức theo từng lớp con tương ứng với mỗi loại thời tiết. Đoạn mã tiếp theo xây dựng ánh xạ classes từ chỉ số nhãn số nguyên (label\_idx) sang tên lớp (class\_name), đồng thời duyệt toàn bộ cây thư mục để thu thập danh sách đường dẫn ảnh img\_paths và vector nhãn tương ứng labels. Đây là metadata đầu vào cho bước tạo Dataset và DataLoader ở các phần sau.

## Bước 2: Chia tập dữ liệu và xây dựng WeatherDataset

```

1 val_size = 0.2
2 test_size = 0.125
3 is_shuffle = True
4
5 X_train, X_val, y_train, y_val = train_test_split(
6     img_paths, labels,
7     test_size=val_size,
8     random_state=seed,
9     shuffle=is_shuffle
10 )
11
12 X_train, X_test, y_train, y_test = train_test_split(
13     X_train, y_train,
14     test_size=test_size,
15     random_state=seed,
16     shuffle=is_shuffle
17 )
18
19 class WeatherDataset(Dataset):
20     def __init__(
21         self,
22         X, y,
23         transform=None
24     ):
25         self.transform = transform
26         self.img_paths = X

```

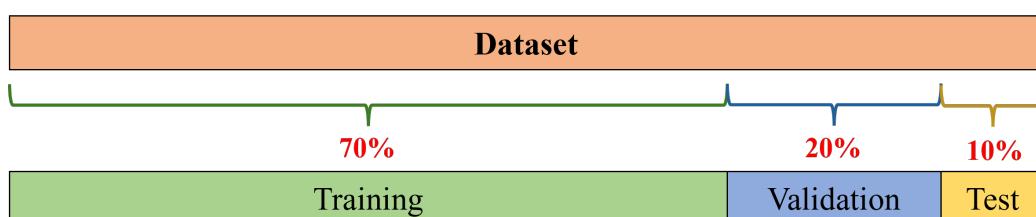
```

27     self.labels = y
28
29     def __len__(self):
30         return len(self.img_paths)
31
32     def __getitem__(self, idx):
33         img_path = self.img_paths[idx]
34         img = Image.open(img_path).convert("RGB")
35
36         if self.transform:
37             img = self.transform(img)
38
39         return img, self.labels[idx]
40
41     def transform(img, img_size=(224, 224)):
42         img = img.resize(img_size)
43         img = np.array(img)[..., :3]
44         img = torch.tensor(img).permute(2, 0, 1).float()
45         normalized_img = img / 255.0
46
47         return normalized_img

```

Ở bước này, ta chia tập ảnh thời tiết thành ba phần: *train*, *validation* và *test*. Tham số `val_size` = 0.2 giúp tách ra khoảng 20% dữ liệu ban đầu làm tập *validation*, phần còn lại dùng để huấn luyện. Sau đó, ta tiếp tục tách một phần nhỏ (`test_size` = 0.125 của tập train ban đầu) thành *test set*, tương đương khoảng 10% dữ liệu gốc, đảm bảo có một tập kiểm tra độc lập để đánh giá cuối cùng. Tham số `random_state` = `seed` và `shuffle` = True đảm bảo việc chia dữ liệu có tính ngẫu nhiên nhưng có thể tái lập.

Lớp `WeatherDataset` kế thừa `torch.utils.data.Dataset` để đóng gói danh sách đường dẫn ảnh X và nhãn y. Phương thức `__getitem__` đọc ảnh bằng PIL, chuyển về không gian màu RGB và áp dụng phép biến đổi `transform` (nếu có) trước khi trả về cặp (`image`, `label`). Hàm `transform` ở đây thực hiện các bước tiền xử lý cơ bản: resize ảnh về kích thước  $224 \times 224$ , chuyển sang NumPy `array`, sắp xếp lại thứ tự chiều về dạng (C, H, W), ép kiểu `float` và chuẩn hóa giá trị pixel về khoảng [0, 1]. Đây là định dạng đầu vào chuẩn cho các kiến trúc CNN hiện đại như ResNet trong các bước huấn luyện tiếp theo.



Hình 8: Mô phỏng cách chia dữ liệu thành các tập train, val, test.

## Bước 3: Khởi tạo Dataset và DataLoader cho train/val/test

```

1 train_dataset = WeatherDataset(
2     X_train, y_train,
3     transform=transform
4 )
5 val_dataset = WeatherDataset(
6     X_val, y_val,
7     transform=transform
8 )
9 test_dataset = WeatherDataset(
10    X_test, y_test,
11    transform=transform
12 )
13
14 train_batch_size = 256
15 test_batch_size = 8
16
17 train_loader = DataLoader(
18     train_dataset,
19     batch_size=train_batch_size,
20     shuffle=True
21 )
22 val_loader = DataLoader(
23     val_dataset,
24     batch_size=test_batch_size,
25     shuffle=False
26 )
27 test_loader = DataLoader(
28     test_dataset,
29     batch_size=test_batch_size,
30     shuffle=False
31 )

```

Từ các danh sách `X_train`, `X_val`, `X_test` và nhãn tương ứng, ta khởi tạo ba đối tượng `WeatherDataset` cho từng tập *train*, *validation* và *test*, cùng dùng chung hàm `transform` để đảm bảo quy trình tiền xử lý nhất quán.

Tiếp theo, ta xây dựng các `DataLoader`:

- `train_loader` sử dụng `batch_size = 256` và `shuffle = True`, giúp mô hình được huấn luyện trên các mini-batch ngẫu nhiên, giảm overfitting và tận dụng song song GPU tốt hơn.
- `val_loader` và `test_loader` dùng `batch_size = 8`, `shuffle = False` để giữ thứ tự dữ liệu ổn định trong quá trình đánh giá, đồng thời không cần xáo trộn vì không tham gia cập nhật trọng số.

Các `DataLoader` này sẽ là nguồn cấp dữ liệu chuẩn cho vòng lặp huấn luyện và đánh giá mô hình ResNet trong các bước tiếp theo.

## Bước 4: Xây dựng mô hình ResNet

### Code Exercise

```

1 class ResidualBlock(nn.Module):
2     def __init__(self, in_channels, out_channels, stride=1):
3         super(ResidualBlock, self).__init__()
4         self.conv1 = nn.Conv2d(
5             in_channels, out_channels,
6             kernel_size=3, stride=stride, padding=1
7         )
8         self.batch_norm1 = nn.BatchNorm2d(out_channels)
9         self.conv2 = nn.Conv2d(
10            out_channels, out_channels,
11            kernel_size=3, stride=1, padding=1
12        )
13         self.batch_norm2 = nn.BatchNorm2d(out_channels)
14
15         self.downsample = nn.Sequential()
16         if stride != 1 or in_channels != out_channels:
17             self.downsample = nn.Sequential(
18                 nn.Conv2d(
19                     in_channels, out_channels,
20                     kernel_size=1, stride=stride
21                 ),
22                 nn.BatchNorm2d(out_channels)
23             )
24         self.relu = nn.ReLU()
25
26     def forward(self, x):
27         shortcut = x.clone()
28         x = self.conv1(x)
29         x = ***Your Code Here***      # (1)
30         x = ***Your Code Here***      # (2)
31         x = ***Your Code Here***      # (3)
32         x = ***Your Code Here***      # (4)
33         x += ***Your Code Here***    # (5)
34         x = self.relu(x)
35
36     return x

```

Ở phần đầu của bước này, bạn cần **hoàn thiện hàm forward** của **ResidualBlock** để mô phỏng đúng một *basic residual block* trong ResNet với nhánh chính (main path) và nhánh tắt (skip connection).

Yêu cầu logic cho từng vị trí **\*\*\*Your Code Here\*\*\***:

- (1) Chuẩn hoá đầu ra của tích chập đầu tiên bằng lớp *batch normalization* tương ứng.
- (2) Áp dụng hàm kích hoạt *ReLU* lên kết quả đã được chuẩn hoá.
- (3) Truyền tensor này qua lớp tích chập thứ hai trong khối.

- (4) Tiếp tục chuẩn hoá đầu ra của tích chập thứ hai bằng lớp *batch normalization* còn lại.
- (5) Áp dụng nhánh tắt (skip connection) lên **shortcut** để đưa về cùng kích thước với nhánh chính (dùng identity hoặc khôi downampling đã khai báo), sau đó cộng kết quả này với đầu ra của nhánh chính.

Tiếp theo, bạn sẽ ghép các khối ResidualBlock lại để xây dựng toàn bộ kiến trúc ResNet cho bài toán *Weather Image Classification*.

### Code Exercise

```

1 class ResNet(nn.Module):
2     def __init__(self, residual_block, n_blocks_lst, n_classes):
3         super(ResNet, self).__init__()
4         self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3)
5         self.batch_norm1 = nn.BatchNorm2d(64)
6         self.relu = nn.ReLU()
7         self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
8         self.conv2 = self.create_layer(residual_block, 64, 64, n_blocks_lst[0], 1)
9         self.conv3 = self.create_layer(residual_block, 64, 128, n_blocks_lst[1], 2)
10        self.conv4 = self.create_layer(residual_block, 128, 256, n_blocks_lst[2], 2)
11        self.conv5 = self.create_layer(residual_block, 256, 512, n_blocks_lst[3], 2)
12        self.avgpool = nn.AdaptiveAvgPool2d(1)
13        self.flatten = nn.Flatten()
14        self.fc1 = nn.Linear(512, n_classes)
15
16    def create_layer(self, residual_block, in_channels, out_channels, n_blocks, stride):
17        :
18        blocks = []
19        first_block = residual_block(in_channels, out_channels, stride)
20        blocks.append(first_block)
21
22        for idx in range(1, n_blocks):
23            block = residual_block(out_channels, out_channels, stride=1)
24            blocks.append(block)
25
26        block_sequential = nn.Sequential(*blocks)
27
28        return block_sequential
29
30    def forward(self, x):
31        x = self.conv1(x)
32        x = self.***Your Code Here*** # (1)
33        x = self.***Your Code Here*** # (2)
34        x = self.***Your Code Here*** # (3)
35        x = self.***Your Code Here*** # (4)
36        x = self.conv3(x)
37        x = self.***Your Code Here*** # (5)
38        x = self.***Your Code Here*** # (6)
39        x = self.***Your Code Here*** # (7)
40        x = self.***Your Code Here*** # (8)
41
42        return x

```

Phần này, bạn **hoàn thiện hàm forward** của lớp **ResNet** để dữ liệu đi qua đầy đủ các thành phần của kiến trúc:

- (1) Áp dụng lớp *batch normalization* ở *stem* ngay sau tích chập đầu tiên.
- (2) Đưa tensor qua lớp *max pooling* để giảm kích thước không gian của feature map.
- (3) Áp dụng hàm kích hoạt *ReLU*.
- (4) Truyền kết quả qua “stage” residual đầu tiên (*conv2*) gồm nhiều *ResidualBlock*.
- (5) Sau khi đi qua *conv3*, tiếp tục cho tensor đi qua các “stage” residual kế tiếp (*conv4* và *conv5*) để tăng dần số kênh và giảm dần kích thước không gian.
- (6) Áp dụng *global average pooling* để gom mỗi feature map về một giá trị duy nhất trên mỗi kênh.
- (7) Làm phẳng (flatten) tensor thu được thành vector đặc trưng 1D cho từng mẫu trong batch.
- (8) Giữ nguyên đầu ra này làm đầu vào cho lớp *fc1* để sinh ra logits tương ứng với *n\_classes* lớp thời tiết.

Hoàn thành cả hai phần trên, bạn đã xây dựng xong backbone ResNet: từ *ResidualBlock* đơn lẻ tới mô hình ResNet đầy đủ cho bài toán phân loại ảnh thời tiết.

## Bước 5: Khởi tạo và huấn luyện mô hình ResNet

```

1 n_classes = len(list(classes.keys()))
2 device = "cuda" if torch.cuda.is_available() else "cpu"
3
4 model = ResNet(
5     residual_block=ResidualBlock,
6     n_blocks_lst=[2, 2, 2, 2],
7     n_classes=n_classes
8 ).to(device)
9
10 def fit(
11     model,
12     train_loader,
13     val_loader,
14     criterion,
15     optimizer,
16     device,
17     epochs
18 ):
19     train_losses = []

```

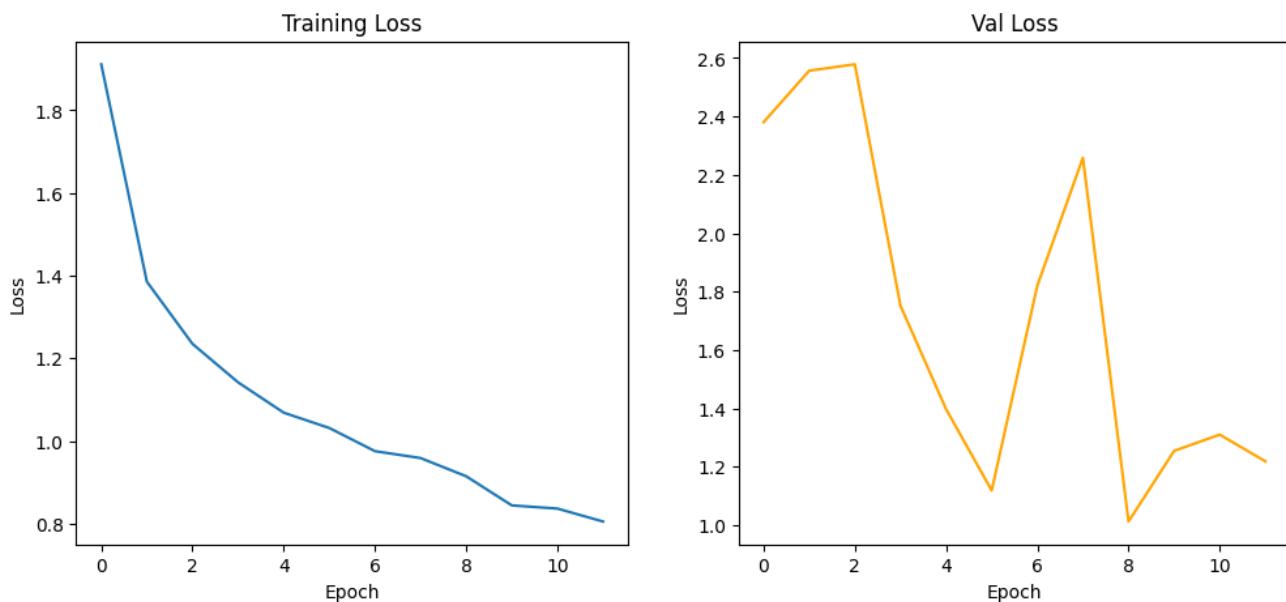
```

20     val_losses = []
21
22     for epoch in range(epochs):
23         batch_train_losses = []
24
25         # Progress bar cho train từng batch
26         model.train()
27         train_pbar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{epochs} - Training")
28
29         for inputs, labels in train_pbar:
30             inputs, labels = inputs.to(device), labels.to(device)
31
32             optimizer.zero_grad()
33             outputs = model(inputs)
34             loss = criterion(outputs, labels)
35             loss.backward()
36             optimizer.step()
37
38             batch_train_losses.append(loss.item())
39
40         train_loss = sum(batch_train_losses) / len(batch_train_losses)
41         train_losses.append(train_loss)
42
43         val_loss, val_acc = evaluate(model, val_loader, criterion, device)
44         val_losses.append(val_loss)
45
46         print(f"EPOCH {epoch+1}: Train loss: {train_loss:.4f} | Val loss: {val_loss:.4f}"
47               f" | Val acc: {val_acc:.4f}")
48
49     return train_losses, val_losses
50
51 lr = 1e-2
52 epochs = 12
53
54 criterion = nn.CrossEntropyLoss()
55 optimizer = torch.optim.SGD(
56     model.parameters(),
57     lr=lr
58 )
59 train_losses, val_losses = fit(
60     model,
61     train_loader,
62     val_loader,
63     criterion,
64     optimizer,
65     device,
66     epochs
67 )

```

Ở bước này, ta khởi tạo mô hình ResNet với số lớp residual tương ứng cấu hình [2, 2, 2, 2] (tương tự ResNet-18), đưa mô hình lên CPU/GPU phù hợp, định nghĩa hàm mất mát

CrossEntropyLoss và tối ưu hoá bằng SGD. Hàm `fit` thực hiện vòng lặp huấn luyện nhiều epoch: mỗi epoch mô hình được train trên `train_loader`, sau đó đánh giá trên `val_loader`, lưu lại `train_loss`, `val_loss` và in ra `val_acc` để theo dõi quá trình học của ResNet trên bài toán phân loại thời tiết.



Hình 9: Biểu đồ loss của quá trình training và validation.

## V. Bài toán Scenes Classification với mô hình DenseNet

Trong phần tiếp theo, chúng ta chuyển sang bài toán *Scenes Classification*: phân loại ảnh theo bối cảnh/quang cảnh (ví dụ: *buildings*, *forest*, *mountain*, *sea*, ...). Khác với bộ dữ liệu thời tiết, các lớp ở đây được phân biệt chủ yếu bởi cấu trúc không gian và bố cục cảnh vật. Bài tập lập trình sẽ hướng dẫn bạn xây dựng và huấn luyện một mô hình DenseNet cho bài toán này. Ở phần này ta chỉ giới thiệu các đoạn code quan trọng, không liệt kê toàn bộ pipeline phụ trợ tránh code quá dài, bạn có thể tham khảo toàn bộ code ở phần phụ lục.



Hình 10: Vài mẫu dữ liệu trong bài toán Scenes Classification.

### Bước 1: Import thư viện và chia dữ liệu

```

1 import os
2 import random
3 from pathlib import Path
4
5 import numpy as np
6 import pandas as pd
7 import matplotlib.pyplot as plt

```

```

8 import torch
9 import torch.nn as nn
10 from PIL import Image
11 from torch.utils.data import Dataset, DataLoader
12 from sklearn.model_selection import train_test_split
13 from tqdm import tqdm
14
15 root_dir = Path("scenes_classification")
16 train_dir = root_dir / "train"
17 val_dir = root_dir / "val"
18
19 seed = 0
20 val_size = 0.2
21 is_shuffle = True
22 X_train, X_val, y_train, y_val = train_test_split(
23     X_train,
24     y_train,
25     test_size=val_size,
26     random_state=seed,
27     shuffle=is_shuffle,
28 )

```

Trong bước này, ta chỉ trích các đoạn code quan trọng (chưa phải toàn bộ pipeline) để minh họa cách chuẩn bị dữ liệu cho bài toán *Scenes Classification*. Đầu tiên, ta import các thư viện cần thiết (xử lý đường dẫn với Path, nạp ảnh với PIL, chia dữ liệu với `train_test_split`, ...). Thư mục gốc `root_dir` được đặt là `scenes_classification` với hai nhánh con `train/` và `val/`. Cuối cùng, ta sử dụng `train_test_split` để tách một phần dữ liệu ảnh cảnh vật ban đầu thành tập `train` và `validation` với tỉ lệ `val_size = 0.2`, kèm theo `seed` cố định và tùy chọn `shuffle` để việc chia dữ liệu vừa ngẫu nhiên vừa có thể tái lập.

## Bước 2: Xây dựng `ScenesDataset` và `DataLoader`

```

1 class ScenesDataset(Dataset):
2     def __init__(self, X, y, transform=None):
3         self.transform = transform
4         self.img_paths = X
5         self.labels = y
6
7     def __len__(self):
8         return len(self.img_paths)
9
10    def __getitem__(self, idx):
11        img_path = self.img_paths[idx]
12        img = Image.open(img_path).convert("RGB")
13        if self.transform:
14            img = self.transform(img)
15        return img, self.labels[idx]
16
17    def transform(img, img_size=(224, 224)):

```

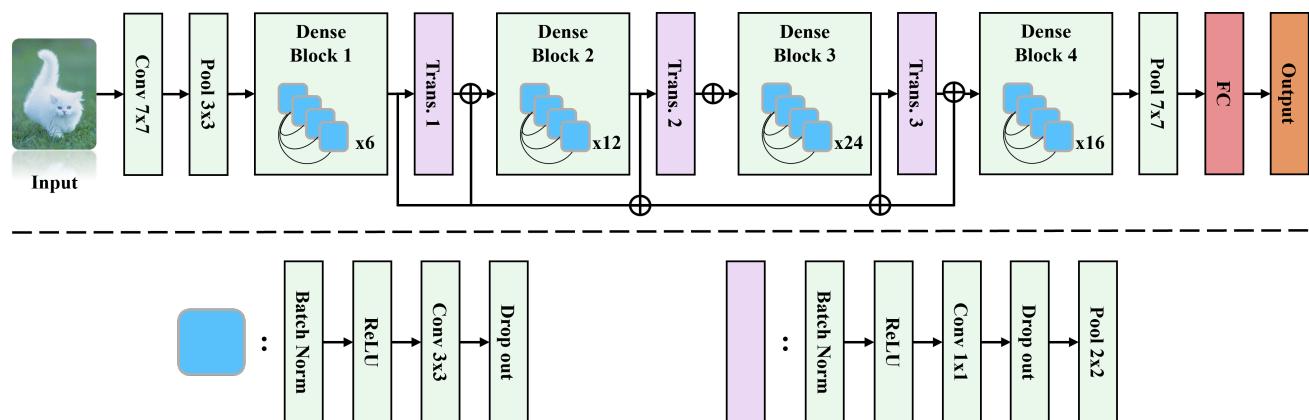
```

18     img = img.resize(img_size)
19     img = np.array(img)[..., :3]
20     tensor_img = torch.tensor(img).permute(2, 0, 1).float()
21     normalized_img = tensor_img / 255.0
22     return normalized_img
23
24 train_dataset = ScenesDataset(X_train, y_train, transform=transform)
25 val_dataset   = ScenesDataset(X_val,   y_val,   transform=transform)
26 test_dataset  = ScenesDataset(X_test,  y_test,  transform=transform)
27
28 train_batch_size = 128
29 test_batch_size  = 64
30
31 train_loader = DataLoader(train_dataset, batch_size=train_batch_size, shuffle=True)
32 val_loader   = DataLoader(val_dataset,   batch_size=test_batch_size,  shuffle=False)
33 test_loader  = DataLoader(test_dataset,  batch_size=test_batch_size,  shuffle=False)

```

Tương tự phần Weather Classification, ở đây ta **chỉ trích các đoạn code quan trọng** để xây dựng pipeline dữ liệu cho Scenes Classification. Lớp `ScenesDataset` gói các đường dẫn ảnh cảnh vật và nhãn tương ứng, đọc ảnh bằng `PIL`, chuyển sang RGB và áp dụng hàm `transform`. Hàm `transform` resize ảnh về kích thước chuẩn  $224 \times 224$ , chuyển sang tensor dạng ( $C, H, W$ ) và chuẩn hóa về khoảng  $[0, 1]$ . Sau đó, ta khởi tạo ba `Dataset` (train/val/test) và các `DataLoader` với batch size phù hợp, sẵn sàng dùng cho bước định nghĩa và huấn luyện mô hình CNN (DenseNet) ở các phần tiếp theo.

### Bước 3: Xây dựng mô hình DenseNet



Hình 11: Kiến trúc mô hình DenseNet

## Code Exercise

```
1 class BottleneckBlock(nn.Module):
2     def __init__(self, in_channels, growth_rate):
3         super().__init__()
4         self.bn1 = nn.BatchNorm2d(in_channels)
5         self.conv1 = nn.Conv2d(in_channels, 4 * growth_rate, kernel_size=1, bias=False)
6         self.bn2 = nn.BatchNorm2d(4 * growth_rate)
7         self.conv2 = nn.Conv2d(4 * growth_rate, growth_rate, kernel_size=3, padding=1,
8                             bias=False)
9         self.relu = nn.ReLU()
10
11    def forward(self, x):
12        res = ***Your Code Here***
13        x = self.bn1(x)
14        x = ***Your Code Here*** # (1)
15        x = ***Your Code Here*** # (2)
16        x = ***Your Code Here*** # (3)
17        x = ***Your Code Here*** # (4)
18        x = ***Your Code Here*** # (5)
19        return torch.cat([res, x], 1)
20
21 class DenseBlock(nn.Module):
22     def __init__(self, num_layers, in_channels, growth_rate):
23         super().__init__()
24         layers = []
25         for i in range(num_layers):
26             layers.append(BottleneckBlock(in_channels + i * growth_rate, growth_rate))
27         self.block = nn.Sequential(*layers)
28
29     def forward(self, x):
30         return self.block(x)
```

Ở phần đầu của bài tập DenseNet này, bạn cần **hoàn thiện hàm forward của BottleneckBlock**. Đây là đơn vị cơ bản trong *dense block*, có cấu trúc chuẩn của DenseNet: chuẩn hoá → kích hoạt → tích chập  $1 \times 1$  (giảm kênh) → chuẩn hoá → kích hoạt → tích chập  $3 \times 3$  (tạo thêm kênh mới), rồi *nối (concatenate)* với tensor đầu vào ban đầu.

Yêu cầu logic cho các vị trí \*\*\*Your Code Here\*\*\*:

- Ở dòng gán `res`, bạn cần lưu lại một bản copy của tensor đầu vào để sau cùng dùng làm nhánh “đặc trưng cũ” khi nối với đặc trưng mới.
- (1) Sau `bn1`, áp dụng hàm kích hoạt *ReLU* lên tensor đã được chuẩn hoá.
- (2) Đưa tensor này qua lớp tích chập  $1 \times 1$  để giảm số kênh xuống *bottleneck* (tỉ lệ liên quan đến `growth_rate`).
- (3) Chuẩn hoá đầu ra của tích chập  $1 \times 1$  bằng lớp `bn2`.
- (4) Tiếp tục áp dụng *ReLU* lần nữa sau chuẩn hoá thứ hai.
- (5) Đưa tensor qua lớp tích chập  $3 \times 3$  để sinh thêm các feature map mới với số kênh bằng `growth_rate`.

Dòng lệnh cuối cùng `torch.cat([res, x], 1)` thực hiện đúng tinh thần *dense connectivity*: nối (theo chiều kênh) đặc trưng ban đầu `res` với đặc trưng mới `x` của bottleneck block. Lớp

DenseBlock đã được xây dựng sẵn để xếp chồng nhiều BottleneckBlock; trong bước này ta chỉ tập trung vào việc *hiểu và hoàn thiện logic bên trong một bottleneck block*, các phần còn lại của kiến trúc DenseNet sẽ được sử dụng ở các bước tiếp theo.

Tiếp theo, bạn sẽ cần hoàn thiện code để ghép các DenseBlock (kèm transition layer) để xây dựng toàn bộ kiến trúc DenseNet cho bài toán Scenes Classification.

### Code Exercise

```

1 class DenseNet(nn.Module):
2     def __init__(self, num_blocks, growth_rate, num_classes):
3         super().__init__()
4         self.conv1 = nn.Conv2d(3, 2 * growth_rate, kernel_size=7, padding=3, stride=2,
5                             bias=False)
6         self.bn1 = nn.BatchNorm2d(2 * growth_rate)
7         self.pool1 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
8         self.dense_blocks = nn.ModuleList()
9         in_channels = 2 * growth_rate
10        for i, num_layers in enumerate(num_blocks):
11            self.dense_blocks.append(DenseBlock(num_layers, in_channels, growth_rate))
12            in_channels += num_layers * growth_rate
13            if i != len(num_blocks) - 1:
14                out_channels = in_channels // 2
15                transitions = nn.Sequential(
16                    nn.BatchNorm2d(in_channels),
17                    nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=False),
18                    nn.AvgPool2d(kernel_size=2, stride=2),
19                )
20                self.dense_blocks.append(transitions)
21                in_channels = out_channels
22        self.bn2 = nn.BatchNorm2d(in_channels)
23        self.pool2 = nn.AvgPool2d(kernel_size=7)
24        self.relu = nn.ReLU()
25        self.fc = nn.Linear(in_channels, num_classes)
26
27    def forward(self, x):
28        x = self.conv1(x)
29        x = ***Your Code Here*** # (1)
30        x = ***Your Code Here*** # (2)
31        x = ***Your Code Here*** # (3)
32        for block in self.dense_blocks:
33            x = block(x)
34            x = ***Your Code Here*** # (4)
35            x = ***Your Code Here*** # (5)
36            x = ***Your Code Here*** # (6)
37            x = x.view(x.size(0), -1)
38        return ***Your Code Here*** # (7)

```

Ở đoạn này, ta **chỉ trích các dòng code quan trọng** trong forward để bạn hoàn thiện kiến trúc DenseNet. Phần `__init__` đã xây dựng sẵn:

- Khối *stem*: tích chập đầu vào  $7 \times 7$ , batch normalization, max pooling.

- Một danh sách `dense_blocks` gồm các `DenseBlock` xen kẽ với các transition layer (BN + Conv  $1 \times 1$  + AvgPool).
- Phần cuối: batch normalization, global average pooling, hàm kích hoạt và một lớp fully-connected cho `num_classes`.

Yêu cầu logic cho từng vị trí **\*\*\*Your Code Here\*\*\*** trong `forward`:

- (1) Sau lớp tích chập đầu tiên, chuẩn hoá feature map bằng batch normalization tương ứng ở *stem*.
- (2) Áp dụng hàm kích hoạt phi tuyến (ReLU) lên kết quả đã chuẩn hoá.
- (3) Dưa tensor qua lớp max pooling ban đầu để giảm kích thước không gian trước khi đi vào các `DenseBlock`.
- (4) Sau khi tensor lần lượt đi qua toàn bộ các phần tử trong `self.dense_blocks` (bao gồm dense block và transition layer), chuẩn hoá feature map cuối cùng bằng batch normalization ở phần *tail*.
- (5) Áp dụng lại ReLU sau batch normalization thứ hai để tăng tính phi tuyến trước khi gom đặc trưng.
- (6) Áp dụng global average pooling (`pool2`) trên toàn bộ không gian  $H \times W$  để thu được một vector theo kênh.
- (7) Sau khi đã `view` về dạng (`batch_size, in_features`), đưa vector đặc trưng này qua lớp fully-connected cuối cùng để thu được logits tương ứng với `num_classes` lớp cảnh vật.

Hoàn thiện BottleneckBlock, DenseBlock và DenseNet ở bước này sẽ giúp bạn nắm được cách DenseNet triển khai *dense connectivity* kết hợp với bottleneck và transition layer trong một kiến trúc CNN sâu cho bài toán Scenes Classification.

## Bước 4: Khởi tạo và huấn luyện DenseNet cho Scenes Classification

```

1 n_classes = len(classes)
2 device = "cuda" if torch.cuda.is_available() else "cpu"
3 model = DenseNet([6, 12, 24, 16], growth_rate=32, num_classes=n_classes).to(device)
4
5 lr = 1e-2
6 epochs = 25
7 criterion = nn.CrossEntropyLoss()
8 optimizer = torch.optim.SGD(model.parameters(), lr=lr)
9
10 def fit(model, train_loader, val_loader, criterion, optimizer, device, epochs):
11     train_losses, val_losses = [], []
12     for epoch in range(epochs):

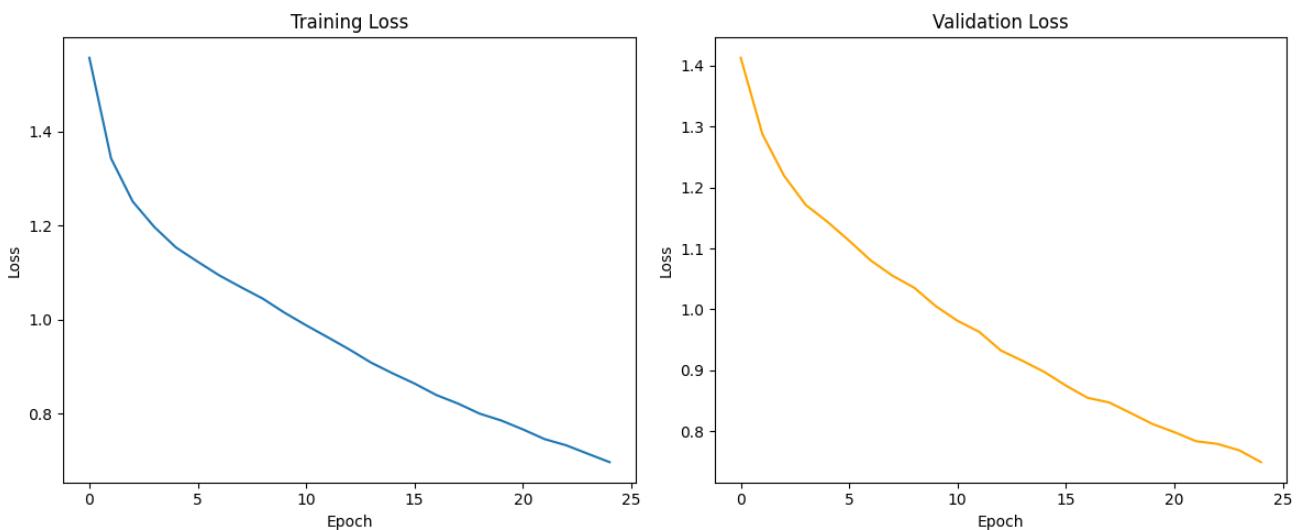
```

```

13     batch_train_losses = []
14     model.train()
15     train_pbar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{epochs} - Training")
16     for imgs, labels in train_pbar:
17         imgs = imgs.to(device)
18         labels = labels.to(device)
19         optimizer.zero_grad()
20         outputs = model(imgs)
21         loss = criterion(outputs, labels)
22         loss.backward()
23         optimizer.step()
24         batch_train_losses.append(loss.item())
25         train_pbar.set_postfix({"loss": loss.item()})
26     epoch_loss = sum(batch_train_losses) / len(batch_train_losses)
27     train_losses.append(epoch_loss)
28     val_loss, val_acc = evaluate(model, val_loader, criterion, device)
29     val_losses.append(val_loss)
30     print(
31         f"Epoch {epoch + 1}/{epochs} - "
32         f"train loss: {epoch_loss:.4f} - "
33         f"val loss: {val_loss:.4f} - "
34         f"val acc: {val_acc:.4f}"
35     )
36     return train_losses, val_losses
37
38 train_losses, val_losses = fit(
39     model,
40     train_loader,
41     val_loader,
42     criterion,
43     optimizer,
44     device,
45     epochs,
46 )

```

Ở bước này, ta khởi tạo mô hình DenseNet với cấu hình [6, 12, 24, 16] và `growth_rate = 32` (tương đương DenseNet-121 thu gọn cho Scenes Classification), đưa mô hình lên CPU/GPU, khai báo hàm mất mát `CrossEntropyLoss` và tối ưu hoá bằng SGD. Hàm `fit` thực hiện vòng lặp huấn luyện nhiều epoch: mỗi epoch mô hình được train trên `train_loader`, sau đó đánh giá trên `val_loader`, ghi lại `train_loss`, `val_loss` và `val_acc` để theo dõi hiệu năng của DenseNet trên bài toán phân loại cảnh.



Hình 12: Biểu đồ loss của quá trình training và validation.

## VII. Câu hỏi trắc nghiệm

1. (Lý thuyết – ResNet) Phát biểu nào sau đây mô tả đúng mục tiêu chính của *residual learning* trong ResNet?
  - (a) Giảm kích thước ảnh đầu vào bằng cách học trực tiếp ánh xạ  $H(x)$  có ít chiều hơn.
  - (b) Thay vì học trực tiếp ánh xạ  $H(x)$ , khôi học phần dư  $F(x)$  sao cho  $H(x) = F(x) + x$ , giúp việc tối ưu hoá ổn định hơn khi mạng sâu.
  - (c) Thay thế hoàn toàn các tầng Convolution bằng các tầng Fully-Connected để tránh vanishing gradient.
  - (d) Không cho phép dùng hàm kích hoạt phi tuyến ở giữa các tầng để giữ nguyên identity mapping.
2. (Lý thuyết – ResNet) Về *skip connection* trong ResNet, nhận định nào sau đây là **đúng**?
  - (a) *Identity shortcut* được dùng khi kích thước không gian và số kênh của đầu vào và đầu ra block giống nhau, còn *projection shortcut* dùng Conv  $1 \times 1$  để điều chỉnh kích thước khi chúng khác nhau.
  - (b) *Projection shortcut* luôn được dùng cho mọi block, kể cả khi kích thước không gian và số kênh đã trùng nhau.
  - (c) *Skip connection* chỉ có tác dụng giảm số tham số của mô hình, không liên quan đến dòng chảy gradient.
  - (d) ResNet không sử dụng *skip connection*; khái niệm này chỉ xuất hiện trong DenseNet.
3. (Lý thuyết – DenseNet) Cơ chế *dense connectivity* trong DenseNet mang ý nghĩa gì về mặt luồng thông tin?
  - (a) Mỗi lớp chỉ nhận đầu vào từ lớp ngay trước nó, nhưng gửi đầu ra đến tất cả các lớp phía sau.
  - (b) Mỗi lớp nhận đầu vào là trung bình cộng của tất cả các feature map trước đó, thay vì phép nối (concatenation).
  - (c) Mỗi lớp trong một *dense block* nhận đầu vào là phép nối theo kênh của tất cả feature map từ các lớp trước đó trong cùng block, giúp tái sử dụng đặc trưng và cải thiện gradient flow.
  - (d) Mỗi lớp chỉ kết nối tới lớp cuối cùng của block, không kết nối tới các lớp trung gian.
4. (Lý thuyết – DenseNet) Mục đích chính của *bottleneck layer* và *transition layer* trong DenseNet là gì?
  - (a) Làm mô hình sâu hơn nhưng không thay đổi số kênh và kích thước không gian.
  - (b) Tăng số kênh sau mỗi block để mô hình học được nhiều đặc trưng hơn, không quan tâm đến chi phí tính toán.
  - (c) *Bottleneck layer* dùng Conv  $1 \times 1$  để giảm số kênh trước Conv  $3 \times 3$ , còn *transition layer* dùng BN-ReLU-Conv  $1 \times 1$ -Pooling để giảm kích thước không gian và (tùy chọn) giảm số kênh, giúp kiểm soát số tham số và FLOPs.

- (d) Chỉ dùng để thay thế các lớp Fully-Connected, không ảnh hưởng đến số kênh hay kích thước không gian.
5. (Tính toán – ResNet) Số tham số trong một *basic residual block*. Xét một *basic block* của ResNet gồm hai lớp Conv  $3 \times 3$  liên tiếp, không dùng bias, kích thước kernel là  $K = 3$ , số kênh vào/ra của cả hai Conv đều bằng  $C$ . Số tham số của một Conv 2D (bỏ qua BatchNorm) được tính bởi:

$$\text{Param}_{\text{conv}} = K^2 \cdot C_{\text{in}} \cdot C_{\text{out}}.$$

Với  $C = 64$ , tổng số tham số của *cả block* (chỉ tính 2 Conv  $3 \times 3$ ) là:

- (a) 18 432  
 (b) 36 864  
 (c) 73 728  
 (d) 147 456
6. (Tính toán – ResNet) Kích thước feature map sau *stem* của ResNet cho ảnh đầu vào  $224 \times 224$ . Giả sử ta dùng *stem* giống ResNet:
- Conv đầu vào: kernel  $7 \times 7$ , stride  $S = 2$ , padding  $P = 3$ .
  - Sau đó là MaxPool: kernel  $3 \times 3$ , stride  $S = 2$ , padding  $P = 1$ .

Với chiều cao/chiều rộng đầu vào  $H = W = 224$ , kích thước không gian sau mỗi phép toán được tính bởi:

$$H_{\text{out}} = \left\lfloor \frac{H + 2P - K}{S} \right\rfloor + 1.$$

Kích thước không gian của feature map sau khi qua **Conv  $7 \times 7$  và MaxPool  $3 \times 3$**  là:

- (a)  $112 \times 112$   
 (b)  $56 \times 56$   
 (c)  $64 \times 64$   
 (d)  $28 \times 28$
7. (Tính toán – ResNet) Số lượng Conv layer trong một ResNet dạng  $[2, 2, 2, 2]$ . Giả sử ta xây dựng ResNet với:
- 1 Conv đầu vào (trong *stem*).
  - 4 *stage* residual liên tiếp, mỗi *stage* có số block lần lượt là  $[2, 2, 2, 2]$ .
  - Mỗi *basic block* gồm đúng 2 lớp Conv.

Tổng số lớp Conv trong mạng (bỏ qua các lớp FC, BN, Pooling, ...) được tính theo công thức:

$$N_{\text{conv}} = 1 + 2 \sum_{i=1}^4 n_i,$$

trong đó  $n_i$  là số block tại stage thứ  $i$ . Giá trị  $N_{\text{conv}}$  là:

- (a) 15  
 (b) 17  
 (c) 18  
 (d) 34
8. (Tính toán – DenseNet) Số kênh sau một *dense block*. Giả sử đầu vào của một *dense block* có  $k_0$  kênh, *growth rate* là  $k$ , số lớp trong block là  $L$ . Theo định nghĩa của DenseNet, số kênh đầu ra sau block được tính bởi:
- $$C_{\text{out}} = k_0 + L \cdot k.$$
- Cho  $k_0 = 64$ ,  $k = 32$  và  $L = 6$ , giá trị  $C_{\text{out}}$  là:
- (a) 160  
 (b) 192  
 (c) 256  
 (d) 320
9. (Tính toán – DenseNet) Ảnh hưởng của *transition layer* với *compression factor*. Giả sử một *dense block* cho đầu ra có  $C_{\text{in}}$  kênh. *Transition layer* phía sau block sử dụng *compression factor*  $\theta \in (0, 1]$ , số kênh sau transition được tính bởi:
- $$C_{\text{out}} = \lfloor \theta \cdot C_{\text{in}} \rfloor.$$
- Nếu  $C_{\text{in}} = 256$  và  $\theta = 0.5$ , thì số kênh sau *transition layer* là:
- (a) 64  
 (b) 128  
 (c) 192  
 (d) 256
10. (Tính toán – DenseNet) Số kênh cuối cùng trong DenseNet cấu hình  $[6, 12, 24, 16]$  với *growth rate*  $k = 32$ . Giả sử:

- Sau Conv đầu vào, số kênh ban đầu là  $C_0 = 2k = 64$ .
- Mỗi *dense block* thứ  $i$  có  $L_i$  lớp, số kênh tăng theo công thức:

$$C_{\text{out}}^{(i)} = C_{\text{in}}^{(i)} + L_i \cdot k.$$

- Giữa các block có *transition layer* với *compression factor*  $\theta = 0.5$ , nên sau *transition*:

$$C_{\text{in}}^{(i+1)} = \lfloor \theta \cdot C_{\text{out}}^{(i)} \rfloor.$$

- Cấu hình số lớp theo từng block là  $[L_1, L_2, L_3, L_4] = [6, 12, 24, 16]$  và không có *transition* sau block cuối cùng.

Số kênh đầu vào của lớp Fully-Connected cuối cùng (tức số kênh sau block thứ 4) là:

- (a) 512
- (b) 768
- (c) 1024
- (d) 2048

# Phụ lục

1. **Hint:** Các file code gợi ý và dữ liệu nếu có được lưu trong thư mục có thể được tải [tại đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp tại [đây](#). Tất cả câu hỏi sẽ được trả lời trong vòng tối đa 4 giờ.

## AIO\_QAs-Verified

🔒 Nhóm Riêng tư · 1,4K thành viên



Hình 13: Hình ảnh group facebook AIO Q&A

### 4. Rubric:

Phần	Kiến Thức	Đánh Giá
1	- Xây dựng và huấn luyện mô hình ResNet cho bài toán Weather Image Classification.	<ul style="list-style-type: none"> <li>- Đánh giá được hiệu quả của ResNet trong phân loại ảnh thời tiết.</li> <li>- Hiểu quy trình end-to-end từ chuẩn bị dữ liệu đến train/validation với ResNet.</li> </ul>
2	- Xây dựng và huấn luyện mô hình DenseNet cho bài toán Scenes Classification.	<ul style="list-style-type: none"> <li>- Nhận diện được vai trò của dense connectivity trong tái sử dụng đặc trưng.</li> <li>- Vận dụng kiến thức DenseNet để cải thiện phân loại cảnh trong thực nghiệm.</li> </ul>