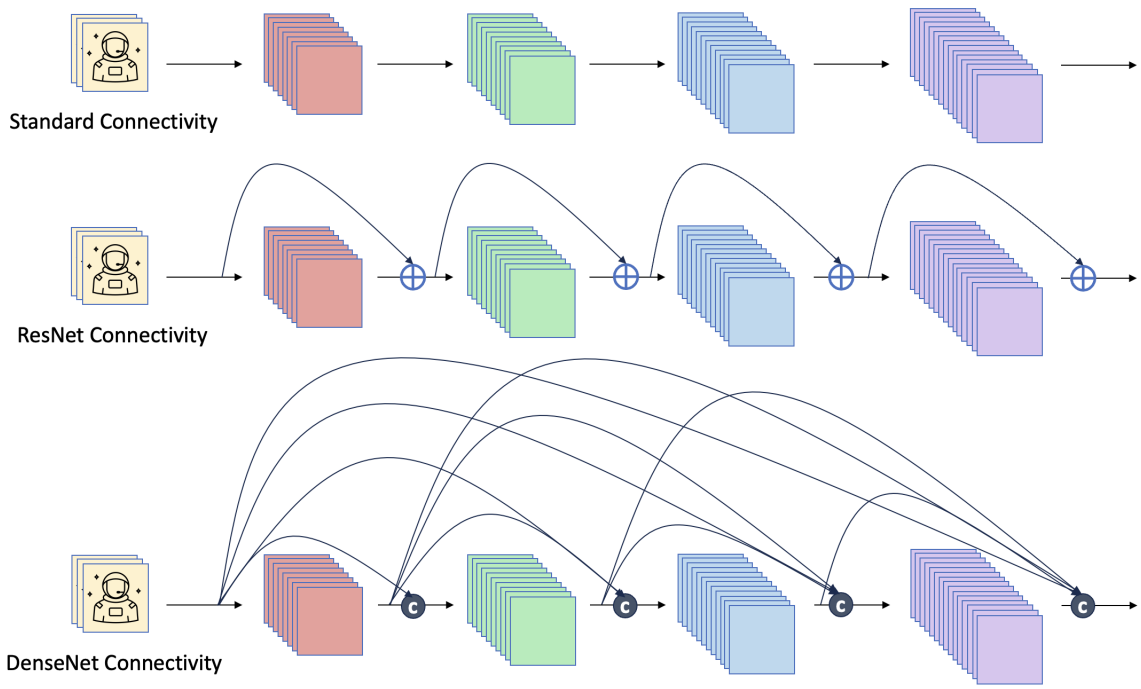
AI VIET NAM – AI COURSE 2024

Exercise: Advanced CNN Architecture

Dinh-Thang Duong, Quang-Vinh Dinh

I. Giới thiệu

Convolutional Neural Networks (CNNs) là một nhánh trong deep neural networks, thường được dùng để giải quyết các bài toán liên quan đến dữ liệu ảnh. Các mạng CNNs với đặc điểm là các lớp tích chập (convolutional layers) để trích xuất đặc trưng từ ảnh, lớp pooling dùng để giảm kích thước ảnh nhằm tăng hiệu suất tính toán và các lớp fully-connected thực hiện phân loại dựa trên các đặc trưng đã trích xuất.



Hình 1: Minh họa một số kiểu kết nối giữa các lớp trong CNN.

Trong bài tập này ở phần lập trình, chúng ta sẽ thực hành cài đặt từ đầu quá trình xây dựng hai mô hình phân loại ảnh sử dụng kiến trúc ResNet và DenseNet, áp dụng vào giải quyết hai bài toán phân loại ảnh đa lớp là Weather Image Classification và Scenes Classification. Đồng thời, ôn tập một số lý thuyết về CNNs thông qua bài tập trắc nghiệm.

II. Bài tập

A. Phần lập trình

- **Weather Classification**

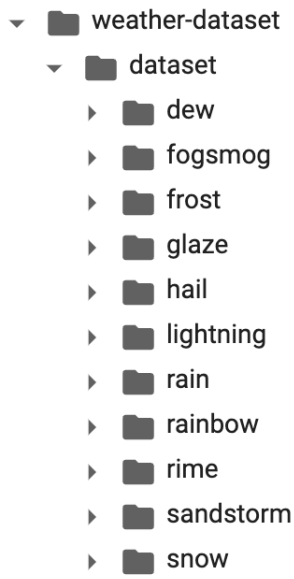
1. **Tải bộ dữ liệu:** Các bạn tải bộ dữ liệu **img_cls_weather_dataset.zip** trong đường dẫn thuộc mục Datasets tại phần III.
2. **Import các thư viện cần thiết:** Trong bài tập này, chúng ta sẽ sử dụng thư viện PyTorch để xây dựng và huấn luyện mô hình deep learning. Thêm vào đó, vì làm việc liên quan đến dữ liệu ảnh, chúng ta sẽ sử dụng thư viện PIL để xử lý:

```
import torch
import torch.nn as nn
import os
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from PIL import Image
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
```

3. **Cố định giá trị ngẫu nhiên:** Để có thể tái tạo lại cùng một kết quả mô hình, chúng ta sẽ cố định cùng một giá trị ngẫu nhiên (seed) cho các thư viện có chứa các hàm tạo giá trị ngẫu nhiên:

```
def set_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

seed = 59
set_seed(seed)
```

4. **Đọc dữ liệu:** Sau khi tải và giải nén bộ dữ liệu, chúng ta sẽ được một thư mục chứa dữ liệu như sau:



Hình 2: Cấu trúc cây thư mục trong bộ dữ liệu ảnh thời tiết.

Để thuận tiện trong việc xây dựng PyTorch datasets, chúng ta sẽ ghi nhận thông tin về các classes, đường dẫn đến tất cả các ảnh cũng như label tương ứng như sau. Nhận thấy tên của các folder con trong thư mục weather-dataset/dataset cũng là tên class. Vì vậy, chúng ta sẽ đọc tên các folder này và đưa vào một dictionary như sau:

```
root_dir = 'weather-dataset/dataset'
img_paths = []
labels = []
classes = {
    label_idx: class_name \
        for label_idx, class_name in enumerate(
            sorted(os.listdir(root_dir))
        )
}
```

Sau đó, ta đọc toàn bộ đường dẫn của các ảnh trong bộ dữ liệu cũng như label tương ứng:

```
img_paths = []
labels = []
for label_idx, class_name in classes.items():
    class_dir = os.path.join(root_dir, class_name)
    for img_filename in os.listdir(class_dir):
        img_path = os.path.join(class_dir, img_filename)
        img_paths.append(img_path)
        labels.append(label_idx)
```

5. **Chia bộ dữ liệu train, val, test:** Với danh sách đường dẫn ảnh và label, chúng ta sẽ chia thành ba bộ dữ liệu train, val, test sử dụng hàm train_test_split() của thư viện scikit-learn như sau:

```
val_size = 0.2
test_size = 0.125
is_shuffle = True
```

```
X_train, X_val, y_train, y_val = train_test_split(
    img_paths, labels,
    test_size=val_size,
    random_state=seed,
    shuffle=is_shuffle
)

X_train, X_test, y_train, y_test = train_test_split(
    X_train, y_train,
    test_size=val_size,
    random_state=seed,
    shuffle=is_shuffle
)
```

6. **Xây dựng class pytorch datasets:** Chúng ta xây dựng class datasets cho bộ dữ liệu weather như sau:

```
class WeatherDataset(Dataset):
    def __init__(
        self,
        X, y,
        transform=None
    ):
        self.transform = transform
        self.img_paths = X
        self.labels = y

    def __len__(self):
        return len(self.img_paths)

    def __getitem__(self, idx):
        img_path = self.img_paths[idx]
        img = Image.open(img_path).convert("RGB")

        if self.transform:
            img = self.transform(img)

        return img, self.labels[idx]
```

7. **Xây dựng hàm tiền xử lý ảnh (transform):** Để đảm bảo dữ liệu ảnh đầu vào được đồng bộ về kích thước và giá trị, chúng ta tự định nghĩa hàm `transform` để tiền xử lý ảnh đầu vào như sau (không sử dụng thư viện `torchvision.transforms`):

```
def transform(img, img_size=(224, 224)):
    img = img.resize(img_size)
    img = np.array(img)[..., :3]
    img = torch.tensor(img).permute(2, 0, 1).float()
    normalized_img = img / 255.0

    return normalized_img
```

   Các kỹ thuật được áp dụng: resize ảnh, đổi về tensor và chuẩn hóa giá trị pixel về khoảng (0, 1).

8. **Khai báo datasets object cho ba bộ train, val, test:** Với class WeatherDataset và hàm chuẩn hóa ảnh, ta tạo ba object datasets tương ứng như sau:

```
train_dataset = WeatherDataset(
    X_train, y_train,
    transform=transform
)
val_dataset = WeatherDataset(
    X_val, y_val,
    transform=transform
)
test_dataset = WeatherDataset(
    X_test, y_test,
    transform=transform
)
```

9. **Khai báo dataloader:** Với ba object datasets trên, ta khai báo giá trị batch size và tạo dataloader như sau:

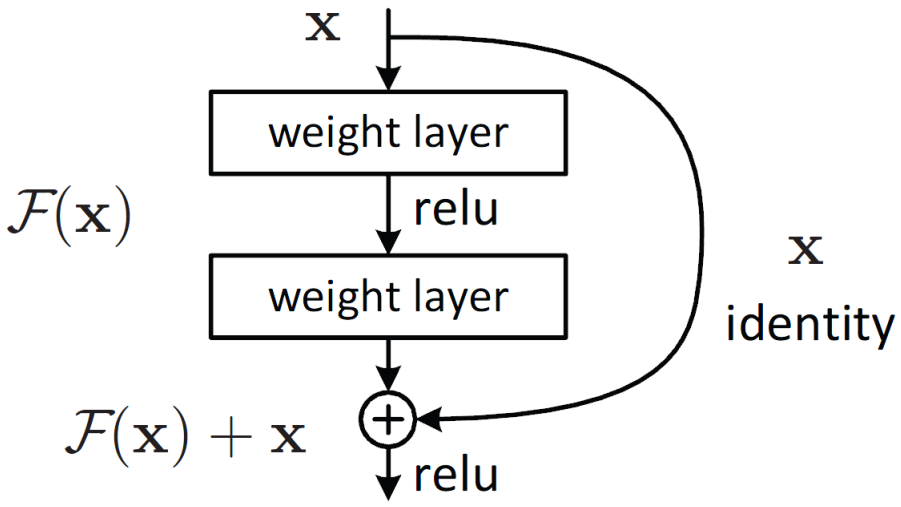
```
train_batch_size = 512
test_batch_size = 8

train_loader = DataLoader(
    train_dataset,
    batch_size=train_batch_size,
    shuffle=True
)
val_loader = DataLoader(
    val_dataset,
    batch_size=test_batch_size,
    shuffle=False
)
test_loader = DataLoader(
    test_dataset,
    batch_size=test_batch_size,
    shuffle=False
)
```

10. **Xây dựng model:** Trong phần này, chúng ta sẽ xây dựng class cho model deep learning với kiến trúc ResNet. Thông tin tổng quan về kiến trúc ResNet được thể hiện ở bảng sau:

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Hình 3: Thông tin chi tiết của từng layer trong kiến trúc ResNet. Nguồn ảnh: link.

Đầu tiên, chúng ta sẽ xây dựng class Residual Block, đây là một thành phần đặc biệt của kiến trúc ResNet so với các mạng CNNs khác, có mô phỏng như hình sau:



Hình 4: Minh họa residual connection. Nguồn ảnh: link.

Trong PyTorch, ta triển khai Residual Block như sau:

```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size
    =3, stride=stride, padding=1)
        self.batch_norm1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels,
    kernel_size=3, stride=1, padding=1)
        self.batch_norm2 = nn.BatchNorm2d(out_channels)

        self.downsample = nn.Sequential()
```

```
        if stride != 1 or in_channels != out_channels:
            self.downsample = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride),
                nn.BatchNorm2d(out_channels)
            )
        self.relu = nn.ReLU()

    def forward(self, x):
        shortcut = x.clone()
        x = self.conv1(x)
        x = self.batch_norm1(x)
        x = self.relu(x)
        x = self.conv2(x)
        x = self.batch_norm2(x)
        x += self.downsample(shortcut)
        x = self.relu(x)

        return x
```

Với ResidualBlock, ta triển khai toàn bộ kiến trúc ResNet như sau:

```
class ResNet(nn.Module):
    def __init__(self, residual_block, n_blocks_lst, n_classes):
        super(ResNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3)
        self.batch_norm1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.conv2 = self.create_layer(residual_block, 64, 64, n_blocks_lst[0], 1)
        self.conv3 = self.create_layer(residual_block, 64, 128, n_blocks_lst[1], 2)
        self.conv4 = self.create_layer(residual_block, 128, 256, n_blocks_lst[2], 2)
        self.conv5 = self.create_layer(residual_block, 256, 512, n_blocks_lst[3], 2)
        self.avgpool = nn.AdaptiveAvgPool2d(1)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(512, n_classes)

    def create_layer(self, residual_block, in_channels, out_channels, n_blocks, stride):
        blocks = []
        first_block = residual_block(in_channels, out_channels, stride)
        blocks.append(first_block)

        for idx in range(1, n_blocks):
            block = residual_block(out_channels, out_channels, stride)
            blocks.append(block)

        block_sequential = nn.Sequential(*blocks)
```

```
        return block_sequential

    def forward(self, x):
        x = self.conv1(x)
        x = self.batch_norm1(x)
        x = self.maxpool(x)
        x = self.relu(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)
        x = self.avgpool(x)
        x = self.flatten(x)
        x = self.fc1(x)

        return x
```

Cuối cùng, khai báo model ResNet bằng đoạn code sau:

```
n_classes = len(list(classes.keys()))
device = 'cuda' if torch.cuda.is_available() else 'cpu'

model = ResNet(
    residual_block=ResidualBlock,
    n_blocks_lst=[2, 2, 2, 2],
    n_classes=n_classes
).to(device)
```

11. **Xây dựng hàm đánh giá model:** Ta xây dựng hàm đánh giá model với đầu vào là model, bộ dữ liệu đánh giá và hàm loss. Hàm này sẽ trả về giá trị loss và accuracy của model trên tập dữ liệu đầu vào:

```
def evaluate(model, dataloader, criterion, device):
    model.eval()
    correct = 0
    total = 0
    losses = []
    with torch.no_grad():
        for inputs, labels in dataloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            losses.append(loss.item())
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    loss = sum(losses) / len(losses)
    acc = correct / total

    return loss, acc
```

12. **Xây dựng hàm huấn luyện model:** Ta triển khai xây dựng hàm huấn luyện mô hình như sau:

```python
def fit(
    model,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    device,
    epochs
):
    train_losses = []
    val_losses = []

    for epoch in range(epochs):
        batch_train_losses = []

        model.train()
        for idx, (inputs, labels) in enumerate(train_loader):
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            batch_train_losses.append(loss.item())

        train_loss = sum(batch_train_losses) / len(batch_train_losses)
        train_losses.append(train_loss)

        val_loss, val_acc = evaluate(
            model, val_loader,
            criterion, device
        )
        val_losses.append(val_loss)

        print(f'EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tVal loss: {val_loss:.4f}')

    return train_losses, val_losses
```

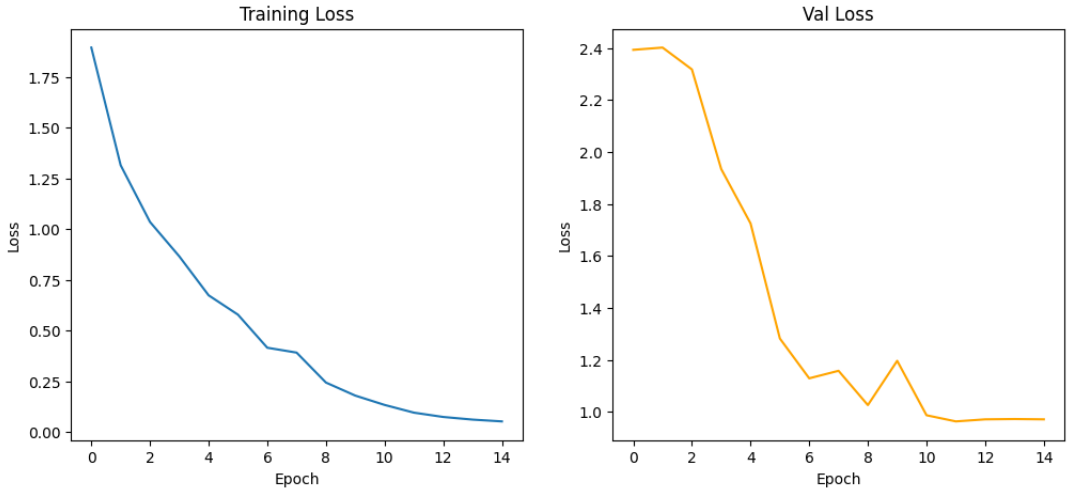
13. **Khai báo hàm loss và thuật toán tối ưu hóa:** Với bài toán phân loại ảnh, ta sử dụng hàm loss `CrossEntropyLoss` và thuật toán tối ưu hóa `Stochastic Gradient Descent (SGD)`. Ngoài ra, ta cũng khai báo giá trị learning rate và số epochs:

```python
lr = 1e-2
epochs = 25

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(
    model.parameters(),
    lr=lr
)
```

14. **Thực hiện huấn luyện:** Với tất cả các tham số đầu vào đã sẵn sàng, ta gọi hàm fit() để bắt đầu quá trình huấn luyện mô hình ResNet:

```
train_losses, val_losses = fit(
    model,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    device,
    epochs
)
```

Với danh sách giá trị loss qua từng epoch trên tập train và val, ta có thể trực quan hóa như sau:



Hình 5: Trực quan hóa kết quả huấn luyện của mô hình trên tập train và tập val theo giá trị loss.

15. **Đánh giá mô hình:** Ta gọi hàm evaluate() để đánh giá performance của model trên hai tập val và test như sau:

```
val_loss, val_acc = evaluate(
    model,
    val_loader,
    criterion,
    device
)
test_loss, test_acc = evaluate(
    model,
    test_loader,
    criterion,
    device
)

print('Evaluation on val/test dataset')
print('Val accuracy: ', val_acc)
print('Test accuracy: ', test_acc)
```

- **Scenes Classification**

1. **Tải bộ dữ liệu:** Các bạn tải bộ dữ liệu **img_cls_scenes_classification.zip** trong đường dẫn thuộc mục Datasets tại phần III.
2. **Import các thư viện cần thiết:**

```
import torch
import torch.nn as nn
import os
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from PIL import Image
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
```

3. **Cố định giá trị ngẫu nhiên:**

```
def set_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

seed = 59
set_seed(seed)
```

4. **Đọc dữ liệu:** Sau khi tải và giải nén bộ dữ liệu, chúng ta sẽ được một thư mục chứa dữ liệu như sau:

Hình 6: Cấu trúc cây thư mục trong bộ dữ liệu ảnh phong cảnh.

So với bộ dữ liệu ở bài trước, ta đã được chia sẵn hai bộ dữ liệu train và val. Tuy nhiên, để đồng bộ, chúng ta sẽ vẫn đi theo hướng code cũ. Đầu tiên, ta vẫn đọc danh sách classes như sau:

```
root_dir = 'scenes_classification'
train_dir = os.path.join(root_dir, 'train')
test_dir = os.path.join(root_dir, 'val')

classes = {
    label_idx: class_name \
        for label_idx, class_name in enumerate(
            sorted(os.listdir(train_dir))
        )
}
```

Tiếp đến, ta đọc lên toàn bộ các đường dẫn ảnh cũng như label tương ứng. Tuy nhiên, ta sẽ coi thư mục val là tập test của bộ dữ liệu và sẽ khai báo danh sách riêng cho bộ này. Như vậy, ta có code như sau:

```
X_train = []
y_train = []
X_test = []
y_test = []

for dataset_path in [train_dir, test_dir]:
    for label_idx, class_name in classes.items():
        class_dir = os.path.join(dataset_path, class_name)
        for img_filename in os.listdir(class_dir):
            img_path = os.path.join(class_dir, img_filename)
            if 'train' in dataset_path:
                X_train.append(img_path)
                y_train.append(label_idx)
            else:
                X_test.append(img_path)
                y_test.append(label_idx)
```

5. **Chia bộ dữ liệu train, val, test:** Vì đã có sẵn bộ dữ liệu train và test, ta chỉ việc chia thêm cho tập val từ tập train như sau:

```
seed = 0
val_size = 0.2
is_shuffle = True

X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train,
    test_size=val_size,
    random_state=seed,
    shuffle=is_shuffle
)
```

6. **Xây dựng class pytorch datasets:** Tương tự ở bài trước, ta xây dựng pytorch dataset cho bộ dữ liệu Scenes như sau:

```
class ScenesDataset(Dataset):
    def __init__(
        self,
```

```
        X, y,
        transform=None
    ):
        self.transform = transform
        self.img_paths = X
        self.labels = y

    def __len__(self):
        return len(self.img_paths)

    def __getitem__(self, idx):
        img_path = self.img_paths[idx]
        img = Image.open(img_path).convert("RGB")

        if self.transform:
            img = self.transform(img)

        return img, self.labels[idx]
```

7. **Xây dựng hàm tiền xử lý ảnh (transforms):** Tương tự như bài trước, ta xây dựng hàm tiền xử lý ảnh như sau:

```
def transform(img, img_size=(224, 224)):
    img = img.resize(img_size)
    img = np.array(img)[..., :3]
    img = torch.tensor(img).permute(2, 0, 1).float()
    normalized_img = img / 255.0

    return normalized_img
```

8. **Khai báo datasets object cho ba bộ train, val, test:** Tương tự như bài trên, ta khai báo ba object datasets tương ứng cho ba bộ dữ liệu train, val, test như sau:

```
train_dataset = ScenesDataset(
    X_train, y_train,
    transform=transform
)
val_dataset = ScenesDataset(
    X_val, y_val,
    transform=transform
)
test_dataset = ScenesDataset(
    X_test, y_test,
    transform=transform
)
```

9. **Khai báo dataloader:** Lưu ý, với colab, các bạn nên cài đặt train batch size = 64 để có thể train được bài này.

```
train_batch_size = 64
test_batch_size = 8

train_loader = DataLoader(
    train_dataset,
    batch_size=train_batch_size,
    shuffle=True
```

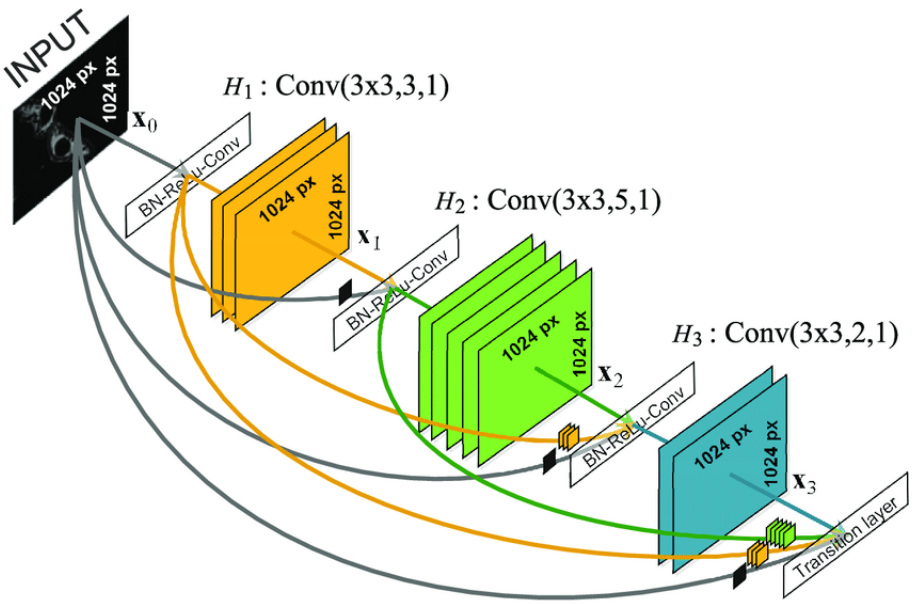
```
)
val_loader = DataLoader(
    val_dataset,
    batch_size=test_batch_size,
    shuffle=False
)
test_loader = DataLoader(
    test_dataset,
    batch_size=test_batch_size,
    shuffle=False
)
```

10. **Xây dựng model:** Trong phần này, chúng ta sẽ xây dựng class cho model deep learning với kiến trúc DenseNet. Thông tin tổng quan về kiến trúc DenseNet được thể hiện ở bảng sau:

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Hình 7: Thông tin chi tiết của từng layer trong kiến trúc DenseNet. Nguồn ảnh: link.

Đầu tiên, chúng ta sẽ xây dựng class Dense Block, có ảnh mô phỏng như hình dưới đây:



Hình 8: Ảnh mô phỏng kiến trúc mạng DenseNet. Nguồn ảnh: link.

```python
class BottleneckBlock(nn.Module):
    def __init__(self, in_channels, growth_rate):
        super(BottleneckBlock, self).__init__()
        self.bn1 = nn.BatchNorm2d(in_channels)
        self.conv1 = nn.Conv2d(in_channels, 4 * growth_rate, kernel_size=1, bias=False)
        self.bn2 = nn.BatchNorm2d(4 * growth_rate)
        self.conv2 = nn.Conv2d(4 * growth_rate, growth_rate, kernel_size=3, padding=1, bias=False)
        self.relu = nn.ReLU()

    def forward(self, x):
        res = x.clone().detach()
        x = self.bn1(x)
        x = self.relu(x)
        x = self.conv1(x)
        x = self.bn2(x)
        x = self.relu(x)
        x = self.conv2(x)
        x = torch.cat([res, x], 1)

        return x

class DenseBlock(nn.Module):
    def __init__(self, num_layers, in_channels, growth_rate):
        super(DenseBlock, self).__init__()
        layers = []
        for i in range(num_layers):
            layers.append(BottleneckBlock(in_channels + i * growth_rate, growth_rate))
        self.block = nn.Sequential(*layers)

    def forward(self, x):
```

```
        return self.block(x)
```

Với DenseBlock, ta triển khai toàn bộ kiến trúc DenseNet như sau:

```
class DenseNet(nn.Module):
    def __init__(self, num_blocks, growth_rate, num_classes):
        super(DenseNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 2 * growth_rate, kernel_size=7, padding=3, stride=2, bias=False)
        self.bn1 = nn.BatchNorm2d(2 * growth_rate)
        self.pool1 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.dense_blocks = nn.ModuleList()
        in_channels = 2 * growth_rate
        for i, num_layers in enumerate(num_blocks):
            self.dense_blocks.append(DenseBlock(num_layers, in_channels, growth_rate))
            in_channels += num_layers * growth_rate
            if i != len(num_blocks) - 1:
                out_channels = in_channels // 2
                self.dense_blocks.append(nn.Sequential(
                    nn.BatchNorm2d(in_channels),
                    nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=False),
                    nn.AvgPool2d(kernel_size=2, stride=2)
                ))
                in_channels = out_channels

        self.bn2 = nn.BatchNorm2d(in_channels)
        self.pool2 = nn.AvgPool2d(kernel_size=7)
        self.relu = nn.ReLU()
        self.fc = nn.Linear(in_channels, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.pool1(x)

        for block in self.dense_blocks:
            x = block(x)

        x = self.bn2(x)
        x = self.relu(x)
        x = self.pool2(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)

        return x
```

Cuối cùng, khai báo model DenseNet bằng đoạn code sau (ở đây ta sẽ sử dụng phiên bản DenseNet-121):

```
n_classes = len(list(classes.keys()))
device = 'cuda' if torch.cuda.is_available() else 'cpu'

model = DenseNet(
```

```
    [6, 12, 24, 16],
    growth_rate=32,
    num_classes=n_classes
).to(device)
```

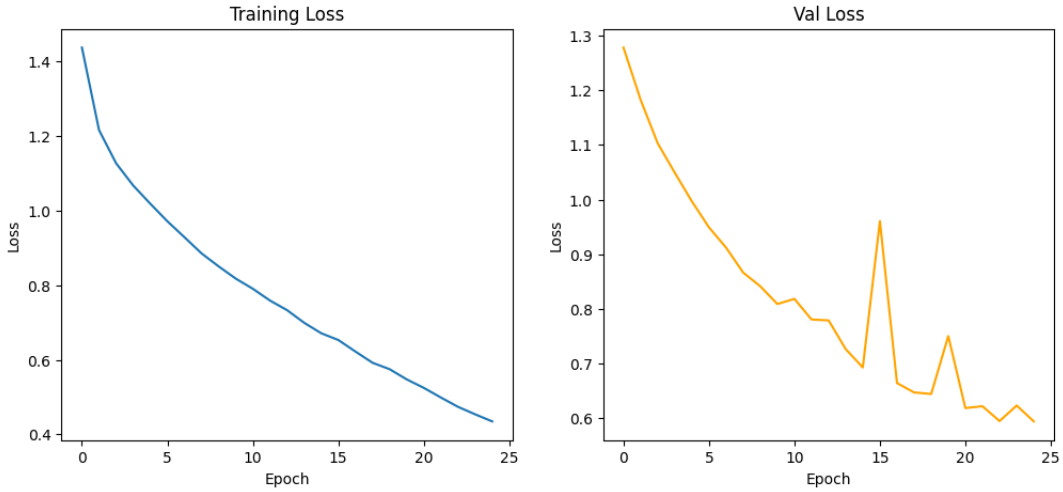
11. **Khai báo hàm loss và thuật toán huấn luyện:** Với bài toán là phân loại ảnh, ta cũng sẽ sử dụng hàm loss CrossEntropy và Stochastic Gradient:

```
lr = 1e-2
epochs = 15

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(
    model.parameters(),
    lr=lr
)
```

12. **Thực hiện huấn luyện:** Sử dụng hàm evaluate() và hàm fit() đã triển khai trong bài trước, chúng ta sẽ huấn luyện model DenseNet như sau:

```
train_losses, val_losses = fit(
    model,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    device,
    epochs
)
```

Với danh sách giá trị loss qua từng epoch trên tập train và val, ta có thể trực quan hóa như sau:



Hình 9: Trực quan hóa kết quả huấn luyện của mô hình trên tập train và tập val theo giá trị loss.

13. **Đánh giá model:** Ta gọi hàm evaluate() để đánh giá performance của model trên hai tập val và test như sau:

```python
val_loss, val_acc = evaluate(
    model,
    val_loader,
    criterion,
    device
)
test_loss, test_acc = evaluate(
    model,
    test_loader,
    criterion,
    device
)

print('Evaluation on val/test dataset')
print('Val accuracy: ', val_acc)
print('Test accuracy: ', test_acc)
```

- **Triển khai mô hình:** Với hai mô hình về phân loại ảnh thời tiết và phân loại ảnh phong cảnh đã huấn luyện được ở phía trên, chúng ta có thể triển khai lên streamlit như ảnh sau (thông tin về code cài đặt và link web streamlit được đề cập tại phần III.):

Hình 10: Ảnh giao diện của web demo.

B. Phần trắc nghiệm

1. Cho đoạn code như hình bên dưới, các bạn khai báo sử dụng BatchNormalization của pytorch với các tham số mặc định. Khi thực thi chương trình, kết quả in ra màn hình là?

```
import torch
import torch.nn as nn
import numpy as np

data = np.array([[[1, 6], [3, 4]]])
data = torch.tensor(data, dtype=torch.float32)

bnorm = nn.BatchNorm2d(1)
data = data.unsqueeze(0)
with torch.no_grad():
    output = bnorm(data)
    print(output)
```

(a) `tensor([[[[-1.3867, 1.3867], [-0.2773, 0.2773]]]).`

(b) `tensor([[[[-1.3842, 1.3998], [-0.2573, 0.2417]]]).`

(c) `tensor([[[[-1.3802, 1.3906], [-0.2891, 0.2787]]]).`

(d) `tensor([[[[-1.3732, 1.4073], [-0.2750, 0.2409]]]).`

2. Cho đoạn code như hình bên dưới, các bạn sử dụng phép toán concatenate (torch.cat) của pytorch với axis=0, để concatenate 2 feature map lại với nhau. Khi thực thi chương trình, kết quả in ra màn hình là?

```
import torch

a = torch.tensor([[1, 2], [3, 4]])
b = torch.tensor([[1, 2], [3, 4]])

a = a.reshape(1, 2, 2)
b = b.reshape(1, 2, 2)


c = torch.cat((a, b))
print(c)
```

(a) `tensor([[[5, 6], [7, 8]], [[9, 10], [11, 12]]]).`

(b) `tensor([[[13, 14], [15, 16]], [[17, 18], [19, 20]]]).`

(c) `tensor([[[1, 2], [3, 4]], [[1, 2], [3, 4]]]).`

(d) `tensor([[[21, 22], [23, 24]], [[25, 26], [27, 28]]]).`

3. Hạn chế nào sau đây trong deep neural networks truyền thống đã được ResNet cải thiện?

(a) Vanishing Gradient.

(b) Underfitting với mạng kích thước nhỏ.

(c) Overfitting với dữ liệu lớn.

(d) Chi phí tính toán cao.

4. Khi thực thi đoạn code sau đây, kết quả in ra màn hình là?

```
import torch
import torch.nn as nn

seed = 1
torch.manual_seed(seed)
input_tensor = torch.tensor([[[[1.0, 2.0], [3.0, 4.0]]]])

conv_layer = nn.Conv2d(in_channels=1, out_channels=1, kernel_size=1)
conv_output = conv_layer(input_tensor)

with torch.no_grad():
    output = conv_output + input_tensor
    print(output)
```

   (a) `tensor([[[[0.1345, 1.2345], [2.1345, 3.1345]]]])`.

   (b) `tensor([[[[-1.5678, -0.5678], [0.4322, 1.4322]]]])`.

   (c) `tensor([[[[2.5432, 3.5432], [4.5432, 5.5432]]]])`.

   (d) `tensor([[[[1.0739, 2.5891], [4.1044, 5.6197]]]])`.

5. Điều nào sau đây là sự khác biệt giữa ResNet-18 và ResNet-34?

   (a) ResNet-34 sử dụng DenseBlock thay vì ResidualBlock.

   (b) ResNet-18 sử dụng loại convolutional layer khác.

   (c) Hàm kích hoạt trong mỗi layer.

   (d) Số lượng layers.

6. Trong ResNet, để thực hiện skip connection add, channel của 2 feature map phải bằng nhau. Khi đó, tác giả paper ResNet đã đề xuất phương pháp gì để thực hiện skip connection khi input và output của một block có số lượng channel khác nhau?

   (a) Zero-padding.

   (b) Convolution layer 1x1.

   (c) Tất cả đều sai.

   (d) A và B đúng.

7. Đặc điểm nào sau đây là của kiến trúc DenseNet?

   (a) Có sử dụng Residual Connections giữa các layer.

   (b) Sử dụng layer max-pooling giữa các layer.

   (c) Ứng dụng 1x1 convolution.

   (d) Có Direct Connection từ bất kì layer nào đến tất cả các layer đứng sau.

8. Điều nào sau đây là lợi điểm của kiến trúc DenseNet?

   (a) Tiêu tốn nhiều tài nguyên tính toán hơn các kiến trúc CNNs khác.

   (b) Hạn chế tình trạng Vanish Gradient.

   (c) Loại bỏ convolutional layer.

   (d) Loại bỏ max-pooling layer.

III. Phụ lục

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại đây.
2. **Hint:** Các file code gợi ý có thể được tải tại đây.
3. **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 bài tập, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
4. **Triển khai streamlit:** Các bạn có thể truy cập vào web streamlit và link GitHub tại đây.
5. **Rubric:**

Mục	Kiến Thức	Đánh Giá
II.A	- Kiến thức về việc cài đặt chương trình huấn luyện mô hình ResNet từ đầu sử dụng PyTorch. - Kiến thức về các thành phần trong kiến trúc ResNet. - Kỹ thuật lập trình PyTorch.	- Nắm được cách cài đặt mô hình ResNet cho bài toán phân loại ảnh từ đầu sử dụng PyTorch. - Nắm được các hàm thành phần liên quan đến ResNet.
II.B	- Kiến thức về việc cài đặt chương trình huấn luyện mô hình DenseNet từ đầu sử dụng PyTorch. - Kiến thức về các thành phần trong kiến trúc DenseNet. - Kỹ thuật lập trình PyTorch.	- Nắm được cách cài đặt mô hình DenseNet cho bài toán phân loại ảnh từ đầu sử dụng PyTorch. - Nắm được các hàm thành phần liên quan đến DenseNet.
III.	- Lý thuyết về ResNet và DenseNet. - Code về PyTorch.	- Nắm được lý thuyết cơ bản về ResNet và DenseNet.

- Hết -