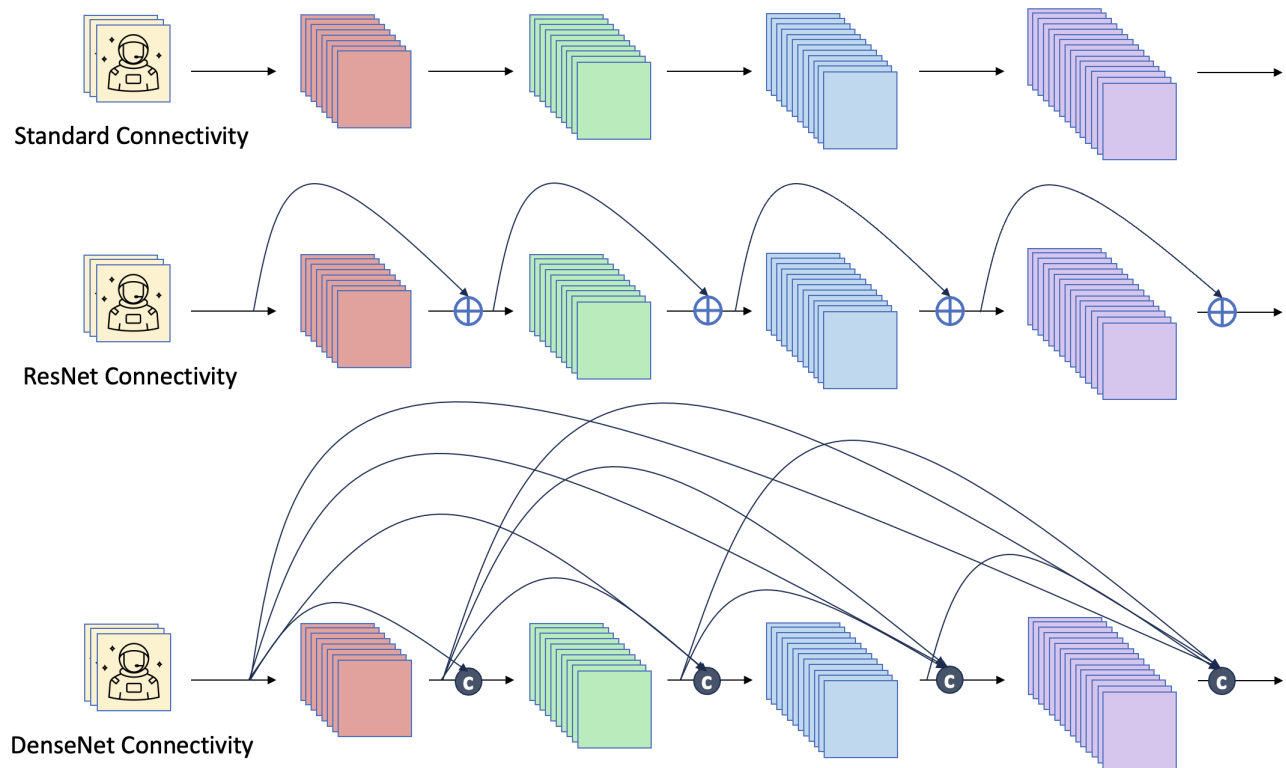


# Convolutional Neural Networks - Exercise

Ngày 12 tháng 11 năm 2023

## Phần 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 trưng 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.



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.

# Phần II: Bài tập

## A. Phần lập trình

### • Weather Classification

1. **Tải bộ dữ liệu:** Cho bộ dữ liệu về phân loại thời tiết trong ảnh, các bạn tải bộ dữ liệu này tại [đây](#).
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:

```
1 import torch
2 import torch.nn as nn
3 import os
4 import numpy as np
5 import matplotlib.pyplot as plt
6
7 from PIL import Image
8 from torch.utils.data import Dataset, DataLoader
9 from sklearn.model_selection import train_test_split
```

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

```
▼ weather-dataset
  ▼ dataset
    ▶ dew
    ▶ fogsmog
    ▶ frost
    ▶ glaze
    ▶ hail
    ▶ lightning
    ▶ rain
    ▶ rainbow
    ▶ rime
    ▶ sandstorm
    ▶ snow
```

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:

```
1 root_dir = 'weather-dataset/dataset'
2 img_paths = []
3 labels = []
4 classes = {
5     label_idx: class_name \
6         for label_idx, class_name in enumerate(
7             sorted(os.listdir(root_dir))
8         )
9 }
```

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

```

1 img_paths = []
2 labels = []
3 for label_idx, class_name in classes.items():
4     class_dir = os.path.join(root_dir, class_name)
5     for img_filename in os.listdir(class_dir):
6         img_path = os.path.join(class_dir, img_filename)
7         img_paths.append(img_path)
8         labels.append(label_idx)

```

4. **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:

```

1 seed = 0
2 val_size = 0.2
3 test_size = 0.125
4 is_shuffle = True
5
6 X_train, X_val, y_train, y_val = train_test_split(
7     img_paths, labels,
8     test_size=val_size,
9     random_state=seed,
10    shuffle=is_shuffle
11 )
12
13 X_train, X_test, y_train, y_test = train_test_split(
14     X_train, y_train,
15     test_size=val_size,
16     random_state=seed,
17     shuffle=is_shuffle
18 )

```

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

```

1 class WeatherDataset(Dataset):
2     def __init__(
3         self,
4         X, y,
5         transform=None
6     ):
7         self.transform = transform
8         self.img_paths = X
9         self.labels = y
10
11     def __len__(self):
12         return len(self.img_paths)
13
14     def __getitem__(self, idx):
15         img_path = self.img_paths[idx]
16         img = Image.open(img_path).convert("RGB")
17
18         if self.transform:
19             img = self.transform(img)
20
21         return img, self.labels[idx]

```

6. **Xây dựng hàm tiền xử lý ảnh (transforms):** Để dữ liệu ảnh đầu vào được đồng bộ về kích thước cũng như đơn giản hóa, chúng ta sẽ xây dựng hàm tiền xử lý ảnh như sau:

```
1 def transform(img, img_size=(224, 224)):  
2     img = img.resize(img_size)  
3     img = np.array(img)[..., :3]  
4     img = torch.tensor(img).permute(2, 0, 1).float()  
5     normalized_img = img / 255.0  
6  
7     return normalized_img
```

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

7. **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:

```
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 )
```

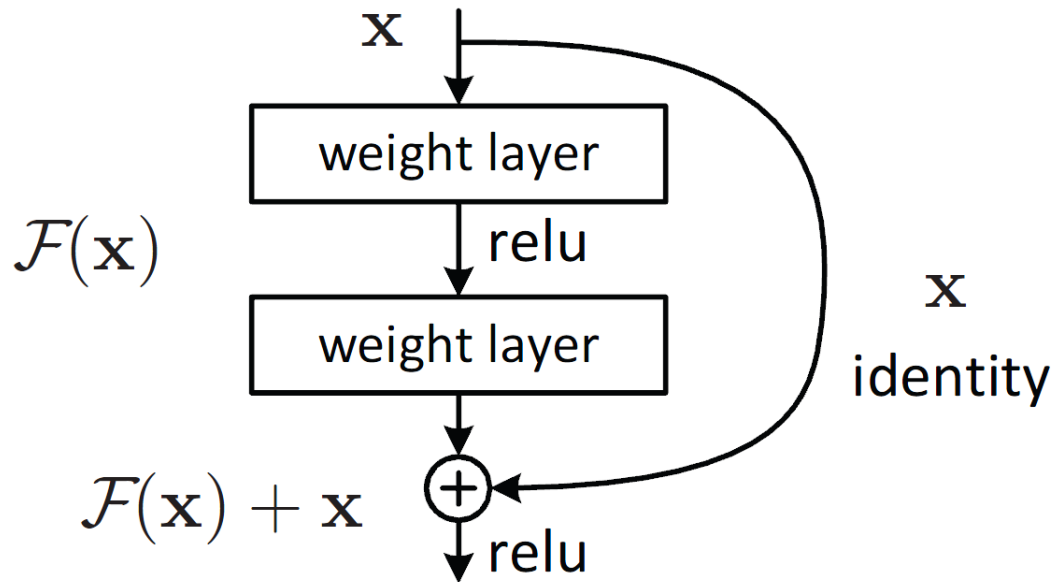
8. **Khai báo dataloader:** Với ba object datasets trên, ta khai báo giá trị batch size và tạo dataloader như sau:

```
1 train_batch_size = 512  
2 test_batch_size = 8  
3  
4 train_loader = DataLoader(  
5     train_dataset,  
6     batch_size=train_batch_size,  
7     shuffle=True  
8 )  
9 val_loader = DataLoader(  
10    val_dataset,  
11    batch_size=test_batch_size,  
12    shuffle=False  
13 )  
14 test_loader = DataLoader(  
15    test_dataset,  
16    batch_size=test_batch_size,  
17    shuffle=False  
18 )
```

9. **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	$112 \times 112$	$7 \times 7, 64, \text{stride } 2$				
conv2_x	$56 \times 56$	$3 \times 3 \text{ max pool, stride } 2$				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	$28 \times 28$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	$14 \times 14$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	$7 \times 7$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	$1 \times 1$	average pool, 1000-d fc, softmax				
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$

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:



Trong PyTorch, ta triển khai Residual Block như sau:

```

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(in_channels, out_channels, kernel_size=3,
5                                 stride=stride, padding=1)
6         self.batch_norm1 = nn.BatchNorm2d(out_channels)
7         self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
8                                 stride=1, padding=1)
9         self.batch_norm2 = nn.BatchNorm2d(out_channels)
10
11        self.downsample = nn.Sequential()

```

```

10         if stride != 1 or in_channels != out_channels:
11             self.downsample = nn.Sequential(
12                 nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=
stride),
13                 nn.BatchNorm2d(out_channels)
14             )
15         self.relu = nn.ReLU()
16
17     def forward(self, x):
18         shortcut = x.clone()
19         x = self.conv1(x)
20         x = self.batch_norm1(x)
21         x = self.relu(x)
22         x = self.conv2(x)
23         x = self.batch_norm2(x)
24         x += self.downsample(shortcut)
25         x = self.relu(x)
26
27     return x

```

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

```

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        blocks = []
18        first_block = residual_block(in_channels, out_channels, stride)
19        blocks.append(first_block)
20
21        for idx in range(1, n_blocks):
22            block = residual_block(out_channels, out_channels, stride)
23            blocks.append(block)
24
25        block_sequential = nn.Sequential(*blocks)
26
27        return block_sequential
28
29    def forward(self, x):
30        x = self.conv1(x)
31        x = self.batch_norm1(x)
32        x = self.maxpool(x)
33        x = self.relu(x)
34        x = self.conv2(x)

```

```

35         x = self.conv3(x)
36         x = self.conv4(x)
37         x = self.conv5(x)
38         x = self.avgpool(x)
39         x = self.flatten(x)
40         x = self.fc1(x)
41
42         return x

```

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

```

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)

```

10. **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:

```

1 def evaluate(model, dataloader, criterion, device):
2     model.eval()
3     correct = 0
4     total = 0
5     losses = []
6     with torch.no_grad():
7         for inputs, labels in dataloader:
8             inputs, labels = inputs.to(device), labels.to(device)
9             outputs = model(inputs)
10            loss = criterion(outputs, labels)
11            losses.append(loss.item())
12            _, predicted = torch.max(outputs.data, 1)
13            total += labels.size(0)
14            correct += (predicted == labels).sum().item()
15
16     loss = sum(losses) / len(losses)
17     acc = correct / total
18
19     return loss, acc

```

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

```

1 def fit(
2     model,
3     train_loader,
4     val_loader,
5     criterion,
6     optimizer,
7     device,
8     epochs
9 ):
10     train_losses = []
11     val_losses = []
12
13     for epoch in range(epochs):
14         batch_train_losses = []

```

```

15     model.train()
16     for idx, (inputs, labels) in enumerate(train_loader):
17         inputs, labels = inputs.to(device), labels.to(device)
18
19         optimizer.zero_grad()
20         outputs = model(inputs)
21         loss = criterion(outputs, labels)
22         loss.backward()
23         optimizer.step()
24
25         batch_train_losses.append(loss.item())
26
27     train_loss = sum(batch_train_losses) / len(batch_train_losses)
28     train_losses.append(train_loss)
29
30     val_loss, val_acc = evaluate(
31         model, val_loader,
32         criterion, device
33     )
34     val_losses.append(val_loss)
35
36     print(f'EPOCH {epoch + 1}: \tTrain loss: {train_loss:.4f} \tVal loss: {
37         val_loss:.4f}')
38
39     return train_losses, val_losses

```

12. **Khai báo hàm loss và thuật toán huấn luyện:** Với bài toán là phân loại ảnh, ta sẽ sử dụng hàm loss CrossEntropy và thuật toán huấn luyện sẽ là Stochastic Gradient Descent (SGD). Bên cạnh đó, ta cũng sẽ khai báo giá trị learning rate và số epochs:

```

1 lr = 1e-3
2 epochs = 50
3
4 criterion = nn.CrossEntropyLoss()
5 optimizer = torch.optim.SGD(
6     model.parameters(),
7     lr=lr
8 )

```

13. **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:

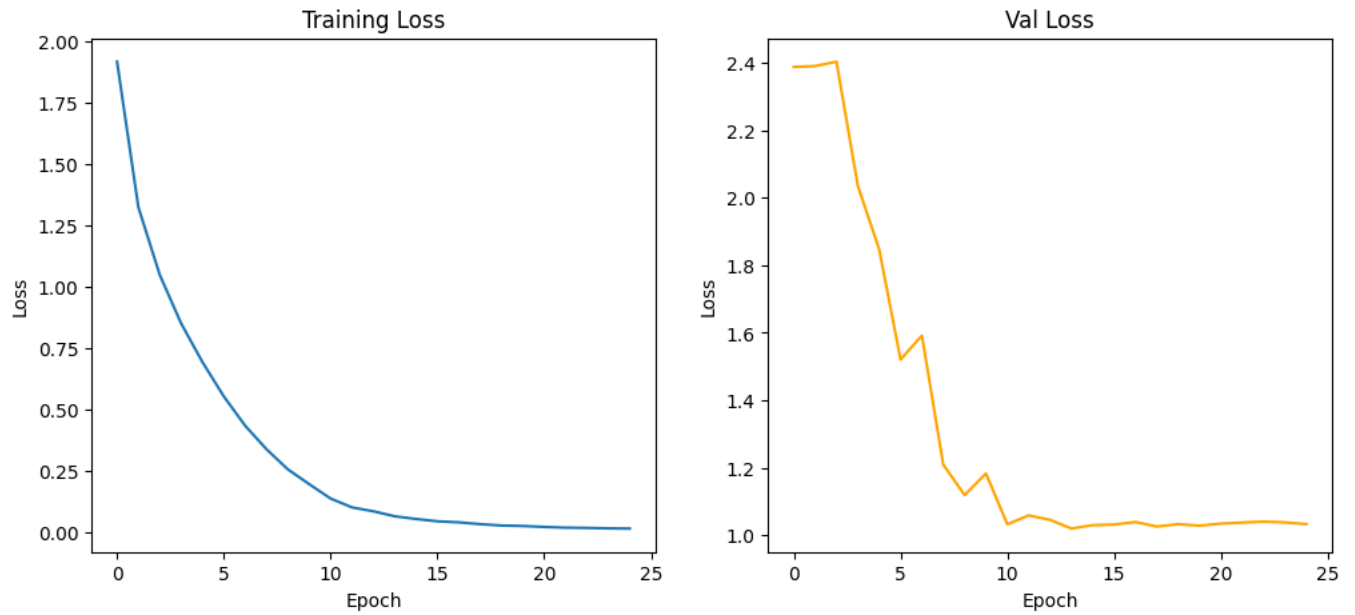
```

1 train_losses, val_losses = fit(
2     model,
3     train_loader,
4     val_loader,
5     criterion,
6     optimizer,
7     device,
8     epochs
9 )

```

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





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

```
1 val_loss, val_acc = evaluate(  
2     model,  
3     val_loader,  
4     criterion,  
5     device  
6 )  
7 test_loss, test_acc = evaluate(  
8     model,  
9     test_loader,  
10    criterion,  
11    device  
12 )  
13  
14 print('Evaluation on val/test dataset')  
15 print('Val accuracy: ', val_acc)  
16 print('Test accuracy: ', test_acc)
```

## • Scenes Classification

1. **Tải bộ dữ liệu:** Cho bộ dữ liệu phân loại thể loại phong cảnh trong ảnh, các bạn tải bộ dữ liệu này tại [đây](#).

2. **Import các thư viện cần thiết:**

```
1 import torch
2 import torch.nn as nn
3 import os
4 import numpy as np
5 import pandas as pd
6 import matplotlib.pyplot as plt
7
8 from PIL import Image
9 from torch.utils.data import Dataset, DataLoader
10 from sklearn.model_selection import train_test_split
```

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

```

└─ scenes_classification
   └─ train
      ├── buildings
      ├── forest
      ├── glacier
      ├── mountain
      ├── sea
      └── street
   └─ val
      ├── buildings
      ├── forest
      ├── glacier
      ├── mountain
      ├── sea
      └── street

```

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:

```
1 root_dir = 'scenes_classification'
2 train_dir = os.path.join(root_dir, 'train')
3 test_dir = os.path.join(root_dir, 'val')
4
5 classes = {
6     label_idx: class_name \
7         for label_idx, class_name in enumerate(
8             sorted(os.listdir(train_dir))
9         )
10 }
```

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:

```

1 X_train = []
2 y_train = []
3 X_test = []
4 y_test = []
5
6 for dataset_path in [train_dir, test_dir]:
7     for label_idx, class_name in classes.items():
8         class_dir = os.path.join(dataset_path, class_name)
9         for img_filename in os.listdir(class_dir):
10            img_path = os.path.join(class_dir, img_filename)
11            if 'train' in dataset_path:
12                X_train.append(img_path)
13                y_train.append(label_idx)
14            else:
15                X_test.append(img_path)
16                y_test.append(label_idx)

```

4. **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:

```

1 seed = 0
2 val_size = 0.2
3 is_shuffle = True
4
5 X_train, X_val, y_train, y_val = train_test_split(
6     X_train, y_train,
7     test_size=val_size,
8     random_state=seed,
9     shuffle=is_shuffle
10 )

```

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

```

1 class ScenesDataset(Dataset):
2     def __init__(
3         self,
4         X, y,
5         transform=None
6     ):
7         self.transform = transform
8         self.img_paths = X
9         self.labels = y
10
11     def __len__(self):
12         return len(self.img_paths)
13
14     def __getitem__(self, idx):
15         img_path = self.img_paths[idx]
16         img = Image.open(img_path).convert("RGB")
17
18         if self.transform:
19             img = self.transform(img)
20
21         return img, self.labels[idx]

```

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

```

1 def transform(img, img_size=(224, 224)):
2     img = img.resize(img_size)

```

```
3     img = np.array(img)[..., :3]
4     img = torch.tensor(img).permute(2, 0, 1).float()
5     normalized_img = img / 255.0
6
7     return normalized_img
```

7. **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:

```
1 train_dataset = ScenesDataset(
2     X_train, y_train,
3     transform=transform
4 )
5 val_dataset = ScenesDataset(
6     X_val, y_val,
7     transform=transform
8 )
9 test_dataset = ScenesDataset(
10    X_test, y_test,
11    transform=transform
12 )
```

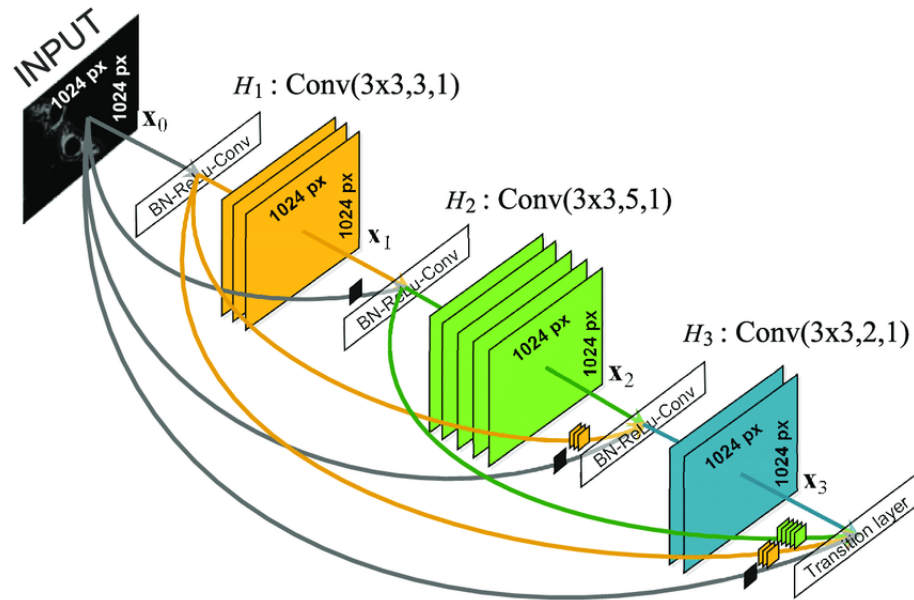
8. **Khai báo dataloader:** Lưu ý, với colab, các bạn nên cài đặt train batch size = 64 để có thể train được bài này.

```
1 train_batch_size = 64
2 test_batch_size = 8
3
4 train_loader = DataLoader(
5     train_dataset,
6     batch_size=train_batch_size,
7     shuffle=True
8 )
9 val_loader = DataLoader(
10    val_dataset,
11    batch_size=test_batch_size,
12    shuffle=False
13 )
14 test_loader = DataLoader(
15    test_dataset,
16    batch_size=test_batch_size,
17    shuffle=False
18 )
```

9. **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	$112 \times 112$	$7 \times 7$ conv, stride 2			
Pooling	$56 \times 56$	$3 \times 3$ max pool, stride 2			
Dense Block (1)	$56 \times 56$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	$56 \times 56$	$1 \times 1$ conv			
	$28 \times 28$	$2 \times 2$ average pool, stride 2			
Dense Block (2)	$28 \times 28$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	$28 \times 28$	$1 \times 1$ conv			
	$14 \times 14$	$2 \times 2$ average pool, stride 2			
Dense Block (3)	$14 \times 14$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	$14 \times 14$	$1 \times 1$ conv			
	$7 \times 7$	$2 \times 2$ average pool, stride 2			
Dense Block (4)	$7 \times 7$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	$1 \times 1$	$7 \times 7$ global average pool			
		1000D fully-connected, softmax			

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



Hình 1: Source: [image](#)

```

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

```

```

10     def forward(self, x):
11         res = x.clone().detach()
12         x = self.bn1(x)
13         x = self.relu(x)
14         x = self.conv1(x)
15         x = self.bn2(x)
16         x = self.relu(x)
17         x = self.conv2(x)
18         x = torch.cat([res, x], 1)
19
20     return x
21
22 class DenseBlock(nn.Module):
23     def __init__(self, num_layers, in_channels, growth_rate):
24         super(DenseBlock, self).__init__()
25         layers = []
26         for i in range(num_layers):
27             layers.append(BottleneckBlock(in_channels + i * growth_rate,
28 growth_rate))
29         self.block = nn.Sequential(*layers)
30
31     def forward(self, x):
32         return self.block(x)

```

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

```

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

```

```

33         for block in self.dense_blocks:
34             x = block(x)
35
36             x = self.bn2(x)
37             x = self.relu(x)
38             x = self.pool2(x)
39             x = x.view(x.size(0), -1)
40             x = self.fc(x)
41
42         return x

```

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

```

1 n_classes = len(list(classes.keys()))
2 device = 'cuda' if torch.cuda.is_available() else 'cpu'
3
4 model = DenseNet(
5     [6, 12, 24, 16],
6     growth_rate=32,
7     num_classes=n_classes
8 ).to(device)

```

10. **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:

```

1 lr = 1e-3
2 epochs = 100
3
4 criterion = nn.CrossEntropyLoss()
5 optimizer = torch.optim.SGD(
6     model.parameters(),
7     lr=lr
8 )

```

11. **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:

```

1 train_losses, val_losses = fit(
2     model,
3     train_loader,
4     val_loader,
5     criterion,
6     optimizer,
7     device,
8     epochs
9 )

```

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

```

1 val_loss, val_acc = evaluate(
2     model,
3     val_loader,
4     criterion,
5     device
6 )
7 test_loss, test_acc = evaluate(
8     model,
9     test_loader,
10    criterion,

```

```
11     device
12 )
13
14 print('Evaluation on val/test dataset')
15 print('Val accuracy: ', val_acc)
16 print('Test accuracy: ', test_acc)
```



## 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?

```

1 import torch
2 import torch.nn as nn
3 import numpy as np
4
5 data = np.array([[[1, 6], [3, 4]]])
6 data = torch.tensor(data, dtype=torch.float32)
7
8 bnorm = nn.BatchNorm2d(1)
9 data = data.unsqueeze(0)
10 with torch.no_grad():
11     output = bnorm(data)
12     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?

```

1 import torch
2
3 a = torch.tensor([[1, 2], [3, 4]])
4 b = torch.tensor([[1, 2], [3, 4]])
5
6 a = a.reshape(1, 2, 2)
7 b = b.reshape(1, 2, 2)
8
9
10 c = torch.cat((a, b))
11 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?

```

1 import torch
2 import torch.nn as nn
3
4 seed = 1
5 torch.manual_seed(seed)
6 input_tensor = torch.tensor([[[[1.0, 2.0], [3.0, 4.0]]]])
7
8 conv_layer = nn.Conv2d(in_channels=1, out_channels=1, kernel_size=2)
9 conv_output = conv_layer(input_tensor)
10
11 with torch.no_grad():
12     output = conv_output + input_tensor
13     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.

- Hết -