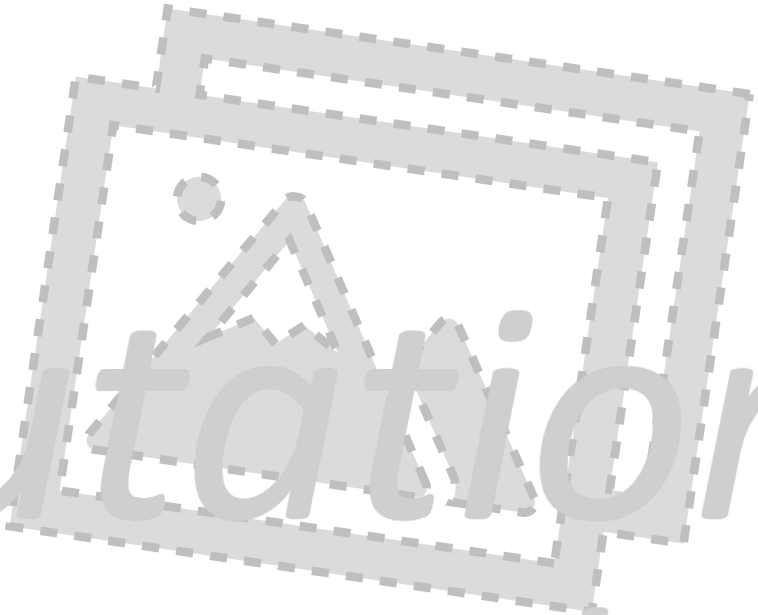
LAB06

Unet & DDPM

计算摄影学 2025春夏

2025/3/25



Computation Photography

BEFORE

- 本次实验内容需要在学在浙大上提交作业，本次作业计入成绩。
- 提交要求：
  - 将**源代码和实验文档**打包。
  - 不要提交训练好的模型文件。
  - 如有需要，请补充 requirements.txt 。
  - 压缩包名称为 Lab6-学号-姓名.zip/7z。
  - 在**4月15日23:59:59**前提交至学在浙大。

实验任务

- **（本周）搭建 Unet 网络**
  - 配置深度学习环境
  - 搭建 Unet 网络并进行推理
- **（下周）基于 MNIST 数据集训练扩散模型**
  - 训练 epsilon – prediction 模型
  - **Bonus**：训练 v-prediction 模型

配置深度学习环境(Cont.)

推荐平台：Linux / WSL

编程语言: python

深度学习框架: pytorch

配置深度学习环境(Cont.)

推荐平台：Linux / WSL

- win10 及以上的 WSL 安装
- 配置 windows terminal 访问 WSL
- 通过 vscode 扩展连接 WSL

编程语言: Python

深度学习框架: Pytorch

配置深度学习环境(Cont.)

推荐平台：Linux / WSL

编程语言: Python

- Python是一种高级的解释型编程语言。与 C++ 这种编译型语言相比，解释型语言没有单独的编译器，而是通过解释器直接执行。而Python代码在转换成字节码后，由Python虚拟机(PVM)解释执行。
- Python非常适合灵活快速的开发，但它的运行速度通常比编译型语言慢。

深度学习框架: Pytorch

Python Basics

C++	Python	说明
`int x = 5;`	`x = 5`	Python无需声明变量类型。也无需分号。
`if (x > 0) { ... }`	`if x > 0:` `    ...`	Python使用缩进代替花括号进行分层。
`for (int i = 0; i < 10; i++) { ... }`	`for i in range(10):` `    ...`	Python以列表为基础的 for 循环。
`while (condition) { ... }`	`while condition:` `    ...`	Python 的 while 循环。
`// commet`	`# comment`	行注释使用#符号

Python Basics(Cont.)

变量和数据类型

```
# 基本数据类型
x = 5                    # 整数
y = 3.14                 # 浮点数
name = "Python"          # 字符串
is_valid = True          # 布尔值

# 复合数据类型
my_list = [1, 2, 3]      # list 列表(类似于动态数组)
my_dict = {"key": "value", "age": 25}  # dict 字典(映射)
my_tuple = (1, 2, 3)     # tuple 元组(不可变列表)
my_set = {1, 2, 3}       # set 集合(不重复元素集合)
```

Python Basics(Cont.)

函数定义

```
# 简单函数
def add(a, b):
    return a + b

# 带默认参数的函数
def greet(name, message="Hello"):
    return f"{message}, {name}!"  # f-string格式化

# 调用函数
result = add(5, 3)      # 结果为8
greeting = greet(“world")  # 结果为"Hello, world!"
```

Python Basics(Cont.)

类和对象

```
class Person:
    # 构造函数
    def __init__(self, name, age):
        self.name = name  # 实例变量
        self.age = age

    # 实例方法
    def introduce(self):
        return f“I am {self.name}. I am {self.age} now."

# 创建对象
student = Person(“Cloud", 20)
print(student.introduce())  # 输出: I am Cloud. I am 20 now.
```

Python Basics(Cont.)

导入和使用模块

```
import numpy as np
import matplotlib.pyplot as plt # 画图使用的包

# 使用导入的模块
data = np.array([1, 2, 3, 4, 5])
mean = np.mean(data)  # 计算平均值

# 仅导入特定函数
from math import sqrt
root = sqrt(16)  # 结果为4
```

Python Basics(Cont.)

文件读写与报错

```
with open ("generate.log", "a") as f:
    for i in range(epoch):
        #training code
        f.write(f"epoch {i} loss {loss}\n")

import os
try:
    # 可能引起 error 的代码，如
    if os.path.exists(f“{file_name}”) == False: # 不确定有没有指定文件
        raise Exception(f"there is no file named {file_name}")
except Exception as e:
    print(e)
```

使用Anaconda进行包管理

下载地址：Download Now | Anaconda

- 在 linux 中使用 wget 命令下载
- wget https://repo.anaconda.com/archive/Anaconda3-2024.10-1-Linux-x86_64.sh

运行安装脚本

- bash Anaconda3-2023.09-0-Linux-x86_64.sh
- 按照提示完成安装，初始化可以选 yes
- 重启终端

使用Anaconda进行包管理(Cont.)

创建环境

- conda create –n torch_env python=3.10
- -n 指定环境名，python指定版本。本实验在 python 3.10 下验证可运行。

激活环境

- conda activate torch_env
- which python 查看python位置，通常在你的环境下。比如：/root/miniconda3/envs/torch_env/bin/python
- pip install 或 conda install 安装包。

配置深度学习环境(Cont.)

推荐平台：Linux / WSL

编程语言: Python

深度学习框架: Pytorch

- PyTorch是目前主流的深度学习框架，其核心是基于张量（Tensor）的计算系统。它的底层实现主要基于C++和CUDA，同时提供友好的Python接口。PyTorch提供了丰富的生态系统，包括计算机视觉（torchvision）和音频处理（torchaudio）等领域的专用工具包。
- 本实验要同时安装 torch 和 torchvision。

  pip install torch=2.6.0　　pip install torchvision=0.21.0

配置深度学习环境(Cont.)

推荐平台：Linux / WSL

编程语言: Python

深度学习框架: Pytorch

- PyTorch是目前主流的深度学习框架，其核心是基于张量（Tensor）的计算系统。它的底层实现主要基于C++和CUDA，同时提供友好的Python接口。PyTorch提供了丰富的生态系统，包括计算机视觉（torchvision）和音频处理（torchaudio）等领域的专用工具包。
- Pytorch 支持使用 CPU 进行训练。

对深度使用 GPU 训练感兴趣的同学，可以参考这篇博客配置 CUDA 和 CUDNN。

Pytorch Basics

张量创建

```
import torch
import numpy as np

x = torch.tensor([1, 2, 3, 4]) # 从列表创建
print(x)
zeros = torch.zeros(2, 3)  # 2x3的全0张量
ones = torch.ones(2, 3)    # 2x3的全1张量
rand = torch.rand(2, 3)    # 2x3的随机张量(0-1均匀分布)
randn = torch.randn(2, 3)  # 2x3的随机张量(标准正态分布)
x_float = torch.tensor([1, 2, 3], dtype=torch.float32) # 指定数据类型

np_array = np.array([1, 2, 3]) # 从NumPy数组创建
tensor_from_np = torch.from_numpy(np_array)  # 深度拷贝请用 copy()

arange = torch.arange(0, 10, step=2)  # tensor([0, 2, 4, 6, 8])
linspace = torch.linspace(0, 10, steps=5) # tensor([0.0, 2.5, 5.0, 7.5, 10.0])
```

Pytorch Basics(Cont.)

张量操作

```python
# 属性
x = torch.randn(3, 4, 5)
print(f"形状: {x.shape}")  # torch.Size([3, 4, 5])
print(f"维度: {x.dim()}")  # 3
print(f"数据类型: {x.dtype}")  # torch.float32

# 变形
y = x.reshape(3, 20)

# 运算
a = torch.tensor([1, 2, 3])
b = torch.tensor([4, 5, 6])
print(a + b)  # 加
print(torch.add(a, b))
print(a * b)  # element-wise 乘
print(torch.matmul(a, b))  # 点积
```

Pytorch Basics(Cont.)

张量操作

```
# 索引与切片
x = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(x[0])  # tensor([1, 2, 3])
print(x[:, 1])  # tensor([2, 5, 8])
print(x[0, 0:2])  # tensor([1, 2])

# 连接张量
c = torch.cat([a, b], dim=0)  # tensor([1, 2, 3, 4, 5, 6])
d = torch.stack([a, b], dim=0)  # tensor([[1, 2, 3], [4, 5, 6]])
```

Pytorch Basics(Cont.)

GPU上的张量

```
# 检查GPU可用性
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f“device: {device}")

# 创建在GPU上的张量
x_gpu = torch.tensor([1, 2, 3], device=device)

# 将已有张量移动到GPU
x = torch.tensor([1, 2, 3])
x_gpu = x.to(device)

# 将张量移回CPU
x_cpu = x_gpu.to('cpu')
```

你可以使用 device 编写设备无关的代码，但注意张量运算时，参与运算的张量需要保证在同一个设备上。

使用Pytorch搭建神经网络

例子:线性模型

```python
import torch
import torch.nn as nn

# 线性模型
class LinearModel(nn.Module):
    # 在 __init__ 定义 operators
    def __init__(self, input_dim, output_dim):
        super(LinearModel, self).__init__()
        self.linear = nn.Linear(input_dim, output_dim) # nn 已经实现的层

    # 在 forward 使用 operators
    def forward(self, x):
        return self.linear(x)

# 创建模型对象
model = LinearModel(input_dim=10, output_dim=1)
```

使用Pytorch搭建神经网络

常用神经网络层和激活 *RECALL 课堂内容*

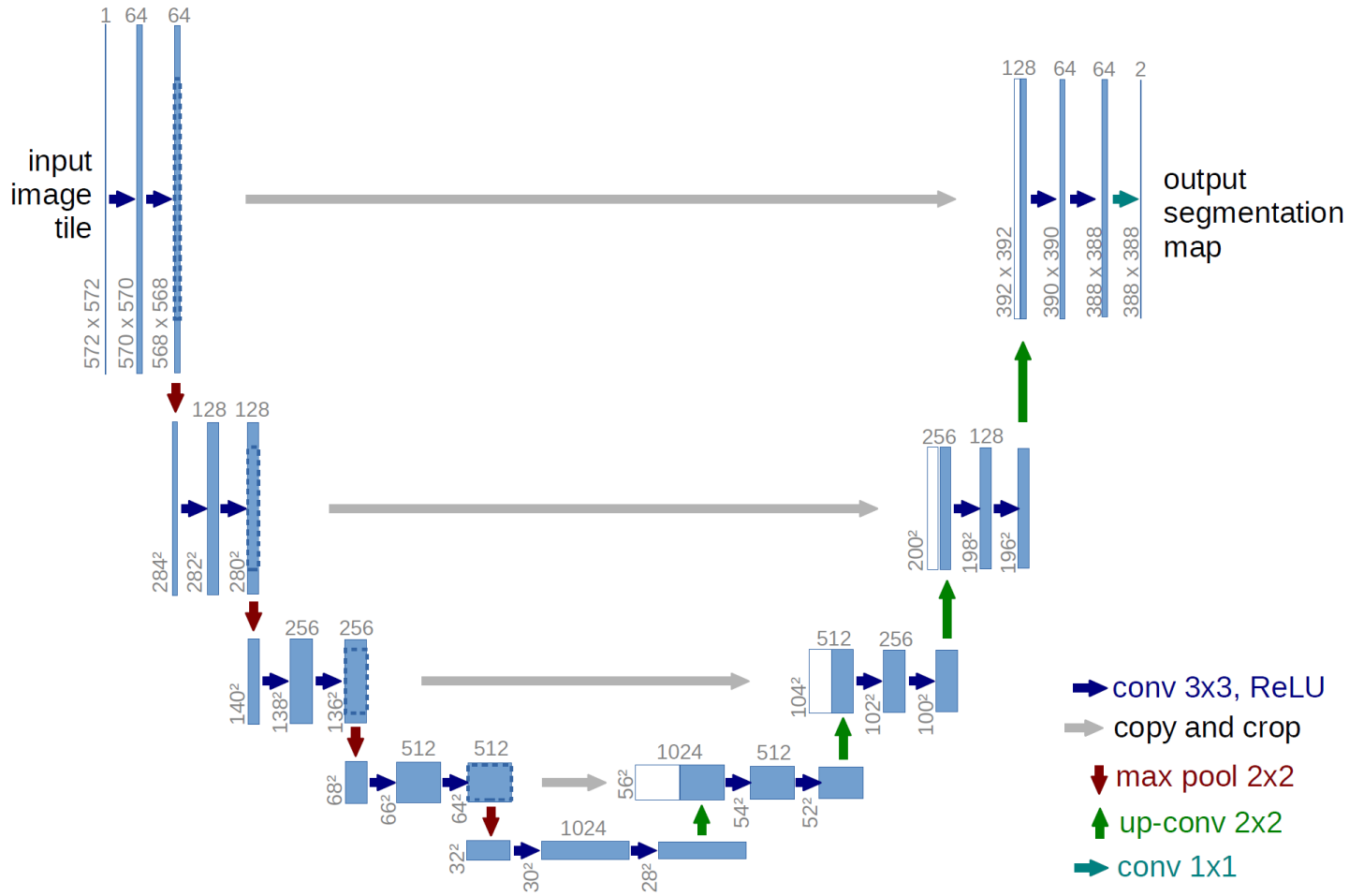
```
# 全连接层，10输入，20输出
linear = nn.Linear(10, 20)
# 2D卷积，3通道输入，16通道输出
conv = nn.Conv2d(3, 16, kernel_size=3, padding=1)
# 2x2最大池化
pool = nn.MaxPool2d(2)
# 50%的dropout
dropout = nn.Dropout(0.5)
# 批标准化
batch_norm = nn.BatchNorm2d(16)

# ReLU激活函数
relu = nn.ReLU()
# Sigmoid激活函数
sigmoid = nn.Sigmoid()
```

搭建 Unet 网络

U-Net 是一个经典的语义分割全卷积网络，最初应用于医疗图像的分割任务，其结构如右图所示。

U-Net 具有一个对称的结构，左边是一个典型的卷积神经网络，右边是一个对称的上采样网络。



搭建 Unet 网络(Cont.)

本实验使用 Unet 完成汽车图像划分的任务。该任务输入为一张彩色rgb图像，输出为一个掩码矩阵，表示图片上每个像素是否属于汽车(0-1)。这个问题可以处理成逐像素的二分类问题。

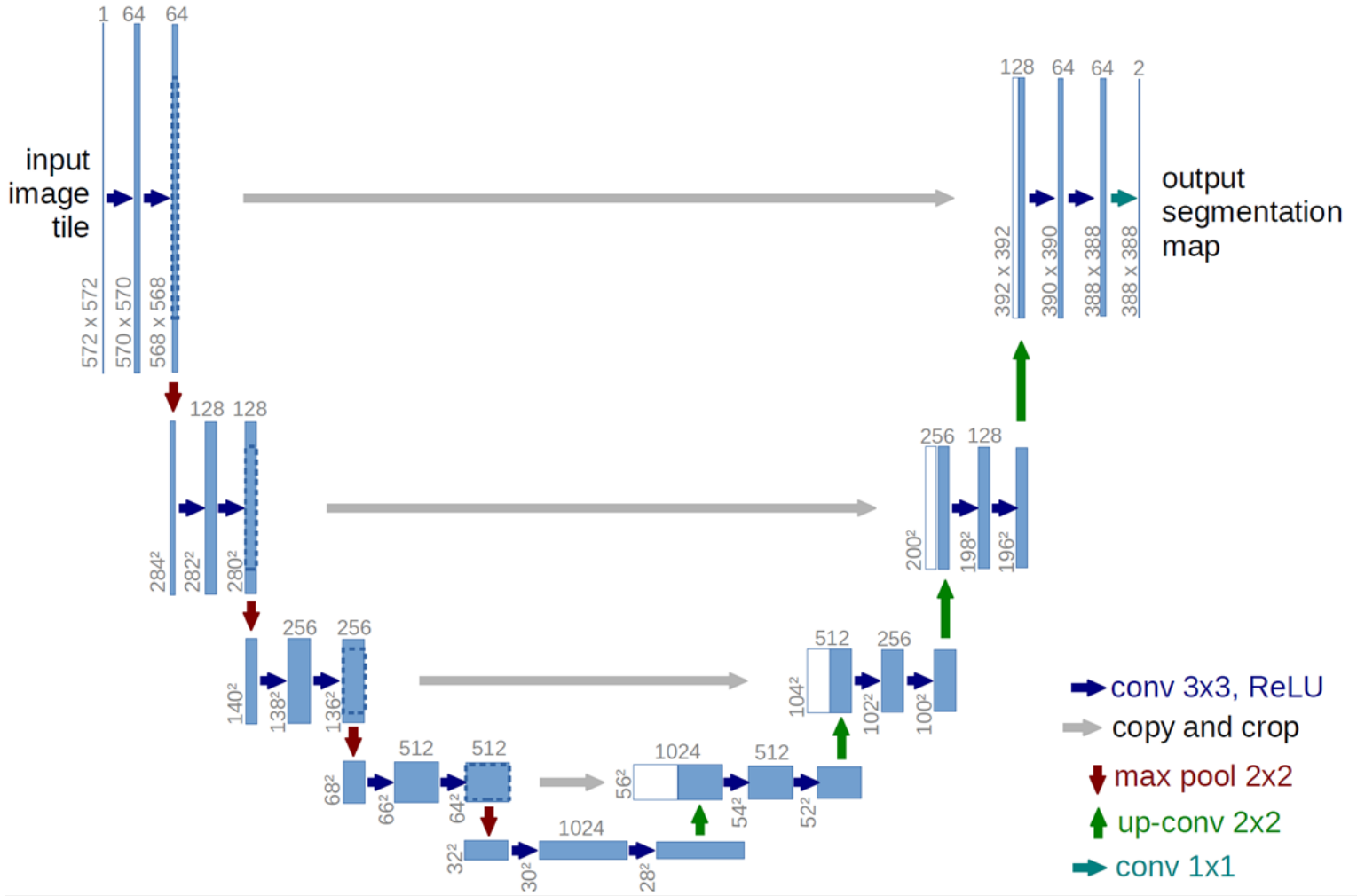
本实验提供已在训练集上训练好的模型 model.pth。

搭建 Unet 网络(Cont.)

在 U-Net 原论文中，左侧向下的结构被称为 **Contracting Path**，由通道数不断增加的卷积层和池化层组成。右侧向上的结构被称为 **Expanding Path**，由通道数不断减少的卷积层和上采样层（反卷积层）组成。

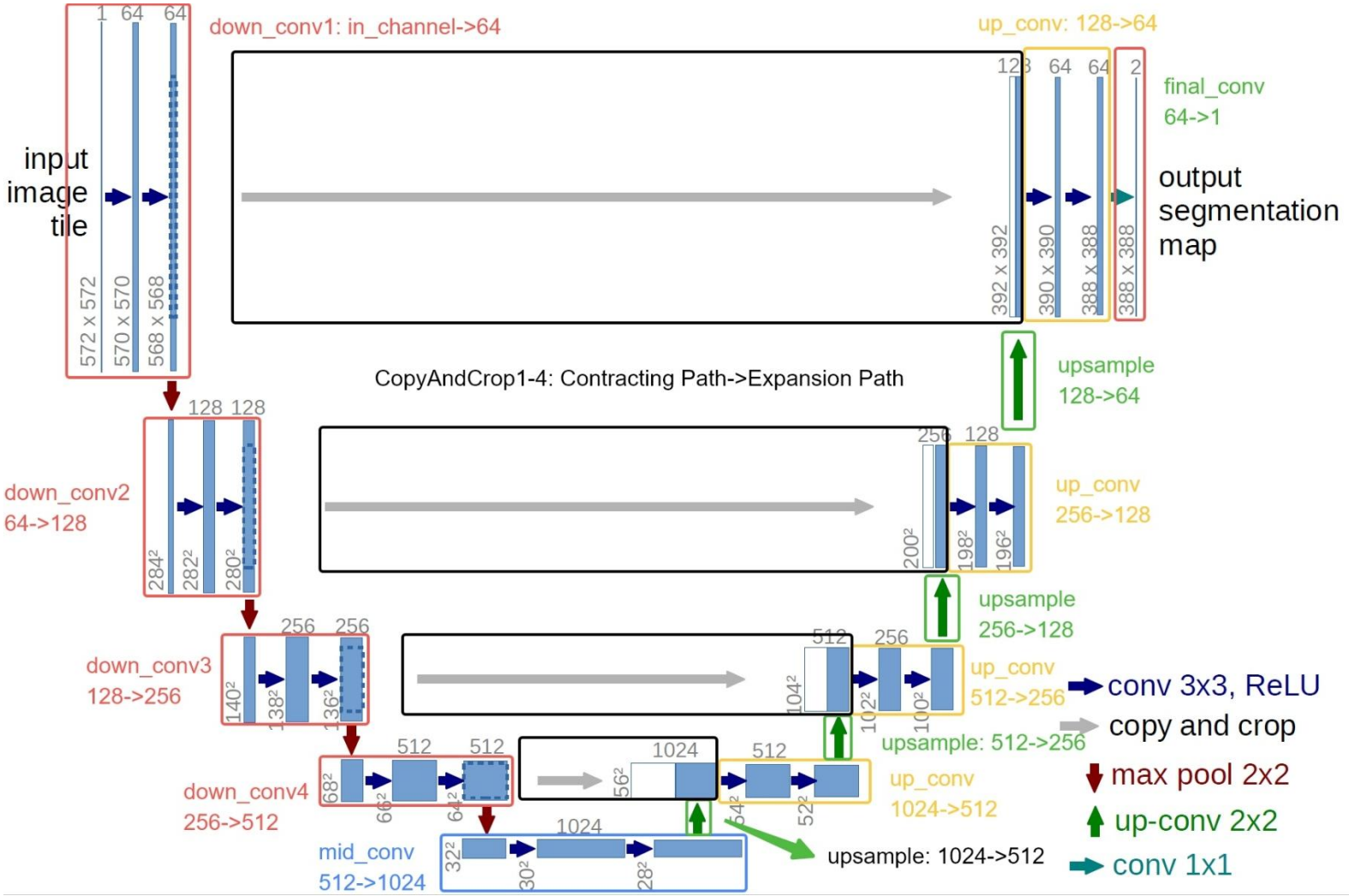
搭建 Unet 网络(Cont.)

特别的是，在 **Expanding Path** 中，每次上采样层都会将 **Contracting Path** 中对应的特征图与自身的特征图进行**拼接**，这样可以保证 Expanding Path 中的每一层都能够利用 Contracting Path 中的信息。*可以和计组中的前递类比。*



搭建 Unet 网络: __init__

实验要求按照右图标注的块完成为提供代码中的 UNet 类的 __init__ 补全按部分卷积层的定义。



搭建 Unet 网络:__init__

前图中已经标明了通道数的变化，注意由于我们进行划分任务，final_conv 的输出通道变成了 1。在通用的网络结构中，使用 out_channels 控制输出通道。

- **down/mid/up_conv:** 由两个3×3卷积核的卷积层组成，padding 和 stride 为 1。每个卷积层后有一个 ReLU 激活函数。
- **final_conv:** 一个 1×1 卷积核的卷积层，padding 为 0，stride 为 1，没有激活函数。

*只需要在有TODO的地方填写，在 nn.Sequential 的括号中正常填写 nn.Conv2d , nn.ReLU即可。**不要自定义网络类**，避免模型因为层命名不一致而加载失败。*

搭建 Unet 网络:forward

在 forward 中，Unet 将 Contracting Path 中的特征图与 Expanding Path 中的特征图进行拼接，以保证 Expanding Path 中的每一层都能够利用 Contracting Path 中的信息。我们以简单的 **CropAndConcat 类**来实现这个功能，只需要实现 forward 方法。

- 取得 shape，其中 b, c, h, w 的含义：batch size / channel / height / width
- 使用 torchvision.transforms.functional.center_crop(...) 对 Contracting Path 中的特征图进行合适裁剪，以保证尺寸一致。
- 使用 torch.cat() 的用法，在合适维度拼接。
- **注意**：实际拼接的顺序为 Expanding Path 中的 feature map 在左，Contracting Path 中的 feature map 在右

搭建 Unet 网络:forward

根据图片实现 Unet 的 forward 函数。

注意保留中间结果以供 CropAndConcat 使用。

搭建 Unet 网络:加载测试

try.py

```
import argparse
import torch
from unet import UNet

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Predict masks from input
images')
    parser.add_argument('--model', '-m', default='model.pth',
                        help='Specify the file in which the model is stored')
    args = parser.parse_args()
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    print(f'Loading model {args.model}')
    print(f'Using device {device}')

    model = UNet(in_channels=3, out_channels=1).to(device)
    state_dict = torch.load(args.model, map_location=device)
    model.load_state_dict(state_dict)

    print('Model loaded')
```

搭建 Unet 网络:加载测试

在终端通过参数 –model 指定模型文件:

```
python try.py --model model.pth
```

加载成功的期望输出如下(设备可能有区别):

```
Loading model model.pth
Using device cpu
Model loaded
```

搭建 Unet 网络:模型推理

在加载提供的模型的基础上，对单张汽车图片 test.jpg 的 mask 进行推断。主要步骤包括：

- 读入单张图片。
- 图片预处理。
- 单张图片[C,H,W]变为模型的输入输出 [B, C, H, W] 的格式
- 用 sigmoid 处理输出分数并寻找合适阈值转换为 0-1 mask。

实验要求实现一个 infer.py，应能够满足在终端通过参数指定模型文件、输入图片、输出图片和阈值。如：

```
python infer.py -m model.pth -i test.jpg -o output.jpg -t 0.5
```

搭建 Unet 网络:模型推理

对于使用 Image.open() 读入的单张图片，这里提供一个非常朴素的预处理函数：

另外，由于模型训练时输入输出均为 256×256 图像，需使用 torch.nn.functional.interpolate 进行插值，以和测试图片尺寸匹配。

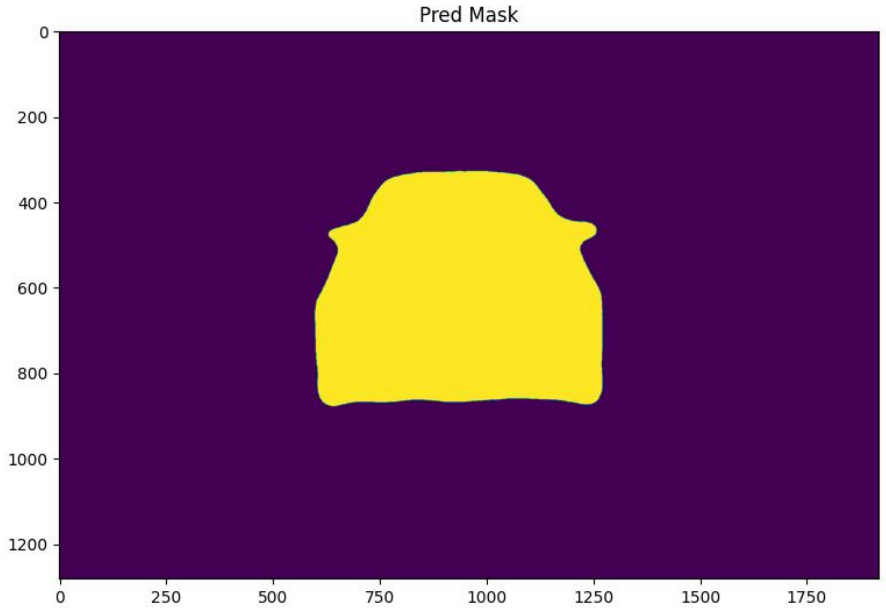
```
def preprocess(img):
    _img = img.resize((256,256))
    w, h =  _img.size
    _h = int(h)
    _w = int(w)
    assert _w > 0
    assert _h > 0
    _img = img.resize((_w, _h))

    _img = np.array(_img)
    if len(_img.shape) == 2:
        _img = np.expand_dims(_img, axis=-1)

    _img = _img.transpose((2, 0, 1))
    if _img.max() > 1:
        _img = _img / 255.
    return _img
```

期望结果


Image
0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750


Pred Mask
0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750

AGAIN

- 两节课的内容合并在一个压缩包，在**4月15日23:59:59**前提交至学在浙大。

Questions are welcome

计算摄影学 2025春夏

2025/3/25

特别感谢：2024年计算摄影学助教周健均对本实验的贡献

Computation Photography