高级语言程序设计

实验报告

学院 计算机学院

班级 计算机科学卓越班

学号 2313903

姓名 付嘉晨

2024 年 5 月 15 日

目录

0 引言

本工作基于 Qt C++ 实现了一个图片编辑器，其主体部分集成了诸多功能。这其中，有如**灰度化**、**均值滤波**、**边缘检测**与**伽马变换**之类的传统图像处理算法的实现；也有本文**最主要的创新点**：一个**用于图像重建的多尺度感知神经网络**。我们在正文中将介绍深度学习技术在图像超分辨率任务上的应用历史以及本工作的 **Motivation**，同时也将介绍我们是如何使用 Qt C++ 搭建这样一个图像处理框架的，希望能对您有帮助！

图 1: 图片编辑器界面图.

在下文中展示的几组图片，可以清晰地看出低分辨率图像与超分辨率图像在细节上的差距。其中，左边一列的图片为低分辨率图片，右边一列为超分辨率得到的图片。

图 2: Set14/comic.png

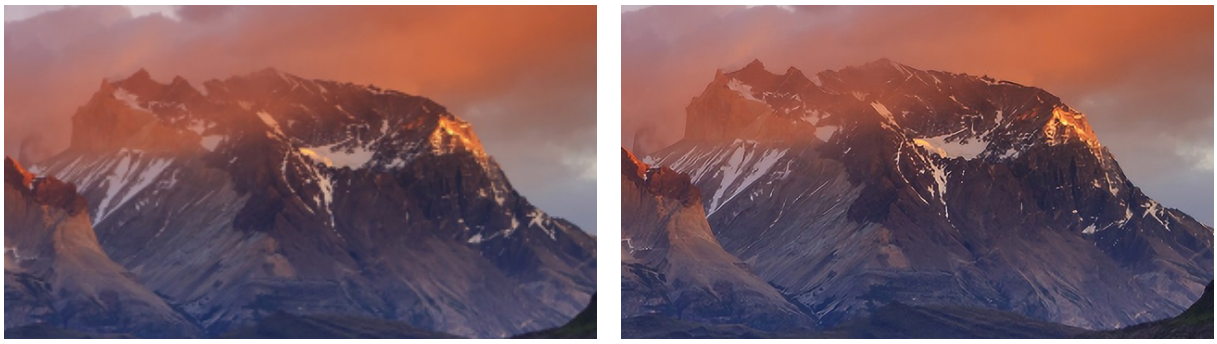

图 3: Wallpaper.png

图 4: Set14/ppt3.png

图 5: Set14/flowers.png

感谢您对本工作的关注！如果您有任何问题，欢迎发送邮件至我的邮箱，我将在看到之后回复！

附：我的邮箱地址为 fjc@mail.nankai.edu.cn

1 作业题目

1.1 中文题目

集成了用于图像重建的多尺度感知神经网络的图片编辑器

1.2 英文题目

Rethinking Pixel-level Predictions with Multi-scale Features: An Image Editor Integrating Multi-scale Perceptual Neural Networks for Image Reconstruction

2 开发环境

Visual Studio2022、Qt Creator 5.0.2、OpenCV4.5.5、LibTorch1.12.0

3 背景介绍

3.1 神经网络

以深度学习模型为代表的神经网络已经成为计算机领域的一项革命性技术，其主要思想为使用神经网络层如**密集连接层 (Dense or Linear)**、**卷积层 (Convolution)** 等及激活函数构造复杂的模型，通过对数据集进行拟合来进行表征的学习。在计算机视觉领域中，常使用的神经网络模型有以下几种：

卷积神经网络 LeCun 等人 [1] 首次将卷积模型用于手写数字识别，而 Krizhevsky 等人 [2] 使用 ImageNet 数据集首次训练出深度卷积网络，并在 ImageNet 竞赛中取得了突破性的胜利。在之后，牛津大学的视觉几何实验室提出了经典分块网络 VGG[3]，深刻影响后来的网络设计；Kaiming 等人 [4] 于 2016 年提出残差网络，利用残差跳跃连接首次训练出上千层深度的网络，这样的残差结构在后来的众多深度学习模型中均被使用。在 2020 年之后，卷积网络的研究热度有所降低，但也有许多有意义的研究出现，这其中，令人影响深刻的有清华大学丁霄汉等人提出的 RepVGG[5]，使用结构重参数化技术重构了 VGG[3] 网络；同样由清华大学丁霄汉等人提出的 RepLKNet[6]，发掘了大核卷积的潜力；FAIR 提出的 ConvNeXt[7]，总结了多年以来的训练经验以及技巧，训练出了性能极佳的卷积神经网络。

Transformer 网络 人类在看到一张图片或者阅读一段文字时，并不是从头到尾顺序地获取信息，而是基于注意力地进行感知。长期以来，学者们都尝试让网络获取注意力，这称之为注意力机制，但学界对于注意力机制的使用长期停留在对网络 “锦上添花”，即在主干网络之后添加注意力机制以获得增强的特征信息。2017 年，谷歌的研究团队创新性

地提出了完全基于注意力机制的用于机器翻译任务的神经网络模型 Transformer[8]，这样的架构设计点起了自然语言处理学界的研究热潮，而计算机视觉学界的研究者们也不断在思考如何将这样一种结构引入视觉任务中，但由于图像的二次复杂度以及低级语义特征，使得像自然语言处理那样的上下文处理变得困难。2020 年，同样来自谷歌的团队提出了 Vision Transformer[9]，首次将 Transformer 架构引入视觉任务；在之后，微软亚洲研究院的团队对 ViT 做出了改进，引入了滑动窗口机制，提出 Swin Transformer[10]，进一步提高了神经网络在 ImageNet 图像分类任务上的准确度，并且为后来的研究提供了思路。

3.2 图像超分辨率

图像超分辨率是一个重要的底层视觉（Low-Level Vision）任务，其目的是从低分辨率的图像中重建出高质量的版本。根据输入的不同可以分为两个子任务：**单一图像超分辨率（SISR）**与**基于多幅图像的超分辨率（MISR）**，本工作聚焦于 SISR 这一任务，并使用深度学习的方法实现超分辨率。多年来，基于深度学习的超分辨率方法主要有以下几类：

基于卷积神经网络的图像超分辨率 自 Dong 等人 [11] 首次将神经网络引入图像超分辨率任务并取得较好的成果以来，大量研究者聚焦于如何使用神经网络进一步提升超分辨率的性能，这其中，有里程碑意义的有：2016 年由 Shi 等人 [12] 提出的 ESPCN 网络，他们引入了**亚像素卷积**这一操作，使图像超分辨率的计算复杂度大幅下降；2017 年由 Lim 等人 [13] 提出的增强深度超分辨率网络，删除了传统网络中非必要的模块以及冗余的操作，在使模型轻量化的同时提升了超分辨率的效果；2018 年由 Zhang 等人 [14] 提出的残差通道注意力网络，首次将注意力机制引入图像超分辨率任务中，在效果比 EDSR[13] 好的同时仅使用了其 $\frac{1}{3}$ 的参数量，证明了注意力机制对超分辨率任务的有效性。

基于生成对抗网络的图像超分辨率 生成对抗网络于 2014 年由 Ian 等人 [15] 提出，其生成图片的能力令学界震惊。2017 年，Ledig 等人 [16] 首次将 GAN[15] 引入图像超分辨率任务中，并且提出了**内容损失**，使整个网络在学习的过程中更加关注重建图像和原始图像的语义特征差异，而非逐个像素之间的颜色和亮度差异。在之后，Wang 等人 [17] 提出的 ESRGAN 网络进一步提高了性能，取得了不错的成绩。

基于 Transformer 的图像超分辨率 2020 年，Yang 等人 [18] 首次将 Transformer[8] 引入图像超分辨率任务中。在之后，Liang 等人 [19] 于 2021 年提出 SwinIR 网络，将 Swin Transformer[10] 引入超分辨率任务，刷新了图像超分辨率的 SOTA（State-Of-The-Art）。2023 年，南开大学 Zhou 等人 [20] 提出 SwinIR[19] 的改进网络 SRFormer，进一步提升了网络的性能。

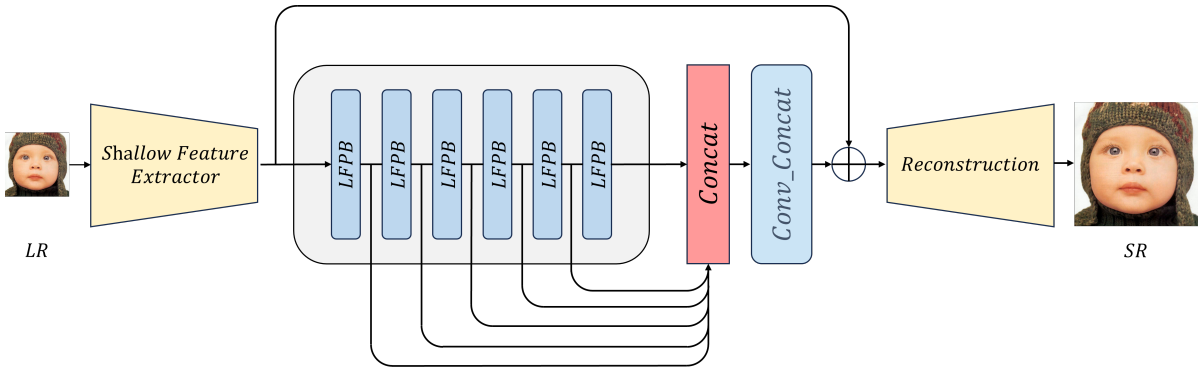

图 6: 多尺度感知神经网络结构图.

基于轻量级神经网络的图像超分辨率 神经网络的特征提取与表达能力令人震惊的同时，如何将其有效地部署备受学界及工业界的思考，这其中，如何将网络用在计算能力较差的移动端设备上是极具现实意义的问题。由于图像处理的二次复杂度以及深度卷积网络的大参数量，使得像 RCAN[14] 这样参数量达到 10M 级别的网络难以用于移动端，而 Transformer 网络如 SwinIR[19] 由于使用了大量的多头自注意力，产生了极大的并行负担，也难以被移动端直接使用。2017 年，Howard 等人 [21] 将深度可分离卷积用于神经网络结构中，大幅降低网络参数量的同时，性能衰减几乎可以忽略不计，给学界提供了一种轻量化模型的思路；2020 年，Haase 等人 [22] 基于对 Howard 等人工作的思考与实验，提出了一种深度可分离卷积的改进版本，蓝图卷积，在参数量小幅度增加的同时，有效地提升了深度可分离卷积的性能。2018 年，Zheng 等人 [23] 提出了信息蒸馏网络，使用一种由多分支架构组成的主干网络来提供蒸馏的语义信息，为轻量级超分辨率的网络设计提供了一种模板级思路；这之后，同样是 Zheng 等人 [24] 基于这一工作提出了改进版本 IMDN，进一步为网络引入多级蒸馏信息，提升了网络性能。2022 年，Li 等人 [25] 基于蓝图卷积 [22] 提出了用于高效率图片超分辨率的卷积神经网络，在参数量仅有 0.3M 的基础下取得了极其优秀的成绩；Kong 等人 [26] 提出 RLFN 网络，去除了先前广为使用的多分支结构，使网络推理提速。

4 研究动机

4.1 多尺度特征对于视觉任务的意义

南开大学的程明明教授在《粒度自适应的图像感知技术》[27] 中提出了这样的一个例子：如果只看一张大图的一小部分，不论是人或是机器都难以准确辨别出物体的类别，而在获得这个小部分周围的信息后，我们能轻易地辨别出这个物体是一个树桩样子的椅子。这充分说明了多粒度感知与上下文信息对于图像感知的重要性。同时，程教授提出了对 ResNet[4] 有效性的另一种思考，即：残差跳跃连接能够使小感受野的特征图与大感受野的特征图混合，给网络提供一种多粒度的信息，有利于图像感知。

而在图像分类、目标检测等高级视觉任务中，由于输入和输出可能分辨率不同、维

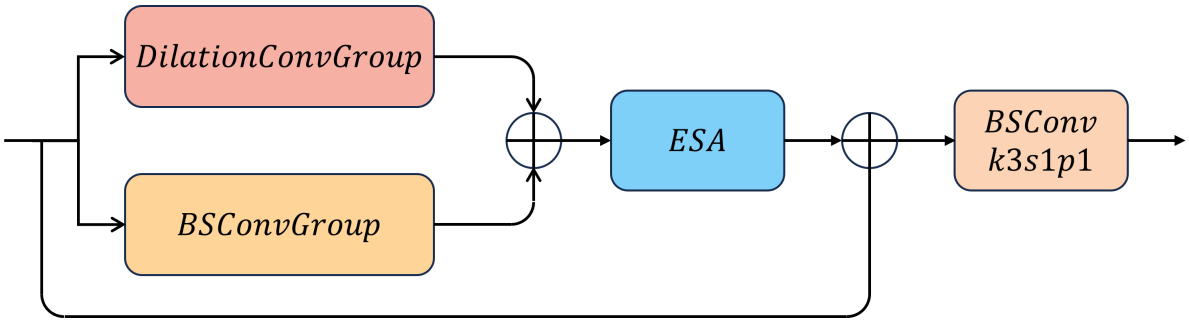

图 7: 多尺度感知神经网络块 LFPB.

度不同，因此可以引入池化等下采样操作来降低特征图的分辨率，一方面可以减小运算复杂度，另一方面可以对特征图中的信息进行融合与筛选，分辨率不同的特征图组成了“多粒度”的信息。而在超分辨率任务中，由于网络需要做的是上采样工作，因此在网络中的降采样操作可能是对模型性能有害的，因此分辨率不同的金字塔样“多粒度”信息提取对于这一任务可能不适用。但我们可以设计一种方法在保持特征图分辨率不变的条件下尽可能地提供不同尺度的信息，以达到“多粒度”的感知效果。出于“提供多尺度感知信息”的目的，我设计了这样一种结构与 IMDN[24] 相似的网络，并使用感受野不同的卷积组合来进行特征的提取，并提供多次特征图融合以混合多尺度信息。这样的网络设计具备从小到大几乎所有尺度的感知信息，实验证明，这样的网络效果十分不错。

4.2 如何在低参数量的条件下获得大的感受野

Chen 等人 [28] 提出的 Deeplab 架构在另一个像素级预测任务：**语义分割**上取得了极为优秀的成果，他们引入了空洞卷积这一卷积层设计，在保持卷积网络参数量不变的同时，提供了更大的感受野，这启发了我们的工作。但同时也有人认为，空洞卷积这一有像素跨越的操作对于像素密集型预测有不良的影响，而超分辨率正是一个像素密集预测的任务。

出于对空洞卷积是否对超分辨率有不良影响的思考，以及为网络提供大感受野的需求，我在网络中引入了空洞卷积。Wang 等人 [29] 的工作说明了如果整个网络都使用有空洞的卷积，那么一定会导致相邻像素的信息无法构建联系，因此，我们并非全部使用带有膨胀的卷积，而是使用膨胀系数 $dilation = [1, 2, 2]$ 的卷积组合，这一组合能提供 11×11 大小的感受野。同时，我们的网络块还包含一个由 3 个卷积核大小为 3×3 的蓝图卷积 [22] 组成的卷积组合，这一组合能提供 7×7 大小的感受野。

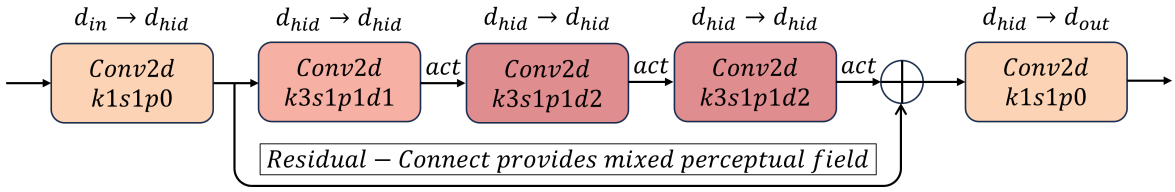

图 8: 膨胀卷积组合.

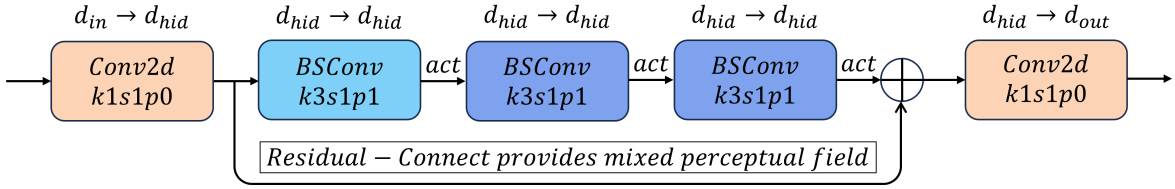

图 9: 蓝图卷积组合.

5 主要流程

5.1 模型训练

我们将上述模型在 DIV2K 数据集上进行训练, 训练一共进行了 1×10^6 个 iter, 训练的批次大小 $batch = 16$, 训练使用的优化器是 Adam 优化器, 初始学习率为 2×10^{-4}, 并使用余弦退火并周期性重启的方式进行训练，重启周期为 $range = [3e5, 2e5, 2e5, 2e5, 1e5]$, 重启权重为 $weight = [1, 0.5, 1, 0.5, 0.5]$，训练的真实图像 $PatchSize = 96$。

我们的训练配置如下，训练的总耗时大约为 24h:

CPU: 12 vCPU Intel(R) Xeon(R) Platinum 8255C CPU @ 2.50GHz

GPU: Nvidia GeForce RTX 3080 10G

memory: 40G

5.2 C++ 部署

我们的目的是将模型部署于 C++ 环境并进行测试，为此，我们定义了几个基本类型及 MainWindow 类用于后续操作及 UI 设计。

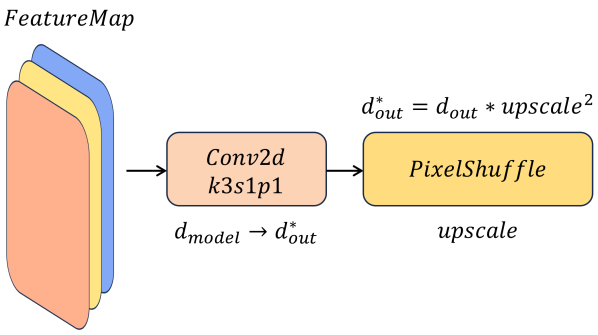

图 10: 网络的重建模块 upsample.

5.2.1 Image 类的定义

```
// Image.h
#pragma once
#define NOMINMAX
#undef min
#undef max
#include "stdafx.h"

class Image {
private:
    std::string image_dir;

public:
    Image(); // 构造方法1
    Image(std::string); // 构造方法2

    at::Tensor feature;
    cv::Mat feature_mat;

    std::string _image_dir();
    void load();
    void load(std::string);
    void ToTensor();
    void load_mat();
    void imshow(std::string);
    void save(std::string);
    void resize(int, int);
};
```

在 Image.h 文件中，我们声明了 Image 类包含的成员属性及函数。在此之中，**ToTensor()** 方法是一种高明的实现，此外，被主函数广泛使用的是公有成员属性 **feature**，以及公有成员函数 **load(std::string)**。保存图片时使用的是公有成员函数 **save(std::string)**，调用 **resize(int, int)** 函数同时修改公有成员属性 **feature** 及 **feature_mat**。我们接下来介绍该类中各个方法的定义。

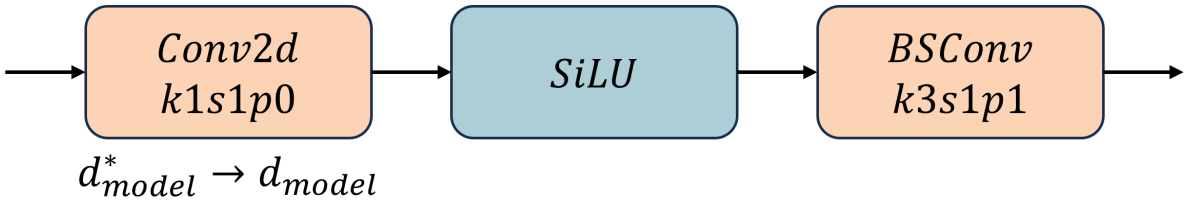

图 11: 特征图拼接之后的卷积组合 Conv_Concat.

无参构造函数 Image::Image() 以下是 Image 类的无参构造函数定义:

```
Image::Image() {
    str::string dir;
    std::cout << "requires a path to load image!";
    std::cin >> dir;
    this->image_dir = dir;
    this->load()
}
```

有参构造函数 Image::Image(std::string) 以下是 Image 类的有参构造函数定义:

```
Image::Image(std::string dir) {
    this->image_dir = dir;
    this->load();
}
```

根据以上两个构造函数的定义，不难发现 void Image::load() 方法是构造的关键，我们接下去介绍这一方法。

无参加载函数 Image::load() 以下是 Image::load() 函数的定义:

```
void Image::load() {
    this->load_mat();
    this->ToTensor();
}
```

load() 函数由两个子部分组成，一个是用于加载 this->feature_mat 的 load_mat() 方法，另一个是将 this->feature_mat 转为 at::Tensor 类并保存到 this->feature 的 ToTensor() 方法，我们在下文将一一介绍。

无参加载函数 Image::load_mat() 以下是 Image::load_mat() 函数的定义：

```
1 void Image::load_mat() {
2     this->feature_mat = cv::imread(this->image_dir);
3 }
```

load_mat() 方法通过调用 cv::imread() 函数来读取 this->image_dir 所存储的图像，该函数返回 cv::Mat 类型变量，我们将其赋值给 this->feature_mat 以保存读取的图像。

有参加载函数 Image::load(std::string path) 以下是该函数的定义：

```
1 void Image::load(std::string path) {
2     this->image_dir = path;
3     this->load();
4 }
```

有参加载函数 load(std::string path) 用于读取新的图片，通过覆写 this->image_dir 来减少对内存的占用，在之后调用类的无参加载函数 load() 自动根据 this->image_dir 更新对象属性。

保存函数 Image::save(std::string save_dir) 以下是该函数的定义：

```
1 void Image::save(std::string save_dir) {
2     cv::imwrite(save_dir, this->feature_mat);
3 }
```

通过调用 cv::imwrite() 函数来将 this->feature_mat 的信息存入指定的保存位置。

显示函数 Image::imshow(std::string window_name) 以下是该函数的定义：

```
1 void Image::imshow(std::string window_name) {
2     cv::imshow(window_name, this->feature_mat);
3 }
```

通过调用 cv::imshow() 函数来将 this->feature_mat 的信息以图像形式显示。

修改函数 Image::resize(int h, int w) 以下是该函数的定义：

```
void Image::resize(int h, int w) {
    cv::resize(this->feature_mat, this->feature_mat,
        cv::Size(w, h), 0, 0, cv::INTER_AREA);
    this->ToTensor();
}
```

我们通过调用 cv::resize 函数来修改读取的图片的尺寸，在之后调用 ToTensor() 方法，自动将修改后的 this->feature_mat 的信息保存到 this->feature 中。

转换函数 Image::ToTensor() 以下是该函数的定义：

```
void Image::ToTensor() {
    cv::Mat img = this->feature_mat.clone();
    int h = img.rows;
    int w = img.cols;
    int channels = img.channels();

    at::Tensor img_t;
    img_t = torch::from_blob(img.data,
                { 1, h, w, channels }, at::kByte);
    img_t = img_t.permute({ 0, 3, 1, 2 });
    img_t = img_t.toType(c10::kFloat).div(255.0);
    img_t.to(c10::DeviceType::CPU);

    this->feature = img_t;
}
```

实现 at::Tensor 和 cv::Mat 类型间的互相转换是实现 C++ 部署深度学习模型的关键一步，网络上常见的方法是创建一个与源 cv::Mat 形状相同的 at::Tensor，然后逐点赋值。这样的操作显然是笨拙的，而且对于具有二次复杂度的高分辨率的图像，赋值操作的时间是无法被忽视的。我们提出了一种更为高明的方法来实现两种类型的转换，这大大提升了我们运行的速度并且节省了内存的使用。

我们后续的操作基于对 feature_mat 的深拷贝。我们构造一个与 feature_mat 大小相同但多了一维的 at::Tensor img_t 并直接将 feature_mat 的值拷贝给 img_t，之后，从 OpenCV 默认的 BGR 色彩格式转换为 RGB 色彩格式，并且将每个像素值范围从 0-255.int 转为 0-1.float，最后将 img_t 转入目标设备，并且将其赋值给 this->feature，这样就完成了 ToTensor() 的操作。

5.2.2 BaseModel 类的定义

```
// BaseModel.h
#pragma once
#define NOMINMAX
#undef min
#undef max
#include "stdafx.h"

class BaseModel {
private:
    torch::jit::script::Module body;
    std::string MODEL_DIR;
    void load();

public:
    BaseModel(std::string);
    std::string _MODEL_DIR();
    at::Tensor forward(at::Tensor);
    at::Tensor forward(std::vector<torch::jit::IValue>);
};
```

在 BaseModel.h 文件中，我们声明了 BaseModel 类包含的成员属性及函数。在此之中，被主函数广泛使用的是其两种 **forward()** 方法，我们将在下文详细介绍。此外，BaseModel 类的私有成员包含一个 Module 类的对象 **body** 以及一个 std::string 类的对象 **MODEL_DIR**，分别代表模型的主体以及模型文件所在磁盘的位置。

有参构造函数 BaseModel::BaseModel(std::string dir) 以下是该函数的定义：

```
BaseModel::BaseModel(std::string dir) {
    this->MODEL_DIR = dir;
    this->load();
}
```

我们将传入的形参 dir 赋值给 BaseModel 对象的私有成员 MODEL_DIR，之后调用私有成员函数 load() 来根据指定地址加载模型。

加载函数 BaseModel::load() 以下是该函数的定义：

```
void BaseModel::load() {
    this->body = torch::jit::load(this->MODEL_DIR);
}
```

我们使用 LibTorch 库中的 torch::jit::load() 函数来根据对象自身的属性 MODEL_DIR 来载入模型，并将结果储存于 BaseModel 类对象的 body 属性中。

推理函数 BaseModel::forward(at::Tensor input) 以下是该函数的定义：

```
at::Tensor BaseModel::forward(at::Tensor input) {
    std::vector<torch::jit::IValue> inputs;
    inputs.push_back(input);
    at::Tensor output = this->body.forward(inputs);

    return output.toTensor();
}
```

由于 torch::jit::script::Module 类的对象的 forward() 方法不能直接传入 at::Tensor 类参数，因此我们定义一个元素类型为 torch::jit::IValue 的 std::vector 容器，用于存储输入的 at::Tensor 数据。之后我们调用 push_back() 方法将输入的 at::Tensor 数据存入容器中。我们调用 BaseModel.body 属性，并使用其 forward() 方法，此时我们定义的 vector 容器便可以作为参数传入模型供模型推理。在最后，我们将推理得到的结果调用.toTensor() 方法，使其转为 at::Tensor 类型，并返回。

推理函数 BaseModel::forward(std::vector<torch::jit::IValue> inputs) 以下是该函数的定义：

```
at::Tensor BaseModel::forward(vector<IValue> inputs) {
    at::Tensor output = this->body.forward(inputs);

    return output.toTensor();
}
```

我们对上文提到的 BaseModel.forward() 方法进行重载，主要区别在于接收参数的不同。如果接收的参数是 at::Tensor 类的对象，那么我们需要将其作类型转换才能调用 Module.forward() 方法；但如果接受的参数是 Module.forward() 方法可以接收的参数，那么我们可以直接调用这一方法而无需作任何的类型转换。

5.2.3 MainWindow 类的定义

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    QImage gray(QImage image);//灰度化
    QImage setRGB(QImage image,
        int value_r,int value_g,int value_b);//调整RGB
    QImage AdjustContrast(QImage image, int value);
    QImage ImageCenter(QImage  qimage,QLabel *qLabel);
    QImage AdjustSaturation(QImage image, int value);
    QImage bianyuan(QImage image);//边缘检测
    QImage  MatToQImage(const cv::Mat& mat);
    QImage junzhi(QImage image);//均值滤波
    QImage fuhe(QImage images);
    QImage gamma(QImage image);
    ...

private slots:
    void on_pushButton_save_img_sr_clicked();
    void on_pushButton_bic_clicked();
    void on_pushButton_sr_clicked();
    ...

private:
    Ui::MainWindow *ui;
    QString origin_path;//目前处理的图片的原图路径
    Image img;
    cv::Mat sr_img;
    ...
}
```

由于 MainWindow 类定义了过多的成员函数和变量，我们只展示其中一部分。MainWindow 类继承 QMainWindow 类，并基于超类的框架定义了如灰度化、调整 RGB、边缘检测等功能函数，以及用于超分辨率存储的 img 和 sr_img 对象。

构造函数 MainWindow::MainWindow(QWidget *parent) 以下是该函数的定义：

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
    , beishu(1)
    ,delay(0)
{
    ui->setupUi(this);
    ...
}
```

该函数主要实现了 ui 界面的建立，以及输出提示性信息，这部分信息在正文中省略。

析构函数 MainWindow:: MainWindow() 以下是该函数的定义：

```
MainWindow::~MainWindow()
{
    delete ui;
    capture.release();
}
```

主要实现了 ui 界面的释放以及捕获的释放。

均值滤波 QImage MainWindow::junzhi(QImage image) 以下是该函数的定义：

```
QImage MainWindow::junzhi(QImage image){
    int kernel[3][3] = {
        {1,1,1},
        {1,1,1},
        {1,1,1}};
    int sizeKernel = 3;
    int sumKernel = 9;
    ...Convolution...
    return image;
}
```

该函数首先定义了一个用于 3×3 大小的的 kernel，其核上每个点的权重都为 1，在使用该核对输入图像进行卷积操作之后，对生成图像的每个像素点的值除以 sumKernel，相当于使用权重为 $\frac{1}{9}$ 的卷积核进行滤波，即一次均值滤波操作。

双三次插值放大 MainWindow::on_pushButton_bic_clicked() 以下是该函数的定义：

```
void MainWindow::on_pushButton_bic_clicked(){
    if(ui->label_show->pixmap() != nullptr){
        cv::Mat upscale;
        Size dsize = Size(2 * img.feature_mat.cols,
                          2 * img.feature_mat.rows);
        cv::resize(img.feature_mat, upscale,
                   dsize, 0, 0, cv::INTER_CUBIC);
        sr_img = upscale.clone();
        ...Show Image...
    }
    else{
        ...Warning...
    }
}
```

在该函数中，我们先对程序是否已经打开图片做出判断：如果 pixmap() 不是一个空指针则代表已经打开了图片。我们对这张打开的图片进行双三次插值的二倍放大，先定义了一个 cv::Mat 类对象 upscale 用于存储放大后的图像，再通过调用 cv::resize 函数来修改实现双三次插值。最后我们将 upscale 的深拷贝赋值给 MainWindow 对象的 sr_img 属性，以达到存储的目的。

超分辨率 MainWindow::on_pushButton_sr_clicked() 以下是该函数的定义：

```
void MainWindow::on_pushButton_sr_clicked(){
    if(ui->label_show->pixmap()!=nullptr){
        at::Tensor feature = img.feature.clone();
        at::Tensor output = Model.forward(feature);
        sr_img = tensor2Mat(output);
        ...Show Image...
    }
    else{
        ...Warning...
    }
}
```

与双三次插值放大函数类似地，我们先检查程序是否已经打开图片。在 pixmap() 不是

空指针的情况下我们通过调用 BaseModel 类对象的 forward() 方法来实现对 img 的放大，并将输出 Tensor 通过 tensor2Mat 函数转为 cv::Mat 并存储于 sr_img 属性中。

值得注意的是，我们传入网络的并不是 img.feature 本身，而是它的深拷贝。这样处理的目的在于防止网络的前向传播直接修改 img.feature 的值。

5.2.4 其它重要函数

转换函数 cv::Mat tensor2Mat(at::Tensor& t) 以下是该函数的定义：

```
cv::Mat tensor2Mat(at::Tensor& t) {
    at::Tensor tmp = t.clone();
    tmp = tmp.squeeze(0).detach()
    tmp = tmp.permute({ 1, 2, 0 });
    tmp = tmp.mul(255).clamp(0, 255)
    tmp = tmp.to(torch::kU8);
    tmp = tmp.to(torch::kCPU);
    int h_dst = tmp.size(0);
    int w_dst = tmp.size(1);

    cv::Mat mat(h_dst, w_dst, CV_8UC3);
    std::memcpy((void*)mat.data, tmp.data_ptr(),
                sizeof(torch::kU8) * tmp.numel());
    return mat.clone();
}
```

与 Image 类中定义的成员函数 Image::ToTensor() 类似地，我们使用一种高明的方式来实现 at::Tensor 类对象到 cv::Mat 类对象的转换，即利用 cv::Mat 类对象的.data 指针进行值拷贝，这样可以大大减小类型转换的时间。

我们首先将传入的希望拷贝的 at::Tensor 类对象进行深拷贝并赋值给函数的局部变量 tmp，这样做的目的是**防止后文转换 cv::Mat 和 at::Tensor 值的尺度时对传入的源 Tensor 造成修改**。我们之后将 tmp 从 4D 变量压缩成 3D 变量，再调用.detach() 方法来分离值和梯度，这是因为**梯度属性与值属性无法在单个 cv::Mat 中存储**。再之后，我们将 RGB 色彩模式转换为 OpenCV 默认的 BGR 色彩模式，我们再将 tmp 每个元素的值的范围从 0-1 转换到 0-255，最后通过调用 memcpy 函数进行转换操作，转换的目标内存空间是我们定义的局部变量 cv::Mat mat 的内存空间。最后我们返回 mat 的深拷贝，即完成类型转换。

6 结语

计算机科学是一门极其考验思维水平、工程能力以及持续努力能力的学科。作为计算机专业的学生，我想：在具体的项目中动手实践能获得的提升，会远远多于在课堂上听取知识带来的提升，而这样的想法也使我义无反顾地开始投入精力到这样一个对于大一学生来说复杂无比的项目，并将它作为我的高级语言程序设计课程大作业。

人类并不是无所不能的，但科研的精神便是：我们会发现有哪些问题是我们所不能的，然后想办法来让我们 Be able to。摄影技术在百年以前便已经出现在世界上，但由于光学技术、材料技术等还仍不成熟，因此长达几十年的时间里，人类所记录下的照片都是语义信息残缺，或者模糊的。我们如何从这些低质量的照片中，尽可能地恢复出彼时真实的情景？这样的问题促使着 Low-Level Vision（底层视觉）不断推进研究：为了解决模糊问题，我们提出了图像超分辨率；为了解决拍摄过程中由于采样速率不同而产生摩尔纹的问题，我们提出了图像去摩尔纹；为了解决带雾的图片难以看清的问题，我们提出了图像去雾... 这些问题人类本来是没有能力解决的，但我们不会因为一开始无法解决，就认定自己永远都无法攻克这一问题；反之，人类在几千年的历史中，总是自信地向前走着，带着一种奔向星辰大海，探尽宇宙奥秘的决心向前走着。我们不断提出问题，不断解决问题，科学技术也因此而发展。

我将在接下来的大学生涯中继续用这样一种科研精神来提升自己，究世物之根本，解古今之难题。

付嘉晨

2024 年 5 月 15 日

参考文献

[1] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[5] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13733–13742, 2021.

[6] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11963–11975, 2022.

[7] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[10] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted

windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[11] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2015.

[12] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016.

[13] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 136–144, 2017.

[14] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. Image super-resolution using very deep residual channel attention networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 286–301, 2018.

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[16] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[17] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *Proceedings of the European conference on computer vision (ECCV) workshops*, pages 0–0, 2018.

[18] Fuzhi Yang, Huan Yang, Jianlong Fu, Hongtao Lu, and Baining Guo. Learning texture transformer network for image super-resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5791–5800, 2020.

[19] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1833–1844, 2021.

[20] Yupeng Zhou, Zhen Li, Chun-Le Guo, Song Bai, Ming-Ming Cheng, and Qibin Hou. Srformer: Permuted self-attention for single image super-resolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12780–12791, 2023.

[21] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[22] Daniel Haase and Manuel Amthor. Rethinking depthwise separable convolutions: How intra-kernel correlations lead to improved mobilenets. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14600–14609, 2020.

[23] Zheng Hui, Xiumei Wang, and Xinbo Gao. Fast and accurate single image super-resolution via information distillation network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 723–731, 2018.

[24] Zheng Hui, Xinbo Gao, Yunchu Yang, and Xiumei Wang. Lightweight image super-resolution with information multi-distillation network. In *Proceedings of the 27th acm international conference on multimedia*, pages 2024–2032, 2019.

[25] Zheyuan Li, Yingqi Liu, Xiangyu Chen, Haoming Cai, Jinjin Gu, Yu Qiao, and Chao Dong. Blueprint separable residual network for efficient image super-resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 833–843, 2022.

[26] Fangyuan Kong, Mingxi Li, Songwei Liu, Ding Liu, Jingwen He, Yang Bai, Fangmin Chen, and Lean Fu. Residual local feature network for efficient super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 766–776, 2022.

[27] Mingming Cheng. 粒度自适应的图像感知技术. https://mmcheng.net/report/.

[28] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets,

atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[29] Panqu Wang, Pengfei Chen, Ye Yuan, Ding Liu, Zehua Huang, Xiaodi Hou, and Garrison Cottrell. Understanding convolution for semantic segmentation. In *2018 IEEE winter conference on applications of computer vision (WACV)*, pages 1451–1460. Ieee, 2018.