

本科毕业设计

|  |  |
| --- | --- |
| **题目：** | 基于深度卷积神经网络的图像融合算法设计 |
|  |  |

|  |  |
| --- | --- |
| **学 院:** | 机械自动化 |
| **专 业:** | 测控技术与仪器 |
| **学 号:** | 201603191006 |
| **学生姓名:** | 朱燊然 |
| **指导教师:** | 朱攀 |
| **日 期:** | 二○二○年五月 |

摘 要

多聚焦图像融合是图像融合领域的一个重要分支，已经广泛地被应用在许多领域之中。多聚焦图像进行融合的研究有着重要意义，多聚焦图像融合技术的目的是将多张同一场景下的不同聚焦的图像进行融合，以使得大部分区域清晰。在基于空间域的图像融合方法中，大多数方法的聚焦评价都是根据人为进行设定带有人为的主观因素。

主要研究内容为：基于现有研究成果上探讨使用卷积神经网络对已经配准了的图像进行多聚焦图像进行融合。主要思想：使用卷积神经网络在构建的数据集上训练得到模型，使用模型对各个待融合的多聚焦图像进行评价，获得每个像素的聚焦程度值组成每张待融合图像的聚焦评价图，根据该图进行融合决策得到最终的融合图像。

使用该方法对多聚焦图像进行融合可以得到相较于任一待融合图像都更为清晰的图像。对于本文的方法，从主观上和客观指标上与其它方法进行对比，在部分图像客观的评价上可以获得相比较于其他方法较优的结果，相较于其他基于空间域方法具有更好地鲁棒性。

关键词： 图像融合； 卷积神经网络； 深度学习

**Abstract**

Multi focus image fusion is an important branch of image fusion, which has been widely used in many fields. In real scenes, due to the optical characteristics of some lenses, there are often the following problems: in the same scene, there are many targets, but these targets are not always in the focus area of the lens, so it will lead to the situation that some areas of the image are clear and some areas are fuzzy. Therefore, the problem of fusing multiple images with different focus in the same scene to get a higher quality image arises.

The main research contents are as follows: Based on the existing research results, the convolution neural network is used to fuse the multi focus image of the registered image. Main idea: use convolution neural network to evaluate the multi focus image to be fused, obtain the focus degree value of each pixel to form the focus degree map of each image to be fused, and make fusion decision according to the map to get the final fused image.

Using this method to fuse the multi focus image can get a clearer image than any image to be fused. For the method in this paper, compared with other methods in subjective and objective indexes, we can get better results in objective evaluation of some images compared with other methods, and have better robustness compared with other spatial methods.

**Keywords**: image fusion; convolutional neural network; deep learning

目 录

[摘 要 I](#_Toc21509)

[Abstract II](#_Toc17250)

[目 录 III](#_Toc12720)

[1 绪论 1](#_Toc9179)

[1.1 图像融合研究背景及意义 1](#_Toc26935)

[1.2 多聚焦图像融合主要算法 2](#_Toc11884)

[1.3 主要评价指标 3](#_Toc25504)

[1.4 存在的问题 5](#_Toc10105)

[1.5 本文主要工作 5](#_Toc7103)

[2 卷积神经网络设计 6](#_Toc232)

[2.1 卷积神经网络概述 6](#_Toc18220)

[2.2 聚焦评价网络结构设计 7](#_Toc8214)

[2.3 神经网络训练集构建 10](#_Toc11114)

[3 多聚焦图像融合 11](#_Toc15272)

[3.1 融合方法思想 11](#_Toc22258)

[3.2 融合方法流程 11](#_Toc14580)

[3.3 实验结果 14](#_Toc28541)

[3.4 小结 18](#_Toc3494)

[结束语 19](#_Toc10129)

[参考文献 20](#_Toc3976)

[致 谢 21](#_Toc15788)

[附 录 22](#_Toc11161)



1 绪论

1.1 图像融合研究背景及意义

随着计算机技术的发展，使得运算能力得到大幅提升，这直接促使了图像处理领域的研究更加快速的展开、图像处理技术快速发展[1]。

图像融合是机器视觉的一项重要内容，其目的十分明确，即将两张或者多张图像进行融合得到具有更高信息价值的融合图像。图像融合的本质是将多图像进行互补，将有效信息集中展现在一张图像中。图像融合在许多领域中扮演了极其重要的角色。在可见光与红外光的融合中，在较暗的光线下，可见光可以表现出环境的轮廓，但可见光难以清晰地表现出人的轮廓，而由于红外光无法呈现出能量相同的环境背景，但可以在较暗的光线下清晰地呈现出高能量的物体，因此将可见光与红外光进行融合，可以将两张图像的信息融合在一起，表现出可见光图像和红外光图像各自清晰的轮廓。在遥感领域，进行多光谱图像和全色图像进行融合，多光谱图像分辨率低但每个光谱图像可以表现出遥感对象的不同的显著特征，全色图像分辨率高，但只有灰度信息。由于多光谱图像的分辨率低于全色图像但多光谱图像有色彩意义，因此将多光谱图像与全色图像融合可以获得高分辨率的彩色意义的图像，可以更高效率地对信息进行判断，提高工作效率。在医学领域，许多医学成像如B超等都是灰度图像，医生需要根据多种医学图像对病人的病情进行判断，因此，将多张图像融合为融合图像，可以显著提高医生对病人疾病的诊断效率，大大提高医疗效率。图像融合在国防领域、军事领域也被广泛应用，在侦查上提高侦查图像判别效率和侦查图像判别准确度。图像融合于八十年代前就已经开始被应用。但多应用与多光谱卫星的遥感图像中。直到八十年代以后，图像融合才开始被其他领域所重视，随后被广泛应用于各个领域之中。早期的图像融合算法简单直接，但融合效果较差，新世纪以来，融合算法的研究取得重大进展，融合的性能得到了大幅提高。截止目前，以美国、德国等欧美国家为代表，已经将图像融合技术应用于商用场合、军事场合之中。并且欧美国家在图像融合算法的研究中处于领先水平。我国对于图像融合算法的研究和应用均较晚，在算法的研究上与国外先进水平仍然存在一定的差距，处于追赶的地位。尽管如此，我国在图像融合算法的研究和应用上仍然取得了巨大成就。

多聚焦图像融合是图像融合领域的一个重要分支，本文重点对多聚焦图像融合问题进行讨论。在现实的场景之中，由于一些镜头的光学特性，往往会出现以下问题：在同一场景下拥有着多个目标，但这多个目标并不总在镜头的聚焦区域之中，因此将会导致图像呈现出部分区域清晰、部分区域模糊的情况。由此产生将多幅同一场景下不同聚焦的图像进行融合得到更高质量的图像的问题。

1.2 多聚焦图像融合主要算法

1. 加权平均法

平均法图像融合主要将两张或者多张图像直接将每个对应的像素按平均系数进行相加融合，得到的图像即为融合图像。

基于平均法的图像融合的主要优点：简单易行，编程效率高，融合计算速度极快。与之相对的是，平均法图像融合缺点也十分明显：图像融合质量不高，由于平均作用，极容易产生模糊图像，因此极少情况下会采用平均法进行图像融合。

1. 局部梯度法

空域法图像融合方法是基于图像空域来处理的。 早期空域法是把图像分块，然后比较原始图像对应位置的聚焦程度，聚焦值大的块作为融合图像的相应块[2]。

对于原始图像，为得分图，其大小与原始图像相同，初始值都为0。以块的方式进行计算，对每个块内的和作为指标，如果图像1的块梯度累积和大于图像2块梯度累积和，则中对应块中的所有元素加1，如果。以滑窗的方式计算每个像素点的对应窗内的图像梯度的累积值作为得分图，选取两张待融合图像中得分较大的作为融合像素点。

对于原始图像1，为像素聚焦评判图：

|  |  |
| --- | --- |
|  | (1.1) |

对于原始图像2，为像素聚焦评判图：

|  |  |
| --- | --- |
|  | (1.2) |

根据和进行决策，选取聚焦像素作为融合像素。

1. 小波变换法

小波变换被广泛地应用在不同的领域之中，包括图像处理领域，小波变换在该领域中主要研究如何进行图像压缩、图像融合[3]。

基于小波分解的图像融合：基于小波变换分解的图像融合主要有两种方式，一是单系数融合，二是区域系数融[4]。

单系数融合：不断地对小波的低频系数部分进行分解，可以得到基于小波变换的多分辨率金字塔。我们认为高频对应着细节，因此，对于多聚焦图像对应的小波系数中，在低频段中直接将小波系数平均求和，在高频段中选取绝对值最大的作为融合小波系数。将最终得到的融合小波系数进行小波逆变换获得融合图像。

区域系数融合：区域系数融合认为待融合的多聚焦图像的高频区域能量越高，对应的聚焦程度越高，因此对于某一个系数，选择区域能量最大的图像的系数作为融合系数。对于低频区域，由于我们认为低频对应模糊，因此仍然采用与单系数融合相同的方式进行融合，即进行平均加权融合。

1. 拉普拉斯金字塔法

基于拉普拉斯金字塔的图像融合[5]：对源图像进行拉普拉斯金字塔进行分解，得到的拉普拉斯金字塔图像，在金字塔图像集合中，顶层是低频分量，其他层是该层的高频分量。我们认为低频分量对应模糊区域，因此直接对金字塔顶部图像对应的像素进行平均。对于非顶部的图像，我们认为高频分量代表了图像的聚焦区域，因此图像中值较大则表明该值更可能是聚焦区域，所以选取值较大对应的作为融合系数。

1.3 主要评价指标

在图像融合领域中，除了设计对图像进行融合的算法外，对图像进行客观有效的评价也是图像融合研究的一项重要内容，至今已经发展了诸多方法对其进行评价[6]。

1. 标准差(STD)

标准差概念来源于概率论统计学中，主要用于度量离散变量的离散程度。标准差被广泛地应用于统计领域，其主要目的是对数据的离散程度进行有效度量。标准差可以通过某个变量的一系列采样取值求出，但在图像中，可以使用灰度直方图来表示变量采样。

对于标准差评价，我们认为，清晰的图像中的各个像素点拥有较大的离散程度。因此可以使用方差来度量图像的清晰程度。

|  |  |
| --- | --- |
|  | (1.3) |

1. 信息熵(IE)

信息熵来自于信息领域。熵值可以表征出变量的不确定度，不确定度越高承载的信息量越大。信息熵越大，则随机事件的信息量越大，该随机事件的不确定度越高。

对于信息熵评价，我们认为，由于清晰的图像比模糊图像有更多的信息论。因此我们认为直方图拥有较大的熵值，所以可以使用熵值来度量图像的清晰程度。我们可以使用图像的直方图作为变量。

|  |  |
| --- | --- |
|  | (1.4) |

1. 平均梯度(AG)

平均梯度是图像融合研究中普遍使用的评价指标之一。均值滤波的角度出发，我们可以看到均值滤波后的图像清晰程度降低，更加模糊化，而均值滤波会降低图像梯度。因此我们可以假设，图像梯度越大，则清晰程度越高。实际研究也表明，图像的梯度越大，则代表图像的像素值的变化越剧烈。而且实验中我们观察到图像的清晰度越高，往往其梯度越大。

1. 互信息(MI)

互信息是一个来源于香农的信息论的概念，互信息可以被主要概括为：某一事件对于另一事件的不确定度的减小程度。假如两个事件相互之间是独立的不相关的，那么这一事件对于另一事件是没有任何信息的，即另一事件的不确定度没有被降低，反之如果两个事件有着深刻而明显的联系，则这一事件可以极大地减小另一事件的不确定度。因此，互信息可以被用于度量图像之间的关系，我们可以将图像的像素灰度作为变量，进行互信息的度量。变量X和变量Y的互信息的定义如下：

|  |  |
| --- | --- |
|  | (1.5) |
|  | (1.6) |

1. 均方根误差(RMSE)

均方根误差可以被用于度量融合图像与参考图像的相似程度，均方根误差越小

说明平均像素误差越小，融合效果越好。均方根误差定义为：

|  |  |
| --- | --- |
|  | (1.7) |

1. 峰值信噪比(PSNR)

峰值信噪比可以作为图像评估质量指标的一个客观标准，在图像领域是一种基础的度量方法。虽然峰值信噪比相比于结构相似度有着一定的局限性，但它仍然是一种十分重要的评价方法，仍然被广泛应用。其计算方式决定了它是一种全参考的评价方式，其主要计算方法如下所示：

|  |  |
| --- | --- |
|  | (1.8) |

峰值信噪比定义为：

|  |  |
| --- | --- |
|  | (1.9) |

1.4 存在的问题

作为图像融合领域中的一个重要分支，多聚焦图像融合算法也可以在大致上划分为两个大类：一是基于空间域的图像融合，二是基于变换域的图像融合。

基于空间域进行多聚焦图像融合由于可以直接在原始的图像上进行，相较于基于变换域的多聚焦图像融合更具有时间复杂度上的优势。因此，基于空间域的方法技术的应用更为广泛。

多数的基于空间域的方法，都依赖于图像的聚焦评价，图像的聚焦评价方法决定了图像融合性能。但是大多数方法的聚焦评价都是根据人为进行设定的，比如基于局部方差或者基于局部梯度等对图像进行聚焦评价，由此将会带来的某些问题，即这些度量包含了人的主观因素，因此这些度量并不能确保准确地评价聚焦。本文针对此问题，通过引入一个卷积神经网络模型，使用该网络模型对一个包含聚焦与非聚焦的图像数据集进行训练学习，使得卷积神经网络可以自动提取聚焦和非聚焦特征，进而较好地排除人工设置特征的主观影响，提高多聚焦融合的有效性。

1.5 本文主要工作

(1) 对多聚焦图像融合的现状进行一般性的阐述。主要阐述了典型常用的多聚焦图像融合方法和图像融合质量指标。

(2) 在本文中提出了构建训练集的方法，使用构建的卷积神经网络模型可以对该训练集进行训练，从而对模型参数进行估计以获得预测能力。

(3) 提出了一种基于空间域的并且基于卷积神经网络的多聚焦图像融合方法，实验表明该方法具有较好的图像融合性能。

2 卷积神经网络设计

2.1 卷积神经网络概述

深度学习是目前人工智能领域的热点领域。深度学习中典型的多层结构的思想来源于人脑的分层结构[7]。在人脑感知过程中，这种分层结构可以从各种各样的信息中有效地对特征进行提取。深度学习本质上是通过构建一个具备多个层次结构的模型对函数进行拟合，通过使用大量的样本进行训练，可以获取到数据样本中有效的信息，并将这些信息用于实际的预测中[8]。

卷积神经网络是深度学习模型的一个主要分支，卷积神经网络已经成为了图像处理领域中的热点。与全连接神经网络相比，卷积神经网络的主要特点体现在其没有使用全连接机制而是使用了权重共享机制。权重共享机制可以极大地减少参数数量，可以在极大地减小网络模型复杂度的同时不会对模型性能造成过大的损失[9]。特别是在面对图像这样的超高维度输入数据时的效果十分显著。在图像处理领域，特征提取一直以来都是一项繁杂的工程。特征的提取需要人工进行设计，如Haar特征、LBP特征、边缘特征等相关的特征。而将卷积神经网络应用在图像处理中，可以将繁重的特征工程任务交给了网络模型。卷积神经网络是一种数据驱动型的算法，其可以自动学习数据集合中的特征[10]。这说明卷积神经网络可以更为容易的应用于不同的场景之中，而不需要根据不同的场景手动地选择不同的特征进行处理。在文献[11]中提到，一定深度的卷积神经网络的特征学习是从低层次到高层次的，其首先学习到的是图像的低层次特征，如边缘和颜色等，然后在中层学习到局部的信息，最终在高层学习到高级特征。在使用卷积神经网络进行学习时，其中的一个重要基础是反向传播算法，其目的是通过训练样本集和样本的目标值来对参数进行估计，反向传播算法可以不断地对参数进行有效调整从而降低网络的误差，达到使用网络进行预测的目的[12]。

虽然卷积神经网络在机器学习领域拥有着巨大的潜力，但由于其对于算力资源的要求巨大，因此并没有马上得到广泛的应用。近年来，计算机技术特别是硬件性能的的快速发展提高，为卷积神经网络的决定性突破创造了有利条件。直到2012年，文献[13]中使用GPU对神经网络进行训练以及使用了Dropout技术和图像增强技术，首次展现出其在图像处理领域的强大能力，从此卷积神经网络不断迭代性能也不断提高，越来越多基于卷积神经网络的优秀架构不断地被学者们提出。

在文献[14]中，研究了神经网络的宽度和深度关于网络拟合能力的影响，该研究指出，神经网络的深度相比较于深度具有更好的性能，可以更好的提升模型的精确度并且降低模型的过拟合效应。

2.2 聚焦评价网络结构设计

2.2.1 设计思路

在基于空间域进行融合时，需要对图像进行聚焦评价。在许多基于空间域的方法中，聚焦评价方法往往是人工设计的，因此这些聚焦评价方法带有了一定的人为主观特性。使用卷积神经网络来对特征进行提取，并通过构建一个样本集合来训练网络使得其具有聚焦评价功能可以在较大程度上降低人为的主观因素影响。

由于网络的深度可以更好地提升网络的性能，因此我们倾向于使用更为深层的网络结构。

由于网络具有自动学习到图像的特征的能力，所以我们认为将图像的边缘特征输入网络中作为先验的信息对于网络性能提升是有效的。我们设计将图像的灰度图像、图像的边缘图像作为输入。

2.2.2 设计过程

本文思想是通过构建一个评价函数，即一个评判模型FCNN。假设待融合图像为A和B，和对应，需要通过模型FCNN分别给出和的聚焦评价指标。

像素点的聚焦与否显然与像素点的局部区域有着很强的相关性，因此，卷积神经网络的感受野必须达到一定范围。

除此以外，我们认为图像的聚焦程度与图像的梯度具有相关性，因此我们对原图像的灰度图分别利用Sobel算子进行X方向和Y方向的卷积滤波。最终将原图像灰度图、X方向Sobel滤波图、Y方向Sobel滤波图作为网络输入。

为了验证使用Sobel算子提取人工特征对于网络性能提升的有效性，我们分别建造了两个网络模型进行训练。从图4.1中可以看出，在相同的超参数训练下，使用了Sobel算子提取人工特征的网络模型在训练速度上显然具有更好地表现，说明了图像的聚焦程度与图像的梯度具有相关性这个先验是有效的。

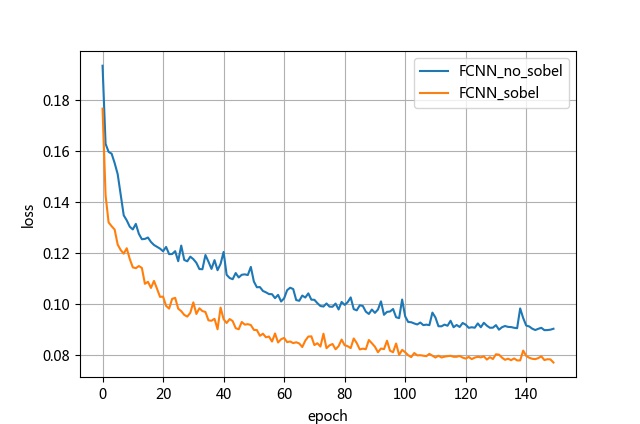


图2.1 Sobel算子有效性检验

像素点的聚焦与否显然与像素点的局部区域有着很强的相关性，因此，卷积神经网络的感受野必须达到一定范围。

由于模型需要达到一定的性能，因此需要兼顾速度性能、预测性能来选择网络的深度和网络的宽度。深度网络在性能上往往要比宽度网络更加强大。我们构建了多个不同深度不同宽度的网络进行性能比较，从图4.2中可以看出，深度网络具有更大优势。

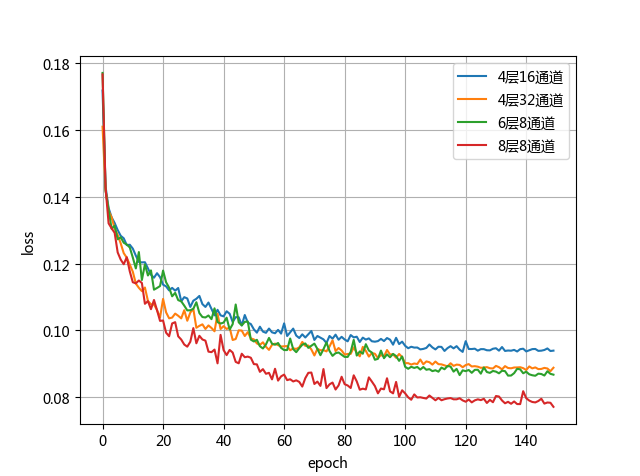


图2.2 不同深度与宽度网络性能对比

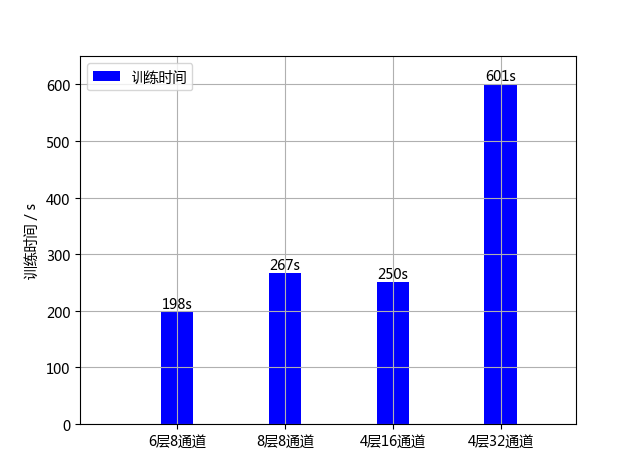


图2.3 不同深度与宽度网络性能对比

2.2.3 设计结果

最终的网络结构图(图3.4)。第1层输入通道为3，输出通道为8；第2到7层输入通道为8，输出通道为8；第8层输入通道为8，输出通道为1；每个卷积层均使用3x3卷积核；除最后一层使用作为激活函数外，其余层均使用作为激活函数。网络的感受野大小为15个像素。

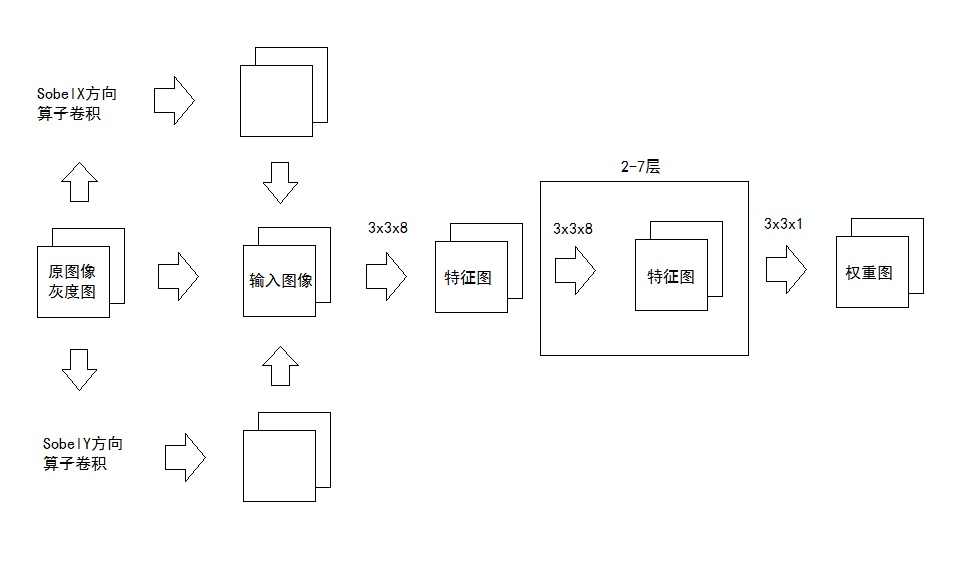


图2.4 网络结构

2.3 神经网络训练集构建

聚焦评价模型FCNN对图像进行评判，但为了获取模型的参数，需要进行训练，因此需要用到大量的训练数据集合。

由于在聚焦图像融合中，大部分图像融合并无标准参考图像，因此无法获得大量训练数据集。经过考察，在多聚焦图像中，非聚焦部分表现模糊，因此对图像进行模糊处理以近似非聚焦部分。并且采用多种模糊处理，包括高斯模糊、均值模糊、下采样处理。

对图像进行如图所示的模糊，作为训练集。左图为输入图像，右图为标签图像。训练一个网络使其对输入的图像中的模糊区域有较高响应值。

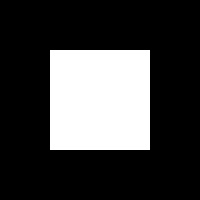


图2.5 训练样本示例

3 多聚焦图像融合

3.1 融合方法思想

由于基于空间域的图像融合相比较于基于变换域的图像融合在时间复杂度上具有更优的性能，因此采用空间域的方式进行融合。

在多聚焦图像融合中基于空间域的方法的重要一步是对原始图像进行聚焦评价。本文设计了一个卷积神经网络模型FCNN，并构建了一个包含有聚焦与非聚焦图像的训练图像集合(具体已由第2章详细描述)。我们使用卷积神经网络FCNN对聚焦与非聚焦图像集合进行训练学习，目的是通过训练使得卷积神经网络模型可以更加有效地提取出聚焦与非聚焦图像区域的特征信息，进而该模型可以对每一张源图像的每一个像素进行聚焦评价并分类。

算法主要的流程可以被概括为：使用卷积神经网络对每一张原始的多聚焦图像进行聚焦评价得到聚焦评价图像，将得到的聚焦评价图按照一定的决策方式对多聚焦图像进行融合，在此我们选择根据聚焦评价选择聚焦程度更高的像素作为融合像素，遍历每一个对应的像素重复以上操作得到最终的融合图像。

3.2 融合方法流程

本算法的核心思想是根据图像的聚焦评价图来进行加权融合，具体方法如下：

1. 在训练集上训练卷积神经网络模型。
2. 将待融合的多聚焦图像和图像输入中，得到聚焦评价图和聚焦评 价图。
3. 对于每一个对应的像素，根据聚焦评价图和，得出中每个

权重，得到权重图。权重的获取方式为：

|  |  |
| --- | --- |
|  | (3.1) |

1. 图像的权重图为：

|  |  |
| --- | --- |
|  | (3.2) |

1. 对图像和图像根据权重图进行图像融合，融合方式为：

|  |  |
| --- | --- |
|  | (3.3) |

1. 输出最终融合图像。



图3.1 输入图像1与输入图像2

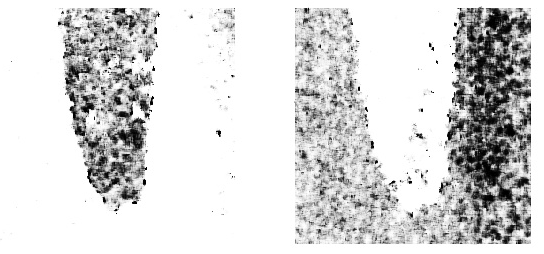


图3.2 图像1聚焦评价图和图像2聚焦评价图

但实验表明，如果直接使用聚焦评价图进行融合，可以发现，即使是在清晰的区域中，大片相连通的且颜色相近的局部区域也会被认为是非聚焦的模糊区域，其结果如下图所示。出现该现象的主要原因可能是：对于大片相连通的且颜色相近的区域，虽然它是实际上的图像聚焦区域，但在局部上看它属于低频区域，图像的低频区域的聚焦度可分性比较差。若根据第(3)步骤进行融合，将会出现以下问题：对于某个区域，虽然图像中该区域是明显属于聚焦区域、图像中该区域是明显属于非聚焦区域，但是有可能会将中非聚焦的像素融合进最终图像中。因此，不能直接使用该方法，需要进行改进。



图3.3 原始权重图

针对以上问题，对以上融合算法进行改进。图像中存在着网络模型难以区分的聚焦程度的局部区域。通过观察，我们可以发现这部分错误判断的聚焦区域表现为区域上的不连续性，可以通过更大的区域范围综合判断出该局部区域是否属于聚焦区域，如果该图像局部区域属于聚焦区域但却被网络划分为非聚焦区域（或者相反），那么表现为类似包含有噪声图像。以此作为依据，对算法进行改进，我们将权重图进行下采样，使用中值滤波将其过滤，最终上采样恢复原始权重图大小，可以有效消除该噪声。

根据以上思想，提出以下改进方法进行多聚焦图像融合：

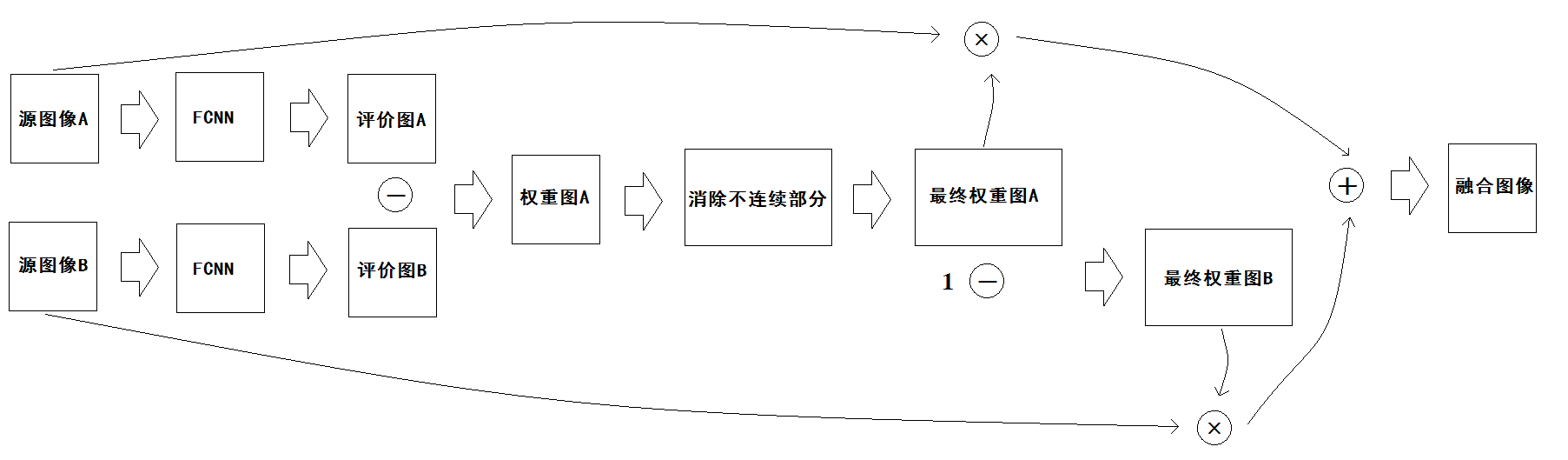


图3.4 流程图

(1) 在训练集上训练卷积神经网络FCNN模型。

(2) 将待融合的多聚焦图像A和图像B输入中，得到聚焦评价图和聚焦评 价图。

(3) 对于每一个对应的像素，根据聚焦评价图和，输出A中每个像素的权重，得到权重图。权重的获取方式为：

|  |  |
| --- | --- |
|  | (3.4) |

1. 对进行下采样得到，对进行中值滤波得到，最后对进行 上采样返回到原始尺寸大小，最终得到图像A的权重图。
2. 图像的权重图为：

|  |  |
| --- | --- |
|  | (3.5) |

1. 对图像A和图像B根据权重图进行图像融合，融合方式为：

|  |  |
| --- | --- |
|  | (3.6) |

1. 输出最终融合图像Y。

图3.5 权重图 图3.6 权重图 图3.7 权重图

3.3 实验结果

3.3.1 模型训练

由于模型本质上是一个回归函数，对输入进行回归。因此我们可以选用MSE函数作为损失函数，我们作如下定义：



y为清晰度评判参考图像矩阵，x为输入待评判图像矩阵，F为网络模型，为模型参数，n为训练样本数量。我们需要将J最小化来对网络模型F进行优化得到最终的网络模型。除此以外，由于使用到Sigmoid激活函数，而该激活函数的有效梯度域在0附近区间，因此我们需要对输入图像进行归一化操作，将图像矩阵范围[0,255]归到[0,1]之间。

模型训练使用基于CPU的Pytorch[15]进行训练，使用基于随机梯度下降的自适应学习率算法Adam[16]。分三批进行训练，每批次训练50轮，每次的学习率分别为，，。最终得到的模型均方差为。

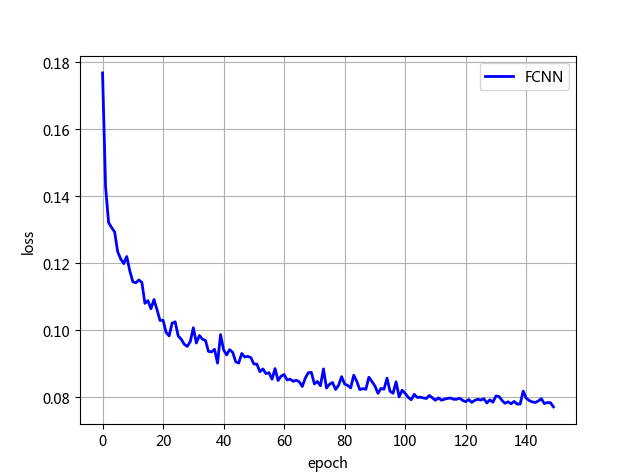


图3.8 训练误差

3.3.2 评价指标

我们选取峰值信噪比(PSNR)、均方误差(RMSE)、均方差(STD)、图像熵(IE)一共六个指标，作为图像融合质量的客观评价指标。这四个指标已经在第一章中详细阐述。

3.3.3 融合结果分析

为了充分体现出本文所提出的方法的有效性，我们决定对同一图像使用多种不同的方法进行融合。在该实验中，我们使用均值法、梯度法、小波变换系数法、小波变换局部系数法、拉普拉斯金字塔法进行融合对比。

图3.9中，(a)为左模糊图像，(b)右模糊图像，(c)为标签图像，(d)(e)(f)(g)(h)为对比所使用的算法得到的多聚焦图像融合结果图像，(d)平均法，(e)梯度法，(f)小波系数法，(g)小波局部系数法，(h)拉普拉斯金字塔法，(i)为本文章所提出的算法得到的多聚焦图像融合结果图像。

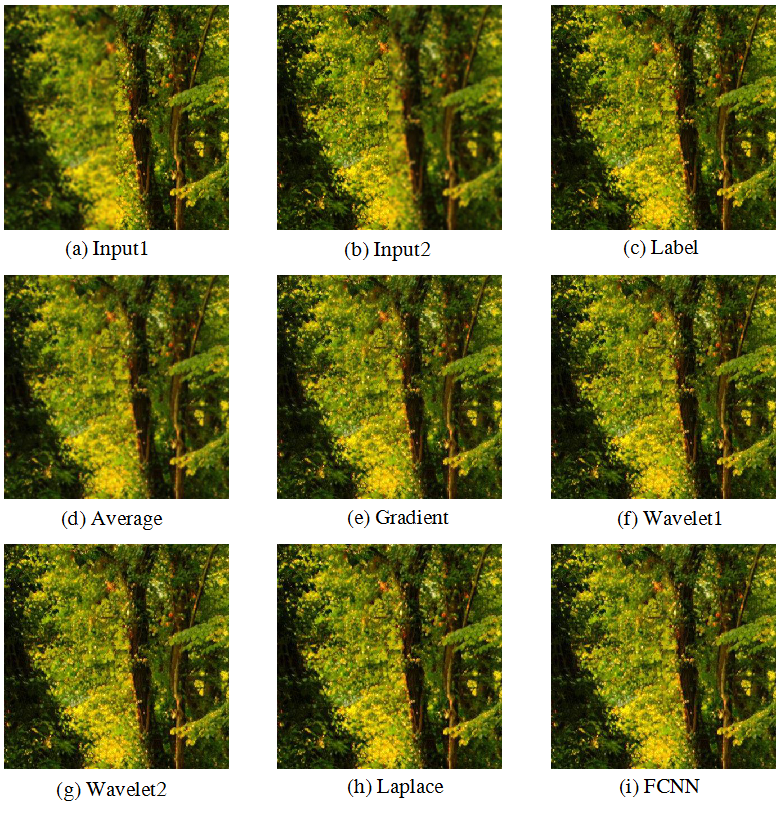


图3.9 第一组图像融合结果

图3.10中，(a)为左模糊图像，(b)右模糊图像，(c)为标签图像，(d)(e)(f)(g)(h)为对比所使用的算法得到的多聚焦图像融合结果图像，(d)平均法，(e)梯度法，(f)小波系数法，(g)小波局部系数法，(h)拉普拉斯金字塔法，(i)为本文章所提出的算法得到的多聚焦图像融合结果图像。实验图像来自于Lytro-Dataset多聚焦图像数据集。

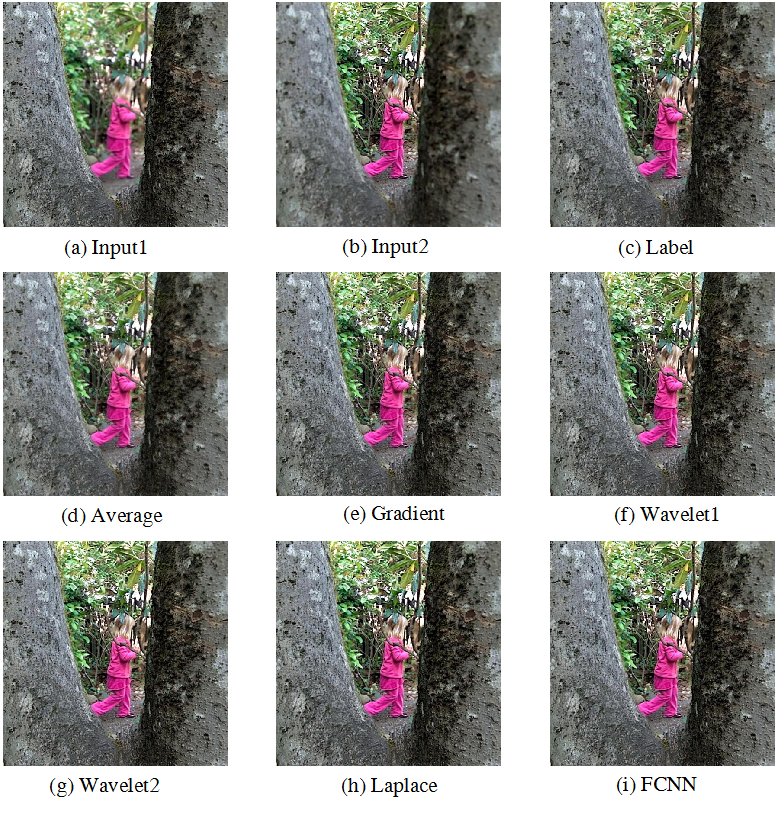


图3.10 第二组图像融合结果

表3.1 第一组融合图像评价

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | STD | IE | RMSE | PSNR |
| FCNN | 48.7078 | 5.1613 | **1.6051** | **44.0209** |
| Laplace | **49.1548** | **5.1671** | 2.0679 | 41.8203 |
| Wave1 | 48.2380 | 5.1524 | 1.8369 | 42.8490 |
| Wave2 | 48.2791 | 5.1536 | 2.1199 | 41.6045 |
| Gradient | 47.7254 | 5.1490 | 3.1546 | 38.1518 |
| Average | 45.9381 | 5.1144 | 5.7530 | 32.9330 |

表3.2 第二组图像融合评价

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | STD | IE | RMSE | PSNR |
| FCNN | 51.2794 | 5.2865 | **3.3946** | **37.0964** |
| Laplace | **52.1555** | **5.2976** | 4.3045 | 35.5885 |
| Wave1 | 51.1867 | 5.2886 | 4.1623 | 35.8792 |
| Wave2 | 51.3151 | 5.2896 | 4.4510 | 35.2821 |
| Gradient | 50.1906 | 5.2790 | 4.8617 | 34.4963 |
| Average | 47.4203 | 5.2326 | 5.7875 | 32.9040 |

表3.10为图3.9的五项客观评价指标。表3.2为图3.10的四项客观评价指标。在所有指标中，本文方法对该图像的融合效果的指标中的两项优于其他算法的指标。

从以上两个实验中，可以看出均值加权融合图像效果相较于标签图像有明显的模糊，基于梯度融合效果相较均值加权法有所提升，基于拉普拉斯、基于小波和本文提出的方法相较于标签图像无明显差异。

从客观评价指标中，可以定量地表达出各种算法的性能程度。说明本文算法具有较好的多聚焦图像融合能力，可以对不同场景不同来源的多聚焦图像进行较好地融合。从图像上看，涉及多分辨率的方法对于图像的影响具有全局特性，这来源于低分辨率下的系数选择。基于聚焦评价的算法的融合图像中，在图像的聚焦与非聚焦的边缘处的融合并不是非常理想，导致出现微小的突出变化，这主要是聚焦评价的不确定性导致的。

3.4 小结

在这一章中，我们详细地描述了模型的训练配置、训练结果与融合结果，并且通过对比实验验证了本文方法的有效性。具体介绍了卷积神经网络模型的训练参数与配置，其中包括损失函数、学习率等。介绍了模型的使用环境、该模型用于算法的仿真结果，并进行了多组对比实验，使用多种客观指标进行评价。最终的实验结果表明，本文算法在多数指标可以取得较好的效果。

结束语

多聚焦图像融合是图像融合领域的一个重要分支。多聚焦图像融合算法的最主要目的十分明确，即为了对聚焦不同的图像进行融合得到相比于源图像更为清晰的图像。

本文主要阐述了多聚焦图像融合技术的背景与现状，介绍了多聚焦图像融合主要算法，详细介绍了主要的图像融合评价指标并着重于研究设计像素级融合方法，并就此提出了一种基于空间域的并且基于卷积神经网络的多聚焦图像融合方法。

本文方法流程清晰明了，只需要将不同聚焦的图像输入卷积神经网络中，根据输入的聚焦评判图对像素进行选择，最终形成连续的聚焦清晰区域。融合效果与融合速度具有一定优势。

但没有任何一种算法是最优的，如在融合效果获得融合速度上往往就会存在着制约关系。本文方法也不例外并非最优。相对于不需要某些训练数据的其他方法，本文方法由于基于卷积神经网络所以算法复杂度明显更高。由于图像中大片的低频域的存在可能会导致融合效果变差。并且，本文方法对于细节较少的图像融合效果一般，细节较多的图像融合效果最好。

在以后的研究中，可以尝试使用卷积神经网络端到端的方式对图像进行融合，即输入两张图像其输出结果即为融合图像。也可以尝试使用其他的图像处理或机器视觉领域技术进行尝试。

参考文献

1. 夏明革,何友,唐小明,等.图像融合的发展现状与展望[J].舰船电子工程,2002,6:2~13
2. 官泽瑾,毛义坪.基于梯度理论的多聚焦图像融合[J].现代计算机机,2020,1:61~66
3. 关雪梅.小波变换图像处理技术研究[J].沧州师范学院学报,2019,35(1):44~46
4. 王卫卫,水鹏朗,宋国乡.小波域多聚焦图像融合算法[J].系统工程与电子技术,2004,26(5):668~671
5. 何乃莹,李蕾,王传恩.一种多聚焦图像融合的拉普拉斯金字塔方法[J].计算机时代,2019,4:18~25
6. 王海晖,彭嘉雄,吴巍,等.多源遥感图像融合效果评价方法研究[J].计算机工程与应用,2003,39(25):33~37
7. 孙志军,薛磊,许阳明,等.深度学习研究综述[J].计算机应用研究,2012,29(8):2806~2810
8. 陈先昌.基于卷积神经网络的深度学习算法与应用研究[D].浙江工商大学硕士学位论文,2013,4~6
9. 周飞燕,金林鹏,董军.卷积神经网络研究综述[J].计算机学报,2017,40(6):1229~1251
10. Ren s, Girshick R, Girshick R, et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks[J]. IEEE Transactions on Pattern Analysis Machine Intelligence,2017,39(6):1137~1149
11. Mingo J M, Aler R. A competence-performance based model to develop a syntactic language for artificial agents [J]. Information Sciences, 2016, 373: 79~94
12. 常亮,邓小明,周明全,等.图像理解中的卷积神经网络[J].自动化学报,2016,42(9):1300~1312
13. Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional

neural networks[C].Advances in neural information processing systems. 2012: 1097~1105

1. Chollet F. Xception: Deep Learning with Depthwise Separable Convolutions[J].2016:

1800~1807

1. 陈云.深度学习框架Pytorch：入门与实践[M].电子工业出版社:北京市,2018:103
2. 张沛阳.深度学习理论综述与研究展望[J].网络安全技术与应用,2020,4:43~44

致 谢

我们的大学四年时间即将过去，我们也将走向社会、走向未来。我们在成长，一直都在成长着。我们不必感慨时间过得飞快，毕竟所有人都走过一样的道路，在这条康庄大道上，最值得我们提起和纪念的便是所有无私地给予我们帮助人们。

在此，感谢我的导师。在即将毕业的这次设计实践中，导师直接给予了我们最多的帮助。在论文的各个阶段中，在导师付出大量时间悉心的指导下，最终选择了正确的航向、航行在广阔的海洋中到达成功的彼岸。

在此，感谢我的每位老师。感谢在大学四年以来，每一位老师给予了我们知识财富，正是老师们在课堂上深入浅出的讲授，让我们看到知识的不同境界，使得我们到达了更高的高度。也在此感谢老师们不仅在学习上也在生活上的关系和帮助。

在此，感谢我的每位同学。感谢在大学四年以来，与各位同学一起拼搏、一起进步，在困难中一直砥砺前行。未来的道路对我们来说仍然很长，我们将继续相互激励一致向前。等到多年以后，也许是下一个十年，再次回首时仍然心潮澎湃。

附 录

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Model\_Frame.py \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

from .Model\_Train import Train

from .Model\_Manage import Manage

import torch,shutil

class Frame:

def \_\_init\_\_(self):

sr\_tip = '\n\n' \

'Need目录在必须存在以下文件\n' \

' - \*\_Model.py 用于生成模型,必须包含模型实现类,模型的键值名称即类名\n' \

' - \*\_Loader.py 用于返回 loader 训练集,必须包含 get\_loader() 方法\n' \

' - \*\_Predict.py 用于返回 预测类,必须包含 predict() 方法' \

'\n\n'

print(sr\_tip)

self.sr\_model\_manage\_dir = './Frame/Model'

self.sr\_model\_dc\_temp\_path = '{}/Model.pt'.format(self.sr\_model\_manage\_dir) # 模型参数临时保存路径

self.sr\_model\_py\_temp\_path = '{}/Model.py'.format(self.sr\_model\_manage\_dir) # 模型脚本临时保存路径

self.sr\_loader\_py\_temp\_path = '{}/Loader.py'.format(self.sr\_model\_manage\_dir) # 加载器脚本临时保存路径

self.sr\_predict\_py\_temp\_path = '{}/Predict.py'.format(self.sr\_model\_manage\_dir) # 预测类脚本临时保存路径

self.sr\_model\_py\_build\_path = './Need/{}/Model.py' # 新模型脚本py文件 路径

self.sr\_loader\_py\_build\_path = './Need/{}/Loader.py'

self.sr\_predict\_py\_build\_path = './Need/{}/Predict.py'

self.model\_key\_name = None # 模型键值命名

self.ins\_model = None # 模型实例，由 build,load 加载

self.ins\_loader = None # 训练数据集加载器

self.ins\_predict = None # 预测类实例

self.dc\_train\_params = None # 训练参数, 由 set\_train\_params() 获得

self.ins\_Train = Train() # 训练类实例

self.ins\_Manage = Manage() # 管理类实例

def init(self):

self.ins\_model = None # 模型实例，由 build,load 加载

self.model\_key\_name = None # 模型键值命名

self.ins\_loader = None # 训练数据集加载器

self.dc\_train\_params = None # 训练参数, 由 set\_train\_params() 获得

# 新建模型

def build\_model(self,sr\_model\_key\_name):

if not self.ins\_Manage.exist\_model\_key\_name(sr\_model\_key\_name):

print('>>> 新建模型 {}.'.format(sr\_model\_key\_name))

# 初始化变量

self.init()

try:

# 将模型脚本临时保存

self.save\_temp\_model\_py(sr\_model\_key\_name)

# 将加载器脚本临时保存

self.save\_temp\_loader\_py(sr\_model\_key\_name)

# 将预测脚本临时保存

self.save\_temp\_predict\_py(sr\_model\_key\_name)

# 添加模型

self.ins\_Manage.add\_model\_item(sr\_model\_key\_name=sr\_model\_key\_name)

# # 导入模型,加载器,预测实例,从新添加的模型导入

# dc = self.ins\_Manage.check\_model\_item(sr\_model\_key\_name,0)

# self.ins\_model = dc['model']

# self.ins\_predict = dc['predict']

# # 模型键值名

# self.model\_key\_name =sr\_model\_key\_name

except:

self.delete\_model(sr\_model\_key\_name)

raise Exception('新建模型失败.')

else:

raise Exception('该模型已存在,无法新建.')

# 加载数据库中已存在的模型

def load\_model(self,sr\_model\_key\_name):

if self.ins\_Manage.exist\_model\_key\_name(sr\_model\_key\_name): # 数据库是否已存在该模型

print('>>> 导入模型 {}.'.format(sr\_model\_key\_name))

# 初始化变量

self.init()

# 导入模型,加载器,预测实例

dc = self.ins\_Manage.check\_model\_item(sr\_model\_key\_name,mode=0)

self.ins\_model = dc['model']

self.ins\_predict = dc['predict']

# 模型键值名

self.model\_key\_name = sr\_model\_key\_name

else:

raise Exception('该模型不存在,无法导入.')

# 模型预测

def predict(self,dc\_input):

Y = self.ins\_predict.predict(self.ins\_model,dc\_input)

return Y

# 设置训练参数

def set\_train\_params(self,lr,epochs,lossf,opt,momentum=0.9,is\_show\_detail=True):

self.dc\_train\_params = {

'lr':lr,

'epochs':epochs,

'lossf':lossf,

'optim':opt,

'momentum':momentum,

'is\_show\_detail':is\_show\_detail

}

# 设置模型提示

def set\_model\_tips(self,sr\_tips):

self.ins\_Manage.set\_model\_tips(self.model\_key\_name,sr\_tips)

# 获取模型提示

def get\_model\_tips(self):

return self.ins\_Manage.get\_model\_tips(self.model\_key\_name)

# 获取模型的所有信息

def get\_model\_all\_info(self):

dc = self.ins\_Manage.check\_model\_item(self.model\_key\_name,mode=2) # 调用管理实例获取模型信息

dc\_info = dc['info']

return dc\_info

# 临时保存模型dc文件

def save\_temp\_model\_dc(self):

torch.save(self.ins\_model.state\_dict(), self.sr\_model\_dc\_temp\_path) # 保存到模型临时存储

# 临时保存模型py脚本

def save\_temp\_model\_py(self,sr\_model\_key\_name):

shutil.copy(self.sr\_model\_py\_build\_path.format(sr\_model\_key\_name), self.sr\_model\_py\_temp\_path)

# 临时保存加载器py脚本

def save\_temp\_loader\_py(self,sr\_model\_key\_name):

shutil.copy(self.sr\_loader\_py\_build\_path.format(sr\_model\_key\_name), self.sr\_loader\_py\_temp\_path)

# 临时保存预测类py脚本

def save\_temp\_predict\_py(self,sr\_model\_key\_name):

shutil.copy(self.sr\_predict\_py\_build\_path.format(sr\_model\_key\_name), self.sr\_predict\_py\_temp\_path)

def get\_loader(self):

dc = self.ins\_Manage.check\_model\_item(self.model\_key\_name,1)

return dc['loader']

# 训练

def train(self):

model = self.ins\_model

self.ins\_loader = self.get\_loader()

loader = self.ins\_loader

dc\_train\_params = self.dc\_train\_params

print('>>> 开始训练.')

dc\_model = self.ins\_Train.train(model=model,loader=loader,dc\_train\_params=dc\_train\_params) # 训练

print('>>> 训练完成.')

self.ins\_model = dc\_model['model'] # 模型更新

self.save\_temp\_model\_dc() # 保存

self.ins\_Manage.update\_model\_item(self.model\_key\_name,dc\_model) # 数据库更新

# 更新加载器py脚本

def update\_loader\_py(self, sr\_model\_key\_name):

self.save\_temp\_loader\_py(sr\_model\_key\_name)

self.ins\_Manage.update\_loader\_py(sr\_model\_key\_name)

# 更新预测类py脚本

def update\_predict\_py(self, sr\_model\_key\_name):

self.save\_temp\_predict\_py(sr\_model\_key\_name)

self.ins\_Manage.update\_predict\_py(sr\_model\_key\_name)

# 删除模型

def delete\_model(self,sr\_model\_key\_name):

if sr\_model\_key\_name == self.model\_key\_name:

self.init()

is\_succ = self.ins\_Manage.delete\_model\_item(sr\_model\_key\_name) # 调用管理实例删除当前模型( 删除模型前必须加载模型 )

if is\_succ:

print('>>> 删除 {} 成功.'.format(sr\_model\_key\_name))

else:

print('>>> 删除 {} 失败.'.format(sr\_model\_key\_name))

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Model\_Manage.py \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import json,os,shutil,torch

class Manage:

def \_\_init\_\_(self):

self.sr\_model\_manage\_dir = './Frame/Model' # 数据库根路径

self.sr\_info\_name = 'Info.json'

self.sr\_model\_dc\_temp\_path = '{}/Model.pt'.format(self.sr\_model\_manage\_dir) # 模型参数临时保存路径,需要与Frame保持一致

self.sr\_model\_py\_temp\_path = '{}/Model.py'.format(self.sr\_model\_manage\_dir) # 模型脚本临时保存路径,需要与Frame保持一致

self.sr\_loader\_py\_temp\_path = '{}/Loader.py'.format(self.sr\_model\_manage\_dir) # 加载器脚本临时保存路径,需要与Frame保持一致

self.sr\_predict\_py\_temp\_path = '{}/Predict.py'.format(self.sr\_model\_manage\_dir) # 加载器脚本临时保存路径,需要与Frame保持一致

self.sr\_data\_path = '{}/Manage.json'.format(self.sr\_model\_manage\_dir) # 主文件路径

self.dc\_manage = None # 数据库主文件

self.load\_manage\_json() # 加载 数据库主文件

# 加载 数据库主文件

def load\_manage\_json(self):

with open(self.sr\_data\_path) as file\_handel:

self.dc\_manage = json.load(file\_handel)

# 判断是否存在模型

def exist\_model\_key\_name(self,sr\_model\_key\_name):

if sr\_model\_key\_name not in self.dc\_manage:

return False

else:

return True

# 更新主文件

def update\_dc\_manage(self):

with open(self.sr\_data\_path,'w') as file\_handel:

json.dump(self.dc\_manage,file\_handel)

# 更新模型py文件

def update\_model\_py(self,sr\_model\_key\_name):

sr\_model\_dir = '{}/{}'.format(self.sr\_model\_manage\_dir, sr\_model\_key\_name) # 模型文件信息目录

sr\_model\_py\_save\_path = '{}/{}\_Model.py'.format(sr\_model\_dir,sr\_model\_key\_name) # 模型py文件路径

shutil.copy(self.sr\_model\_py\_temp\_path, sr\_model\_py\_save\_path) # 复制 模型py文件

# 更新加载器py文件

def update\_loader\_py(self,sr\_model\_key\_name):

sr\_model\_dir = '{}/{}'.format(self.sr\_model\_manage\_dir, sr\_model\_key\_name) # 模型文件信息目录

sr\_loader\_py\_save\_path = '{}/{}\_Loader.py'.format(sr\_model\_dir,sr\_model\_key\_name) # 加载器py文件路径

shutil.copy(self.sr\_loader\_py\_temp\_path, sr\_loader\_py\_save\_path) # 复制 加载器py文件

# 更新预测类py文件

def update\_predict\_py(self, sr\_model\_key\_name):

sr\_model\_dir = '{}/{}'.format(self.sr\_model\_manage\_dir, sr\_model\_key\_name) # 模型文件信息目录

sr\_prdict\_py\_save\_path = '{}/{}\_Predict.py'.format(sr\_model\_dir,sr\_model\_key\_name) # 加载器py文件路径

shutil.copy(self.sr\_predict\_py\_temp\_path, sr\_prdict\_py\_save\_path) # 复制 加载器py文件

# 获取预测类实例

def get\_predict\_instance(self,sr\_model\_key\_name):

# 从py文件中实例化预测实例

sr\_model\_dir = '{}/{}'.format(self.sr\_model\_manage\_dir, sr\_model\_key\_name)

sr\_predict\_py\_save\_path = '{}/{}\_Predict.py'.format(sr\_model\_dir,sr\_model\_key\_name) # 预测类py文件路径

sr\_exec = sr\_predict\_py\_save\_path.replace('.py', '').replace('.', '').replace('/', '.')[1:]

sr\_exec = 'from {} import Predict'.format(sr\_exec)

exec(sr\_exec)

predict = eval('Predict()')

return predict

# 获取加载器实例

def get\_loader\_instance(self,sr\_model\_key\_name):

# 从py文件中实例化加载器

sr\_model\_dir = '{}/{}'.format(self.sr\_model\_manage\_dir, sr\_model\_key\_name)

sr\_loader\_py\_save\_path = '{}/{}\_Loader.py'.format(sr\_model\_dir,sr\_model\_key\_name) # 加载器py文件路径

sr\_exec = sr\_loader\_py\_save\_path.replace('.py', '').replace('.', '').replace('/', '.')[1:]

sr\_exec = 'from {} import Loader'.format(sr\_exec)

exec(sr\_exec)

loader = eval('Loader().get\_loader()')

return loader

# 获取模型实例,注意非导入参数,因为调用时不一定存在dc参数py路径, load\_model()才进行dc参数py导入

def get\_model\_instance(self,sr\_model\_key\_name):

# 从py文件中实例化模型

sr\_model\_dir = '{}/{}'.format(self.sr\_model\_manage\_dir, sr\_model\_key\_name) # 模型文件信息目录

sr\_model\_py\_save\_path = '{}/{}\_Model.py'.format(sr\_model\_dir,sr\_model\_key\_name) # 模型py文件路径

sr\_exec = sr\_model\_py\_save\_path.replace('.py', '').replace('.', '').replace('/', '.')[1:]

sr\_exec = 'from {} import Model'.format(sr\_exec)

exec(sr\_exec) # 动态导入模型类,如 from Model.CNN.Model import CNN

model = eval('Model()'.format(sr\_model\_key\_name)) # 模型实例化, model = CNN()

return model

# 根据模型键值名加载模型

def load\_model(self, sr\_model\_key\_name):

sr\_model\_dir = '{}/{}'.format(self.sr\_model\_manage\_dir, sr\_model\_key\_name) # 模型文件信息目录

sr\_model\_dc\_save\_path = '{}/{}\_Model.pt'.format(sr\_model\_dir, sr\_model\_key\_name) # 模型dc文件路径

model = self.get\_model\_instance(sr\_model\_key\_name)

model.load\_state\_dict(torch.load(sr\_model\_dc\_save\_path))

return model

# 添加模型

def add\_model\_item(self,sr\_model\_key\_name):

if sr\_model\_key\_name not in self.dc\_manage:

self.dc\_manage[sr\_model\_key\_name] = {}

self.update\_dc\_manage() # 更新 主文件

# 创建目录

sr\_model\_dir = '{}/{}'.format(self.sr\_model\_manage\_dir, sr\_model\_key\_name) # 模型文件信息目录

os.mkdir(sr\_model\_dir)

# 将模型py脚本复制到目录中

self.update\_model\_py(sr\_model\_key\_name)

# 将加载器py脚本复制到目录中

self.update\_loader\_py(sr\_model\_key\_name)

# 将预测类py脚本复制到目录中

self.update\_predict\_py(sr\_model\_key\_name)

# 模型dc参数文件

model = self.get\_model\_instance(sr\_model\_key\_name)

sr\_model\_dc\_save\_path = '{}/{}\_Model.pt'.format(sr\_model\_dir,sr\_model\_key\_name) # 模型dc文件路径

torch.save(model.state\_dict(),sr\_model\_dc\_save\_path) # 保存模型参数

sr\_model\_info\_path = '{}/{}'.format(sr\_model\_dir, self.sr\_info\_name) # 模型信息路径

# 添加 模型信息

dc\_info\_json = {'train\_log': []}

with open(sr\_model\_info\_path,'w') as file\_handle:

json.dump(dc\_info\_json, file\_handle)

return True

else:

raise Exception('模型 {} 已存在'.format(sr\_model\_key\_name))

# 更新模型

def update\_model\_item(self,sr\_model\_key\_name,dc\_model):

if sr\_model\_key\_name not in self.dc\_manage:

raise Exception('模型 {} 不存在'.format(sr\_model\_key\_name))

else:

sr\_model\_dir = '{}/{}'.format(self.sr\_model\_manage\_dir, sr\_model\_key\_name) # 模型文件信息目录

sr\_model\_dc\_save\_path = '{}/{}\_Model.pt'.format(sr\_model\_dir,sr\_model\_key\_name) # 模型文件路径

sr\_model\_info\_path = '{}/{}'.format(sr\_model\_dir,self.sr\_info\_name) # 模型信息路径

# 更新 模型dc参数文件

shutil.copy(self.sr\_model\_dc\_temp\_path, sr\_model\_dc\_save\_path)

# 更新 模型信息

dc\_train\_log = dc\_model['train\_log']

dc\_train\_info = {

'lr': dc\_train\_log['lr'],

'epochs': dc\_train\_log['epochs'],

'lossf': dc\_train\_log['lossf'],

'optim': dc\_train\_log['optim'],

'momentum': dc\_train\_log['momentum'],

'time': dc\_train\_log['time'],

'loss': dc\_train\_log['loss']

}

with open(sr\_model\_info\_path) as file\_handle:

dc\_info\_json = json.load(file\_handle)

dc\_info\_json['train\_log'].append(dc\_train\_info)

with open(sr\_model\_info\_path,'w') as file\_handle:

json.dump(dc\_info\_json,file\_handle)

return True

# 删除模型

def delete\_model\_item(self,sr\_model\_key\_name):

if sr\_model\_key\_name in self.dc\_manage:

sr\_model\_dir = '{}/{}'.format(self.sr\_model\_manage\_dir, sr\_model\_key\_name) # 模型文件信息目录

shutil.rmtree(sr\_model\_dir) # 删除 模型目录

self.dc\_manage.pop(sr\_model\_key\_name) # 删除 在数据库主文件中的模型键值

self.update\_dc\_manage() # 更新 数据库主文件

return True

else:

print('>>> 不存在该模型.')

return False

# 查看模型

def check\_model\_item(self,sr\_model\_key\_name,mode):

sr\_model\_dir = '{}/{}'.format(self.sr\_model\_manage\_dir, sr\_model\_key\_name) # 模型文件信息目录

sr\_model\_json\_path = '{}/{}'.format(sr\_model\_dir,self.sr\_info\_name) # 模型信息路径

dc = {

'model': None,

'loader':None,

'predict':None,

'info': None

}

if mode == 0:

# 导入模型和预测类

model = self.load\_model(sr\_model\_key\_name)

predict = self.get\_predict\_instance(sr\_model\_key\_name)

dc['model'] = model

dc['predict'] = predict

elif mode == 1:

# 导入加载器

loader = self.get\_loader\_instance(sr\_model\_key\_name)

dc['loader'] = loader

elif mode == 2:

# 导入模型信息

with open(sr\_model\_json\_path) as file\_handle:

dc\_info\_json = json.load(file\_handle)

dc['info'] = dc\_info\_json

elif mode == 3:

# 导入所有

model = self.load\_model(sr\_model\_key\_name)

loader = self.get\_loader\_instance(sr\_model\_key\_name)

predict = self.get\_predict\_instance(sr\_model\_key\_name)

with open(sr\_model\_json\_path) as file\_handle:

dc\_info\_json = json.load(file\_handle)

dc['model'] = model

dc['loader'] = loader

dc['predict'] = predict

dc['info'] = dc\_info\_json

return dc

# 设置模型提示

def set\_model\_tips(self,sr\_model\_key\_name,sr\_tips):

self.dc\_manage[sr\_model\_key\_name]['tips'] = sr\_tips

self.update\_dc\_manage()

# 获取模型提示

def get\_model\_tips(self,sr\_model\_key\_name):

return self.dc\_manage[sr\_model\_key\_name]['tips']

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Model\_Train.py \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import torch.nn as nn

import torch.optim as optim

import time

class Train:

def \_\_init\_\_(self):

pass

def train(self, model, loader, dc\_train\_params,):

lr = dc\_train\_params['lr']

epochs = dc\_train\_params['epochs']

lossf\_sel = dc\_train\_params['lossf']

opt\_sel = dc\_train\_params['optim']

momentum = dc\_train\_params['momentum']

is\_show\_detail = dc\_train\_params['is\_show\_detail']

if lossf\_sel == 'mse':

loss\_func = nn.MSELoss()

elif lossf\_sel == 'smo':

loss\_func = nn.SmoothL1Loss()

elif lossf\_sel == 'bce':

loss\_func = nn.BCELoss

else:

raise Exception('loss function')

if opt\_sel == 'adam':

optimizer = optim.Adam(model.parameters(),lr=lr)

elif opt\_sel == 'sgd':

optimizer = optim.SGD(model.parameters(), lr=lr, momentum=momentum)

else:

raise Exception('optimizer function')

dc\_model = {

'model':model,

'train\_log':None,

}

train\_log = {

'lr':lr,

'epochs':epochs,

'lossf':lossf\_sel,

'optim':opt\_sel,

'momentum':momentum,

'loss':None

}

# 模型训练

ls\_loss = []

ls\_rate = []

start = time.time()

for epoch in range(epochs):

print()

train\_lossSum = 0

for step, (batch\_x, batch\_y) in enumerate(loader):

# 前向计算获得输出

Y = model(batch\_x)

# 与模型连接

loss = loss\_func(Y, batch\_y)

# 梯度初始化归零,准备优化

optimizer.zero\_grad()

# 反向传播,更新梯度

loss.backward()

# 根据计算得到的梯度,结合优化器参数进行模型参数更新

optimizer.step()

train\_loss = loss.item()

train\_lossSum += train\_loss

if is\_show\_detail:

print('{}-{}: {}'.format(epoch, step, train\_loss))

train\_lossSum = train\_lossSum / len(loader)

print()

print('{}-mean: {}'.format(epoch, train\_lossSum))

ls\_loss.append(train\_lossSum)

if len(ls\_loss) > 1:

fl\_rateFirst = ls\_loss[-1] / ls\_loss[0]

fl\_rateLast = ls\_loss[-1] / ls\_loss[-2]

ls\_rate.append(fl\_rateFirst)

print('{}-rate-compare with first: {}'.format(epoch, fl\_rateFirst))

print('{}-rate-compare with last: {}'.format(epoch, fl\_rateLast))

print()

end = time.time()

use\_time = end - start

train\_log['time'] = use\_time

train\_log['loss'] = ls\_loss

dc\_model['train\_log'] = train\_log

return dc\_model

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FCNN\_Predict.py \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class Predict:

def predict(self,model,dc\_input):

import torch

import cv2 as cv

import numpy as n

X = dc\_input['X']

# 模型预测

X = cv.cvtColor(X, cv.COLOR\_RGB2GRAY)

X = X / 255.0

X = X[n.newaxis, n.newaxis, :, :]

X = torch.from\_numpy(X).float()

Y = model(X)

Y = Y.detach().numpy()

Y = Y[0, 0, :, :]

return Y

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FCNN\_Model.py \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import torch.nn.functional as F

import torch.nn as nn

import torch

class Model(nn.Module): # 定义模型

def \_\_init\_\_(self):

nn.Module.\_\_init\_\_(self)

sobelX = [

[-1,0,1],

[-2,0,2],

[-1,0,1],

]

sobelY = [

[-1, -2, -1],

[0, 0, 0],

[1, 2, 1],

]

sobelX = torch.FloatTensor(sobelX).unsqueeze(0).unsqueeze(0)

self.sobelX = nn.Parameter(data=sobelX, requires\_grad=False)

sobelY = torch.FloatTensor(sobelY).unsqueeze(0).unsqueeze(0)

self.sobelY = nn.Parameter(data=sobelY, requires\_grad=False)

std\_channels = 8

self.layer1 = nn.Sequential(

# 1

nn.Conv2d(

in\_channels=3,

out\_channels=std\_channels,

kernel\_size=3,

stride=1,

padding=1,

),

nn.ReLU(),

# 2

nn.Conv2d(

in\_channels=std\_channels,

out\_channels=std\_channels,

kernel\_size=3,

stride=1,

padding=1,

),

nn.ReLU(),

# 3

nn.Conv2d(

in\_channels=std\_channels,

out\_channels=std\_channels,

kernel\_size=3,

stride=1,

padding=1,

),

nn.ReLU(),

# 4

nn.Conv2d(

in\_channels=std\_channels,

out\_channels=std\_channels,

kernel\_size=3,

stride=1,

padding=1,

),

nn.ReLU(),

# 5

nn.Conv2d(

in\_channels=std\_channels,

out\_channels=std\_channels,

kernel\_size=3,

stride=1,

padding=1,

),

nn.ReLU(),

# 6

nn.Conv2d(

in\_channels=std\_channels,

out\_channels=std\_channels,

kernel\_size=3,

stride=1,

padding=1,

),

nn.ReLU(),

# 7

nn.Conv2d(

in\_channels=std\_channels,

out\_channels=std\_channels,

kernel\_size=3,

stride=1,

padding=1,

),

nn.ReLU(),

# 8

nn.Conv2d(

in\_channels=std\_channels,

out\_channels=1,

kernel\_size=3,

stride=1,

padding=1,

),

nn.Sigmoid(),

)

def forward(self, X):

XX = F.conv2d(X[:,0].unsqueeze(1), self.sobelX, padding=1)

XY = F.conv2d(X[:,0].unsqueeze(1), self.sobelY, padding=1)

size = X.shape

size = list(size)

size[1] = 3

size = tuple(size)

XN = torch.zeros(size=size)

XN[:, 0, :, :] = X

XN[:, 1, :, :] = XX

XN[:, 2, :, :] = XY

Y = self.layer1(XN)

return Y

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FCNN\_Loader.py \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class Loader:

def \_\_init\_\_(self):

# Loader 所在目录

self.sr\_now\_dir = './Need'

def get\_loader(self):

import torch.utils.data as Data

import matplotlib.pyplot as p

import numpy as n

import cv2 as cv

import torch

sr\_input1 = self.sr\_now\_dir + '/FCNN4567/Sample/Input/{}.jpg'

sr\_label = self.sr\_now\_dir + '/FCNN4567/Sample/Label/{}.jpg'

label\_first = p.imread(sr\_label.format(0))

shape = label\_first.shape

heigh = shape[0]

width = shape[1]

Xnum = 100

# 只需要灰度图

X = n.zeros((Xnum, 1, heigh, width))

Y = n.zeros((Xnum, 1, heigh, width))

for i in range(0, Xnum):

input1 = cv.cvtColor(cv.imread(sr\_input1.format(i)), cv.COLOR\_BGR2GRAY) / 255

label = cv.cvtColor(cv.imread(sr\_label.format(i)), cv.COLOR\_BGR2GRAY) / 255

X[i, 0, :, :] = input1

Y[i, 0, :, :] = label

Xtorch = torch.from\_numpy(X).float()

Ytorch = torch.from\_numpy(Y).float()

torch\_dataset = Data.TensorDataset(Xtorch, Ytorch)

loader = Data.DataLoader(

dataset=torch\_dataset,

batch\_size=1,

shuffle=True,

)

return loader