

**计算摄影学课程项目报告**

|  |  |
| --- | --- |
| 项目名称： | 单目深度估计 |
| 姓 名： | 李鹏飞 黄奇浩 李欣怡 |
| 学 号： |  |
| 电子邮箱： |  |
| 联系电话： |  |
| 导 师： | 章国锋 |

2018 年 06 月 26 日

单目深度估计

1. 项目介绍
   1. 项目背景

本项目利用深度神经网络实现了单目图像的深度估计任务。

// TODO by 欣怡

深度估计的传统方法和深度学习方法简单介绍

* 1. 项目的功能和要求

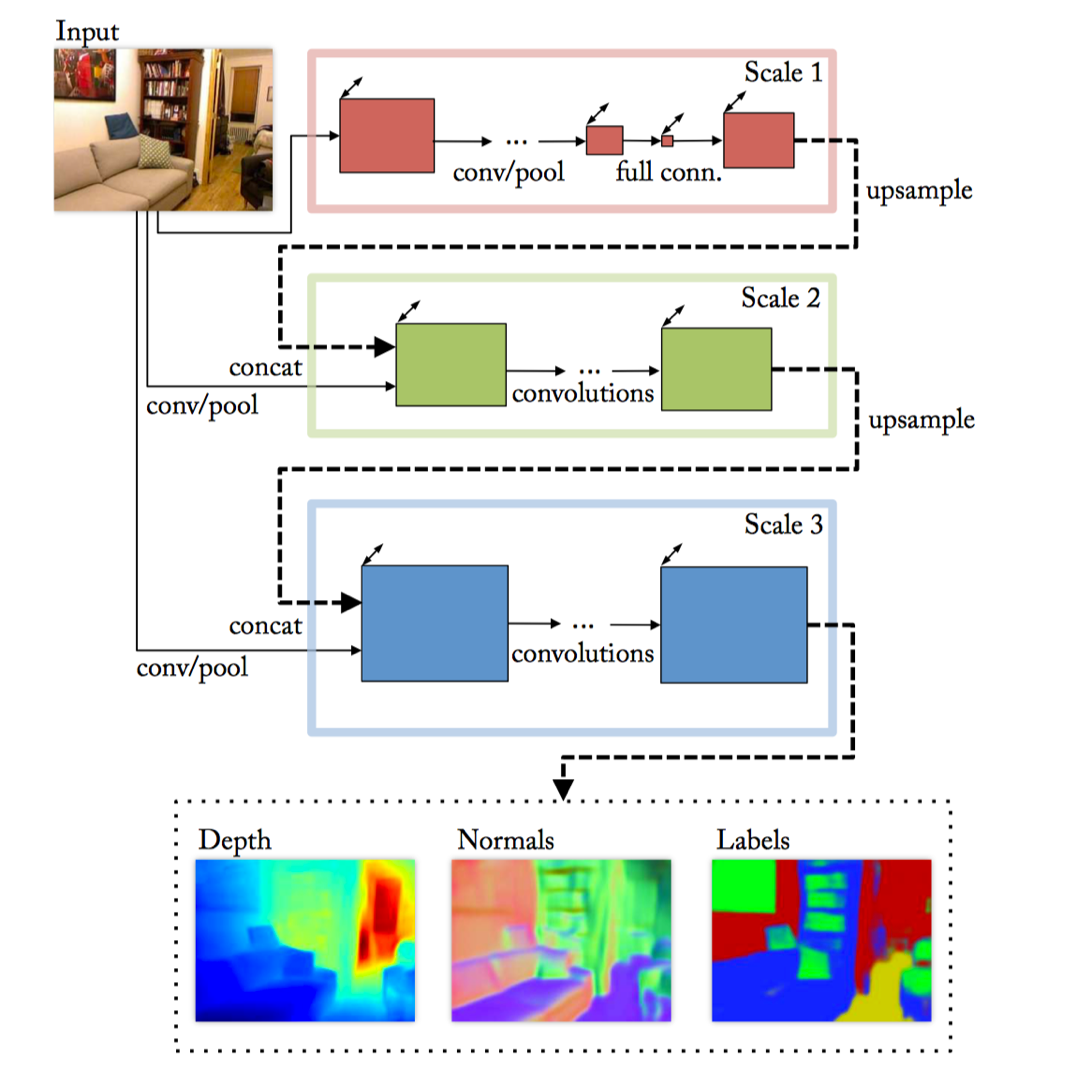
// TODO by 欣怡

根据NYU数据集（需要介绍一下），训练和测试深度估计模型。

// 可以参考论文中的前几段总结

1. 基本原理
   1. 深度估计模型

在学习过程中，我们首先尝试和参考了纽约大学David Eigen等人名为 《Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture》的论文。该篇论文提出了一种多层次的卷积神经网络，应用于单目图像Depth，Normal，Labels三种估计任务，如下图所示。



该模型通过三种不同功能的网络，将不同层级的图像特征组合起来。其中，Scale1和Scale2用于预测较为粗粒度的全局图像深度信息，并输入Scale3中，用于优化Scale3中更为细致的局部图像训练过程中，得到最终的图像深度估计结果。

* 1. 涉及的方法

我们采用了较为经典的卷积结构用于捕捉图像特征，采用特定损失函数用于评价网络预测深度图和训练图像所对应的深度图标签，并反向传播更新网络参数。

* + 1. 网络架构

设计和实现过程中，我们将Scale1和 Scale2统一成一个网络Net12，用于训练和测试。其中Scale1的基本结构采用了VGG Net的网络结构设计，并去除最后的两个全连接层，改为out1 = out1.view(self.N, 64, 19, 14)的结构，即reshape得到Nx64x19x14的输出。为匹配Scale2的Tensor维度，再将其双线性unsample得到Nx64x76x56的输出，添入到Scale2的网络中。Scale2经过一次卷积和池化后，合并Scale1的上述输入，并经过一系列卷积和池化后得到Nx1x74x55的输出。

Scale3（Net3）的训练过程和Net12相似，同样需要将Net12的输出Nx1x74x55双线性unsample，适当裁剪到Nx1x147x109的维度。图像首先经过Net3的卷积和池化后，与上述Net12的输出变换拼接，再采取若干次卷积和池化得到最终的预测输出，Nx1x147x109，N张147x109的单通道深度图。

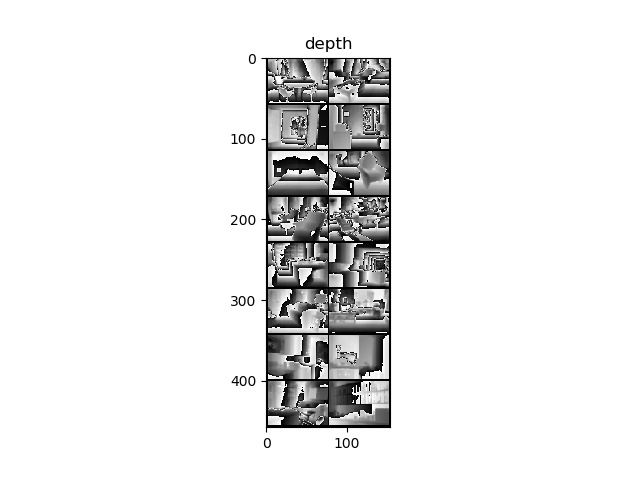
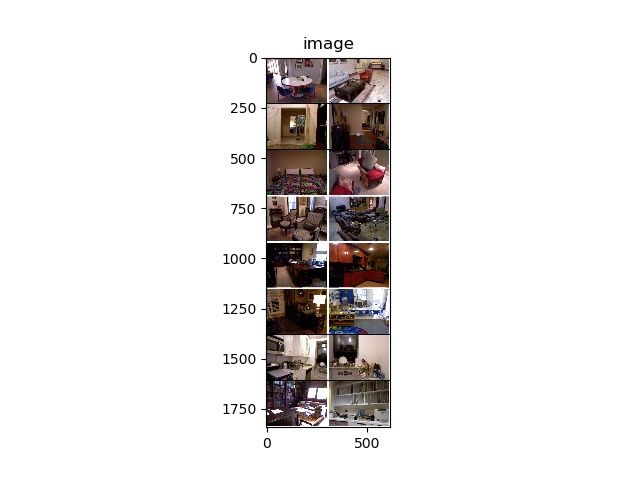
两个网络的具体参数设置可翻阅本篇报告中的附录A和B。

* + 1. 损失函数

// TODO by 鹏飞

* + 1. 数据处理

由于三个网络的输入输出和预测拼接流程不同，我们需要对数据集做训练、测试、验证集的数量分割，并对图像和相应深度图做尺度换边、大小裁剪等具体操作，最终得到如下图所示的一批图像和深度图。



* + 1. 训练、测试步骤

**训练**过程分两步：

**第一步：**训练Net12，本地保存网络参数。得到训练和测试loss和准确率。

**第二步：**加载Net12和对应的网络参数，设置为仅作预测模式。训练Net3，在网络卷积运算中加入Net12的预测输入。得到训练和测试loss和准确率。

整体**测试**步骤：加载Net12和Net3和对应的网络参数，均设置为预测模式，得到输入图像的深度图显示。

1. 项目框架
   1. PyTorch和CUDA

框架版本和计算环境

// TODO by 鹏飞

* 1. 代码基本结构和模块

├── checkpoints // 目录

│   ├── net12.pth // 训练后本地保存的Net12网络模型参数

│   └── net3.pth // 训练后本地保存的Net3网络模型参数

├── data // 目录 NYU数据集

├── data.py

// 实现数据集的读取和分割 以及 继承PyTorch DataSet类，实现我们自己的MyDataSet类，实现数据变换

├── lossF.py

// 基于PyTorch实现损失函数类MyLoss和准确率计算函数 count\_acc

├── main.py // Net12和Net3的整体图像预测

├── net.py // 基于PyTorch实现Net12（Scale12）和Net3（Scale3）

├── param\_utils.py // 训练的超参数设计

├── trainNet12.py // 训练和测试Net12

├── trainNet3.py // 训练和测试Net3

1. 实现细节
   1. 网络结构

Scale12继承了PyTorch中的nn.Module模块，我们的设计由两部分组成：\_\_init\_\_定义网络结构，forward中组合各个结构。

class Scale12(nn.Module):

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

super(Scale12, self).\_\_init\_\_()

self.N = N

self.layer1\_1 = nn.Sequential(

nn.Conv2d(3, 64, kernel\_size=3, stride=1),

nn.ReLU(),

nn.Conv2d(64, 64, kernel\_size=3, stride=1),

nn.ReLU(),

nn.MaxPool2d(kernel\_size=2, stride=2)

)

self.layer1\_2 = …

…

self.layer1\_5 = …

self.layer1\_6 = nn.Sequential(

nn.Linear(512 \* 9 \* 7, 4096),

nn.ReLU(),

nn.Dropout()

)

self.layer1\_7 = = nn.Sequential(

nn.Linear(4096, 64 \* 19 \* 14),

nn.ReLU(),

nn.Dropout()

)

self.layer2\_1 = nn.Sequential(

nn.Conv2d(3, 96, kernel\_size=9, stride=2, padding=1),

nn.ReLU(),

nn.MaxPool2d(kernel\_size=3, stride=2)

)

self.layer2\_2 = …

self.layer2\_3 = …

self.layer2\_4 = …

def forward(self, x):

out1 = self.layer1\_1(x)

…

out1 = out1.view(self.N, 64, 19, 14)

# reshape Nx17024 to Nx64x19x14

m1 = nn.Upsample(scale\_factor=4, mode='bilinear')

out1 = m1(out1)

…

out2 = torch.cat([out2, out1], 1)

…

return out2

上述代码中的self.layer1\_x为Scale1中的网络结构。self.layer1\_1即我们定义的一个序列结构，通过nn.Sequential()序列化地组合了卷积（被卷积的图像为3通道，64个卷积核，每个卷积核的大小为3x3，步长为1）、ReLu、卷积、ReLu、最大池化（池化区域为2x2，步长为2），共5步操作。Scale1采用了VGG Net的网络结构（即layer1\_1 到 layer1\_5），为输出匹配尺寸的图像输出，我们在layer1\_6和layer1\_7将Tensor变换为64\*19\*14。

其中self.layer2\_x为Scale2中的网络结构。我们采用unsample函数和cat函数，实现了Scale1对Scale2的拼接输入操作。

Scale3的网络结构定义与上述相似，具体参数和定义可以查看附录和具体代码。不同的是，由于Net3训练过程中，Net12已经完成训练步骤，所以需要先实例化Net12，并加载对应参数，作为参数输入到Scale3的\_\_init\_\_中。并在forward中做unsample操作，以匹配Tensor维度。

class Scale3(nn.Module):

def \_\_init\_\_(self, net12):

def forward(self, x):

…

out2 = self.net12.forward(x)

m2 = nn.Upsample(scale\_factor=2, mode='bilinear')

out2 = m2(out2)

…

out3 = torch.cat([out3, out2], 1)

…

return out3

* 1. 损失函数

// TODO by 鹏飞

* 1. 数据处理

// TODO by 欣怡

* 1. 训练、测试

训练Net12时，我们采取kaiming-normal方法，初始化网络结构中权重，init.kaiming\_normal(m.weight)，并用init.uniform(m.bias)初始化bias

model = Scale12(BATCH\_SIZE)

init\_weight(model.state\_dict())

采用我们设计的MyLoss（）函数计算损失，采用动量、随机梯度下降法优化。

criterion = MyLoss()

optimizer\_ft = optim.SGD(model.parameters(), lr=LR, momentum=MOMENTUM)

exp\_lr\_scheduler = lr\_scheduler.StepLR(optimizer\_ft, step\_size=STEP\_SIZE, gamma=GAMMA)

model\_final = train\_model(model=model,

criterion=criterion,

optimizer=optimizer\_ft,

scheduler=exp\_lr\_scheduler,

num\_epochs=EPOCHS,

delta=DELTA)

torch.save(model\_final.state\_dict(), path + "/checkpoints/net12.pth")

完成计算后，将模型参数保存在本地。

训练网络Net3时，大体上与训练Net12相似，不同的是，需要先用本地模型参数初始化Net12，并作为Net3的输入。

net12 = Scale12(BATCH\_SIZE)

net12\_model\_wts = path + "/checkpoints/net12.pth"

net12.load\_state\_dict(torch.load(net12\_model\_wts))

net12.eval()

model = Scale3(net12)

init\_weight(model.state\_dict())

为整体预测，仅需执行main.py，其中Net12和Net3分别加载网络参数，设置为测试模式，对输入图像做深度估计，可视化地得到结果。

1. 实验结果与分析
   1. 实验结果

// 不同参数设置下的表格和图片

* 1. 比较、分析

// 前人论文结果对比

// 实验结果的对比需要5、6个例子，并且 side by side 地 Highlight diff

1. 结论与体会

**小组分工**

李鹏飞：

黄奇浩：

李欣怡：

**参考文献**

**附录**

1. **Scale1和Scale2网络结构**

Scale12(

(layer1\_1): Sequential(

(0): Conv2d(3, 64, kernel\_size=(3, 3), stride=(1, 1))

(1): ReLU()

(2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1))

(3): ReLU()

(4): MaxPool2d(kernel\_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil\_mode=False)

)

(layer1\_2): Sequential(

(0): Conv2d(64, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): ReLU()

(2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(3): ReLU()

(4): MaxPool2d(kernel\_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil\_mode=False)

)

(layer1\_3): Sequential(

(0): Conv2d(128, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): ReLU()

(2): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(3): ReLU()

(4): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(5): ReLU()

(6): MaxPool2d(kernel\_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil\_mode=False)

)

(layer1\_4): Sequential(

(0): Conv2d(256, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): ReLU()

(2): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(3): ReLU()

(4): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(5): ReLU()

(6): MaxPool2d(kernel\_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil\_mode=False)

)

(layer1\_5): Sequential(

(0): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): ReLU()

(2): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(3): ReLU()

(4): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(5): ReLU()

(6): MaxPool2d(kernel\_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil\_mode=False)

)

(layer1\_6): Sequential(

(0): Linear(in\_features=32256, out\_features=4096, bias=True)

(1): ReLU()

(2): Dropout(p=0.5)

)

(layer1\_7): Sequential(

(0): Linear(in\_features=4096, out\_features=17024, bias=True)

(1): ReLU()

(2): Dropout(p=0.5)

)

(layer2\_1): Sequential(

(0): Conv2d(3, 96, kernel\_size=(9, 9), stride=(2, 2), padding=(1, 1))

(1): ReLU()

(2): MaxPool2d(kernel\_size=(3, 3), stride=(2, 2), dilation=(1, 1), ceil\_mode=False)

)

(layer2\_2): Sequential(

(0): Conv2d(160, 64, kernel\_size=(5, 5), stride=(1, 1), padding=(2, 2))

(1): ReLU()

)

(layer2\_3): Sequential(

(0): Conv2d(64, 64, kernel\_size=(5, 5), stride=(1, 1), padding=(2, 2))

(1): ReLU()

)

(layer2\_4): Sequential(

(0): Conv2d(64, 1, kernel\_size=(5, 5), stride=(1, 1), padding=(2, 2))

(1): ReLU()

)

)

1. **Scale3网络结构**

Scale3(

(layer3\_1): Sequential(

(0): Conv2d(3, 96, kernel\_size=(9, 9), stride=(2, 2))

(1): ReLU()

(2): MaxPool2d(kernel\_size=(2, 2), stride=(1, 1), dilation=(1, 1), ceil\_mode=False)

)

(layer3\_2): Sequential(

(0): Conv2d(97, 64, kernel\_size=(5, 5), stride=(1, 1), padding=(2, 2))

(1): ReLU()

)

(layer3\_3): Sequential(

(0): Conv2d(64, 64, kernel\_size=(5, 5), stride=(1, 1), padding=(2, 2))

(1): ReLU()

)

(layer3\_4): Sequential(

(0): Conv2d(64, 1, kernel\_size=(5, 5), stride=(1, 1), padding=(2, 2))

(1): ReLU()

)

)