**认知科学与类脑计算**

**大作业**

**设**

**计**

**报**

**告**

|  |  |
| --- | --- |
| 题目： | 临床脊椎检测定位 |
| 姓名： | 李思霏 |
| 学号： | 201800171028 |

2021年12月

摘要：在临床实践中，由于透视图像中解剖结构的重叠性和脊椎的高重复性，手工标记难以达到较高成功率，设计一种自动化脊椎标记工具很有必要。但在医疗图像分析应用中，数据集十分有限，训练卷积神经网络很有挑战性。本系统结合热力图思想进行问题转化设计了基于U-net网络的自动化脊椎标记工具，可以实现精确的脊椎定位。

关键字：自动脊椎定位、Unet、热力图、医疗影像处理

**目录**

[**一、 问题描述 1**](#_Toc91012189)

[**二、 问题分析 1**](#_Toc91012190)

[**三、 方案介绍 2**](#_Toc91012191)

[**1. U-net网络介绍 3**](#_Toc91012192)

[**四、 程序设计 3**](#_Toc91012193)

[**1. 实现步骤 3**](#_Toc91012194)

[**2. 代码及注释 4**](#_Toc91012195)

[**五、 结果及结果分析 18**](#_Toc91012196)

[**1. 网络学习能力分析 18**](#_Toc91012197)

[**2. 高斯分布不同σ选取的影响 18**](#_Toc91012198)

[**3. 根据预测结果对图像标注 19**](#_Toc91012199)

[**六、 总结分析 19**](#_Toc91012200)

## 问题描述

脊椎标记可以为多种脊柱疾病提供诊断参考，如脊柱侧弯和后突、脊柱骨折等。随着久坐人群数目增长，身患脊柱相关疾病的人数与日俱增，设计自动脊椎标记工具可以一定程度缓解对相关专业医生的需求。而医疗领域数据集规模普遍较小，利用小数据集结合CNN训练模型仍有较大挑战。本系统要求设计一种自动化脊椎标记工具，要求在给定的胸部透视图像数据集上脊椎标记成功率达到80%以上（预测标记点与真值小于1cm视为成功）。

## 问题分析

对脊椎进行定位需要网络模型能够准确的学到位置信息，而提供的数据集中含训练数据集403张脊椎影像，测试数据集135张影像，数据集规模较小，因此需要选择合适的网络模型能够充分利用有限的数据集，同时可获得精确的定位信息。

数据集中的标签为图像中每块脊椎中心点坐标，以图像作为输入直接回归坐标，会引入稠密的全连接层，模型规模会很大，因此需要对问题进行转化，可转化为图像拟合问题避免全连接层，但是若只为坐标点处像素赋值，整张图像的信息量很小，学习过程中，稍微一点偏差就会带来较大的梯度，学习困难。可以考虑热力图的思想，以坐标点为中心借助高斯分布生成一定范围内的圆晕，便于网络学习。进行问题转化后，学习目标发生改变，网络除需要学到精确的位置信息，还需学到图像的语义信息，拟合出圆晕。

综上所述，需要选择一个适用于医疗影像领域，可以同时学到图像的语义信息和定位信息的网络。

## 方案介绍

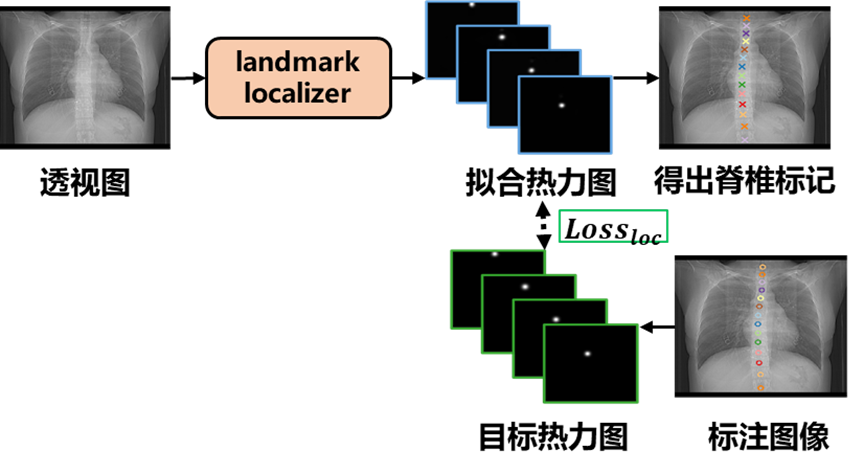
通过问题分析和调研，确定本设计的最终方案，方案示意图如图1所示：模型选择上本设计选用了常用于医疗影像分割领域的网络U-net；问题转化上，改变了拟合目标，将坐标回归问题转化为图像拟合问题，同时引入了热力图的思想。

图1：方案示意图

根据标注图像，借助二维高斯分布公式（1）生成目标热力图，人体共有26块脊椎，因此目标热力图有26个通道与之一一对应，图像中不包含的脊椎块对应通道值为全0，包含的脊椎快对应位置会有圆晕。将透视图作为网络输入，令模型输出拟合目标热力图，预测热力图和目标热力图之间差别即为误差，随着训练的进行，预测热力图会逐渐逼近目标热力图。预测热力图每个通道中最亮的点的坐标即为对应脊椎预测的坐标，因此预测热力图经后处理便可得到预测坐标，可根据预测坐标为原图像做出标记。

（1）

1. U-net网络介绍

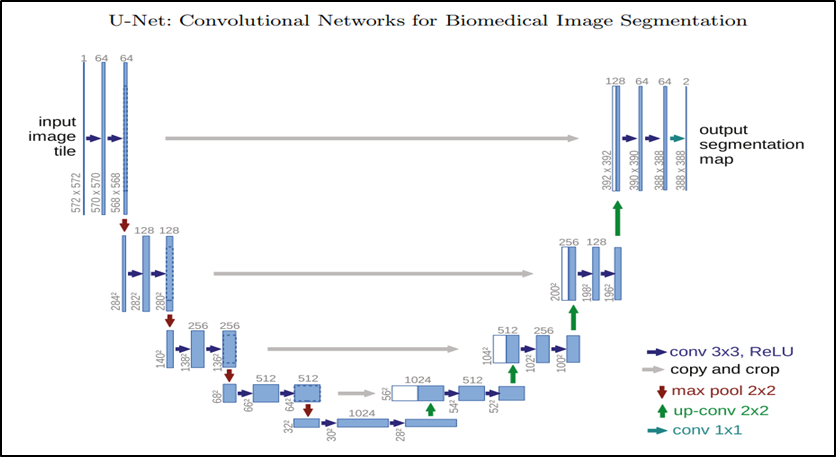
U-net在图像分割中有杰出表现，能很好的拟合目标图像，利用少数图片就可进行端到端训练。U-net结构示意图如图2所示，整体呈现U字形，下采样提取语义信息，上采样可以精确定位，可以更好的拟合图像；多尺度特征融合保留了更多的信息，上采样下采样对应位置进行拼接，有利于应对小数据集。

图2：U-net网络结构示意图

## 程序设计

1. 实现步骤

（1）生成目标热力图：读取标记文件，存储始末脊椎序号，依据二维高斯分布公式、选取合适σ，生成26通道热力图，并对生成的热力图进行归一化。

（2）修改U-net网络：将输出通道修改为26通道（原网络为图像分割的二分类任务）。

（3）选择优化器、调参：经不断尝试最终选用SGD优化器、σ=12、学习率采用分阶段调整的方法，损失函数选用BCEWithLogits损失。

（4）计算坐标及正确率：根据预测结果得出各个通道最亮点的坐标，并与标记文件对比计算正确率（单个像素代表真实距离0.75mm，因此预测结果与真实坐标间距离小于13.33即预测正确），测试集正确率可达84.83%，平均偏差距离2.174mm。

（5）依据预测坐标标注图像，根据预测坐标在原图像上绘制小红圈，将脊椎标记出来。

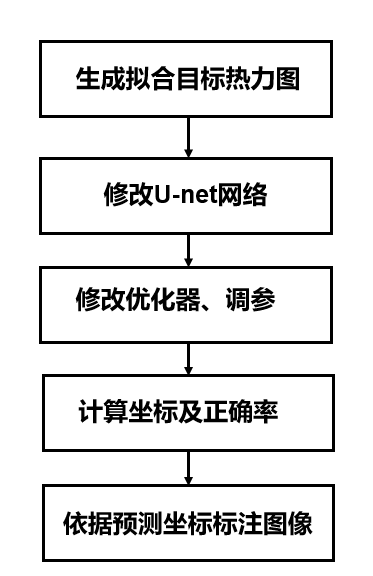


图3：设计实现步骤示意图

1. 代码及注释

本设计共有六个python文件，unet\_model.py是网络模型类文件，unet\_parts.py是实现Unet网络的基本组件，U-net.py是训练文件，prediction.py是预测并存储预测坐标的文件，accuracy.py是计算测试集正确率的文件，test.py是为单个样本图像添加脊椎定位标记的文件。

1. unet\_model.py

#模型文件

import torch.nn.functional as F

from unet\_parts import \*

class UNet(nn.Module):

    def \_\_init\_\_(self, n\_channels, n\_classes, bilinear=True):

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

        self.n\_channels = n\_channels

        self.n\_classes = n\_classes

        self.bilinear = bilinear

        self.inc = DoubleConv(n\_channels, 64)

        self.down1 = Down(64, 128)#下采样

        self.down2 = Down(128, 256)

        self.down3 = Down(256, 512)

        factor = 2 if bilinear else 1

        self.down4 = Down(512, 1024 // factor)

        self.up1 = Up(1024, 512 // factor, bilinear)#上采样

        self.up2 = Up(512, 256 // factor, bilinear)

        self.up3 = Up(256, 128 // factor, bilinear)

        self.up4 = Up(128, 64, bilinear)

        self.outc = OutConv(64, n\_classes)

    def forward(self, x):

        x1 = self.inc(x)

        x2 = self.down1(x1)

        x3 = self.down2(x2)

        x4 = self.down3(x3)

        x5 = self.down4(x4)

        x = self.up1(x5, x4)

        x = self.up2(x, x3)

        x = self.up3(x, x2)

        x = self.up4(x, x1)

        logits = self.outc(x)

        return logits

1. unet\_parts.py

""" Parts of the U-Net model """

import torch

import torch.nn as nn

import torch.nn.functional as F

class DoubleConv(nn.Module):#二次卷积模块，不改变尺寸大小，只改变通道数

    """(convolution => [BN] => ReLU) \* 2"""

    def \_\_init\_\_(self, in\_channels, out\_channels, mid\_channels=None):

        super().\_\_init\_\_()

        if not mid\_channels:

            mid\_channels = out\_channels

        self.double\_conv = nn.Sequential(

            nn.Conv2d(in\_channels, mid\_channels, kernel\_size=3, padding=1),

            nn.BatchNorm2d(mid\_channels),

            nn.ReLU(inplace=True),

            nn.Conv2d(mid\_channels, out\_channels, kernel\_size=3, padding=1),

            nn.BatchNorm2d(out\_channels),

            nn.ReLU(inplace=True)

        )

    def forward(self, x):

        return self.double\_conv(x)

class Down(nn.Module):#下采样模块

    """Downscaling with maxpool then double conv"""

    def \_\_init\_\_(self, in\_channels, out\_channels):

        super().\_\_init\_\_()

        self.maxpool\_conv = nn.Sequential(

            nn.MaxPool2d(2),

            DoubleConv(in\_channels, out\_channels)

        )

    def forward(self, x):

        return self.maxpool\_conv(x)

class Up(nn.Module):#上采样模块

    """Upscaling then double conv"""

    def \_\_init\_\_(self, in\_channels, out\_channels, bilinear=True):

        super().\_\_init\_\_()

        # if bilinear, use the normal convolutions to reduce the number of channels

        if bilinear:

            self.up = nn.Upsample(scale\_factor=2, mode='bilinear', align\_corners=True)

            self.conv = DoubleConv(in\_channels, out\_channels, in\_channels // 2)

        else:

            self.up = nn.ConvTranspose2d(in\_channels , in\_channels // 2, kernel\_size=2, stride=2)

            self.conv = DoubleConv(in\_channels, out\_channels)

    def forward(self, x1, x2):#拼接模块

        x1 = self.up(x1)

        # input is CHW

        diffY = x2.size()[2] - x1.size()[2]

        diffX = x2.size()[3] - x1.size()[3]

        x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2,

                        diffY // 2, diffY - diffY // 2])

        # if you have padding issues, see

        # https://github.com/HaiyongJiang/U-Net-Pytorch-Unstructured-Buggy/commit/0e854509c2cea854e247a9c615f175f76fbb2e3a

        # https://github.com/xiaopeng-liao/Pytorch-UNet/commit/8ebac70e633bac59fc22bb5195e513d5832fb3bd

        x = torch.cat([x2, x1], dim=1)

        return self.conv(x)

class OutConv(nn.Module):#输出卷积

    def \_\_init\_\_(self, in\_channels, out\_channels):

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

        self.conv = nn.Conv2d(in\_channels, out\_channels, kernel\_size=1)

    def forward(self, x):

        return self.conv(x)

1. U-net.py

import json

import numpy as np

import numpy.matlib

import matplotlib.pyplot as plt

from PIL import Image

import os

import torch

import torch.nn.functional as F

import torch.utils.data as Data

from unet\_model import UNet

#数据集路径

train\_path = './dataset/train/'

val\_path = './dataset/val/'

#图片尺寸、脊椎数目、训练集测试集数目

IMAGE\_WIDTH = 512

IMAGE\_HEIGHT = 512

num\_vertebreas = 26

num\_train = 403

num\_val = 135

#初始化输入矩阵和目标矩阵

X\_train = np.zeros((num\_train, IMAGE\_HEIGHT, IMAGE\_WIDTH))

Gauss\_map = np.zeros((num\_train, num\_vertebreas, IMAGE\_HEIGHT, IMAGE\_WIDTH))

#元素值为所在行数矩阵

x1 = np.arange(IMAGE\_WIDTH)

x\_map = np.matlib.repmat(x1, IMAGE\_HEIGHT, 1)

#元素值为所在列数矩阵

y1 = np.arange(IMAGE\_HEIGHT)

y\_map = np.matlib.repmat(y1, IMAGE\_WIDTH, 1)

y\_map = np.transpose(y\_map)

R = 12#高斯分布δ

k = 0

first = []#存储图片中包含的第一个脊椎的标号

last = []#存储图片中包含的最后一个脊椎的标号

position = np.zeros((num\_train, num\_vertebreas, 2))#初始化脊椎位置矩阵

for i,fn in enumerate(os.listdir(train\_path)):

    if '.json' in fn:#保证标签和图片的顺序一致

        X\_train[k] = plt.imread(train\_path + fn[:-4]+'png')#读取图片

        X\_train[k] = (X\_train[k]-np.min(X\_train[k]))/(np.max(X\_train[k])-np.min(X\_train[k]))#归一化

        with open (train\_path + fn) as f:

            landmarks = json.load(f)

            #first = landmarks[0]['label']

            first.append(landmarks[0]['label']-1)#存储首尾标号索引

            last.append(landmarks[-1]['label']-1)

            for j,landmark in enumerate(landmarks):#为第i个文件中第j个脊椎标记生成Heatmap

                center\_x = landmark['X']

                center\_y = landmark['Y']

                position[k][j+first[k]][0] = center\_x#标签矩阵

                position[k][j+first[k]][1] = center\_y

                mask\_x = np.matlib.repmat(center\_x, IMAGE\_HEIGHT, IMAGE\_WIDTH)

                mask\_y = np.matlib.repmat(center\_y, IMAGE\_HEIGHT, IMAGE\_WIDTH)

                #Gauss\_map[i][j+first-1] = np.sqrt((x\_map-mask\_x)\*\*2+(y\_map-mask\_y)\*\*2)

                Gauss\_map[k][j+first[k]] = ((x\_map-mask\_x)\*\*2+(y\_map-mask\_y)\*\*2)/(2\*R\*R)#x\_map-mask\_X得出距离

                #Gauss\_map[i][j+first-1] = 100\*np.exp(-0.5\*Gauss\_map[i][j+first-1]/R)

                Gauss\_map[k][j+first[k]] = 1/(2\*np.pi\*R\*R) \* np.exp(-Gauss\_map[k][j+first[k]])

                Max = np.max(Gauss\_map[k][j+first[k]])

                Min = np.min(Gauss\_map[k][j+first[k]])

                Gauss\_map[k][j+first[k]] = (Gauss\_map[k][j+first[k]]-Min)/(Max-Min)#热力图归一化

        k += 1

Y\_train = Gauss\_map

#model\_path = './model/130modelR1210.pkl'#kequ

device = torch.device('cuda:0' if torch.cuda.is\_available() else 'cpu')#初始化模型

model = UNet(n\_channels=1, n\_classes=num\_vertebreas).to(device)

#model.load\_state\_dict(torch.load(model\_path, map\_location={'cuda:0':'cuda:1'}))#kequ

#model = torch.nn.DataParallel(model)

model.train()

optim = torch.optim.SGD(model.parameters(), lr = 1e-1, momentum=0.9, weight\_decay=3e-4)#初始化优化器

#dataloader = ...

epochs = 100

BATCH\_SIZE = 4

x\_train = torch.from\_numpy(X\_train)#变量类型转换为模型输入所需的类型

x\_train = x\_train.to(torch.float32)

y\_train = torch.from\_numpy(Y\_train)

y\_train = y\_train.to(torch.float32)

Dataset = Data.TensorDataset(x\_train, y\_train)#数据集包

loader = Data.DataLoader(dataset = Dataset, batch\_size = BATCH\_SIZE, shuffle = False, num\_workers = 0,)

Loss = torch.nn.BCEWithLogitsLoss()#初始化损失函数

#lossSum = 0

for i in range(epochs):#训练

    lossSum = 0

    #for X, y in zip(x\_train,y\_train):

    k = 0

    for X, y in loader:

        #X = torch.unsqueeze(X, 0)

        X = torch.unsqueeze(X, 1)

        #y = torch.unsqueeze(y, 0)

        X = X.to(device)  # [N, 1, H, W]将变量转到GPU中

        y = y.to(device)  # [N, H, W] with class indices (0, 1)

        model.train()

        prediction = model(X)  # [N, 2, H, W]

        loss = Loss(prediction, y)

        prediction\_np = prediction.data.cpu().detach().numpy()[0]#取单个batch中的一个样本

        y\_np = y.data.cpu().detach().numpy()[0]

        #prediction\_np\_show = np.mean(prediction\_np,axis=0,keepdims=False)

        #y\_np\_show = np.mean(y\_np,axis=0,keepdims=False)

        prediction\_np\_show = prediction\_np[10]#取一个样本中的一块脊椎的拟合情况进行观察

        y\_np\_show = y\_np[10]

        plt.subplot(121)

        plt.imshow(prediction\_np\_show)

        plt.subplot(122)

        plt.imshow(y\_np\_show)

        plt.savefig("./tmp\_image/{}.png".format(k))

        plt.clf()

        lossSum += loss #优化

        optim.zero\_grad()

        loss.backward()

        optim.step()

        #print('step:',k, 'lossSum:', lossSum)

        k += 1

        if k==2:#输出第二个batch当前的预测结果

            model.eval()

            pred\_train = model(X)

            y0 = y.data.cpu().detach().numpy()[0]#取出这个batch中的第一个样本

            y0\_flatten = y0.reshape(1,num\_vertebreas, -1)

            max\_index = np.argmax(y0\_flatten, axis=2)

            y\_label = np.floor((max\_index+1)/512)#计算出预测的位置

            x\_label = (max\_index+1)%512 -1

            y\_label[x\_label<0] -= 1

            x\_label[x\_label<0] = 511

            #输出预测结果和标签

            print('X\_label:', x\_label)

            print('Y\_label', y\_label)

            print('position:',position[4+0])

            pred\_train = pred\_train.cpu().detach().numpy()

            pred\_flatten = pred\_train.reshape(pred\_train.shape[0],num\_vertebreas, -1)#铺平feature map

            index = np.argmax(pred\_flatten, axis=2)#计算这个batch所有的坐标

            y\_axis = np.floor((index+1)/512)

            x\_axis = (index+1)%512 -1

            y\_axis[x\_axis<0] -= 1

            x\_axis[x\_axis<0] = 511

            x\_axis = x\_axis.reshape(x\_axis.shape[0], x\_axis.shape[1], 1)

            y\_axis = y\_axis.reshape(y\_axis.shape[0], y\_axis.shape[1], 1)

            pred\_position = np.concatenate((x\_axis,y\_axis), axis=2)#坐标拼接

            print('pred\_position:', pred\_position[0])

            errors = []#计算本batch的距离误差

            for m in range(pred\_position.shape[0]):

                error = np.sum(np.square(pred\_position[m,first[4+m]:last[4+m]] - position[m+4,first[4+m]:last[4+m]]))

                errors.append(error/(last[4+m]-first[4+m]+1))

            #errorsum += error

            print('errors:',errors)

    print('epoch',i,': loss:',lossSum)

    # if i % 10 ==0:

    #     torch.save(model.state\_dict(),'./1e-3modelR101'+str(i)+'.pkl')

torch.save(model.state\_dict(), './model.pkl')

1. prediction.py

from unet\_model import UNet

import numpy as np

import torch

from torch import nn

import torch.nn.functional as F

import os

import json

import matplotlib.pyplot as plt

import torch.utils.data as Data

from PIL import Image

from PIL import ImageFilter

from torchvision import transforms

num\_val = 135

num\_vertebreas = 26

model\_path = './model/150modelR12.pkl'

#model\_path = './1e-3modelR10130.pkl'

val\_path = './dataset/val/'

device = torch.device('cuda:2' if torch.cuda.is\_available() else 'cpu')#设置使用的GPU

#加载模型

net = UNet(n\_channels=1,n\_classes=num\_vertebreas).to(device).train()

net.load\_state\_dict(torch.load(model\_path, map\_location={'cuda:0':'cuda:2'}))

#net = nn.DataParallel(net)

#读取测试集X，y

X\_val = np.zeros((num\_val,512,512))

position = np.zeros((num\_val, num\_vertebreas, 2))

k = 0

first = []

last = []

#生成训练集的Heatmap

for i,fn in enumerate(os.listdir(val\_path)):#i代表第i个文件

    if '.json' in fn:

        X\_val[k] = plt.imread(val\_path + fn[:-4]+'png')

        X\_val[k] = (X\_val[k]-np.min(X\_val[k]))/(np.max(X\_val[k])-np.min(X\_val[k]))

        with open (val\_path + fn) as f:

            landmarks = json.load(f)

            first.append(landmarks[0]['label'] -1)

            last.append (landmarks[-1]['label'] -1)#直接存储索引

            #print(first[k])

            for j,landmark in enumerate(landmarks):#j代表第i个文件中第j个标记

                position[k][j+first[k]][0] = landmark['X']

                position[k][j+first[k]][1] = landmark['Y']

        k += 1

fn\_set = []#存储读取文件顺序

for i,fn in enumerate(os.listdir(val\_path)):#i代表第i个文件

    if '.json' in fn:

        fn\_set.append(fn)

#预测并得出坐标

x\_val = torch.from\_numpy(X\_val)

x\_val = x\_val.to(torch.float32)

x\_val = torch.unsqueeze(x\_val, 1)

x\_val = x\_val.to(device)

x\_batch = Data.DataLoader(dataset = x\_val, batch\_size = 4, shuffle = False, num\_workers = 0,)

for num,X in enumerate(x\_batch):#由于GPU的限制只能四张四张预测

    #print('X:',X.size())

    prediction = net(X)

    prediction\_np = prediction.data.cpu().detach().numpy()[0]

    #y\_np = y.data.cpu().detach().numpy()[0]

    prediction\_np\_show = np.mean(prediction\_np,axis=0,keepdims=False)#对预测结果通道取均值

    plt.subplot(111)

    plt.imshow(prediction\_np\_show,plt.cm.gray)#展示预测结果

    plt.savefig("./val\_image/{}.png".format(num))#存储预测结果

    #计算预测坐标并存储为对应文件

    prediction = prediction.cpu().detach().numpy()#计算预测坐标并存储为对应文件

    pred\_flatten = prediction.reshape(X.shape[0], num\_vertebreas, -1)

    index = np.argmax(pred\_flatten,axis=2)

    y = np.floor((index + 1)/512)

    #print(x)

    x = (index + 1)%512 -1

    y[x<0] -= 1

    x[x<0] = 511

    x = x.reshape(x.shape[0],x.shape[1],1)

    y = y.reshape(y.shape[0],y.shape[1],1)

    pred\_position = np.concatenate((x,y), axis=2)

    if num < 33:#由于测试集总数不能整除4

        list1 = range(4)

    else:

        list1 = range(3)

    for m in list1:

        np.savetxt('./dataset/mark/' + fn\_set[num\*4+m][:-4]+'txt', pred\_position[m], fmt='%f',delimiter=',')#存储预测的文件

1. accuracy.py

import os

from os import listdir

import numpy as np

import json

label\_path = './dataset/val/'#预测标记文件路径

mark\_path = './dataset/mark/'#真实标记文件路径

num\_right=0

all\_point=0

label = np.zeros((26,2))

for i,fn in enumerate(os.listdir(label\_path)):

    if 'json' in  fn:

        mark = np.loadtxt(mark\_path+fn[:-4]+'txt', delimiter=',')

        with open(label\_path+fn) as f:

            labels = json.load(f)

        first = labels[0]['label']-1#读取标签文件

        last = labels[-1]['label']-1

        for i,landmark in enumerate(labels):

            label[i+first][0] = landmark['X']

            label[i+first][1] = landmark['Y']

        Mark = mark[first:last+1]#读取预测文件

        Label = label[first:last+1]

        error  = np.sqrt(np.sum(np.square(Mark-Label),axis=1))#计算预测值与真实值像素距离

        right = error[error<13.33]#一个像素代表0.75mm，真实距离小于1cm视为预测正确

        num\_right += right.shape[0]#预测正确的脊椎数目

        all\_point += Label.shape[0]#总共预测的脊椎数目

        print(label)

        print(mark)

        print(error)

    accuracy = num\_right /all\_point#计算正确率

    print(accuracy)

1. test.py

from unet\_model import UNet

import numpy as np

import torch

from torch import nn

import torch.nn.functional as F

import os

import json

import matplotlib.pyplot as plt

import torch.utils.data as Data

from PIL import Image

from PIL import ImageFilter

from torchvision import transforms

import cv2

import  argparse

num\_vertebreas = 26

#定义参数，便于测试时调整侧视图像和模型

parser = argparse.ArgumentParser()

parser.add\_argument('--model-path', type=str, default='./model/150modelR12.pkl', help='the path of model' )

parser.add\_argument('--image-path', type=str, default='./test/LIDC-IDRI-0020.png', help='the path of image' )

args = parser.parse\_args()

#定义设备

device = torch.device('cuda:0' if torch.cuda.is\_available() else 'cpu')

#初始化模型并加载

net = UNet(n\_channels=1,n\_classes=num\_vertebreas).to(device).train()

net.load\_state\_dict(torch.load(args.model\_path))

#net.load\_state\_dict(torch.load(model\_path, map\_location={'cuda:0':'cuda:2'}))

#读取数据

X\_test = plt.imread(args.image\_path)

X\_test = (X\_test-np.min(X\_test))/(np.max(X\_test)-np.min(X\_test))

x\_test = torch.from\_numpy(X\_test)

x\_test = x\_test.to(torch.float32)

x\_test = torch.unsqueeze(x\_test, 0)#转化为pytorch网络可以接收的输入维度

x\_test = torch.unsqueeze(x\_test, 0)

x\_test = x\_test.to(device)

#预测并进行后处理

with torch.no\_grad():

    prediction = net(x\_test)

    prediction = prediction.cpu().detach().numpy()

    #print(prediction.shape)

    pred\_flatten = prediction.reshape(x\_test.shape[0], num\_vertebreas, -1)

    index = np.argmax(pred\_flatten,axis=2)#得出最亮点的索引

    #计算最亮点在图像中的坐标

    y = np.floor((index + 1)/512)

    #print(x)

    x = (index + 1)%512 -1

    y[x<0] -= 1

    x[x<0] = 511

    #形成坐标对

    x = x.reshape(x.shape[0],x.shape[1],1)

    y = y.reshape(y.shape[0],y.shape[1],1)

    pred\_position = np.concatenate((x,y), axis=2)

    pred\_position = np.squeeze(pred\_position, 0)

    #print(pred\_position)

#加载标签文件便于检测测试结果

with open(args.image\_path[:-3]+'json') as f:

    labels = json.load(f)

#确定图像中包含的脊椎

first = labels[0]['label']-1#读取标签文件

last = labels[-1]['label']-1

pred\_position = pred\_position[first:last+1]

print(pred\_position)

#根据预测的坐标在图像上绘制红色圆圈加一体现

image = cv2.imread(args.image\_path)

for i in range(np.shape(pred\_position)[0]):

    cv2.circle(image, (int(pred\_position[i][0]),int(pred\_position[i][1])), radius=4, thickness=2, color=(0,0,255) )

#存储标记后的图像

#cv2.imshow('result', image)

save\_path = './result/'+args.image\_path[7:]

cv2.imwrite(save\_path, image)

## 结果及结果分析

1. 网络学习能力分析

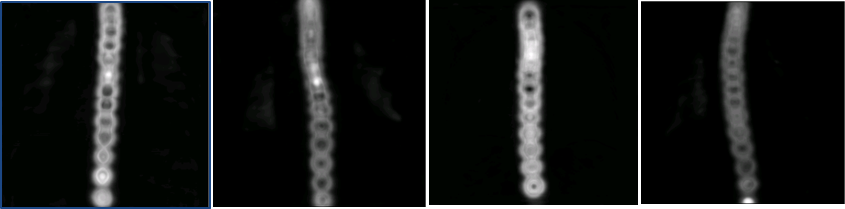


图4 拟合热力图按通道取均值效果展示

由图4可以看出，网络可以成功拟合目标热力图，对发生歪斜的脊柱也有较好响应，说明Unet网络在该场景下有较好的学习能力。

1. 高斯分布不同σ选取的影响



图5不同σ的拟合热力图效果展示

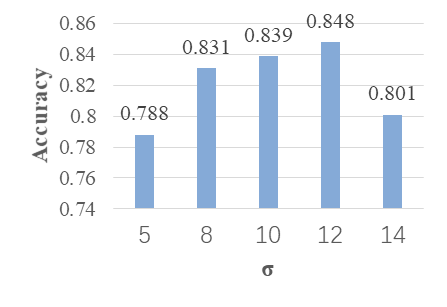
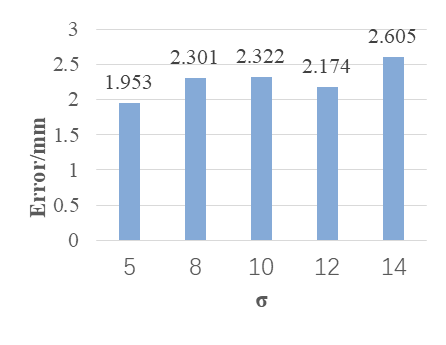


图6 不同σ对正确率和平均偏差距离的影响

图5清晰的呈现了不同σ对网络学习的影响，σ过小会使网络难以拟合，σ过大会使圆晕模糊，通过对图6分析可知σ=12时精度最大，为84.93%，平均偏差距离最小为2.174mm。

1. 根据预测结果对图像标注

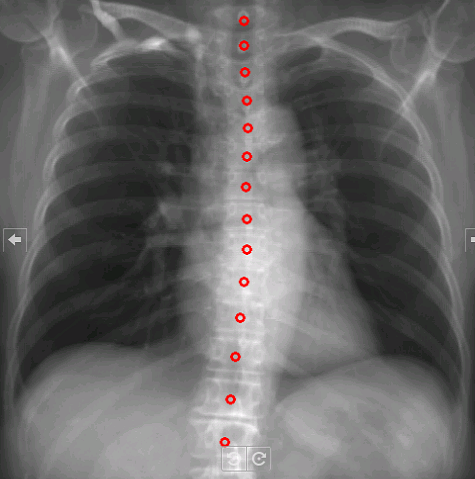


图7 输入图像和标注后图像对比

图7展示了将图像输入模型后可根据预测结果为图像添加的脊椎标注，说明网络可以精准定位脊椎位置。

## 总结分析

本设计在使用较小的数据集的情况下，引入热力图思想，利用Unet网络成功实现了精度较高的脊椎自动化标定工作，特别是局部精度，平均偏差距离可达2.174mm，远超实际需求，也可以将预测结果直观体现在输入图像上，具有较高的实用性。不足之处是网络会对个别位置的脊椎误判，预测为其他位置脊椎的坐标，后续可以考虑采用二阶段定位确保局部和全局都精确。