**国防科技大学研究生课程报告**

课程名称： 数学建模

项目名称： 岩石样本智能识别

授课学期： 2021年春季

学员姓名： 魏一 王闯

学 号： 20023135 20023136

**岩石样本智能识别**

**摘 要**

本文提出了一种岩石样本识别模型，用于分类岩性，达到了。。的效果。同时使用了无监督方法进行图片分割求得了含油百分比。Blabla。。。

**关键词：**岩石样本， 识别模型

**目 录**

[**1.** **题目简介** - 4 -](#_Toc67316863)

[**2.** **岩石样本识别模型** 4](#_Toc67316864)

[2.1 超分 4](#_Toc67316865)

[2.2 基于 4](#_Toc67316866)

[**3.** **岩石含油面积百分比** 2](#_Toc67316867)

[3.1 含油面积的计算 2](#_Toc67316868)

[3.2 岩石面积的计算 2](#_Toc67316869)

[3.2.1 二值化 2](#_Toc67316870)

[3.2.2 自适应阈值分割 2](#_Toc67316871)

[3.2.3 无监督的图片分割 2](#_Toc67316872)

[**4.** **结论** 1](#_Toc67316873)

[**参考文献** 2](#_Toc67316874)

[附录A 含油面积百分比求取算法 3](#_Toc67316875)

# **题目简介**

在油气勘探中，岩石样本识别是一项即基础又重要的环节；在矿产资源勘探中，尤其是在固体金属矿产资源勘探中，岩石样本识别同样发挥着不可估量的作用；岩石样本的识别与分类对于地质分析极为重要。目前岩石样本识别的方法主要有重磁、测井、地震、遥感、电磁、地球化学、手标本及薄片分析方法等方法，而采用图像深度学习的方法建立岩石样本自动识别分类模型是一条新的途径。

现有样本数据是采用工业相机在录井现场对岩屑和岩心样品进行拍照，分别在暗箱内拍摄白光和荧光两种相片，如图 1和图 2所示。白光灯下拍摄的相片是用于提取颜色、纹理、粒度等特征识别岩性，荧光灯下拍摄的相片是用于识别含油气性（石油在紫外线照射下具有的发光特征，其中的绿色和黄色部分是含油的，见图 2）。



图 1：白光灯下拍摄的照片

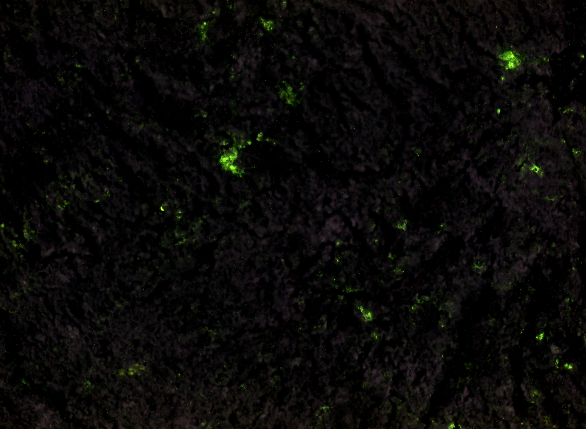


图 2：荧光灯下拍摄的照片

本文将在第2章介绍岩石样本识别模型，然后第3章介绍岩石含油面积百分比的计算，第0章是结论。

# **岩石样本识别模型**

## 2.1 超分

影像可以

## 2.2 基于

神经

# **岩石含油面积百分比**

岩石含油面积百分比的计算可以分为两部分进行：含油面积的计算和岩石面积的计算。这两部分的比值即为含油面积百分比。含油面积可以通过颜色阈值统计，岩石面积则使用了无监督图片分割算法。

## 3.1 含油面积的计算

岩石图片中含油部分在荧光下呈现黄绿色，使用荧光图片可以很容易分辨含油区域。我们首先将图片从BGR转化为HSV图，选定黄绿颜色区间进行二值化处理。最后统计白色像素值总和即为含油面积。由于图片像素值较大，使用了len函数进行快速统计色块面积。

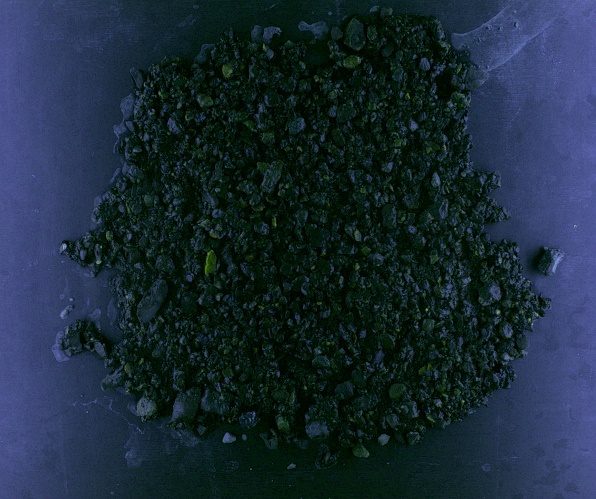


图 3：荧光照射下的图片

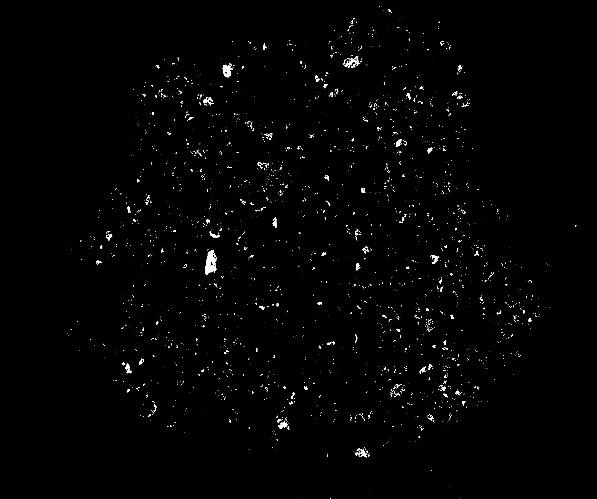


图 4：二值化处理后的图片

图 3和图 4展示了对含油面积的提取效果。此统计方法对数据集中的bmp和jpg图效果都比较好，故采用为最终方案。

## 3.2 岩石面积的计算

岩石面积的计算可分为两种：一是bmp图片例如图 1的面积计算，可以直接使用图片的总像素值；二是jpg图片例如图 3的面积计算，需要进行图片分割提取岩石部分。我们使用了多种方法进行图片分割，有些效果好有些不好，接下来从简到繁说明我们对jpg图片分割提取的多种探索。

### 3.2.1 二值化

首先我们想到的方法是使用颜色阈值剔除蓝色背景。使用HSV图选定蓝颜色区间进行二值化处理，分别处理了白光图和荧光图两种图片。得到的图 5和图 6效果非常不理想。白光灯下会有岩石块和背景色相近，荧光灯下上边界和下边界背景接近岩石块色，并且岩石部分内部存在白色块也无法剔除。

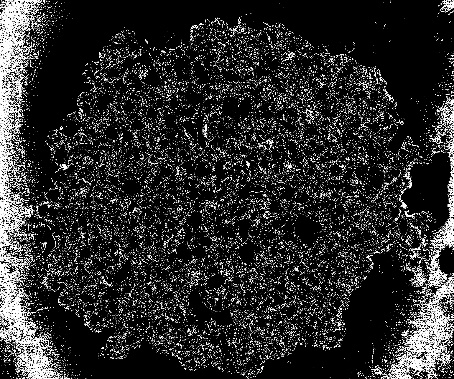


图 5：白光图二值化



图 6：荧光图二值化

### 3.2.2 自适应阈值分割

Opencv提供了adaptiveThreshold函数进行自适应阈值法分割，它的思想不是计算全局图像的阈值，而是根据图像不同区域亮度分布，计算其局部阈值，所以对于图像不同区域，能够自适应计算不同的阈值。

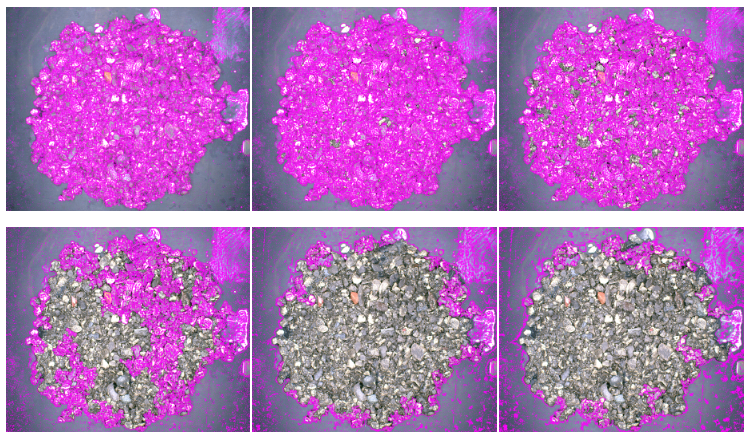


图 7：参数blocksize对分割结果的影响

在实验过程中，我们发现blocksize对分割结果影响非常大。图 7展示的是blocksize分别取11、21、31、51、101以及201时取得的分割图。分割效果没有达到理想状态，参数引起的波动太大且单个参数不能普遍使用于整个数据集。同时这种方法还把右侧和右上方的干扰痕迹引入。

### 3.2.3 无监督的图片分割

经过前两次的失败，我们决定采用神经网络进行图片分割。由于数据集很少，选用了无监督的图片分割算法对单张图进行处理。算法采用的是曾[1][2]优化的Kanezaki[5]提出的无监督图片分割。Kanezaki提出的算法主体伪代码如图 8所示。作者先使用经典机器学习算法为输入图片预分类，为语义信息相同的小区块打上相同标签，然后使用深度学习分类语义分割的结果，最终聚合相同语义的小区块得到输出结果。曾在这个基础上修改了预分类的slic[3]算法为felzenszwalb[4]算法，调整了卷积网络结构，实现了运行速度数倍提升。

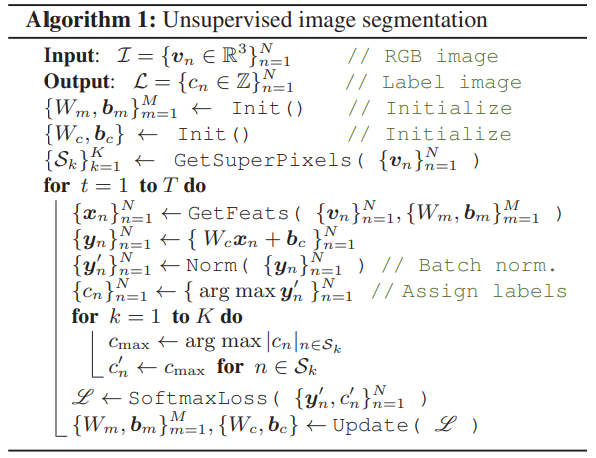


图 8：Kanezaki的算法[5]

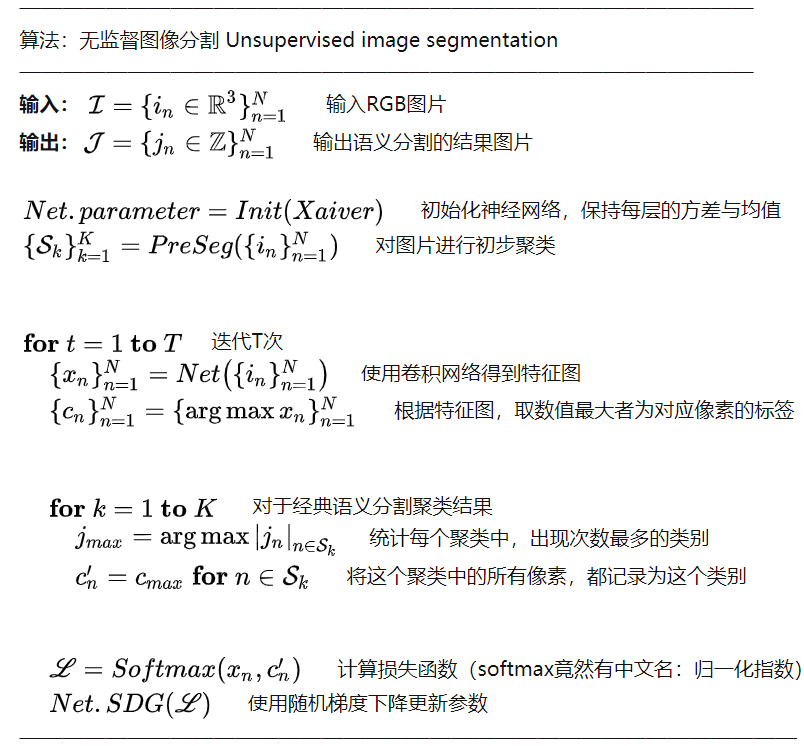


图 9：曾优化后的算法[1]

在使用算法的过程中，我们发现了由于图片较大，分割算法运行时间过长。我们首先对图片进行了多次下采样，将缩放后的图片作为输入进行分割。经过对比多个比例的缩放结果，缩放对面积求取影响不大，带来的时间收益十分可观。图 10展示了采样后的输入图，图 11则是它的分割结果。中心区域的棕色块与原图岩石面积比较接近，分割效果较好。



图 10：下采样后的图片



图 11：分割后的图片

# **结论**

本文

# **参考文献**

1. 曾的优化部分. https://zhuanlan.zhihu.com/p/68528056
2. 曾优化后的算法. <https://github.com/Yonv1943/Unsupervised-Segmentation>
3. Achanta R, Shaji A, Smith K, et al. SLIC superpixels compared to state-of-the-art superpixel methods[J]. IEEE transactions on pattern analysis and machine intelligence, 2012, 34(11): 2274-2282.
4. Felzenszwalb P F, Huttenlocher D P. Efficient graph-based image segmentation[J]. International journal of computer vision, 2004, 59(2): 167-181.
5. Kanezaki A. Unsupervised image segmentation by backpropagation[C]//2018 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, 2018: 1543-1547.

附录A 含油面积百分比求取算法

|  |
| --- |
| 算法 A1 含油面积求取算法 |
| img = cv2.imread(path2, 1) # 读取图片  hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV) # 转化为hsv图  low\_hsv = np.array([26, 43, 46]) # 设定黄绿颜色区间  high\_hsv = np.array([77, 255, 255])  mask = cv2.inRange(hsv, lowerb=low\_hsv, upperb=high\_hsv) # 二值化，白色部分为黄绿色  black = len(mask.astype(np.int8)[mask==255]) |

|  |
| --- |
| 算法 A2 岩石面积分割算法 |
| class Args(object):  train\_epoch = 2 \*\* 7  mod\_dim1 = 64  mod\_dim2 = 32  gpu\_id = 0  min\_label\_num = 2 # if the label number small than it, break loop  max\_label\_num = 6 # if the label number small than it, start to show result image.  class MyNet(nn.Module):  def \_\_init\_\_(self, inp\_dim, mod\_dim1, mod\_dim2):  super(MyNet, self).\_\_init\_\_()  self.seq = nn.Sequential(  nn.Conv2d(inp\_dim, mod\_dim1, kernel\_size=3, stride=1, padding=1),  nn.BatchNorm2d(mod\_dim1),  nn.ReLU(inplace=True),  nn.Conv2d(mod\_dim1, mod\_dim2, kernel\_size=1, stride=1, padding=0),  nn.BatchNorm2d(mod\_dim2),  nn.ReLU(inplace=True),  nn.Conv2d(mod\_dim2, mod\_dim1, kernel\_size=3, stride=1, padding=1),  nn.BatchNorm2d(mod\_dim1),  nn.ReLU(inplace=True),  nn.Conv2d(mod\_dim1, mod\_dim2, kernel\_size=1, stride=1, padding=0),  nn.BatchNorm2d(mod\_dim2),  )  def forward(self, x):  return self.seq(x)  def run(image):  start\_time0 = time.time()  args = Args()  torch.cuda.manual\_seed\_all(1943)  np.random.seed(1943)  os.environ['CUDA\_VISIBLE\_DEVICES'] = str(args.gpu\_id) # choose GPU:0  '''segmentation ML'''  seg\_map = segmentation.felzenszwalb(image, scale=32, sigma=0.5, min\_size=64)  seg\_map = seg\_map.flatten()  seg\_lab = [np.where(seg\_map == u\_label)[0]  for u\_label in np.unique(seg\_map)]  '''train init'''  device = torch.device("cuda" if torch.cuda.is\_available() else 'cpu')  tensor = image.transpose((2, 0, 1))  tensor = tensor.astype(np.float32) / 255.0  tensor = tensor[np.newaxis, :, :, :]  tensor = torch.from\_numpy(tensor).to(device)  model = MyNet(inp\_dim=3, mod\_dim1=args.mod\_dim1, mod\_dim2=args.mod\_dim2).to(device)  criterion = torch.nn.CrossEntropyLoss()  optimizer = torch.optim.SGD(model.parameters(), lr=5e-2, momentum=0.9)  image\_flatten = image.reshape((-1, 3))  color\_avg = np.random.randint(255, size=(args.max\_label\_num, 3))  show = image  '''train loop'''  start\_time1 = time.time()  model.train()  for batch\_idx in range(args.train\_epoch):  '''forward'''  optimizer.zero\_grad()  output = model(tensor)[0]  output = output.permute(1, 2, 0).view(-1, args.mod\_dim2)  target = torch.argmax(output, 1)  im\_target = target.data.cpu().numpy()  '''refine'''  for inds in seg\_lab:  u\_labels, hist = np.unique(im\_target[inds], return\_counts=True)  im\_target[inds] = u\_labels[np.argmax(hist)]  '''backward'''  target = torch.from\_numpy(im\_target)  target = target.to(device)  loss = criterion(output, target)  loss.backward()  optimizer.step()  '''show image'''  un\_label, lab\_inverse = np.unique(im\_target, return\_inverse=True, )  if un\_label.shape[0] < args.max\_label\_num: # update show  img\_flatten = image\_flatten.copy()  if len(color\_avg) != un\_label.shape[0]:  color\_avg = [np.mean(img\_flatten[im\_target == label], axis=0, dtype=np.int) for label in  un\_label]  for lab\_id, color in enumerate(color\_avg):  img\_flatten[lab\_inverse == lab\_id] = color  show = img\_flatten.reshape(image.shape)  print('Loss:', batch\_idx, loss.item())  if len(un\_label) < args.min\_label\_num:  break  time0 = time.time() - start\_time0  time1 = time.time() - start\_time1  print('PyTorchInit: %.2f\nTimeUsed: %.2f' % (time0, time1))  return show |