دانشکده مهندسی کامپیوتر

گزارش پروژه پایانی درس مبانی بینایی کامپیوتر

استاد درس:
دکتر محمدرضا محمدی

اعضای گروه:
هدیه اسحقی دیزنجه
مهدیه نادری

بهمن ۱۴۰۱

بسمه تعالی

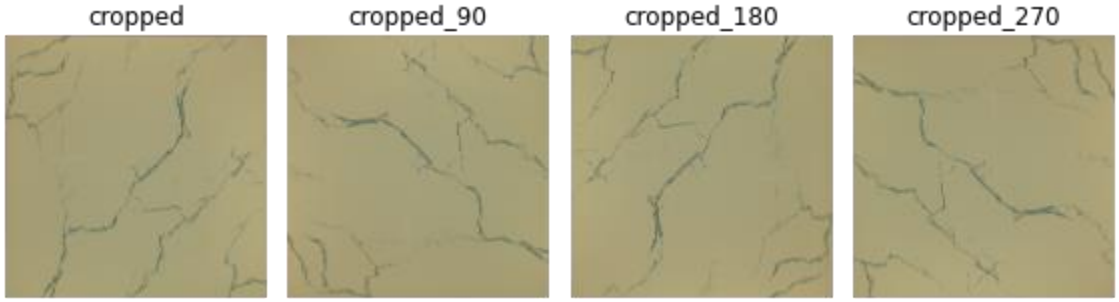
چکیده:

ما برای انجام این پروژه ابتدا یک سری پیش پردازش روی تصویر اصلی انجام دادیم و سپس برای پیدا کردن ترک ها روی تصاویر از fasterRCNN استفاده کردیم. در واقع پروژه را در دو بخش پیش پردازش و ماشین لرنینگ انجام دادیم.

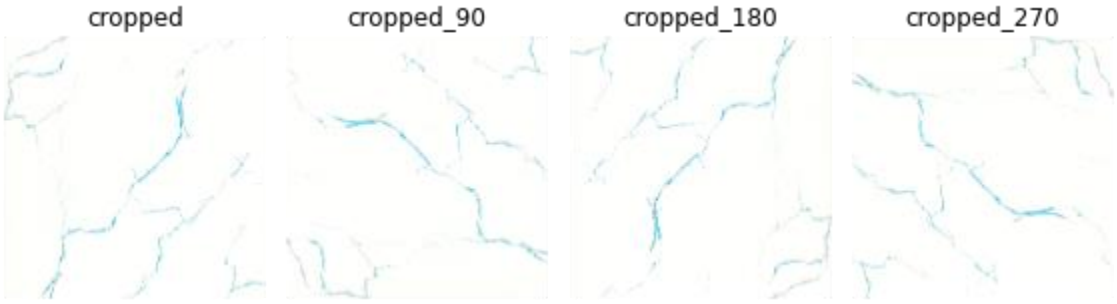
پیش پردازشها:

ابتدا یک تابع نوشتیم که بخش کاشی را از ناحیه تصویر جدا میکند. سپس تصویر برش خورده را ۴ بار به میزان ۹۰ درجه چرخاندیم.
پس از آن برای هر یک از ۴ حالت histogram matching را انجام دادیم. تا بعد از آن بتوانیم تصاویر را با یک ترشلد ثابت به باینری تبدیل کنیم.
سپس برای آنکه پترن را از تصویر کم کنیم، روی پترن dilation انجام دادیم تا قسمتهای طرح آن کمی بزرگ تر شود. سپس یک تابع نوشتیم که تصویر و پترن dilate شده را دریافت میکند و هر جایی از پترن که سیاه است را(طرح کاشی سیاه و بک گراند سفید میباشد) معادلش را در کاشی به سفید تبدیل میکند. اینطوری قسمت پترن نیز در کاشی اصلی سفید میشود و جزو بک گراند به حساب می آید و هرچه باقی می ماند می توانیم امیدوار باشیم که ترک است.

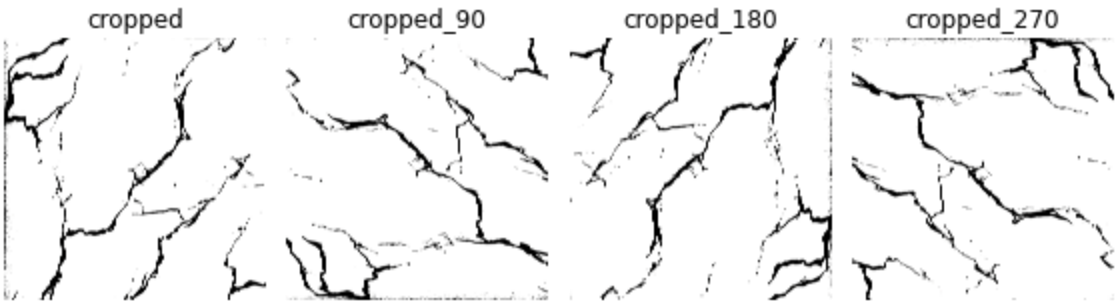
خروجی بریده شده تصویر که ابعاد آن با ابعاد پترن آن کاشی برابر است:



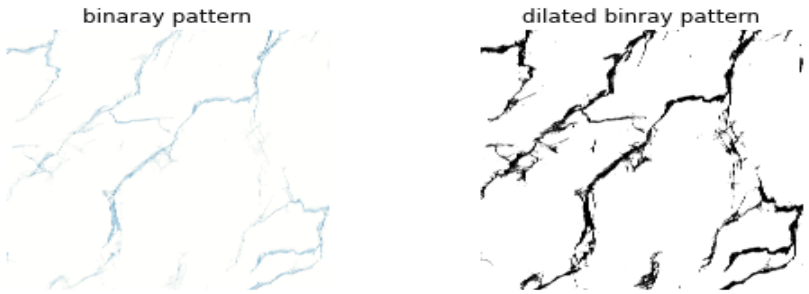
خروجی عملیات histogram matching:



خروجی پس از باینری کردن تصویر:



خروجی پس از انجام عملیات dilation روی پترن:



مچ ترین کاشی از بین کاشی های ۹۰ درجه چرخانده شده که تابع برای ما پیدا کرد و میبینیم که دقیقا با کاشی مچ است:
(در ارائه راجع به score این کاشی مچ شده پرسیده شده بود که الان پس پرینت متوجه شدیم که score آن 1282031 میباشد)

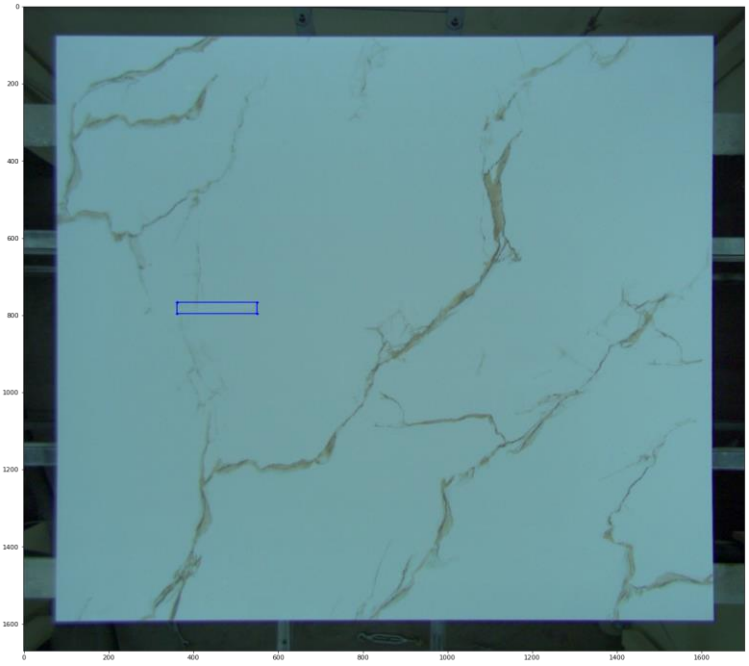
```
all_input_to_check = [cv.bitwise_not(binary_1), cv.bitwise_not(binary_2), cv.bitwise_not(binary_3), cv.bitwise_not(binary_4)]
matchest = find_matchest(all_input_to_check, cv.bitwise_not(dilated_pattern))
matchest = cv.bitwise_not(matchest)

imshow(matchest)
```

```
1282031
```

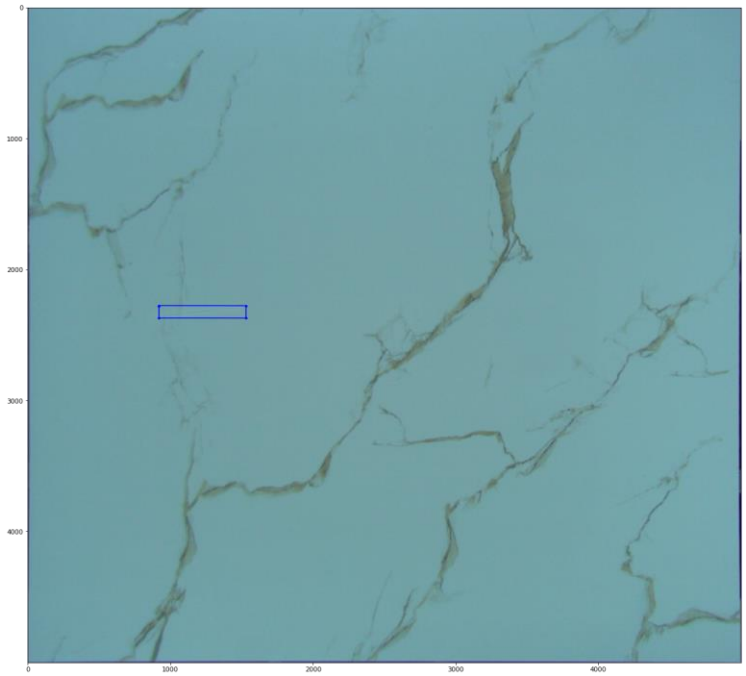
خروجی نهایی تصویر کاشی که پترن از آن کم شده است:

برای نقاط ترک نیز با استفاده از همان ماتریس تبدیلی که کاشی را تشخیص داده بودیم توانستیم نقاط ترک را نیز اسکیل و جابجا کنیم تا مختصات آنها را روی سرامیک بریده شده پیدا کنیم:

شکل زیر نقاط ترک قبل از برش کاشی را نشان میدهد:



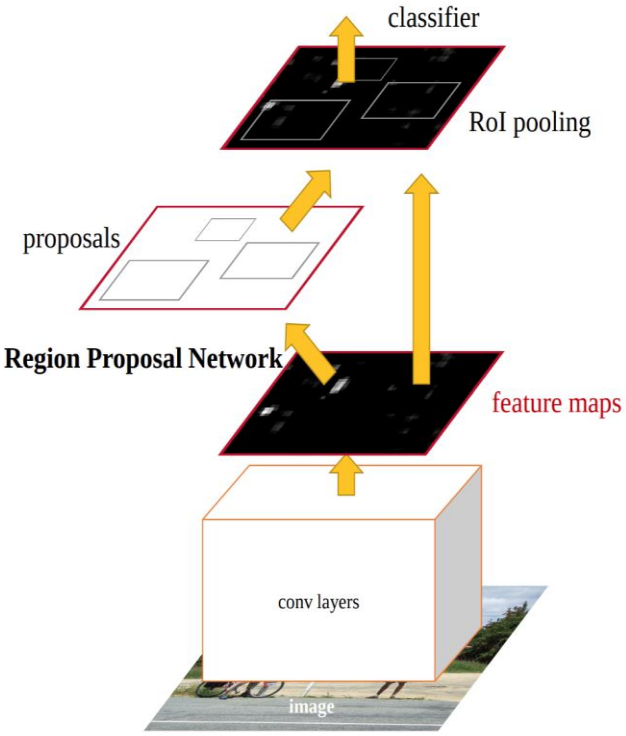
شکل زیر نیز نقاط ترک را پس بریده شدن کاشی نشان میدهد:



همانطور که مشاهده می شود این نقاط به درستی به دست آمده است.

مدل:

ما تصمیم گرفتیم این تسک را با کمک روش های Object Detection پیاده سازی کنیم. با کمی جست و جو متوجه شدیم Fast R-CNN یکی از بهترین شبکه هایی است که میتواند ما را برای دست یابی به این هدف یاری کند. این مدل از دو ماژول تشکیل شده است. ماژول اول یک شبکه عصبی عمیق کانولوشنی که نواحی پیشنهادی را مشخص (RPN: Region Proposal Network) می کند و ماژول دوم همان Fast R-CNN است که از نواحی پیشنهادی بدست آمده استفاده می کند.
بعد از انتخاب این شبکه به پیاده سازی بخش های مختلف آن پرداختیم. برای بخش شبکه ی کانولوشنی معمولا از دو شبکه ی resnet یا vggnet استفاده میشود.



در این شبکه ما از مدل 16vgg برای استخراج ویژگی های تصویر استفاده کردیم. سپس پروپوزال های مناسب تولید کردیم و در نهایت با کمک ROI Pooling عملیات classifier و Box Regression را انجام دادیم. ما در پیاده سازی های خود از کتابخانه ی torchو torchvision استفاده کردیم.

الف) استخراج ویژگی:

با توجه به ابعاد ورودی لایه ی اول 16vgg ابتدا تصویر را به سایز ۸۰۰ در ۸۰۰ رساندیم. سپس مقیاس موثر روی تصویر را در باکس ها هم اعمال نمودیم.
سپس مدل اولیه را ساختیم و فیچرهای آن را ذخیره کردیم و با استفاده از این ویژگی ها یک مدل sequential ساختیم.

```
model = torchvision.models.vgg16(pretrained=True).to(device)
fc = list(model.features)
```

```
[109]  1 d_img = torch.zeros((1, 3, 800, 800)).float()
       2 req_features = []
       3 k = d_img.clone().to(device)
       4 out_channels=0
       5 for i in fc:
       6   k = i(k)
       7   if k.size()[2] < 800//16:
       8     break
       9   req_features.append(i)
      10   out_channels = k.size()[1]
      11
      12
```

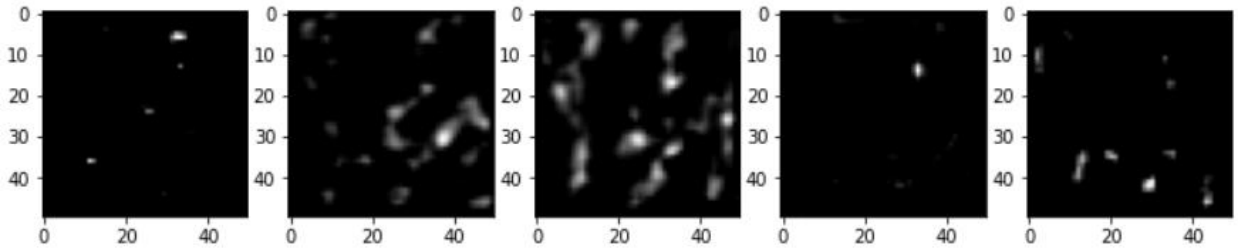
Convert this list to sequential models

```
[110]  1 faster_rcnn_fc_extractor = nn.Sequential(*req_features)
```

```
 1 transform = transforms.Compose([transforms.ToTensor()])
 2 imgTensor = transform(img_r).to(device)
 3 imgTensor = imgTensor.unsqueeze(0)
 4 out_map = faster_rcnn_fc_extractor(imgTensor)
 5 print(out_map.size())
```

```
torch.Size([1, 512, 50, 50])
```

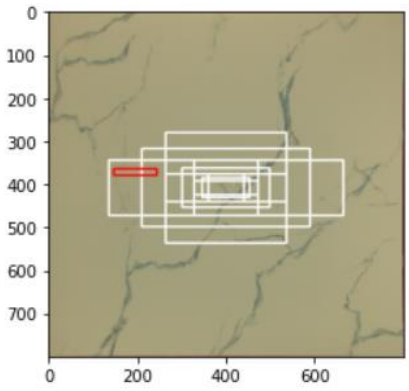
۵ نمونه از این فیچر ها در تصویر زیر قابل مشاهده هستند.



ب)تولید anchor boxها:

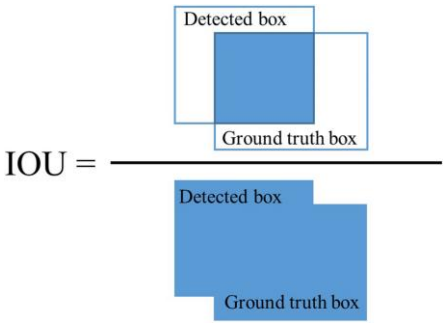
ابتدا ۲۵۰۰ نقطه از تصویر انتخاب شدند. سپس حول این نقاط با مقیاس های مختلف و شکل های مختلف مربعی، مستطیلی عمودی و مستطیلی افقی ۲۲۵۰۰ باکس تولید شدند.

نمونه ای از باکس های تولید شده حول یک نقطه و ترک روی کاشی در تصویر قابل مشاهده هستند.



برای آموزش شبکه RPN به ازاء هر Anchor box در هر خانه از تصاویر آموزشی عدد صفر یا یک اختصاص می دهیم. عدد ۱ را در حالت به یک anchor box اختصاص می دهیم:۱) در صورتی که anchor box مورد نظر بیشترین IoU را با محدوده واقعی آن شی داشته باشد. ۲) درصورتی که anchor box مورد نظر دارای IoU بالاتر از ۰,۷ با محدوده واقعی شی داشته باشد.

برای محاسبه IoU نیز از رابطه زیر استفاده می شود:



برای انتخاب باکس های مناسب تابع محاسبه گر IOU آن را نوشتیم. بعد از بدست آوردن معیار ها ماکسیمم iou را برای هر anchor پیدا کردیم. همچنین ماکسیمم iou برای هر ground truth box را پیدا کردیم و بعد هم با مقایسه ی ترشلد ها برچسب ها را درنظر گرفتیم.

ج)RPN:

در این مرحله ما ۳ لایه ی کانولوشنی قرار دادیم تا بر اساس تصویر و پروپوزال ها و باکس هایی که داریم بهترین پروپوزال ها را نگه دارد و به عبارتی تشخیص دهد که کدام ها مناسب ترین هستند و به ازای هر باکس یک انکر داشته باشیم.

```
in_channels = 512
mid_channels = 512
n_anchor = 9

conv1 = nn.Conv2d(in_channels, mid_channels, 3, 1, 1).to(device)
conv1.weight.data.normal_(0, 0.01)
conv1.bias.data.zero_()

reg_layer = nn.Conv2d(mid_channels, n_anchor*4, 1,1,0).to(device)
reg_layer.weight.data.normal_(0, 0.01)
reg_layer.bias.data.zero_()

cls_layer = nn.Conv2d(mid_channels, n_anchor*2, 1,1,0).to(device) #softmax
cls_layer.weight.data.normal_(0, 0.01)
cls_layer.bias.data.zero_()
```

```
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
x = conv1(out_map.to(device))
pred_anchor_locs = reg_layer(x)
pred_cls_scores = cls_layer(x)
```

د) classification و Box Regression:

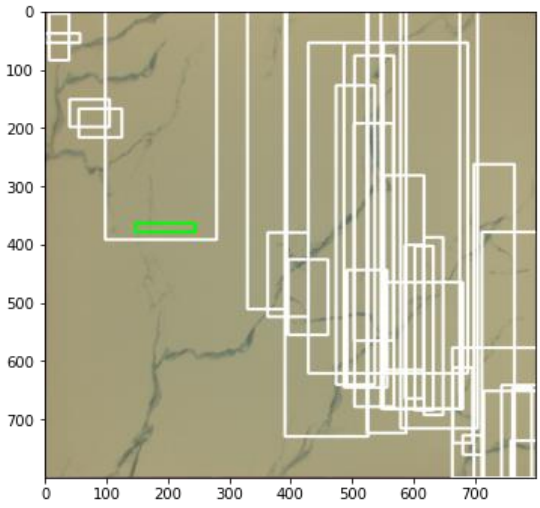
در این مرحله ما با استفاده از لایه های کاملا متصل یک لایه با تابع Loss کراس انتروپی طراحی کردیم. همچنین برای تفکیک کردن باکس ها پسک هایی را بر اساس ترشلد های مشخصی طراحی کردیم و مقدار تابع ضرر را برای هر rpn طبق فرمول محاسبه کردیم.

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*).$$

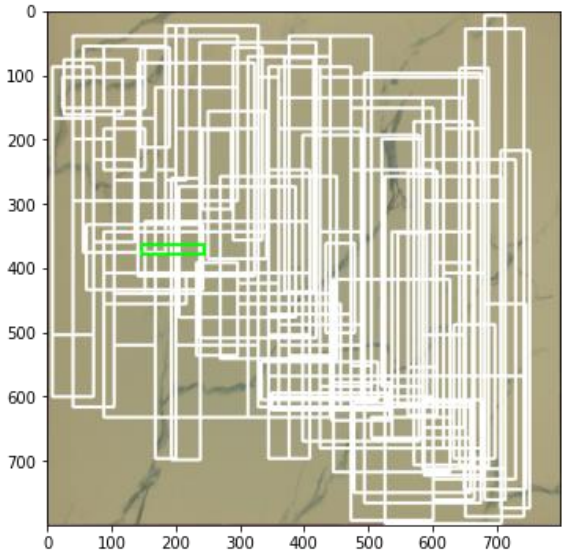
```
rpn_lambda = 10
N_reg = (gt_rpn_score> 0).float().sum()
rpn_loc_loss = rpn_loc_loss.sum() / N_reg
rpn_loss = rpn_cls_loss + (rpn_lambda * rpn_loc_loss)
print(rpn_loss)
```

در نهایت به پیدا کردن NMS(Non-maximum supression) بین anchor ها پرداختیم. این کار به تنهایی چند مرحله دارد چون باید roi ها حساب شوند و بعد از یک لایه ی ROI Pooling استفاده شود.

نمونه ای از roi ها با لیبل ۱ روی یکی از مینی بچ های ۱۲۸ تایی:



نمونه ای از roi ها با لیبل ۰ یا همان بک گراند روی یکی از مینی بچ های ۱۲۸ تایی:



قطعه کد خروجی با RoiMAXPooling:

```
output = []
rois = indices_and_rois.data.float()
rois[:, 1:].mul_(1/16.0)
rois = rois.long()
num_rois = rois.size(0)
for i in range(num_rois):
  roi = rois[i]
  im_idx = roi[0]
  im = out_map.narrow(0, im_idx, 1)[..., roi[2]:(roi[4]+1), roi[1]:(roi[3]+1)]
  tmp = adaptive_max_pool(im)
  output.append(tmp[0])
output = torch.cat(output, 0)
print(output.size)
```

این شبکه فقط روی یک تصویر اجرا شد. اگر با یک حلقه ی for مدل را روی تمام تصاویر اجرا کنیم، وزن ها و باکس های بهتری استخراج میشوند.

منابع:

https://www.youtube.com/watch?v=4yOcsWg-7g8

https://virgool.io/moneytoo/%DA%A9%D8%B4%D9%81-%D8%A7%D8%B4%DB%8C%D8%A7%D8%A1-%D8%AF%D8%B1-%D8%AA%D8%B5%D9%88%DB%8C%D8%B1-r-cnn-%D9%88-fast-r-cnn-%D9%88-faster-r-cnn-%D9%88-yolo-qv3czowuty7m