تشخیص ترک های موجود در کاشی

۱۴۰۱/۱۱/۰۴

اعضای گروه:

امیرحسین سماوات (98521252)

بابک بهکام کیا (98521099)

استاد :

دکتر محمدی

چکیده

چکیده پروژه این است که تصاویری از کاشی به ما داده می شود وما باید ترک ها را از روی طرح های موجود هر نوع کاشی پیدا کنیم و آن ها را شناسایی کنیم. در این پروژه، ما اول سعی کردیم که برای هر جفت عکس کاشی و pattern، در حد امکان این دو عکس را از لحاظ رنگی و ابعاد مانند هم بکنیم. در ادامه از مدل U-net از قبل آموزش دیده، استفاده کردیم زیرا تعداد داده محدودی برای آموزش در دسترس داریم.

خواندن دیتاست

در اولین مرحله پروژه، یک کپی از دیتاست را در درایو خود ذخیره می کنیم. در مرحله بعد عکس هایی که فرمت "jpg" یا "png" دارند را در لیستی به اسم images و لیبل های متناظر آنها را در لیستی به نام labels ذخیره می کنیم. در ادامه، مراحل مشابهی را برای load کردن pattern های کاشی ها انجام می دهیم.

```
pth= f"{path_babak}Dataset/dataset"

images = []
labels = []
skip = False

for file in os.listdir(pth):
  if skip:
    skip = False
    continue
  if file.endswith(".jpg"):
    images.append(cv2.imread(os.path.join(pth,file))[74:1590,74:1590])
  elif file.endswith(".png"):
    images.append(cv2.imread(os.path.join(pth,file))[74:1590,74:1590])
  elif file.endswith(".json"):
    labels.append(json.load(open(os.path.join(pth,file))))
  elif file.endswith(".bmp"):
    skip = True
    continue
```

در این مرحله برای از بین بردن پس زمینه موجود در کاشی که ارتباطی با آن ندارد، 74 پیکسل از هر طرف عکس ها را در نظر نمیگیریم.

تطبیق هیستوگرام

در این مرحله برای اینکه pattern های موجود در حد امکان شبیه عکس از کاشی های باشند، از تطبیق هیستوگرام استفاده می کنیم.

```python
new_patterns = [None] * len(pattern_path)
pattern_visited = [False] * len(pattern_path)
for i in range(len(images)):
  reference = images[i]

  pattern_name = labels[i]['pattern']
  pattern_img = None
  for j in range(len(pattern_path)):
    if (not pattern_visited[j]) and pattern_path[j].split('/')[-1] == pattern_name:
      pattern_img = patterns[j]
      pattern_visited[j] = True
      source = pattern_img
      matched = match_histograms(source, reference, multichannel=True)
      new_patterns[j] = matched
      break
```

پیدا کردن نقاط کلیدی

با توجه به داک پروژه، عکس کاشی ها ممکن است نسبت به pattern ها چرخیده باشند اما می دانیم که درجه این چرخش مضربی از 90 است.

برای این منظور، تابع find_keypoints را می نویسیم. در این تابع نقاط کلیدی عکس کاشی و pattern استخراج می شوند و در مرحله بعد بهترین نقاط را به تابع find_rotation_degree می دهیم تا مقدار چرخیدن عکس را بدست بیاوریم.

در این تابع با توجه به مختصات هر جفت نقطه کلیدی به دست آمده که با همدیگر match شدند، مقدار چرخش را حساب می کنیم. در نهایت آن درجه ای که بیشترین رای را بین نقاط دارد به عنوان مقدار چرخش انتخاب می شود. (برای مثال، اگر نقطه ای در عکس کاشی در قسمت بالا و چپ تصویر باشد و نقطه متناظرش در pattern، در قسمت پایین و راست تصویر باشد به این نتیجه می رسیم که عکس 180 درجه چرخیده است.)

https://pysource.com/2018/07/20/find-similarities-between-two-images-with-opencv-and-python/

https://docs.opencv.org/3.4/d2/d29/classcv_1_1KeyPoint.html

https://docs.opencv.org/3.4/d4/de0/classcv_1_1DMatch.html

هم اندازه کردن عکس کاشی و طرح کاشی

در این قسمت چون اول از همه میدانیم عکس کاشی دورش بدرد نمیخورد پس ابتدا قسمت های اضافه را crop میکنیم سپس هم اندازه کاشی و هم طرح کاشی را هم اندازه می‌کنیم زیرا در ادامه قرار است طرح کاشی و عکس کاشی را از هم کم کنیم . نکته ای که باید ذکر شود این است که در اینجا ما طرح های کاشی داریم که ازشون هیچ عکسی نداریم پس باید آن طرح ها را شناسایی کنیم و کنار بگذاریم.

```
scale = 1516/5000
patterns = []

for i in range(len(new_patterns)):
  if type(new_patterns[i]) != type(None):
    patterns.append(cv2.resize(new_patterns[i], dsize=(0,0), fx=scale, fy=scale))
  else:
    patterns.append(None)
```

دوران عکس

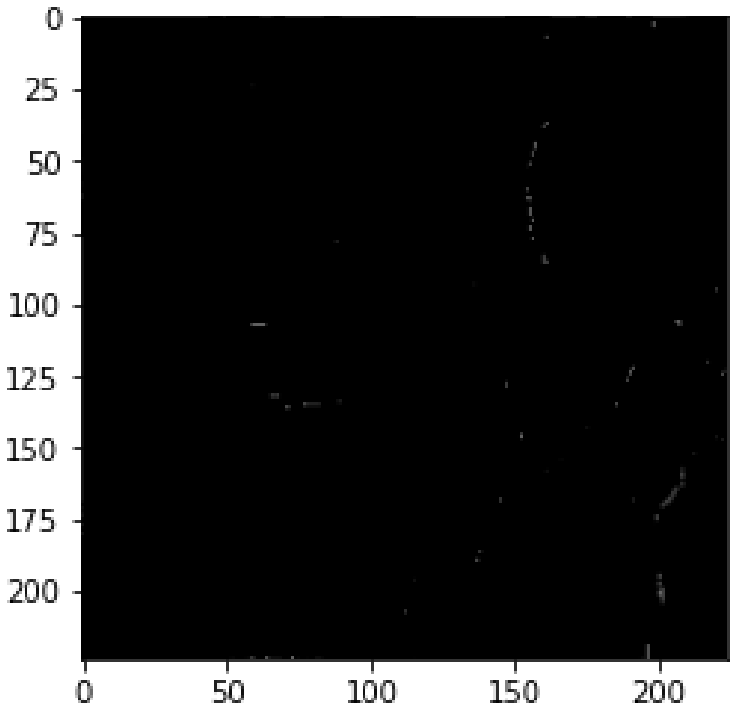
همانطور که پیشتر گفته شد عکس کاشی ممکن است دورانی داشته باشند یعنی ممکن است ۹۰ درجه یا ۱۸۰ درجه یا ۲۷۰ درجه نسبت به طرح کاشی دوران پیدا کرده باشد ابتدا ما کدی نوشتیم که خود دوران را برای ما انجام دهد سپس هر عکس را چه مقدار باید دوران بدهیم را نیز قبلا بدست آورده بودیم در بخش پیدا کردن نقاط کلیدی استفاده میکنیم و عکس ها را دوران می دهیم.

```
def rotate_image(img, deg):
  if deg == '90':
    return cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)

  elif deg == '270':
    return cv2.rotate(img, cv2.ROTATE_90_COUNTERCLOCKWISE)

  elif deg == '180':
    return cv2.rotate(img, cv2.ROTATE_180)

  else:
    return img

rotated_patterns = []
for i in range(len(rotation_deg)):
  pattern_name = labels[i]['pattern']
  for j in range(len(pattern_path)):
    if pattern_path[j].split('/')[-1] == pattern_name:
      rotated_patterns.append(rotate_image(patterns[j], rotation_deg[i]))
      break
```

حذف pattern از کاشی

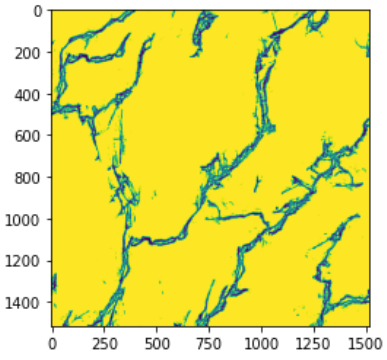
در این بخش قرار است که طرح کاشی را از خود عکس کاشی حذف کنیم . باید در اینجا از قواعد مورفولوژی و همین طور آستانه گذاری حدی استفاده کنیم. ابتدا با آستانه گذاری حدی هر پیکسل را به نسبت مربع پیکسل های مجاور مقایسه میکرد که اگر بزرگتر از عدد آستانه بود آن پیکسل را یک میکرد و اگر نه صفر میکرد. سپس با ساختن دو کرنل ۷در ۷ و همین طور ۳در۳ به صورت دایره ای آمدیم خود طرح کاشی را ابتدا سایش زدیم سپس یکم گسترش و در آخر باز زدیم که طرح کاشی به اندازه خوبی از عکس کاشی برسد که کم می‌کنیم بتوانیم ترک های کاشی را به خوبی بدست بیاوریم.

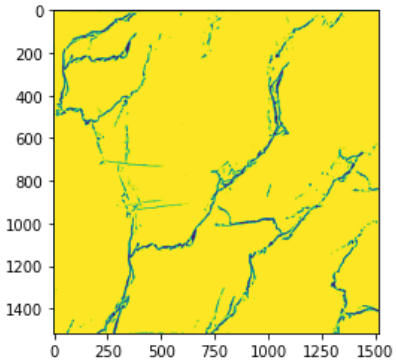
در آخر نیز عکس را به سایز ۲۲۴ می‌رسانیم .



دلیل استفاده از مورفلوژی روی pattern این بود که اندازه خطوط در pattern ها کلفت تر از در عکس های کاشی است.

(سمت راست عکس کاشی و سمت چپ pattern است)





```
input_images = []
for i in range(len(images)):
  img = images[i]
  img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY )
  bin1 = cv2.adaptiveThreshold(img,100,cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY,21,5)

  kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(7,7))
  kernel2 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))


  pat = cv2.cvtColor(rotated_patterns[i], cv2.COLOR_BGR2GRAY )
  bin2 = cv2.adaptiveThreshold(pat,100,cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY,21,5)

  eroded = cv2.erode(bin2,kernel,iterations=5)
  eroded2 = cv2.erode(eroded,kernel2,iterations=3)
  dilate = cv2.dilate(eroded2,kernel)
  opening = cv2.morphologyEx(dilate, cv2.MORPH_OPEN, kernel2,iterations=2)

  temp = cv2.subtract(opening, bin1)
  temp = cv2.resize(temp, dsize=(0,0), fx=224/1516, fy=224/1516)

  input_images.append(cv2.cvtColor(temp, cv2.COLOR_GRAY2BGR))
```

https://www.geeksforgeeks.org/how-to-subtract-two-images-using-python-opencv/

بدون مدل

در این بخش می‌توانیم بدون استفاده از شبکه خودمان با قوانین مورفولوژی بیایم انقدر با سایش و باز روی طرح کاشی عملیات انجام بدهیم که زمانی که از خود کاشی طرح را کم میکنیم فقط برای ما قسمت ترک را به جا بگذارد.

```
kernel=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
kernel2=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(7,7))

open = cv2.erode(th1,kernel2,iterations=6)
eroded = cv2.erode(open,kernel,iterations=7)
opening = cv2.morphologyEx(eroded, cv2.MORPH_OPEN,kernel2, iterations=2)
```

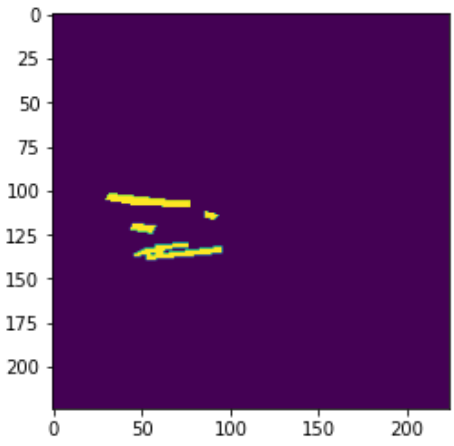
```
subtracted = cv2.subtract(opening, th2)
dilated = cv2.dilate(subtracted,kernel2,iterations=2)
open = cv2.morphologyEx(dilated, cv2.MORPH_OPEN,kernel, iterations=3)
```

نتیجه

آماده کردن لیبل مناسب برای مدل

برای اینکه میخواهیم از U-net به عنوان مدل خود استفاده کنیم، پس نیاز داریم که یک عکسی که پیکسل های مربوط به ترک ها را سفید کرده است را به عنوان لیبل برای مدل آماده کنیم.

```
masked_labels = []

for i in range(len(labels)):
  mask = np.zeros((224, 224))
  for j in range(len(labels[i]['shapes'])):
    points = []
    for point in labels[i]['shapes'][j]['points']:
      points.append([int((point[0] - 74) * (224/1516)), int((point[1] - 74) * (224/1516))])
    cv2.fillPoly(mask, pts=[np.array(points)], color=(255, 255, 255))

  masked_labels.append(mask)
```



https://www.geeksforgeeks.org/draw-a-filled-polygon-using-the-opencv-function-fillpoly/

مدل

در این بخش از یک مدل U-net که از قبل آموزش دیده (زیرا دیتای کافی برای آموزش دادنش از اول را نداریم) را load می کنیم. در اولین تجربه، ابعاد ورودی شبکه را تغییر نمی دهیم (همان (224, 224) می ماند) و دیتای خود را برای اینکه به مدل بدهیم resize می کنیم.

این مدل 5 بار downsample می شود و ابعادش از 224 به 7 می رسد و در نهایت بعد از upsampling ها به ابعاد (224 , 224, 1) می رسد.

حال hyper-parameter های مدل را مشخص می کنیم.

```
out_model_path = 'model.h5'
epochs = 100
patience = 10
batch_size = 8
learning_rate = 0.001


optim = Adam(learning_rate=learning_rate)
model.compile(optimizer=optim, loss=dice_coef_loss, metrics=[dice_coef])

callbacks = [
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-9, min_delta=0.00001, verbose=1, mode='min'),
    EarlyStopping(monitor='val_loss', patience=patience, verbose=0),
    ModelCheckpoint('zf_unet_224_temp.h5', monitor='val_loss', save_best_only=True, verbose=0),
]
```

از callback های مقابل استفاده می کنیم تا مدل بتواند در صورت نیاز learning rate خود را کاهش دهد، همچنین اگر بعد از مدتی پیشرفت نکند، فرایند آموزش متوقف شود. یک callback هم برای ذخیره checkpoint ها استفاده می کنیم.

در این مدل از تابع dice_coef_loss به عنوان loss function استفاده شده است. در این تابع در واقع خروجی مدل و خروجی اصلی را and می کند. (یعنی اگر قسمت های سفید روی هم بیفتند مقدار متغیر intersection را بیشتر می کند.) در نهایت منفی intersection را برمیگرداند تا بتوانیم مقدار این متغیر را حداکثر کنیم.

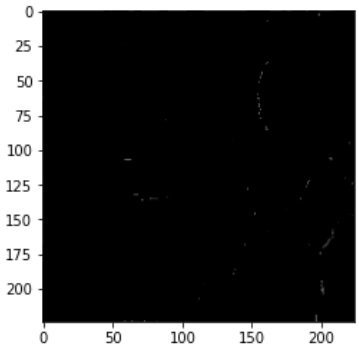
```
def dice_coef(y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2.0 * intersection + 1.0) / (K.sum(y_true_f) + K.sum(y_pred_f) + 1.0)


def dice_coef_loss(y_true, y_pred):
    return -dice_coef(y_true, y_pred)
```

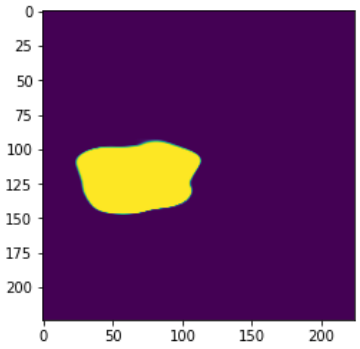
در نهایت دیتاست train را درست می کنیم و به مدل می دهیم تا fine tuning را آغاز کند.

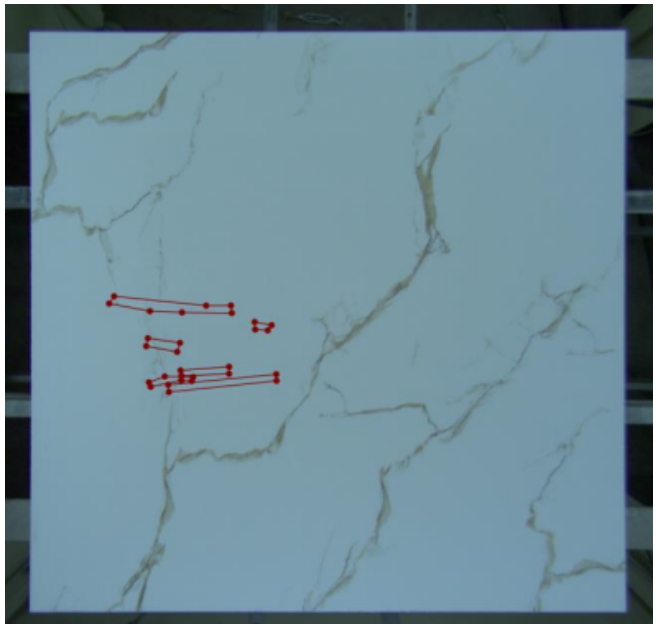
```
input_images_train = np.concatenate([input_images[:200]], axis=0 )
masked_labels_train = np.concatenate([masked_labels[:200]], axis=0 )
masked_labels_train = np.expand_dims(masked_labels_train, axis=-1)
```

https://github.com/ZFTurbo/ZF_UNET_224_Pretrained_Model

نتایج







کار های آینده

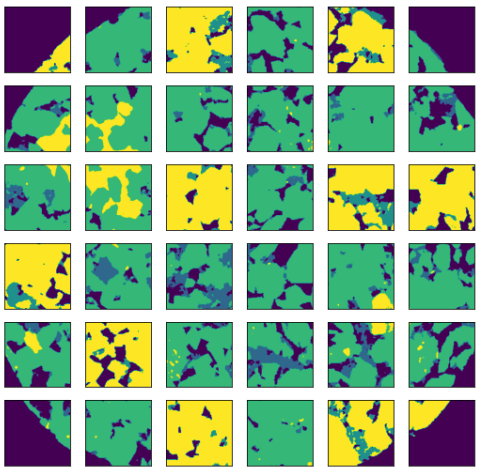
I. تغییر ابعاد ورودی مدل

ما در ادامه ابعاد مدل را به 1536 تغییر دادیم تا نیاز نباشد عکس ها را به resize 224 کنیم. اما به دلیل حجم بالای عکس ها قابلیت پیاده سازی آنها در کولب را نداشتیم. (همچنین به دلیل محدودیت وقت پروژه، نتوانستیم دیتا را به صورت batch در داخل کولب load کنیم.)

II. تقسیم عکس ورودی به قسمت های مساوی

در واقع در این روش بجای تغییر دادن ابعاد ورودی مدل، خود عکس ورودی را به قسمت های (224, 224) تقسیم می کنیم و به مدل به عنوان ورودی می دهیم. همچنین خروجی مدل را نیز باید در آخر به همدیگر وصل کنیم.

مثال برای این ایده:



https://github.com/bnsreenu/python_for_image_processing_APEER/blob/master/tutorial120_applying_trained_unet_model_to_large_images.ipynb

منابع

https://docs.opencv.org/4.x/d7/dff/tutorial_feature_homography.html

https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga4b3841447530523e5272ec05c5d1e411

https://stackoverflow.com/questions/50945385/python-opencv-findhomography-inputs

https://thinkinfi.com/image-alignment-and-registration-with-opencv/

https://www.tutorialspoint.com/opencv-python-implementing-feature-matching-between-two-images-using-sift

- در این پروژه با آقای کمبرانی، خانم سبزواری، خانم نزاکتی و خانم دیوانی آذر مشورت کردیم.