به نام خدا

گزارش پروژه درس مبانی بینایی کامپیوتر
نام مدرس: دکتر محمدرضا محمدی

اعضای گروه:
فرزان رحمانی ۹۹۵۲۱۲۷۱
گلبرگ سپهرآرا ۹۹۵۲۱۳۳۴

موضوع پروژه:
تشخیص ترک های موجود در کاشی

خلاصه ای از پروژه

چکیده این پروژه این است که تصاویری از کاشی با طرح ای متناظر آن به ما داده می شود. ما نیز باید ترک های موجود در آن را تشخیص دهیم و بگوییم کدام pixel ها ترک هستند و کدام یک نیستند. در این پروژه ما ایده های مختلفی را تست کردیم و نتایج مختلفی بدست آوردیم.

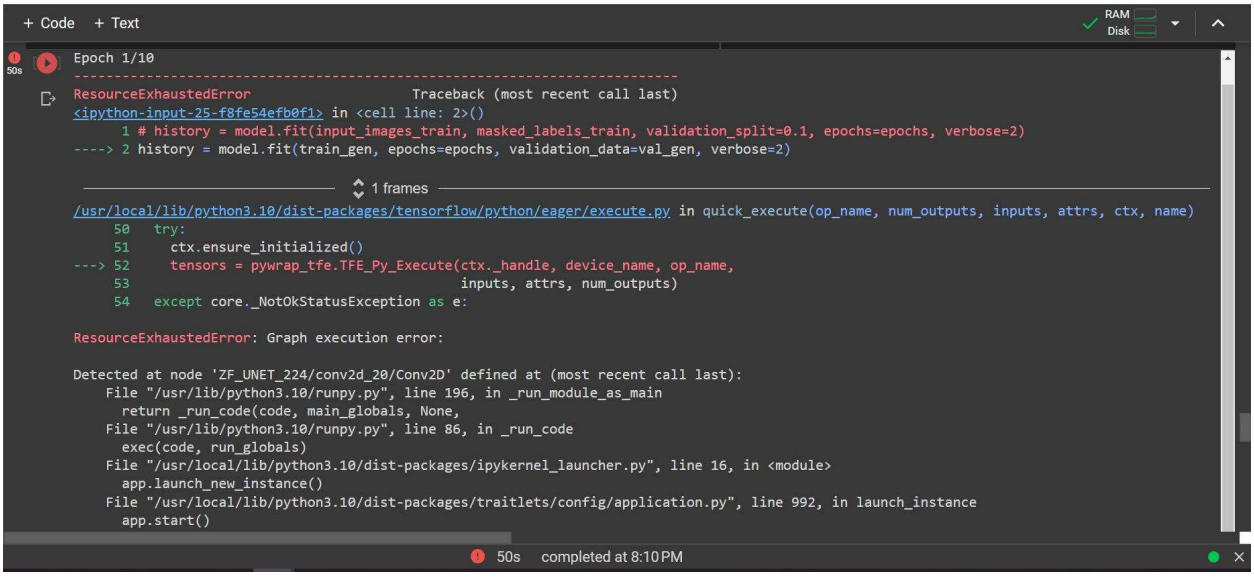
ایده ایی که به ذهن ما رسید

- Data preprocessing:
تطبیق هیستوگرام برای مشابه کردن رنگ های کاشی و الگو از لحاظ رنگی و شدت نور و کیفیت.
حذف کردن چرخش به این علت که بعضی از داده ها(عکس کاشی ها) ۰، ۹۰، ۱۸۰ یا ۲۷۰ درجه نسبت به الگو چرخیده بودند.
کم کردن الگو از عکس تا اینکه فقط ترک ها باقی بمانند.
بریدن اطراف تصاویر کاشی ها
- Data augmentation:
به علت کمبود داده های آموزشی از داده افزایی استفاده کنیم.
- استفاده از مدل های مختلف:
مدل هایی با skip connection
مدل هایی با residual connection
- Transfer learning:
به علت کمبود داده آموزشی می توانیم از مدل از پیش آموزش دیده و fine-tuning رو آن استفاده کنیم.
- استفاده از معماری Siamese:
با توجه به اینکه هم تصویر و هم الگوی مربوطه را داریم می توانیم هر دو را به شبکه بدهیم تا مدل خودش یاد بگیرد که طرح ها را تشخیص دهید و آن ها را به عنوان ترک شناسایی نکند و در عین حال ترک ها را به خوبی تشخیص دهد.

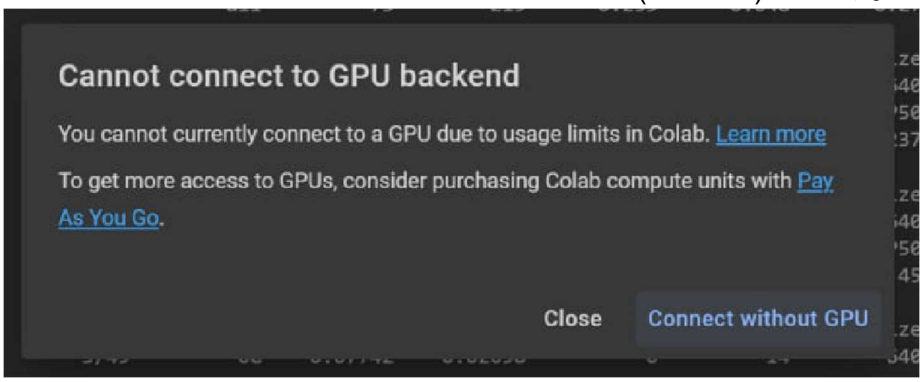
چالش ها

- ترک ها با الگو ها قاطی میشدند.

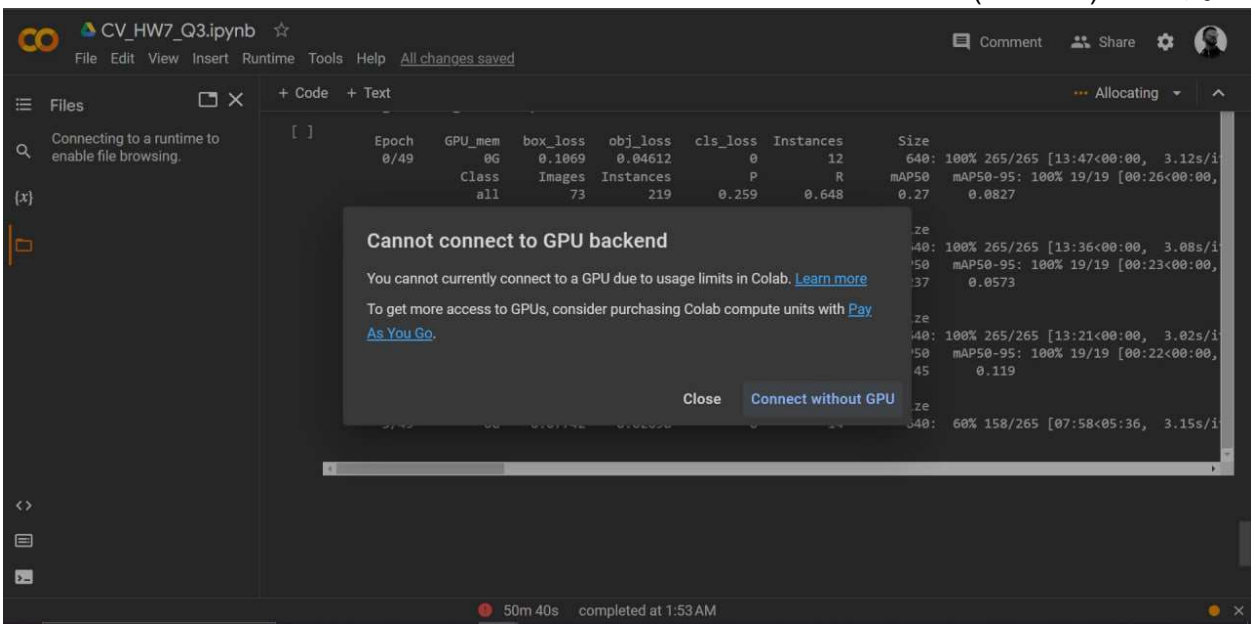
- محدودیت RAM برای شبکه های بزرگ



- محدودیت GPU (شتاب دهنده)



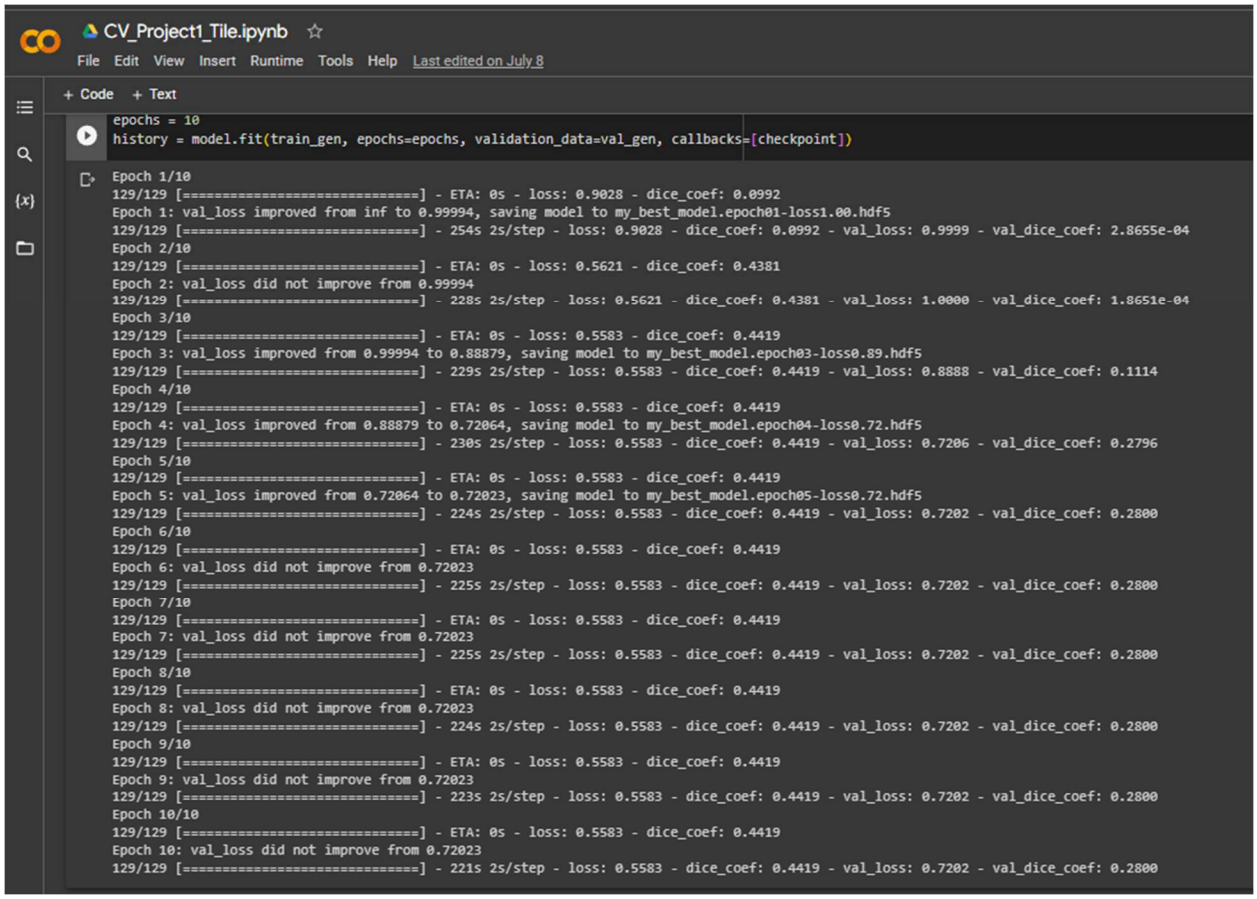
- محدودیت TPU (شتاب دهنده)



- محدودیت زمانی با توجه به مشکلات بالا
- محدودیت داده آموزشی
- دقیق نبودن داده های آموزشی
- محدودیت زمانی برای آموزش مدل با تعداد epoch بالا
- به دلیل مشکلات اینترنت و کمبود حافظه و شتاب دهنده در کولب بخشی از زمان صرف اجرا گرفتن های چندین باره میشد که اگر این مشکلات به وجود نمی آمد میتوانستیم ایده های بیشتری را تست کنیم و مدل کاملتری ارائه کنیم.

اجرا و فهمیدن کد پیشنهادی داده شده

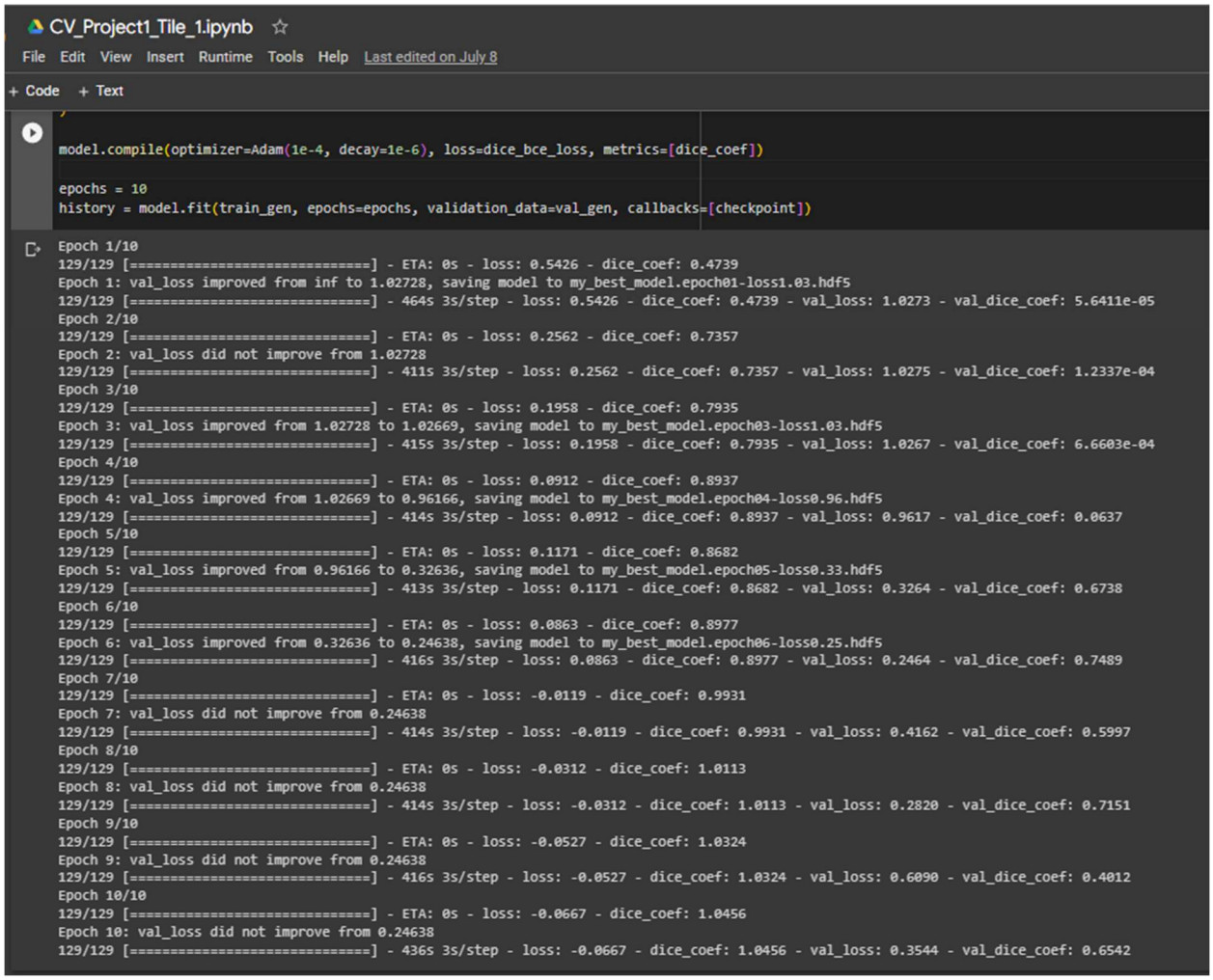
در ابتدا کد داده شده در سوال را به طور کامل خواندیم تا موضوع پروژه را بهتر درک کنیم.نخوه خواندن داده ها و همچنین مرتب و آماده سازی داده در آن پیاده سازی شده بود. همچنین داده ها را برای آموزش و تست جدا کرده و از یک مدل UNET ساده استفاده کرده بود. در نهایت آن را اجرا کردیم و نتیجه آن را دیدیم.



فایل پیوست شده: CV_Project1_Tile.ipynb

اضافه کردن الگو به ورودی مدل

در این مرحله ایده ای که به ذهن ما رسید این بود که علاوه بر خود تصویر ، الگو را هم به آن بدهیم. برای این کار ورودی مدل را بجاش ۳ کانال به ۶ کانال تغییر دادیم. در واقع سه کانال اول عکس مطلوب و سه کانال بعدی طرح کاشی بودند. برای این کار کد داده شده را اندکی تغییر دادیم. همان طور که انتظار داشتیم نتیجه بهتری را کسب کردیم.



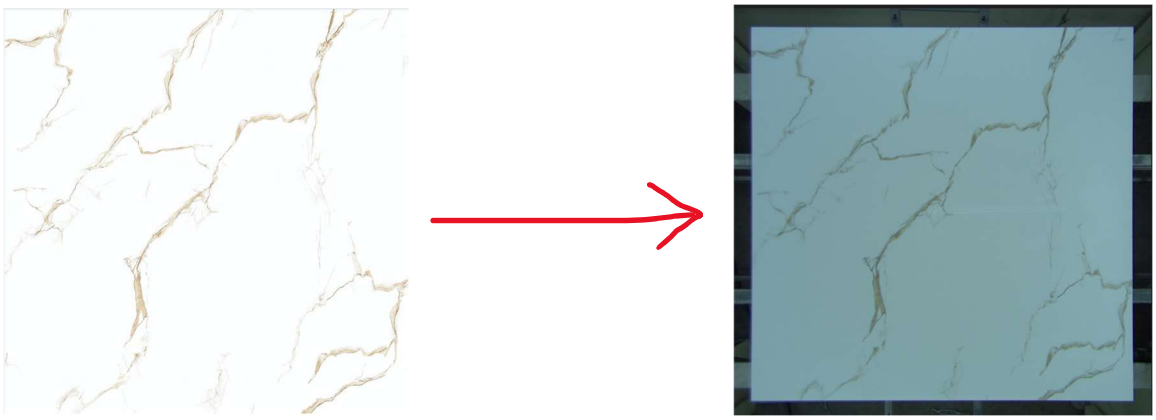
فایل پیوست شده: CV_Project1_Tile_1.ipynb

پیش پردازش داده ها

در این بخش ایده که قرار است پیاده سازی کنیم این است عکس کاشی و طرح متناظر را با هم ترکیب کنیم. در واقع طرح را از عکس کاشی کم کنیم و سپس تصویر باقی مانده را به شبکه بدهیم.

برای این کار گام های زیر را طی میکنیم:

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



کد آن:

```
# histogram matching on pattern
# change the pattern to match the histogram of the image
reference = img
source = pattern
matched_pattern = match_histograms(source, reference, multichannel=True)
```

۲. پیدا کردن نقاط کلیدی و استفاده از آنها برای حذف چرخش:

برای این منظور، تابع find_keypointsرا می نویسیم .در این تابع نقاط کلیدی عکس کاشی و patternاستخراج می شوند و درمرحله بعد بهترین نقاط را به تابع find_rotation_degreeمی 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 ،patternدر قسمت پایین و راست تصویر باشد به این نتیجه می رسیم که عکس 180 درجه چرخیده است).

منابع:

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

کد آن به شرح زیر است:

```python
# https://pysource.com/2018/07/20/find-similarities-between-two-images-with-opencv-and-python/
def find_keypoints(img1, img2):
  original = img1 # queryImage
  image_to_compare = img2 # trainImage
  height, width, c = original.shape

  sift = cv.xfeatures2d.SIFT_create()
  kp_1, desc_1 = sift.detectAndCompute(original, None)
  kp_2, desc_2 = sift.detectAndCompute(image_to_compare, None)

  index_params = dict(algorithm=0, trees=5)
  search_params = dict()
  flann = cv.FlannBasedMatcher(index_params, search_params)

  matches = flann.knnMatch(desc_1, desc_2, k=2)

  good_points = []
  ratio = 0.7
  for m, n in matches:
      if m.distance < ratio*n.distance:
          good_points.append(m)

  no_of_matches = len(good_points)
  # result = cv.drawMatches(original, kp_1, image_to_compare, kp_2, good_points, None)

  deg = find_rotation_degree(good_points, kp_1, kp_2)
  return deg
```

```python
def find_rotation_degree(good_points, kp_1, kp_2):
  # kp_1[i].pt -> mokhtasat in tile image
  # kp_2[i].pt -> mokhtasat in pattern
  # good_points[i].queryIdx -> keypoint in tile
  # good_points[i].trainIdx -> keypoint in pattern

  degrees = {'0' : 0, '90': 0, '180' : 0, '270' : 0} # all degrees can be rotated
  for good_point in good_points:
    key1_index = good_point.queryIdx
    key2_index = good_point.trainIdx

    x_1, y_1 = kp_1[key1_index].pt # 0 to 1600
    x_2, y_2 = kp_2[key2_index].pt # 0 to 5000
    # print(f"({x_1}, {y_1}) -> ({x_2}, {y_2})")

    # 800,2500 ?
    # ------------------------------
    if x_1 < 800 and y_1 < 800: # left top
      if x_2 < 800 and y_2 < 800:
        degrees['0'] += 1
      elif x_2 < 800 and y_2 > 800:
        degrees['90'] += 1
      elif x_2 > 800 and y_2 > 800:
        degrees['180'] += 1
      elif x_2 > 800 and y_2 < 800:
        degrees['270'] += 1
    # ------------------------------
    elif x_1 < 800 and y_1 > 800: # left botton
      if x_2 < 800 and y_2 < 800:
        degrees['270'] += 1
      elif x_2 < 800 and y_2 > 800:
        degrees['0'] += 1
      elif x_2 > 800 and y_2 > 800:
        degrees['90'] += 1
      elif x_2 > 800 and y_2 < 800:
        degrees['180'] += 1
    # ------------------------------
    elif x_1 > 800 and y_1 > 800: # right top
      if x_2 < 800 and y_2 < 800:
        degrees['180'] += 1
      elif x_2 < 800 and y_2 > 800:
        degrees['270'] += 1
      elif x_2 > 800 and y_2 > 800:
        degrees['0'] += 1
      elif x_2 > 800 and y_2 < 800:
        degrees['90'] += 1
    # ------------------------------
    elif x_1 > 800 and y_1 < 800: # right botton
      if x_2 < 800 and y_2 < 800:
        degrees['90'] += 1
      elif x_2 < 800 and y_2 > 800:
        degrees['180'] += 1
      elif x_2 > 800 and y_2 > 800:
        degrees['270'] += 1
      elif x_2 > 800 and y_2 < 800:
        degrees['0'] += 1
  result = None
  # print(degrees)
  m = 0
  for k in degrees.keys(): # finding the most repeated degree
    if degrees[k] > m:
      m = degrees[k]
      result = k
  return result
```

همچنین قبل از این دو مرحله ابعاد کاشی و طرح آن را یکی میکنیم.

فایل پیوست شده: CV_Project1_Tile_2.ipynb

۳. حذف طرح از کاشی (منها کردن)

در این بخش قرار است که طرح کاشی را از خود عکس کاشی حذف کنیم. در اینجا از قواعد مورفولوژی برای گسترش الگو و همین طور آستانه گذاری حدی(یافتن ترک و الگو) استفاده کنیم .ابتدا با آستانه گذاری حدی هر پیکسل را بهه نسبت مربع پیکسل های مجاور مقایسه میکنیم که اگر بزرگتر از عدد آستانه بود آن پیکسل را یک و اگر نه صفر میگنیم.پس از آن ترک و طرح ۰ میشوند و پس زمینه ۱. سپس با ساختن دو کرنل ۷در ۷و همین طور ۳در ۳به صورت دایره ای آمدیم خود طرح کاشی را ابتدا چند گسترش می دهیم سپس یک بار سایش و در آخر عملگر باز میزنیم(برای حذف نقاط سفید داخل طرح) تا طرح کاشی بزرگتر شود. در نهایت طرح گسترش یافته را از کاشی کم میکنیم تا فقط ترک ها باقی بمانند.

منابع:

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

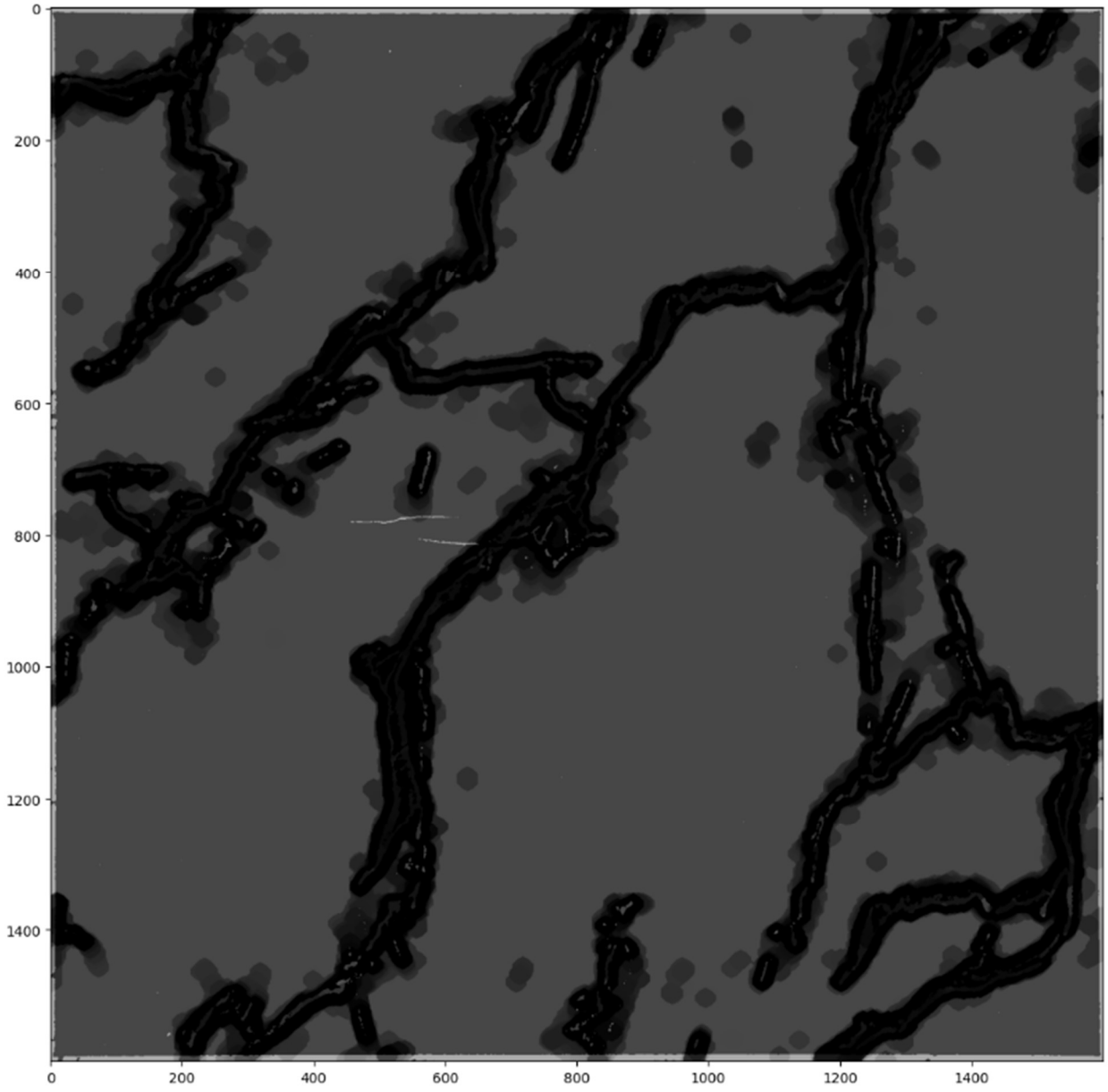
https://docs.opencv.org/3.4/dd/d4d/tutorial_js_image_arithmetics.html

```python
def subtract_with_morpholgy(image, pattern):
  gray_image = cv.cvtColor(image, cv.COLOR_BGR2GRAY )
  binary_image = cv.adaptiveThreshold(gray_image,100,cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY,21,5)
  # convert to binary image (background is white, foreground is black)

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


  gray_pattern = cv.cvtColor(pattern, cv.COLOR_BGR2GRAY )
  binary_pattern = cv.adaptiveThreshold(gray_pattern,100,cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY,21,5)

  # growing the pattern
  eroded = cv.erode(gray_pattern,kernel,iterations=5)
  eroded_2 = cv.erode(eroded,kernel2,iterations=3)
  dilated = cv.dilate(eroded_2,kernel)
  opened = cv.morphologyEx(dilated, cv.MORPH_OPEN, kernel2,iterations=2) # removing the white pixels inside

  res = cv.subtract(opened, binary_image) # https://www.geeksforgeeks.org/how-to-subtract-two-images-using-p
  # res = cv.resize(temp, dsize=(0,0), fx=224/1516, fy=224/1516)

  res = cv.cvtColor(res, cv.COLOR_GRAY2BGR)

  return res
```

در نهایت این سه گام را در یک تابع خلاصه کردیم از آن در کلاس class Tiles(keras.utils.Sequence) استفاده کردیم.

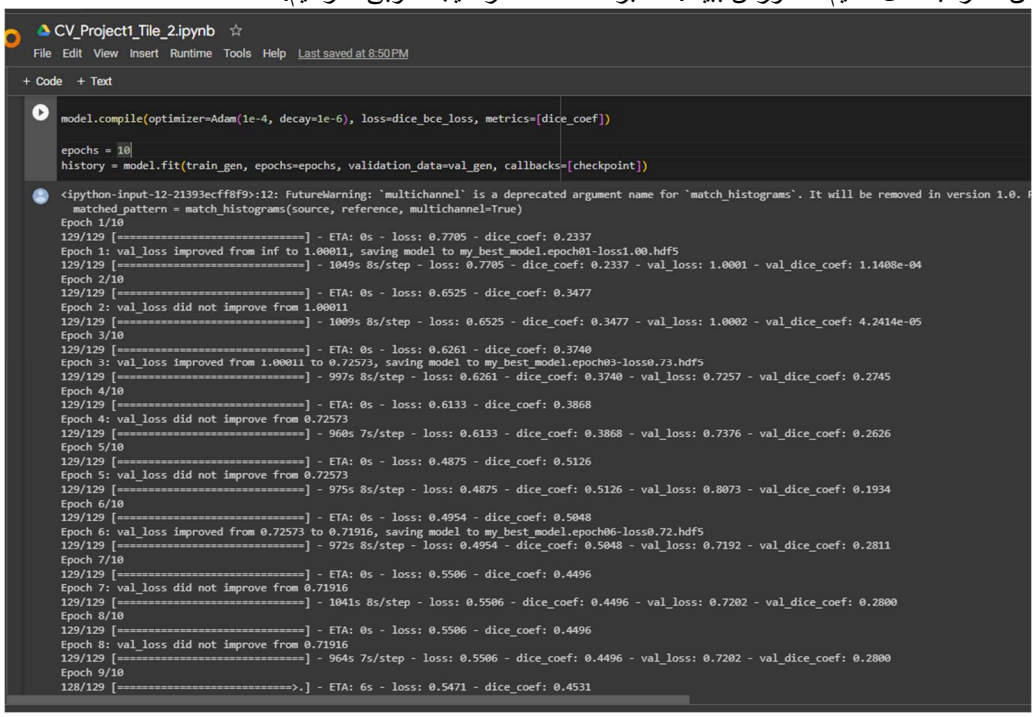
```python
from skimage.exposure import match_histograms

def subtract_pattern_from_image(img, pattern):
    """
        preprocessing and subtracting patten from image
    """

    # histogram matching on pattern
    # change the pattern to match the histogram of the image
    reference = img
    source = pattern
    matched_pattern = match_histograms(source, reference, multichannel=True)

    # removing rotation
    rotate_degree = find_keypoints(img, matched_pattern)
    rotated_matched_pattern = rotate_image(matched_pattern, rotate_degree) # also can use original pattern size 5000*5000 taher than resied 1600*1600

    # subtracting pattern from image
    subtracted_image = subtract_with_morpholgy(img, rotated_matched_pattern)

    return subtracted_image
```

خروجی این تابع تا حدودی ترک ها را تشخیص می دهد حتی بدون استفاده از شبکه عصبی ولی نتیجه مطلوبی ندارد.



در نهایت داده های آموزشی را با استفاده از تابع بالا ترکیب نمودیم. در واقع طرح های مورد نظر را از کاشی ها کم کردیم. سپس

آن ها را به مدل دادیم تا آموزش ببیند. اما بر خلاف انتظار نتیجه خوبی نگرفتیم.



تغییر مدل و استفاده از مدل ZF_UNET_224

پس از آنکه از مدل بالا نتیجه خوبی حاصل نشد تصمیم به تغییر مدل کردیم. همچنین چون وزن های از پیش آموزش دیده این مدل در اینترنت وجود داشت فکر کردیم که شاید نتیجه بهتری بدهد. این مدل نوعی از UNET هاست که ناحیه بندی(segmentation) کردن تصویر کاربرد دارد. عوض کردن مدل کار زیادی داشت و به ارور های مختلفی برخورد کردیم و بخش زیادی از کد را عوض کردیم. اما پس از اجرای آن باز هم نتیجه خوبی حاصل نشد. این نوع شبکه را با مشورت آقای بهکام کیا پیدا کردیم.

منابع:

https://arxiv.org/abs/1505.04597
https://www.kaggle.com/c/dstl-satellite-imagery-feature-detection
https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93
https://kaggle.com/zfturbo
https://github.com/ZFTurbo/ZF_UNET_224_Pretrained_Model/releases/download/v1.0/zf_unet_224.h5

فایل پیوست شده: CV_Project1_Tile_3.ipynb

```python
def ZF_UNET_224(dropout_val=0.2, weights=None):
    inputs = Input((224, 224, INPUT_CHANNELS))
    # inputs = Input((1600, 1600, INPUT_CHANNELS))
    axis = 3
    filters = 32

    conv_224 = double_conv_layer(inputs, filters)
    pool_112 = MaxPooling2D(pool_size=(2, 2))(conv_224)

    conv_112 = double_conv_layer(pool_112, 2*filters)
    pool_56 = MaxPooling2D(pool_size=(2, 2))(conv_112)

    conv_56 = double_conv_layer(pool_56, 4*filters)
    pool_28 = MaxPooling2D(pool_size=(2, 2))(conv_56)

    conv_28 = double_conv_layer(pool_28, 8*filters)
    pool_14 = MaxPooling2D(pool_size=(2, 2))(conv_28)

    conv_14 = double_conv_layer(pool_14, 16*filters)
    pool_7 = MaxPooling2D(pool_size=(2, 2))(conv_14)

    conv_7 = double_conv_layer(pool_7, 32*filters)

    up_14 = concatenate([UpSampling2D(size=(2, 2))(conv_7), conv_14], axis=axis)
    up_conv_14 = double_conv_layer(up_14, 16*filters)

    up_28 = concatenate([UpSampling2D(size=(2, 2))(up_conv_14), conv_28], axis=axis)
    up_conv_28 = double_conv_layer(up_28, 8*filters)

    up_56 = concatenate([UpSampling2D(size=(2, 2))(up_conv_28), conv_56], axis=axis)
    up_conv_56 = double_conv_layer(up_56, 4*filters)

    up_112 = concatenate([UpSampling2D(size=(2, 2))(up_conv_56), conv_112], axis=axis)
    up_conv_112 = double_conv_layer(up_112, 2*filters)

    up_224 = concatenate([UpSampling2D(size=(2, 2))(up_conv_112), conv_224], axis=axis)
    up_conv_224 = double_conv_layer(up_224, filters, dropout_val)
```

نتیجه :

```
+ Code  + Text                                                                    Reconnect

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),
]

[ ] # history = model.fit(input_images_train, masked_labels_train, validation_split=0.1, epochs=epochs, verbose=2)
    history = model.fit(train_gen, epochs=epochs, validation_data=val_gen, verbose=2)

    <ipython-input-60-21393ecff8f9>:12: FutureWarning: `multichannel` is a deprecated argument name for `match_histograms`. It will be removed in version 1.
      matched_pattern = match_histograms(source, reference, multichannel=True)
    Epoch 1/10
    129/129 - 973s - loss: -1.6250e-01 - dice_coef: 0.1625 - val_loss: -2.8458e-01 - val_dice_coef: 0.2846 - 973s/epoch - 8s/step
    Epoch 2/10
    129/129 - 944s - loss: -4.5002e-01 - dice_coef: 0.4500 - val_loss: -2.8458e-01 - val_dice_coef: 0.2846 - 944s/epoch - 7s/step
    Epoch 3/10
    129/129 - 946s - loss: -4.5246e-01 - dice_coef: 0.4525 - val_loss: -2.8458e-01 - val_dice_coef: 0.2846 - 946s/epoch - 7s/step
    Epoch 4/10
    129/129 - 924s - loss: -4.5220e-01 - dice_coef: 0.4522 - val_loss: -2.8458e-01 - val_dice_coef: 0.2846 - 924s/epoch - 7s/step
    Epoch 5/10
    129/129 - 919s - loss: -4.5250e-01 - dice_coef: 0.4525 - val_loss: -2.8458e-01 - val_dice_coef: 0.2846 - 919s/epoch - 7s/step
    Epoch 6/10
    129/129 - 946s - loss: -4.5227e-01 - dice_coef: 0.4523 - val_loss: -2.8458e-01 - val_dice_coef: 0.2846 - 946s/epoch - 7s/step
    Epoch 7/10
    129/129 - 941s - loss: -4.5074e-01 - dice_coef: 0.4507 - val_loss: -2.8458e-01 - val_dice_coef: 0.2846 - 941s/epoch - 7s/step
    Epoch 8/10
    129/129 - 961s - loss: -4.5251e-01 - dice_coef: 0.4525 - val_loss: -2.8458e-01 - val_dice_coef: 0.2846 - 961s/epoch - 7s/step
    Epoch 9/10
```

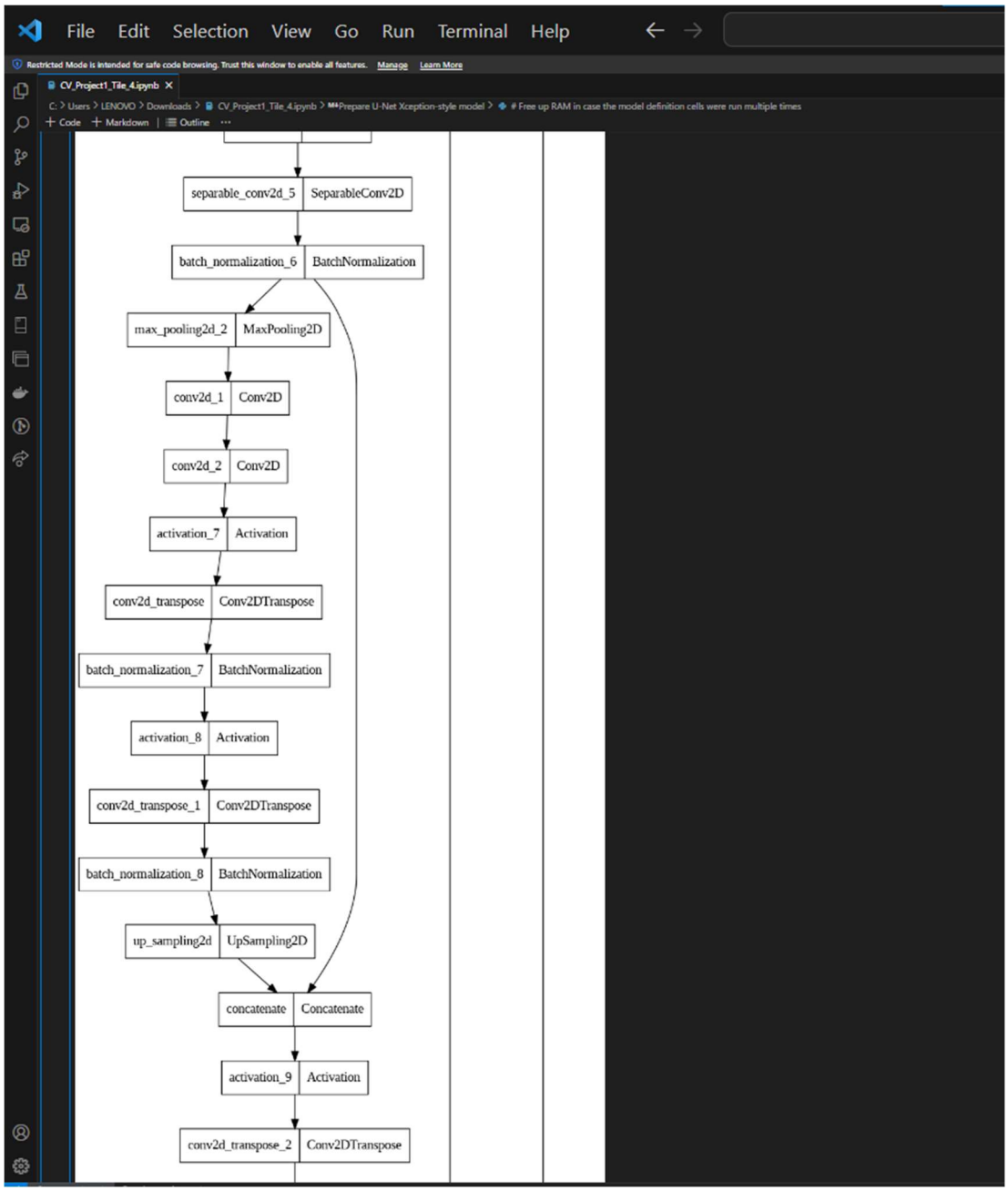
پیاده سازی UNET به روش دیگر

با توجه به گراف کشیده شده در مدلی که در صورت سوال داده شده بود دیدیم که این مدل از skip connection استفاده نمی کند. به همین دلیل آن را عوض کردیم. برای پیاده سازی از لینک زیر کمک گرفتیم. ولی باز هم نتیجه مطلوبی حاصل نشد. و با توجه به کمبود زمان و مدت زمان نیاز به آموزش آن بعد از چند epoch آن را متوقف کردیم و نذاشتیم بیشتر train بشود.

منبع:

https://pyimagesearch.com/2022/02/21/u-net-image-segmentation-in-keras/

فایل پیوست شده: CV_Project1_Tile_4.ipynb

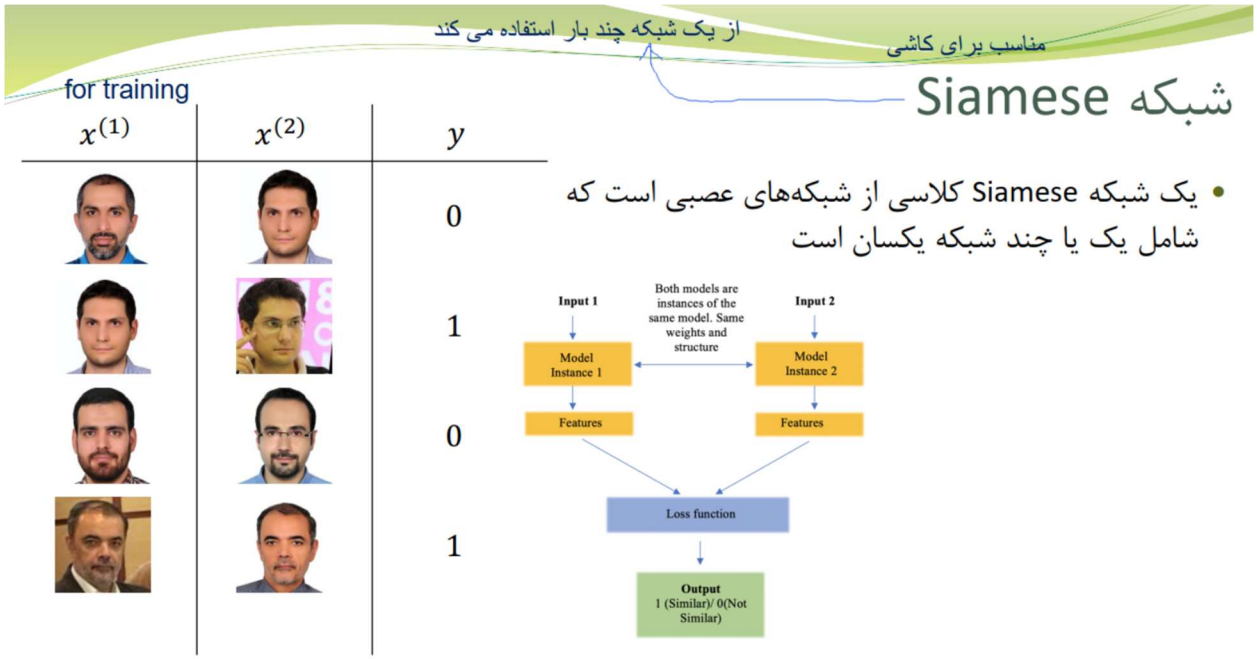


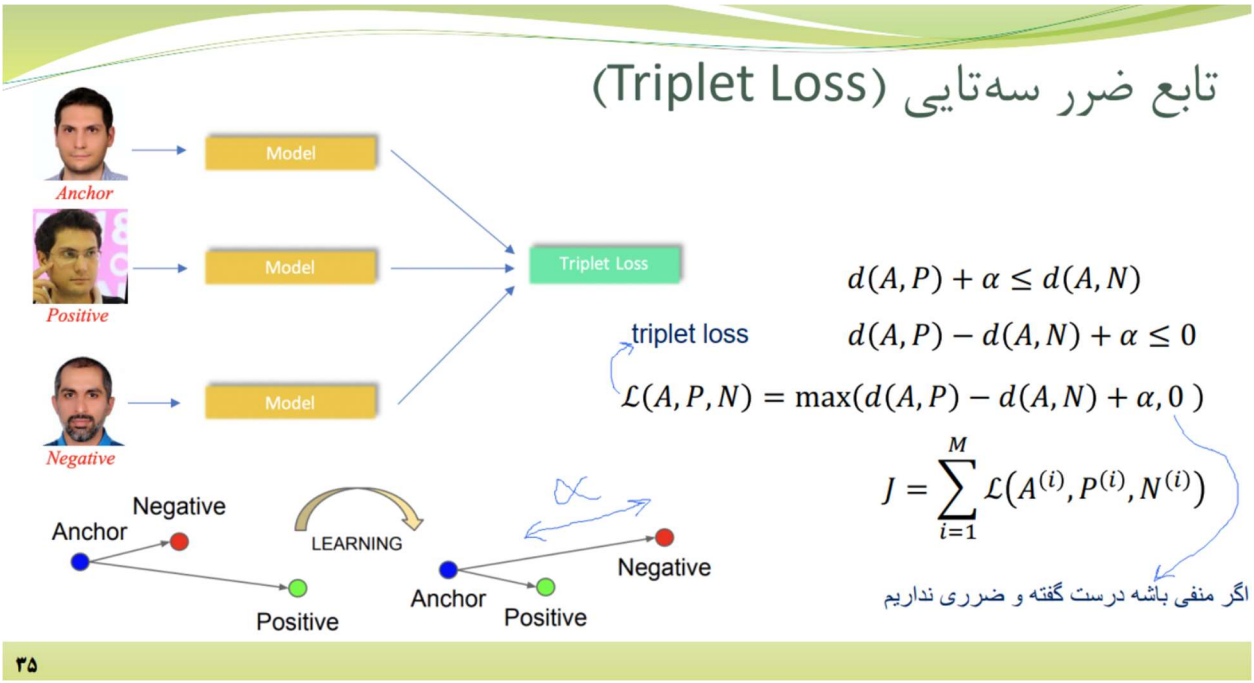
نتایج مدل:

```
+ Code + Markdown | Run All | Clear All Outputs | Outline ...                Select Kernel

    checkpoint = ModelCheckpoint(
        filepath=filepath,
        monitor='val_loss',
        verbose=1,
        save_best_only=True,
        mode='min'
    )

    model.compile(optimizer=Adam(1e-4, decay=1e-6), loss=dice_bce_loss, metrics=[dice_coef])

    epochs = 10
    history = model.fit(train_gen, epochs=epochs, validation_data=val_gen, callbacks=[checkpoint])

                                                                          Python

...  <ipython-input-12-21393ecff8f9>:12: FutureWarning: `multichannel` is a deprecated argument name for `match_histograms`. It will be removed in version 1
       matched_pattern = match_histograms(source, reference, multichannel=True)
     Epoch 1/10
     129/129 [==============================] - ETA: 0s - loss: 0.9993 - dice_coef: 0.0037
     Epoch 1: val_loss improved from inf to 0.99730, saving model to my_best_model.epoch01-loss1.00.hdf5
     129/129 [==============================] - 985s 8s/step - loss: 0.9993 - dice_coef: 0.0037 - val_loss: 0.9973 - val_dice_coef: 0.0029
     Epoch 2/10
     129/129 [==============================] - ETA: 0s - loss: 0.9971 - dice_coef: 0.0042
     Epoch 2: val_loss improved from 0.99730 to 0.99728, saving model to my_best_model.epoch02-loss1.00.hdf5
     129/129 [==============================] - 963s 7s/step - loss: 0.9971 - dice_coef: 0.0042 - val_loss: 0.9973 - val_dice_coef: 0.0029
     Epoch 3/10
      36/129 [=======>......................] - ETA: 10:10 - loss: 0.9969 - dice_coef: 0.0042
```

استفاده از معماری Siamese

پس از شکست های پی در پی در مراحل قبل با اندک رمق باقی مانده برایمان به سراغ این نوع شبکه ها رفتیم. با توجه به اینکه این نوع از شبکه های عصبی چندین ورودی میگیرند و همچنین تمام مراحل قبلی نتیجه خوب و قابل قبولی نداشتند حال به پیاده سازی این ایده می پردازیم. با مشورت هایی که با خانم سبزه‌واری داشتیم به این نتیجه رسیدیم. در واقع با این نوع پیاده سازی عملیات حذف کردن مطلوب را به شبکه میسپاریم.





منابع:

https://keras.io/examples/vision/siamese_network/
https://medium.com/data-science-in-your-pocket/understanding-siamese-network-with-example-and-codes-e7518fe02612
https://towardsdatascience.com/siamese-networks-introduction-and-implementation-2140e3443dee

https://builtin.com/machine-learning/siamese-network

https://towardsdatascience.com/siamese-networks-introduction-and-implementation-2140e3443dee

برای پیاده سازی وقت کافی نداشتیم. ولی معماری مدلی که ایده پیاده سازی آن را داشتیم به شکل زیر است.

```
+ Code  + Text

input = layers.Input((1600, 1600, 3))
x = tf.keras.layers.BatchNormalization()(input)
x = layers.Conv2D(4, (5, 5), activation="relu")(x)
x = layers.AveragePooling2D(pool_size=(2, 2))(x)
x = layers.Conv2D(16, (5, 5), activation="relu")(x)
x = layers.AveragePooling2D(pool_size=(2, 2))(x)
x = layers.Conv2D(32, (5, 5), activation="relu")(x)
x = layers.AveragePooling2D(pool_size=(2, 2))(x)
x = layers.Conv2D(64, (5, 5), activation="relu")(x)
x = layers.AveragePooling2D(pool_size=(2, 2))(x)
x = layers.Conv2D(128, (5, 5), activation="relu")(x)
x = layers.AveragePooling2D(pool_size=(2, 2))(x)
x = layers.Flatten()(x)

x = tf.keras.layers.BatchNormalization()(x)
x = layers.Dense(10000, activation="relu")(x)
embedding_network = keras.Model(input, x)


input_1 = layers.Input((1600, 1600, 3))
input_2 = layers.Input((1600, 1600, 3))

# As mentioned above, Siamese Network share weights between
# tower networks (sister networks). To allow this, we will use
# same embedding network for both tower networks.

tower_1 = embedding_network(input_1)
tower_2 = embedding_network(input_2)

merge_layer = layers.Lambda(euclidean_distance)([tower_1, tower_2])
normal_layer = tf.keras.layers.BatchNormalization()(merge_layer)
output_layer = layers.Dense(1, activation="sigmoid")(normal_layer)
siamese = keras.Model(inputs=[input_1, input_2], outputs=output_layer)
```

همچنین می توانیم به جای ورودی دادن یک تصویر بزرگ به شبکه از ایده ی sliding window استفاده کنیم. اگر مجموع نقاط روشن موجود در یک پنجره از ماسک بیشتر از ۰ باشد به این معنی است که پنجره در ناحیه تر کها بوده است و آن را به قسمت داده های ترک دار اضافه میکنیم. در غیر این صورت پنجره ها شامل طرح یا پس زمینه اند. بدین ترتیب دیتا ستی میسازیم که شامل پنجره هایی که حاوی ترک هستند و پنجره هایی که ترک ندارند. و از آنها برای شبکه Siamese کوچتری نسبت به شکل بالا استفاده میکنیم.

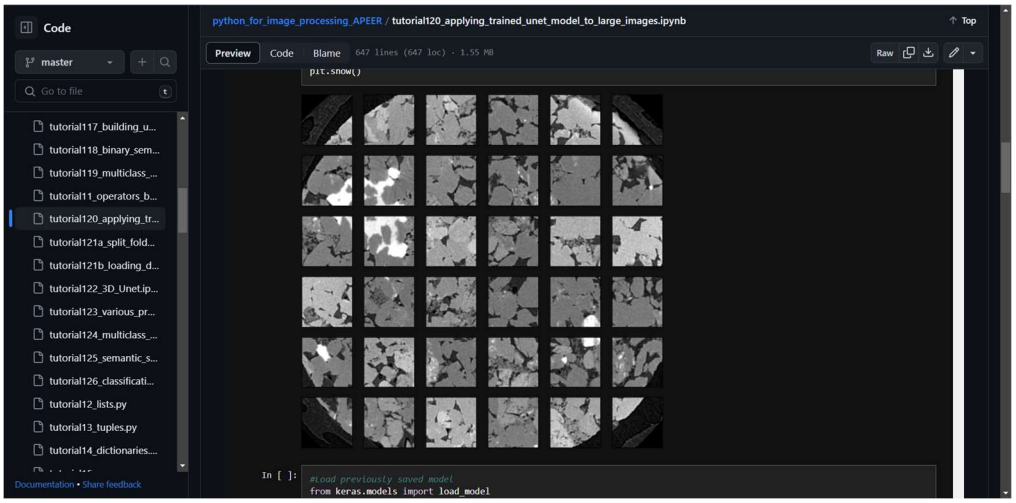
فایل های پیوست شده: CV_Project1_Tile_5.ipynb, CV_Project1_Tile_6.ipynb

ایده پیشنهادی برای پیاده سازی در آینده:

تقسیم عکس کاشی به قسمت های مساوی و دادن هر یک به مدل و سپس وصل کردن آنها به یکدیگر

اطلاعات بیشتر و منبع:

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



پاسخ به سوال مطرح شده در مستند پروژه به کمک chat GPT:
یکی دیگر از چالشها این است که برچسب های تهیه شده برای ترکها به صورت کاملا دقیق برچسبگذاری نشده اند و در محدوده مشخص شده ممکن است برخی پیکسل ها دچار ترک نشده باشند (برچسبگذاری دقیق نیاز به زمان و هزینه بسیار بیشتری دارد.) شما باید راهکاری برای مقابله با این چالش پیشنهاد کنید. سپس، با استفاده از دادههای در اختیار گرفته، مدلی آموزش داده شود که بتواند محلهای دارای ترک را شناسایی کند.

To address the challenge of inaccurately labeled tiles with cracks, we can follow a semi-supervised learning approach that combines both labeled and unlabeled data. This way, we can make better use of limited labeled data, leverage a larger amount of unlabeled data, and reduce the time and money spent on accurate labeling. Here's a step-by-step plan:

1. **Data Preprocessing**: Preprocess the data to remove noise, inconsistencies, and irrelevant details. This can include resizing the images, converting to grayscale if necessary, and normalizing pixel values.
2. **Initial Model Training**: Train an initial deep learning model (such as a Convolutional Neural Network) using the limited labeled dataset. This model will serve as a starting point for the semi-supervised learning process.
3. **Pseudo-Labeling**: Apply the initial model to the unlabeled dataset and generate predictions, which are considered as "pseudo-labels." Select only the high-confidence pseudo-labels to create a new labeled dataset.
4. **Data Augmentation**: Perform data augmentation techniques (like rotations, flips, and translations) to increase the size of both the original labeled dataset and the pseudo-labeled dataset. This helps improve model generalization and performance.
5. **Model Fine-tuning**: Combine the original labeled dataset with the high-confidence pseudo-labeled dataset and retrain the model. This process iteratively improves the model's performance.
6. **Validation**: Split the initial labeled dataset into a training and validation set. Evaluate the model's performance on the validation set throughout the fine-tuning process to prevent overfitting and ensure that the model generalizes well to unseen data.
7. **Hyperparameter Tuning**: Experiment with different hyperparameters (e.g., learning rate, batch size, and network architecture) to optimize the model's performance.
8. **Model Evaluation**: Once the model has been fine-tuned and its performance has reached a satisfactory level, evaluate it on a separate test set. This will provide an unbiased estimate of the model's ability to identify tile cracks in unseen data.

Remember that it's essential to monitor the model's performance and make necessary adjustments throughout the process. This semi-supervised learning approach should help you mitigate the challenge of inaccurate labels and train a model capable of identifying tile cracks effectively.

منابع:

https://keras.io/examples/vision/siamese_network/

https://stackoverflow.com/questions/50298329/error-5-image-is-empty-or-has-incorrect-depth-cv-8u-in-function-cvsift

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

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

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

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

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

https://stackoverflow.com/questions/16002709/how-to-apply-ransac-on-surf-sift-and-orb-matching-results

پایان