مقدمه :

در این تکلیف ما قصد داشتیم با استفاده از مدل U-net که برای انجام تسک های semantic segmentation روی تصاویرانجام می شود، استفاده کنیم و روی دیتاست HoVer-Net که شامل عکس های چند لایه بافت شناسی می باشد اجرا کنیم تا یکسری هسته های بافتی را با توجه به کلاس هایی که در فایل توضیحات دیتاست در فایل Readme این دیتاست بود به کمک مدل هوش مصنوعی مشخص کنیم. این نوع داده های در علوم پاتولوژی و آسیب شناسی کاربرد دارند و استفاده می شوند.

درک دیتاست HoVer-Net:

دیتاست مورد نظر از لینک https://warwick.ac.uk/fac/cross_fac/tia/data/hovernet/ قابل دریافت می باشد. فایل مورد نظر به فرمت zip می باشد و بعد از unzip کردن فایل به ۲ فایل میرسیم که از فولدر CoNSeP برای load کردن دیتاست استفاده شده است. پوشه CoNSeP شامل ۲ پوشه train و test و یک فایل Readme می باشد که در رابطه ساختار فایل های دیتاست و کلاس های مسئله می باشد . در *شکل ۱* این جزئیات قابل مشاهده می باشند.

```
Dataset Description:

Each ground truth file is stored as a .mat file, with the keys:
'inst_map'
'type_map'
'inst_type'
'inst_centroid'

'inst_map' is a 1000x1000 array containing a unique integer for each individual nucleus. i.e the map ranges from 0 to N, where 0 is the background and N is the number of nuclei

'type_map' is a 1000x1000 array where each pixel value denotes the class of that pixel. The map ranges from 0 to 7, where 7 is the total number of classes in CoNSeP.

'inst_type' is a Nx1 array, indicating the type of each instance (in order of inst_map ID)

'inst_centroid' is a Nx2 array, giving the x and y coordinates of the centroids of each instance (in order of inst map ID).

Note, 'inst_type' and 'inst_centroid' are only used while computing the classification statistics.

The values within the class map indicate the category of each nucleus.

Class values: 1 = other
              2 = inflammatory
              3 = healthy epithelial
              4 = dysplastic/malignant epithelial
              5 = fibroblast
              6 = muscle
              7 = endothelial
```

شکل ۱

همانطور که در شکل بالا مشخص است، لیبل های اصلی یا همان فایل های ground truth که ما از type_map استفاده میکنیم به صورت فایل های mat. ذخیره شده اند که می توان به کمک کتاب خانه scipy، اطلاعات کلید type_map را که به صورت دیکشنری هست و حاوی یک ماتریس ۱۰۰۰*۱۰۰۰ که سایز اولیه هر عکس نیز میباشد است و هر عدد در این ماتریس لیبل پیکسل متناظر با آن پیکسل در عکس مورد نظر است را می توان استخراج کرد. توجه

شود که به ازای هر عکس در فایل images ، یک فایل متناظر با آن که فرمت mat. موجود است که از هر کدام اطلاعات type_map را استخراج میکنیم.در *شکل ۲* و *شکل ۳* یکسری اطلاعات استخراج شده مربوط به فایل train10 را مشاهده میکنیم.

Data visualization:

```
[ ]  base_folder = 'CoNSeP'
     sub_folder = ['Train', 'Test']
     imgs, lbls, ovr, cls = 'Images', 'Labels', 'Overlay', 'imgs'
     sample = 'train_10.mat'

     # below line returns dict of ground truth datas with keys: inst_map, type_map, inst_type, inst_centroid
     # so for this project we need only 'type_map' and we extract it's datas.
     mat_info = scipy.io.loadmat(os.path.join(base_folder, sub_folder[0], lbls, sample))
     print('Labels for {} : \n\n'.format(sample), mat_info['type_map'])
     print('\nimage {} shape:'.format(sample), mat_info['type_map'].shape)
```

```
Labels for train_10.mat :

 [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 3. 3. 3.]
 [0. 0. 0. ... 3. 3. 3.]
 [0. 0. 0. ... 3. 3. 3.]]

image train_10.mat shape: (1000, 1000)
```

شکل ۲

```
[ ]  print('max label value in data set: ', np.max(mat_info['type_map']))
     print('min label value in data set: ', np.min(mat_info['type_map']))
     unique, counts = np.unique(mat_info['type_map'], return_counts=True)
     print('\noccurance of each label in {}:\n'.format(sample),dict(zip(unique, counts)))
```

```
max label value in data set:  7.0
min label value in data set:  0.0

occurance of each label in train_10.mat:
 {0.0: 722036, 1.0: 184, 2.0: 67718, 3.0: 131867, 5.0: 77327, 7.0: 868}
```

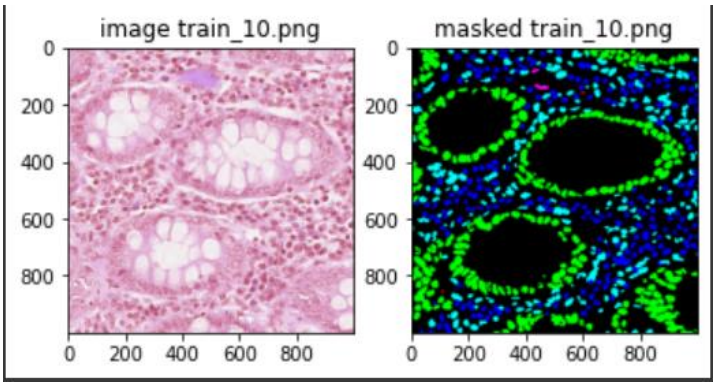
شکل ۳

این دیتاست شامل ۲۷ سمپل برای train و ۱۴ عدد سمپل برای test می باشد.

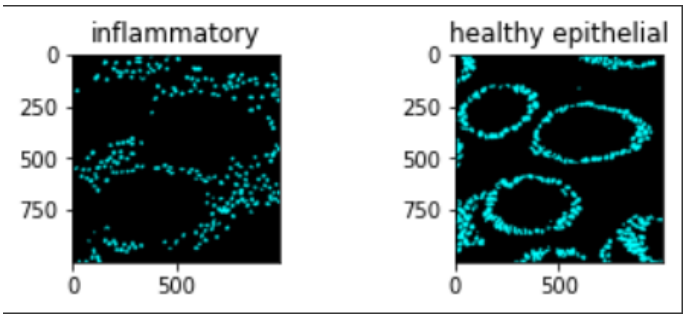
در ادامه می بایست از لیبل های هر عکس، عکس متناظر mask آن را بسازیم چون دیتاست مورد نظر فایل های ماسک را نداشته است.

این مسئله به با مدل چند کلاسه و تک کلاسه(بر اساس لیبل های هر پیکسل) پیاده سازی شده است به همین دلیل یک سری فایل های ماسک را برای همه کلاس ها می سازیم، و یکسری به ازای هر کلاس، فایل ها ی ماسک را میسازیم که

فقط پیکسل های آن کلاس در فایل ماسک مورد نظر مشخص باشند. برای ساختن فایل های ماسک از روی لیبل ها، از تابع label2rgb در ماژول skimage کمک میگیریم در *شکل ۴* و *شکل ۵* نمونه آن آورده شده.



شکل ۴ پیکسل کلاس های متفاوت در هر ماسک



شکل ۵ پیکسل های هر کلاس در هر mask جداگانه

در ادامه برای هر کدام از روش های multiclass segmentation و binary segmentation، فایل های جداگانه ماسک را در پوشه های مخصوص به خود برای train که برای آموزش نیاز است و برای test که برای صرفا evaluate کردن مدل نهایی می باشد، آماده میکنیم.

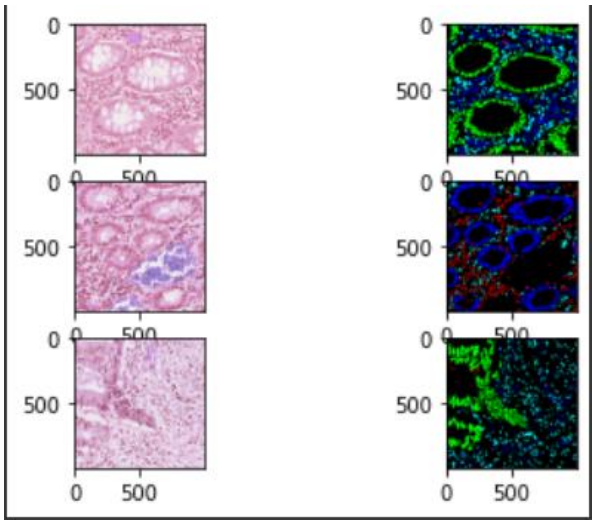
در *شکل ۶* و *شکل ۷* به ترتیب برای مدل چند کلاسه و مدل باینری آورده شده است.

```
[ ]  # dataGen = ImageDataGenerator(rescale=1./255, brightness_range=(0.8, 1.2))
     # train_datagen = dataGen.flow_from_directory('CoNSeP/Train/Images', batch_size=32, target_size=(1000, 1000), class_mode='categorical')

     def maskLabels(base_folder, sub_folder, lbls):    # this function is for multi-class mask generation.

         sub = 'Train' if sub_folder == 'Train' else 'Test'
         for label_file in tqdm( os.listdir(os.path.join(base_folder, sub, lbls)), desc='converting labels to mask for {}: '.format(sub) ):
             mat_label = scipy.io.loadmat(os.path.join(base_folder, sub, lbls, label_file))['type_map']
             maskImage = label2rgb(mat_label, colors=colors, bg_label=0, bg_color=(0, 0, 0))
             cv2.imwrite(os.path.join(base_folder, sub, 'maskedLabels', cls, '{}.png'.format(label_file.split('.')[0])), maskImage)

     maskLabels(base_folder, sub_folder[0], lbls)   # for train
     maskLabels(base_folder, sub_folder[1], lbls)   # for test

     converting labels to mask for Train: 100%|██████████| 27/27 [00:05<00:00,  4.60it/s]
     converting labels to mask for Test: 100%|██████████| 14/14 [00:03<00:00,  4.61it/s]
```

شکل ۶

```
def binary_maskCreator(base_folder, sub_folder, class_labels, write=True):
    sub = 'Train' if sub_folder == 'Train' else 'Test'

    for i in tqdm(range(1, len(class_labels))):

        for label_file in tqdm( os.listdir(os.path.join(base_folder, sub, lbls)), desc='extracting label ({}) to mask in {}: '.format(class_labels[i], sub)):
            mat_label = scipy.io.loadmat(os.path.join(base_folder, sub, lbls, label_file))['type_map']
            mat_label[mat_label == i] = 1
            mat_label[mat_label != 1] = 0
            maskImage = label2rgb(mat_label, colors=[colors[3]], bg_label=0, bg_color=(0, 0, 0))

            if write:
                cv2.imwrite(os.path.join(base_folder, sub, 'binary', str(i), '{}.png'.format(label_file.split('.')[0])), maskImage)
```

شکل ۷



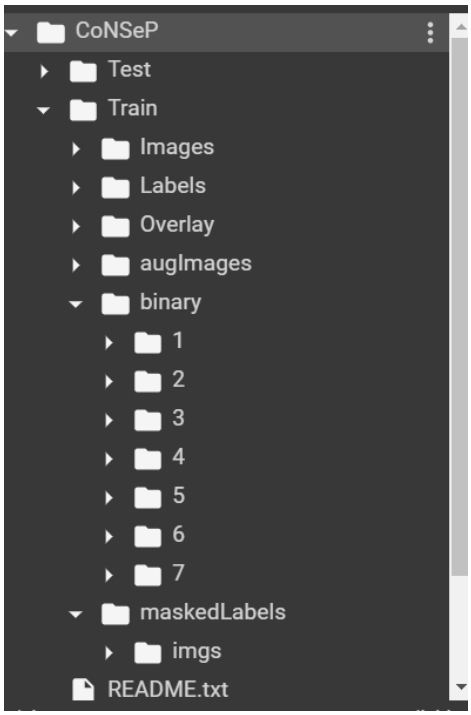
شکل ۸

شکل ۸ نیز چند سمپل کنار هم هستند که برای اطمینان از صحت ذخیره شدن فایل ها در پوشه مورد نظر آورده شده اند.

انجام Data preprocessing and Augmentation:

همانطور که گفته شد، با استفاده از فایل های mat. فایل های ماسک برای سگمنت کردن باینری(به ازای هر کلاس) و سگمنت کردن چند کلاسه در دایرکتوری های مورد نظر توسط توابع مربوطه ذخیره می شوند.

در *شکل ۹* سلسله مراتب دایرکتوری ها برای نگه داری دیتا های train آورده شده است.

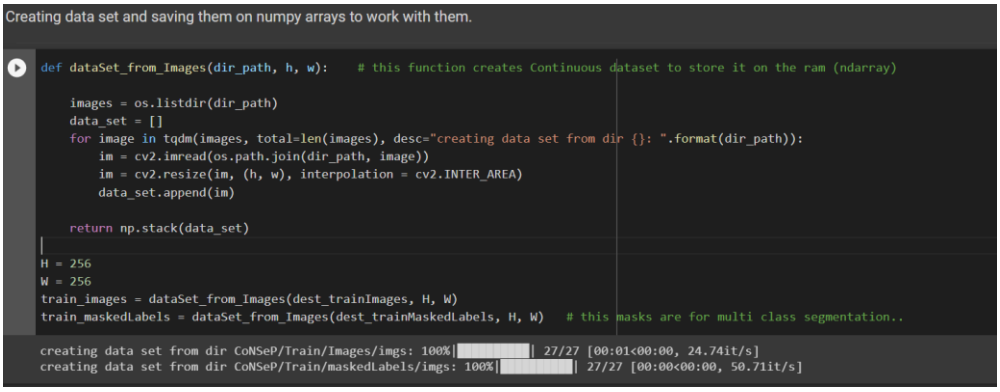


شکل ۹

پوشه باینری شامل چند زیر پوشه است که به شماره کلاس ها شماره گذاری شده که محتوایات هر کدام عکس های ماسک تولید شده مختص به هر کلاس با شماره پوشه می باشد.

پوشه augImages صرفا یک پوشه debug برای اطمینان از صحت انجام augmentation دیتا ها هنگام ترین می باشد که توسط ImageDataGenerator در زیر پوشه های این پوشه نوشته می شوند. این زیر پوشه ها به اسم های image و multiclass و binary هستند که به ترتیب augment شده عکس های اصلی، ماسک چند کلاسه ها و ماسک های باینری به تفکیک کلاس، در آن ها موجود می باشد. این موارد هنگام train شدن هر مدل در داخل پوشه مخصوص به خودش نوشته می شود.

در ادامه تابع data_set_from_Images را داریم دلیل نوشتن این تابع این است که برای دادن داده ها به ImageDataGenerator، باید از تابع flow یا flow_from_dir استفاده کنیم ولی استفاده از تابع flow_from_dir امتحان شد ولی برای مدل ها قابل استفاده نبود و ارور ابعاد دیتا را میداد به همین دلیل برای استفاده از تابع flow این کار انجام گرفته است. در این تابع نیز یک پیش پردازش انجام میگیرد و آن هم این است که ابعاد تصاویر با حفظ ratio تصویر به 256*256 یعنی توانی از ۲ تبدیل میشوند که برای مدل و محاسبات آن بهتر و سریعتر می باشد. تابع گفته شده در *شکل ۱۰* قابل مشاهده می باشد.



شکل ۱۰

در ادامه به انواع پیش پردازش ها و augmentation هایی که روی دیتا ها انجام گرفته است می پردازیم

در ابتدا باید گفت دیتاست موجود در بخش train تنها شامل ۲۷ عدد سمپل دیتا می باشد به همین دلیل برای جلوگیری از underfit شدن مدل، باید از روش augment کردن دیتا استفاده کنیم.

در ابتدا باید گفت که برای train کردن مدل ها از ImageDataGenerator در کراس استفاده شده است که به ترتیب شامل:

- چرخش رندوم عکس ها به اندازه ۴۰ درجه برای بسط دادن دید مدل و ایجاد تفاوت با عکس اصلی.
- Zoom in و zoom out کردن روی تصاویر برای بسط دادن بیشتر دید مدل
- شیف دادن عرضی عکس باز هم به دلیل بسط دادن دید مدل
- Flip کردن عکس ها به صورت افقی
- پوشاندن گوشه ها یا قسمت هایی از عکس که به واسطه augment کردن دیتا به وجود آمده اند.
- در نهایت نرمال کردن پیکسل های عکس ها
- شییر کردن نیز امتحان شد ولی دقت مدل را کمی محدود میکرد به همین خاطر در فرایند آموزش دیگر از آن استفاده نشد.

روش های دیگر پیش پردازش و augmentation نیز تست شدند ولی برای مدل ها یا تفاوت چندانی در دقت و loss مدل نداشتند و یا این ۲ پارامتر را بدتر میکردند.

توجه شود که ما به خاطر داشتن داده های mask به عنوان لیبل، باید داده های mask نیز همراه با داده های image ها augment و پیش پردازش بکنیم تا مدل به درستی دیتا ها را یاد بگیرد.

در ادامه تصاویر موارد توضیح داده شده، *شکل ۱۱*، *شکل ۱۲*، *شکل ۱۳* آورده شده است.

Parameters for DataGenerator:

```
[ ]  augImages = 'augImages'
     data_gen_args = dict(

           rotation_range=40,
           zoom_range=[0.8, 1.2],
           fill_mode='nearest',
           width_shift_range=0.2,
         #   shear_range=0.2,
           horizontal_flip=True ,
           rescale = 1/255.,
     )
```

شکل ۱۱

```
imgs_dataGen = ImageDataGenerator(**data_gen_args)
masks_dataGen = ImageDataGenerator(**data_gen_args)

imgs_dataGen.fit(train_images, augment=True, seed=1)
imgs_dataGen.fit(train_maskedLabels, augment=True, seed=1)

# image_generator = imgs_dataGen.flow_from_directory(
#     os.path.join(base_folder, sub_folder[0], imgs),
#     class_mode=None,
#     target_size = (256, 256),
#     batch_size = 4,
#     save_to_dir = os.path.join(base_folder, sub_folder[0], augImages, 'image'),
#     save_format="png",
#     seed=1
# )

# mask_generator = masks_dataGen.flow_from_directory(
#     os.path.join(base_folder, sub_folder[0], 'maskedLabels'),
#     class_mode=None,
#     target_size = (256, 256),
#     batch_size = 4,
#     save_to_dir = os.path.join(base_folder, sub_folder[0], augImages, 'mask'),
#     save_format="png",
#     seed=1
# )

image_generator = imgs_dataGen.flow(
    train_images,
    batch_size = 4,
    save_to_dir = os.path.join(base_folder, sub_folder[0], augImages, 'image'),
    # save_format="png",
    seed=1
)

mask_generator = masks_dataGen.flow(
    train_maskedLabels,
    batch_size = 4,
    save_to_dir = os.path.join(base_folder, sub_folder[0], augImages, 'multiclass','mask'),
    # save_format="png",
    seed=1
)

train_generator = zip(image_generator, mask_generator)
```

شکل ۱۲

Image Generators for each class for binary segmentation:

```
[ ]  binary_masks_dataGen = ImageDataGenerator(**data_gen_args)
     binary_masks_dataGen.fit(train_maskedLabels, augment=True, seed=1)

     train_generators = []

     for i in range(0, len(class_labels)-1):

         bin_mask_generator = masks_dataGen.flow(
             masked_batches[i],
             batch_size = 4,
             save_to_dir = os.path.join(base_folder, sub_folder[0], augImages, 'binary', 'mask'),
             # save_format="png",
             seed=1
         )
         train_generators.append(zip(image_generator, bin_mask_generator))
```

شکل ۱۳

توجه شود که همانطور که در *شکل ۱۳* مشخص است به ازای هر کلاس، یک بچ از عکس های ماسک داریم که train_generator هر کدام از کلاس ها را با توجه به این نکته می سازیم و این train_generator ها در لیست train_generators قرار می گیرند. و هنگام استفاده از تابع fit برای ترین کردن مدل، train_generator کلاس مورد نظر را به آن تابع می دهیم.

توجه شود که برای Augment کردن دیتا از کتابخانه imgaug در پایتون نیز استفاده شده است که خروجی های مثال های آن در notebook موجود است ولی فقط جنبه آموزشی داشته و از آن ها برای train کردن مدل ها استفاده نشده است.

در ادامه به مدل ها و transfer learning می پردازیم.

مدل ها و transfer learning:

در این پروژه برای پیاده سازی شبکه U-net و انجام segmentation از کتابخانه آماده در github استفاده شده است که لینک آن https://github.com/qubvel/segmentation_models می باشد و از این لینک قابل دریافت است.

مدل های U-net برای binary و multiclass سگمنتیشن، برمبنای resnet32 می باشد و وزن های مدل از دیتاست imagenet گرفته شده اند که کمک بسیاری به segment کردن تصاویر کردند.همچنین برای fine-tuning مدل و آپدیت وزن های قبلی مدل قبل از compile مدل trainable مدل مورد نظر برابر True قرار

میگیرد تا هنگام train رو دیتاست مسئله ما وزن های مدل به روز شوند. این موارد برای حالت سگمنتشن multiclass در *شکل ۱۴*، *شکل ۱۵*، *شکل ۱۶* آمده است.

```
[ ]  model = sm.Unet('resnet34', classes=3, activation='softmax', encoder_weights='imagenet', encoder_freeze=True)
     model.summary()
```

شکل ۱۴

```
model.trainable = True     # we set this true for fine tuneing.
model.compile(optimizer=Adam(learning_rate=1e-5),
    loss=sm.losses.dice_loss,
    metrics=[sm.metrics.iou_score, sm.metrics.FScore()])
model.summary()
```

شکل ۱۵

```
# pre train decoder
model.fit(train_generator, epochs=2, steps_per_epoch=TRAIN_LENGTH // 4, batch_size=4)
set_trainable(model) # set all layers trainable and recompile model
```

```
[ ]  hsMulti = model.fit(train_generator, epochs = 120, steps_per_epoch=TRAIN_LENGTH // 4, batch_size=4)
     # model.fit(x=X, y=Y, epochs = 10, steps_per_epoch=len(X) // 4, batch_size=4)
```

شکل ۱۶

برای train مدل برای سنجش خطا از معیار dice استفاده شده است و برای accuracy از معیار iou و Fscore استفاده گردیده است.

توجه شود که ما در ابتدا لایه آخر مدل یا همان decoder را در حد ۲ epoch آموزش می دهیم یا pre-train می کنیم که در *شکل ۱۶* مشخص است و سپس کل مدل را train می کنیم.

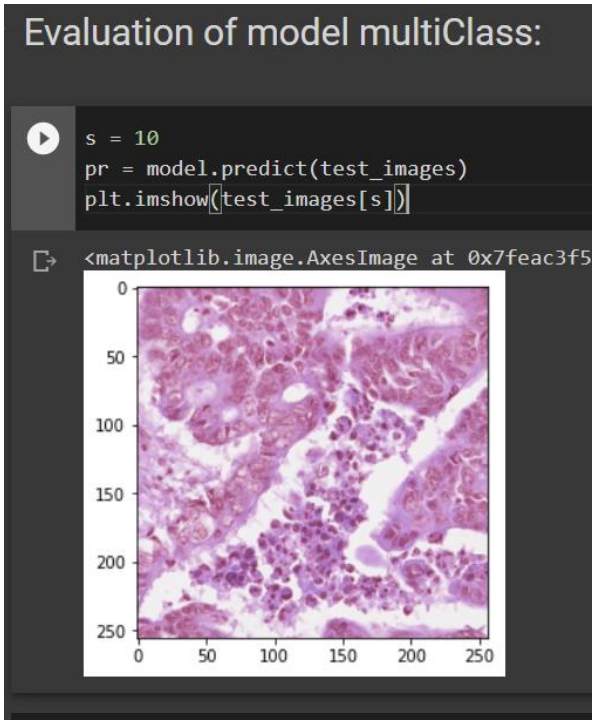
این نکته نیز حائز اهمیت است که به این دلیل که تعداد داده ها در این مسئله بسیار کم می باشد، تعداد batch ها در این مسئله ۴ عدد در نظر گرفته شده است.

و در نهایت پس فیکس کردن هایپر پارامتر ها مدل مالتی کلاس را train میکنیم.

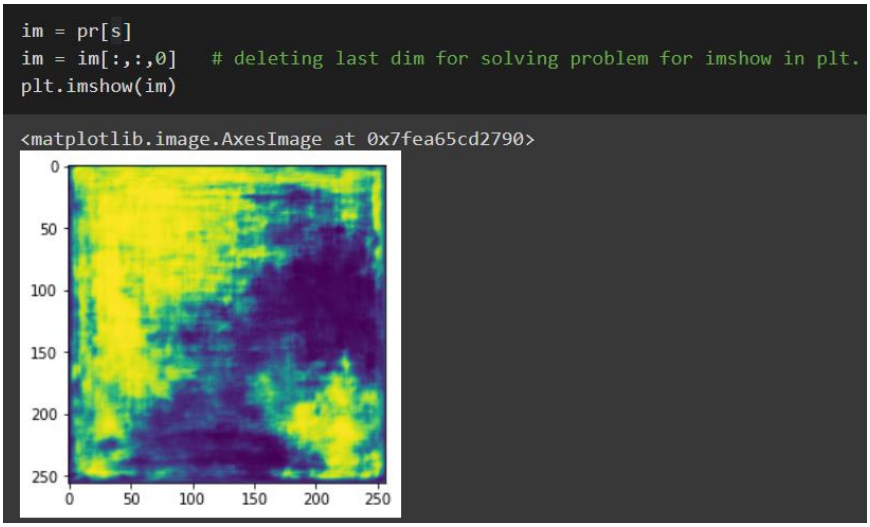
Score ها و loss بدست آمده برای این مدل بعد از train عبارتند از:

```
Epoch 120/120
6/6 [==============================] - 2s 323ms/step - loss: 0.8270 - iou_score: 0.1007 - f1-score: 0.1731
```

پس train شدن مدل مالتی کلاس به سراغ ارزیابی آن به کمک داده های تست میرویم(*شکل ۱۷*، *شکل ۱۸*، *شکل ۱۹*) :

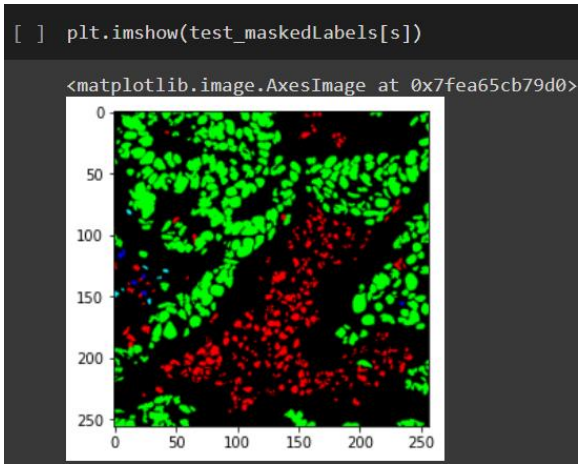


شکل ۱۷



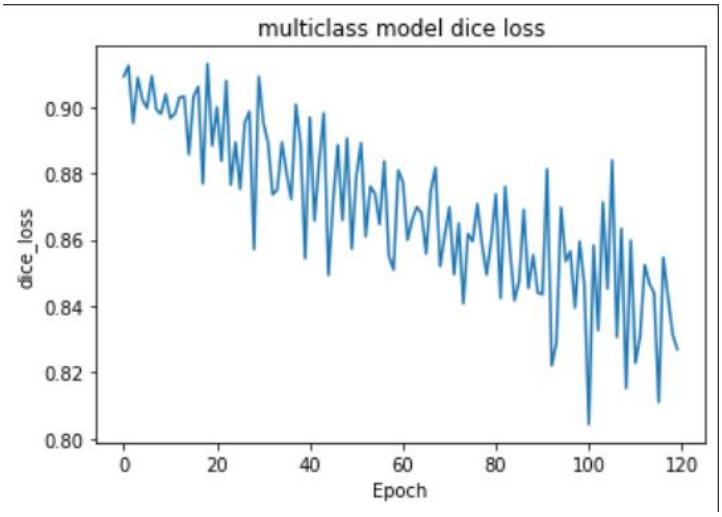
شکل ۱۸

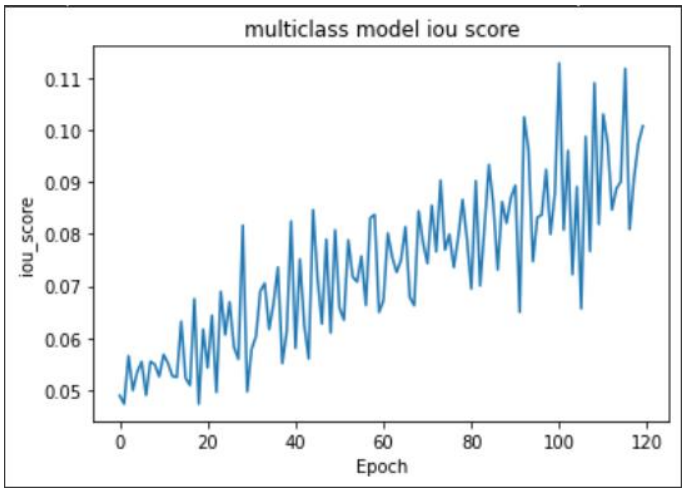
شکل ۱۸ در واقع خروجی شبکه U_net برای سمپل s ست test است. که مشاهده می شود که تا حدودی کلاس رنگ سبز در *شکل ۱۹* درست تشخیص داده شده اند.



شکل ۱۹

نمودار های training curve برای مدل مالتی کلاس(نمودار های زیر بر اساس اطلاعات train می باشند):





و در نهایت برای ارزیابی دقت تست خواهیم داشت:

```
a = model.evaluate(test_images, test_maskedLabels, verbose=2)
```

```
1/1 - 1s - loss: 0.9392 - iou_score: 0.0319 - f1-score: 0.0608
```

در ادامه نیز به سراغ binary segmentation می رویم که کلاس ۵ (fibroblast) را بررسی می کنیم:

در ادامه معماری مدل سگمنت کننده باینری آورده شده است:

```
model_class5 = sm.Unet('resnet34', classes=1, activation='sigmoid', encoder_weights='imagenet', encoder_freeze=True)
model_class5.summary()
```

```
model_class5.trainable = True   # for fine tuning
model_class5.compile(optimizer=Adam(learning_rate=1e-5),
    #loss=sm.losses.binary_focal_loss,
    loss = sm.losses.dice_loss,
    metrics=[sm.metrics.iou_score, sm.metrics.FScore(), 'binary_accuracy'])
model_class5.summary()
```

```
# pretrain model decoder
class_label = 4
model_class5.fit(train_generators[class_label], epochs=3, steps_per_epoch=TRAIN_LENGTH // 4, batch_size=4)
```

```
Epoch 1/3
6/6 [==============================] - 39s 277ms/step - loss: 0.9508 - iou_score: 0.0256 - f1-score: 0.0492 - binary_accuracy: 0.5665
Epoch 2/3
6/6 [==============================] - 4s 278ms/step - loss: 0.9623 - iou_score: 0.0193 - f1-score: 0.0370 - binary_accuracy: 0.5691
Epoch 3/3
6/6 [==============================] - 2s 277ms/step - loss: 0.9437 - iou_score: 0.0304 - f1-score: 0.0576 - binary_accuracy: 0.5525
<keras.callbacks.History at 0x7faabd2407d0>
```

```
hs5 = model_class5.fit(train_generators[class_label], epochs = 200, steps_per_epoch=TRAIN_LENGTH // 4, batch_size=4)
```

Score ها و loss بدست آمده برای این مدل بعد از train عبارتند از:

```
Epoch 200/200
6/6 [==============================] - 2s 273ms/step - loss: 0.8957 - iou_score: 0.0564 - f1-score: 0.1032 - binary_accuracy: 0.7392
```

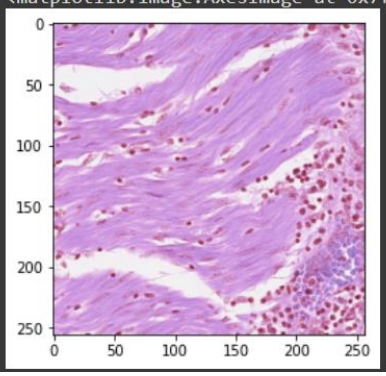
بعد از train مدل باینری برای تشخیص کلاس۵ (fibroblast) سراغ ارزیابی آن به کمک داده های تست میرویم:

به ترتیب سمپل ورودی، خروجی مدل برای کلاس شماره ۵ و لیبل اصلی(شکل ۲۰، شکل ۲۱، شکل ۲۲)

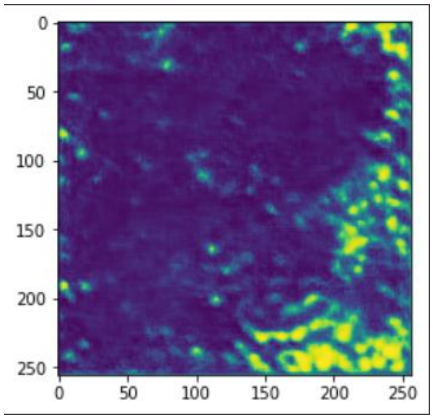
Evaluation on test data for class 5 model:

```
[ ]  s = 1
     pr = model_class5.predict(test_images)
     plt.imshow(test_images[s])
```

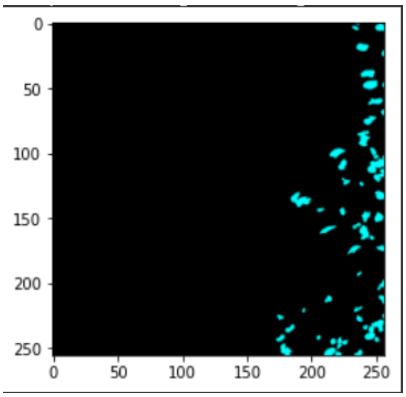
```
<matplotlib.image.AxesImage at 0x7faa73d03ed0>
```



شکل ۲۰

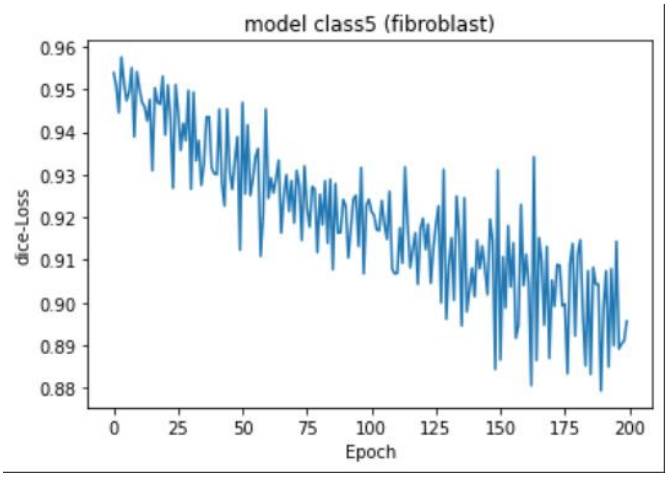


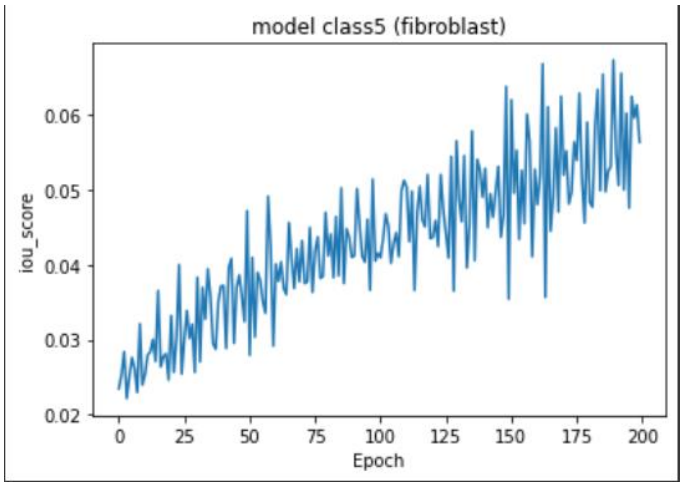
شکل ۲۱

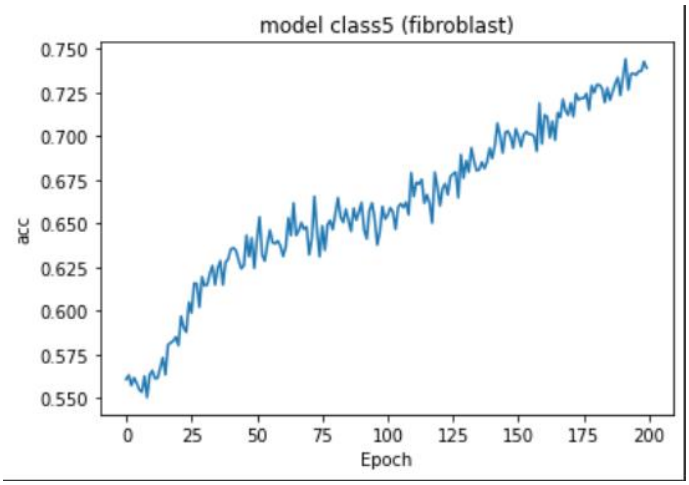


شکل ۲۲

نمودار های training curve برای مدل مالتی کلاس(نمودار های زیر بر اساس اطلاعات train می باشند):







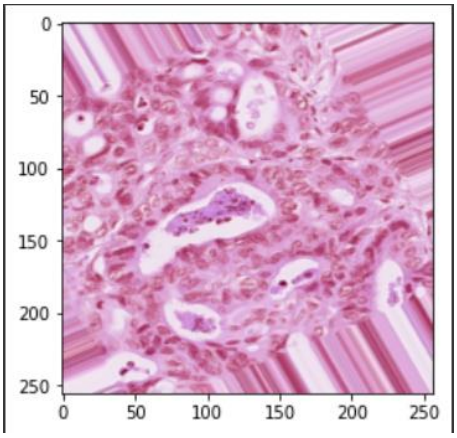
و در نهایت برای ارزیابی دقت تست خواهیم داشت:

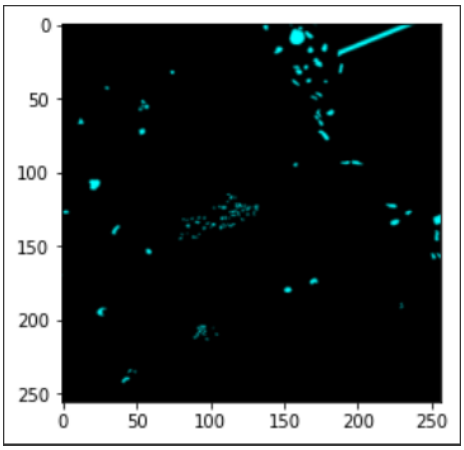
```
[ ]  a = model_class5.evaluate(test_images, test_maskedLabels, verbose=2)

     1/1 - 1s - loss: 0.8596 - iou_score: 0.0769 - f1-score: 0.1404 - binary_accuracy: 0.8042
```

در این تمرین برای کلاس های ۱ یعنی (other) و کلاس ۶ (muscle) نیز مدل train شده است که اطلاعات بیشتر آن ها در notebook می باشد.

به دلیل محدودیت GPU دیگر برای کلاس های دیگر مدل نشد.

در ادامه نیز تعدادی عکس augment شده آورده شده است که با ImageDataGenerator داخل کراس تولید شده اند و در فرایند آموزش مدل ها استفاده شده اند:





- ✓ **جزئیات بیشتر در داخل فایل notebook موجود می باشند.**
- ✓ **لینک notebook:**

https://colab.research.google.com/drive/1qDOp8eJy5tMK9VG5m_g6t0PJyDUZh52C?usp=sharing

- ✓ **فایل ارسالی شامل notebook با اسم**

HoVer_Net_semantic_segmentation_AlirezaRashidi و فایل pdf گزارش می باشد.