به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین سوم

نام و نام خانوادگی	آرمان فروزش – مهسا ندافی قهنویه
شماره دانشجویی	810100490 - 111946
تاریخ ارسال گزارش	1401.09.18

فهرست

شکل‌ها

جدول‌ها

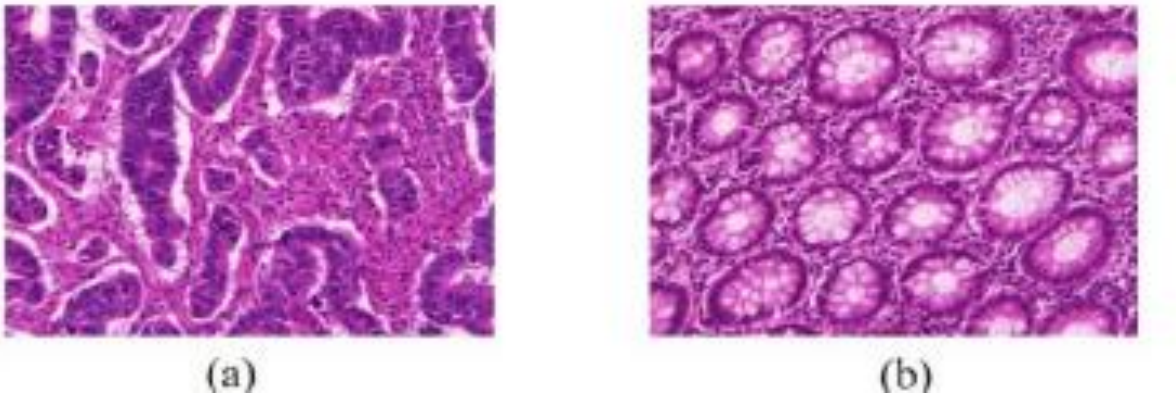
پاسخ 1. آشنایی با یادگیری انتقالی Transfer Learning

۱-۱. گزارش مقاله ResNet

در این مقاله برای تشخیص سرطان کولورکتال از شبکه ی ResNet18 , ResNet50 استفاده شده است بدین منظور از دیتاست Warwick QU gland segmentation (GlaS) استفاده شده است این دیتاست شامل 165 داده بصورت عکس هایی با پسوند bmp. و یک فایل csv. می باشد عکس ها بصورت نرمال و همچنین با پسوند anno_ ذخیره شده اند که در مجموع 330 عکس را در اختیار ما قرار میدهد.

در این مقاله از عکس های معمولی و ستون grade (GlaS) ویژگی ها برای دسته بندی داده ها در 2 کلاس سالم و بیمار (benign and malignant) استفاده شده است.

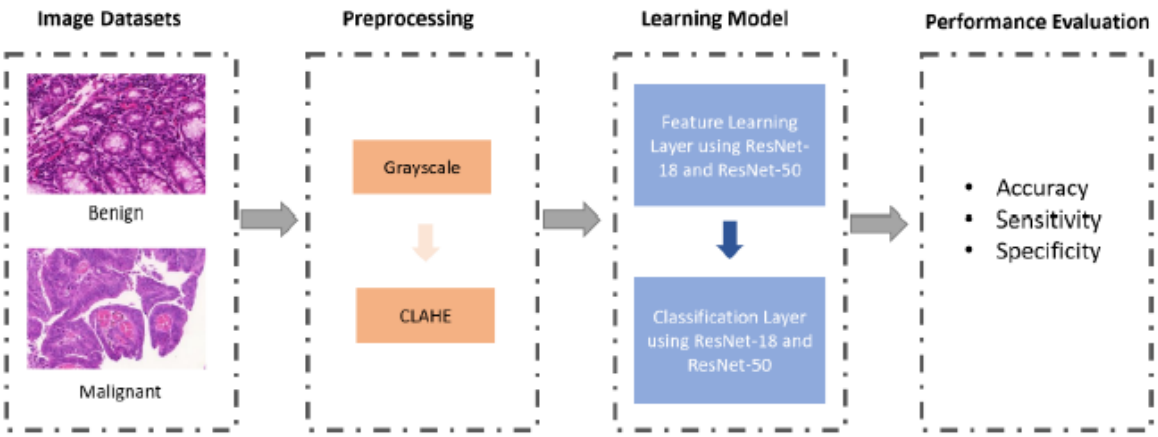
عکس های این دیتاست در شکل زیر دیده میشود:



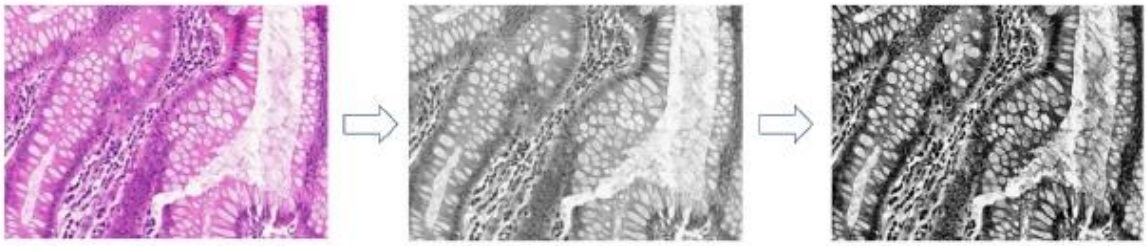
عکس 1-1: **a)Malignant b)Benign**

پس از لود دیتاست پیش پردازش داده ها ابتدا با تغییر ابعاد آنها برای تغییر به سایز مناسب برای ورود به شبکه و سپس با تغییر تصویر rgb به تصویر grayscale ادامه یافته است در نهایت چون تصویر نهایی کنتراست خوبی نداشته است از CLAHE استفاده شده است.

پس از پیش پردازش مدل های گفته شده در سه حالت 80 درصد داده ترین و 20 درصد داده ی تست ، 75 درصد داده ترین و 25 درصد داده ی تست و 60 درصد داده ترین و 40 درصد داده ی تست آزمایش شده و معیار های Accuracy,Sensitivity و Specificity برای آنها گزارش شده است.

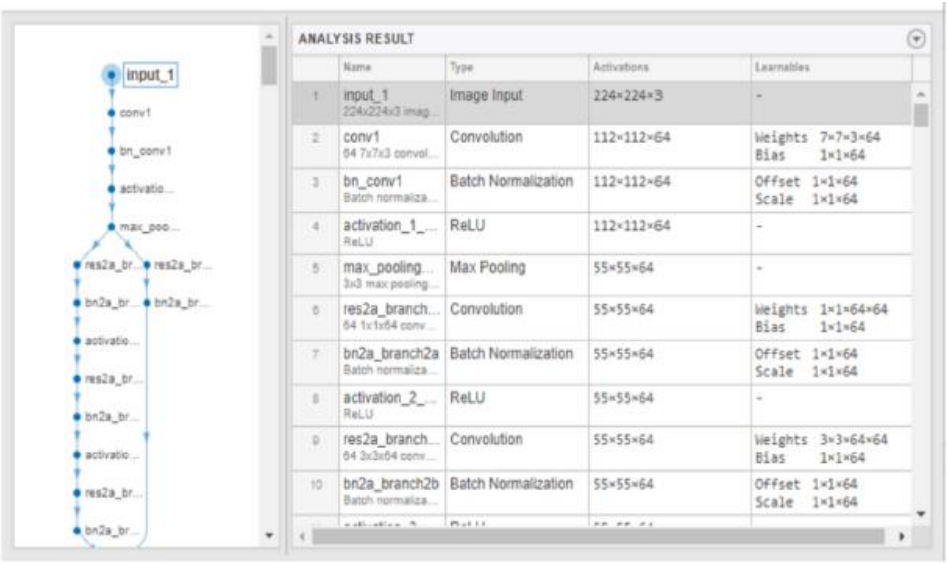


شکل 1-2: الگوریتم مورد استفاده در مقاله



شکل 1-3: تصاویر پیش پردازش شده

نتایج پیش پردازش را میتوان با نتایج شبیه سازی شده در بخش چهارم سوال مقایسه کرد.



شکل 1-4: معماری **ResNet50**

جدول 1-1: نتایج استفاده از **ResNet50**

Training Data: Testing Data	*Accuracy*	*Sensitivity*	*Specificity*
60%:40%	77%	60%	92%
75%:25%	88%	89	87%
80%:20%	88%	93%	83%

جدول 1-2: نتایج استفاده از **ResNet18**

Table 1. Comparison of Accuracy, Sensitivity, Specificity Value in Several Testing Datasets for ResNet-18

Training Data: Testing Data	*Accuracy*	*Sensitivity*	*Specificity*
60%:40%	73%	64%	83%
75%:25%	81%	96%/	63%
80%:20%	85%	83%	87%

هایپر پارامترهای استفاده شده شامل SGD برای optimizer که از کاهش گرادیان استفاده میکند و binary cross-entropy برای loss function می باشد.

با توجه به نتایج مقاله ResNet50 از ResNet18 بهتر عمل میکند و با تعداد بیشتر داده های ترین میتوان نتیجه ی قابل قبول تری بگیریم. با مدل های پیشنهاد شده میتوان از 73 تا 83 درصد دقت برای تشخیص سرطان کولن بگیریم.

2-1.شبکه ی ResNet

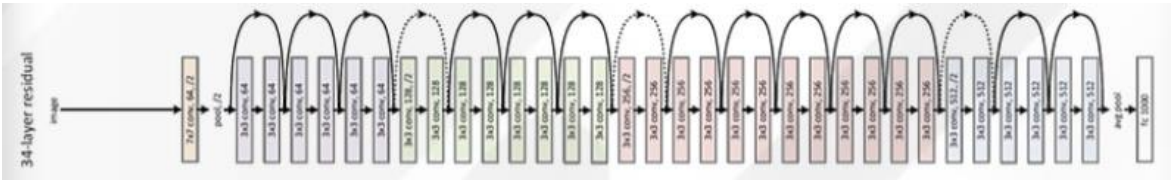
- معماری شبکه

شبکه‌یResNet این است که به ما امکان آموزش شبکه‌های عصبی بسیار عمیق با بیش از ۱۵۰ لایه را داد. قبل این شبکه شبکه‌های عصبی مانند ZFNet,AlexNet و VGGNet بسیار عمیق، به‌دلیل مشکل محوشدگی گرادیان (Vanishing Gradient)، دچار مشکل می‌شدند. اتصالات اضافی (Residual Connections) راه‌حلی بود که شبکه ResNet برای حل مشکل شبکه‌های عمیق ارائه کرد.

این شبکه از جمله شبکه هایی می باشد که برای شناسایی تصاویر و شناسایی الگو استفاده میشود.

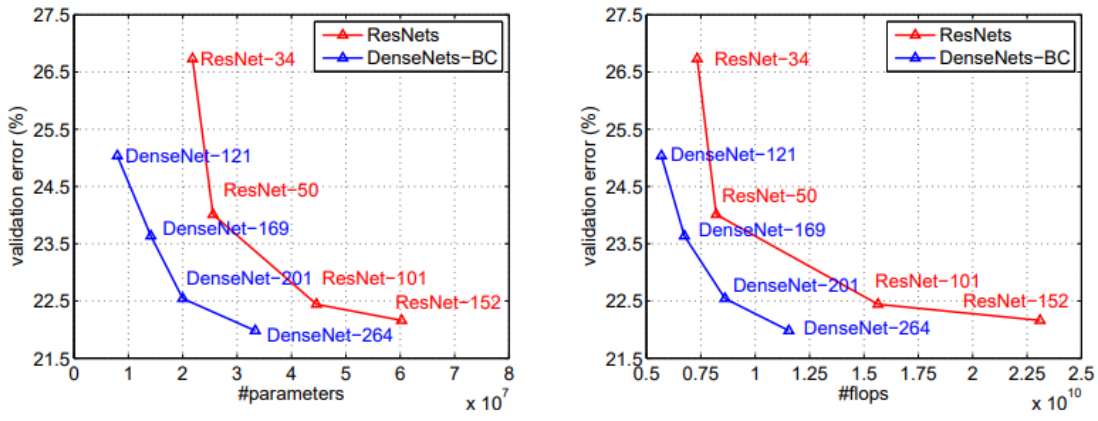
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

شکل 1-5:معماری انواع **Resnet**



شکل 1-6:نحوه ی اتصال لایه ها در شبکه ی **ResNet**

پ معرفی مسیرهای اتصال کوتاه در شبکه به خصوص معرفی شبکه resnet باعث شد تا بتوان عمق شبکه های عصبی را تا حدود زیادی افزایش داد اما این کار منجر به افزایش تعداد پارامترهای شبکه میشود که روند بهینه سازی را کند میکند . اما در شبکه های dense-net استفاده از داده های چندین مرحله قبل به عنوان ورودی در لایه ی های بعدی و همچنین concatenate کردن آنها به جای استفاده از جمع باعث data flow بهتر این شبکه نسبت به resnet میشود و باعث می شود که بتواند با تعداد پارامتری های کمتری عملکرد بهتری را از خود نشان دهد.

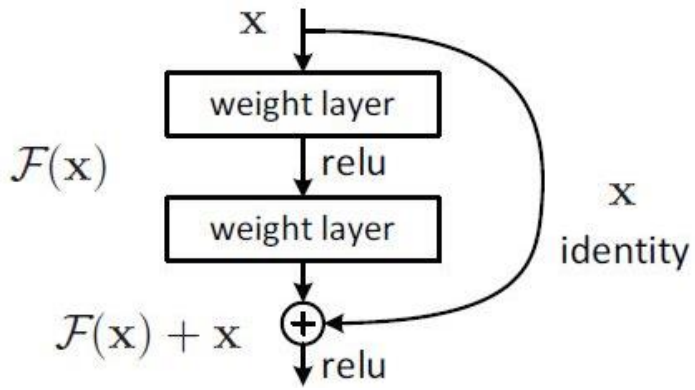


شکل 1-7: مقایسه **resnet , densenets**

- پیش پردازش داده

همانطور که مشخص است ورودی داده های این شبکه (224,224,3) می باشد که با cv2.resize میتوان آنها را ریسایز کرد یا یک لایه ی resizing در ابتدای شبکه استفاده کرد همچنین خروجی باید به شکل one-hot باشد.

همچنین نوع استفاده ی دوباره از داده ها در شکل زیر دیده میشود:



شکل 1-8:نوع ورودی داده ها به لایه ی جدید

1-3. شبکه انتخابی قابلیت تشخیص چه نوع عکس هایی را دارد. اگر عکسی داخل آن دسته نباشد چه-میشود؟ راه حل چیست؟

شبکه های از پیش آموزش داده شده بر روی تصاویر ImageNet آموزش داده شده و کلاس بندی بر روی آنها انجام شده است این مجموعه داده شامل 1281167 تصویر برای آموزش و 5000 تصویر برای ولیدیشن که در 1000 کلاس قرار دارند. پس اگر داده ها برای آموزش دوباره به کلاس بندی داده های ImageNet نزدیک تر باشند شبکه قابلیت تشخبص را به خوبی خواهد داشت.

این داده ها باید دارای سایز (224,224,3) در ورودی خود و rgb باشند که میتوان با افزودن لایه های ورودی به نتیجه دلخواه رسید.

حال اگر عکس داده شده در کلاس بندی داده ها نباشد میتوان با قرار دادن چند لایه پس از شبکه ش ResNet به نتیجه ی مطلوب رسید که داده های دیتاست مقاله در کلاس بندی دیتای ImageNet قرار ندارند.

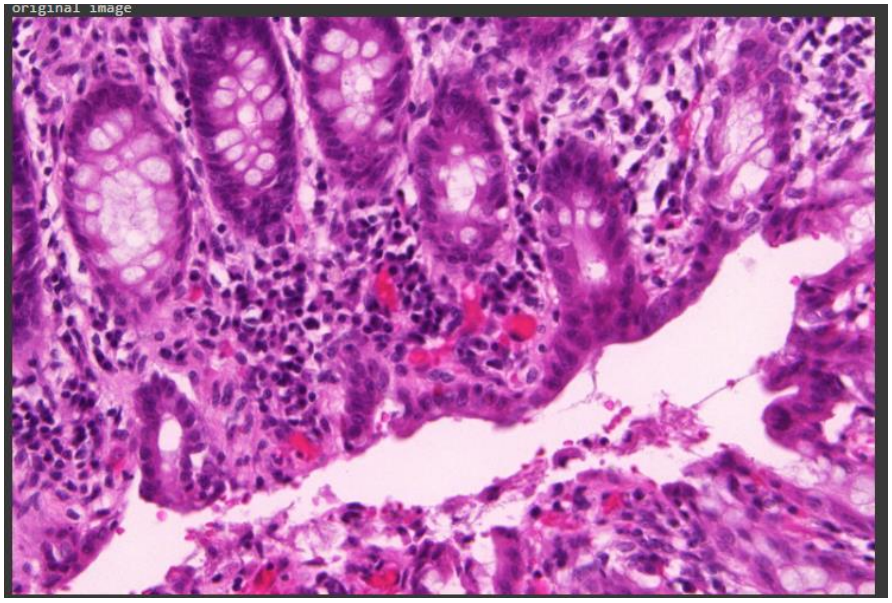
1-4 . دیتاست

این دیتاست شامل 165 داده بصورت عکس هایی با پسوند bmp. و یک فایل csv. می باشد عکس ها بصورت نرمال و همچنین با پسوند anno_ ذخیره شده اند که در مجموع 330 عکس را در اختیار ما قرار میدهد ابعاد این عکس ها تا 775*522 پیکسل هستند.

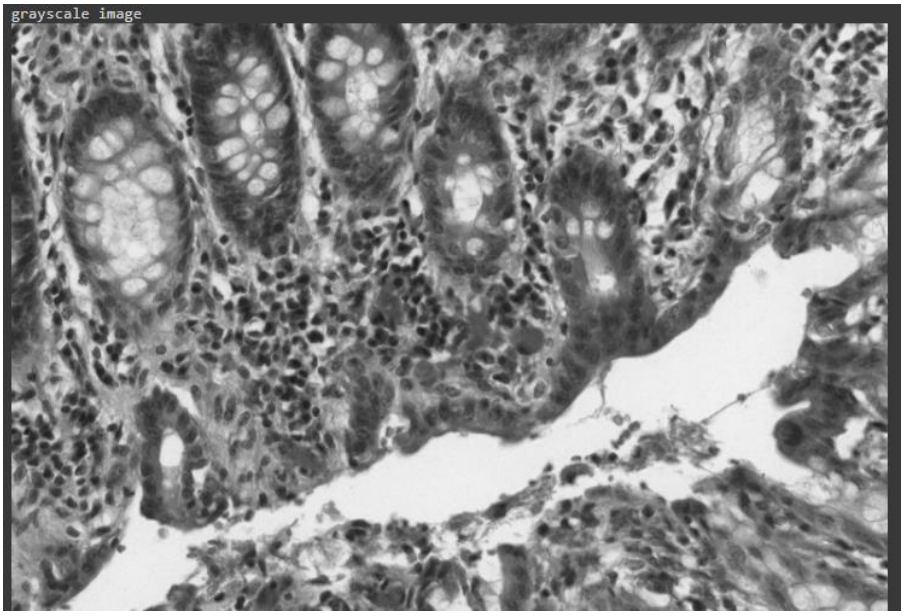
در این بخش دیتاست را آپلود کرده و اطلاعات مهم همچنین اینکه در هر ستون دیتاست چند داده ی Nan داریم در زیر مشاهده میشود:

```
 dataset information:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 165 entries, 0 to 164
Data columns (total 4 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   name                                    165 non-null    object
 1   patient ID                              165 non-null    int64
 2    grade (GlaS)                           165 non-null    object
 3    grade (Sirinukunwattana et al. 2015)   165 non-null    object
dtypes: int64(1), object(3)
memory usage: 5.3+ KB

 sum of Nan data:

name                                    0
patient ID                              0
 grade (GlaS)                           0
 grade (Sirinukunwattana et al. 2015)   0
dtype: int64
```

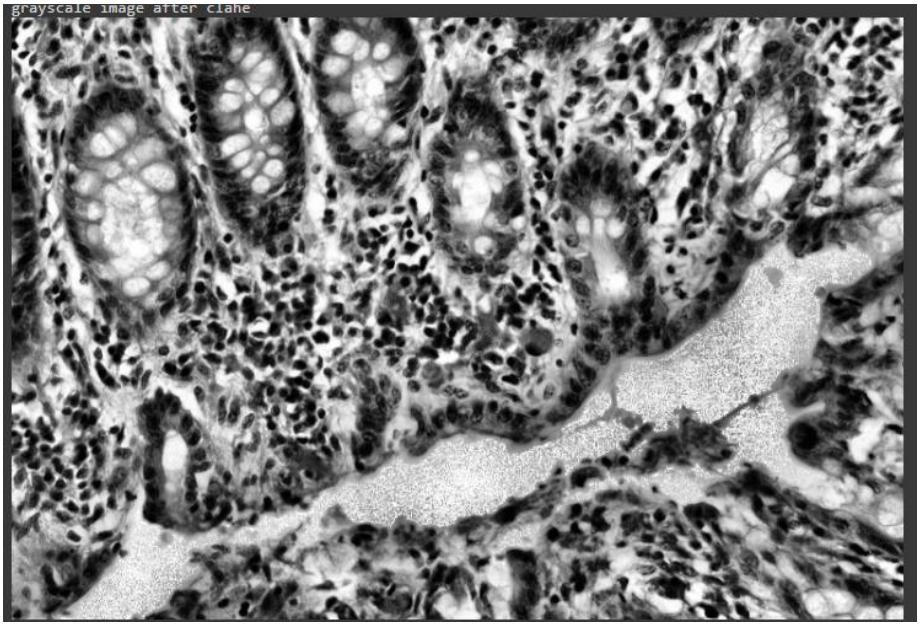
یک نمونه از تصاویر این دیتاست در شکل زیر دیده میشود:



شکل 1-9: نمونه ای از دیتاست



شکل 1-10:تصویر **grayscale**



شکل 1-11: تصویر بعد از اعمال **CLAHE**

به منظور کلاس بندی داده ها فقط با تصاویر معمولی کار میکنیم و تصاویر با پسوند anno. را حذف میکنیم سپس داده ها را به سایز 224*224 برای ورود به شبکه تغییر شکل میدهیم طبق مقاله ابتدا داده ها را grayscale کرده و چون طبق شکل 1-10 کیفیت و کنتراست مورد انتظار را ندارد از CLAHE استفاده میکنیم که شکل 1-11 را نتیجه میدهد.

کد پیش پردازش داده ها:

```
#delete anno.bmp images
imgs = []
for i in range (330):
  path=images[i].split("/",7)
  anno = path[7].split('_',2)
  if len(anno) < 3:
    imgs.append(images[i])

x = []
for i in range(165):
    temp = tf.keras.preprocessing.image.load_img(
        path = imgs[i])
    img = np.array(temp.convert('RGB'))
    img = cv2.resize(img,(224,224))
    gryimg = np.asarray(cv2.cvtColor(np.array(img), cv2.COLOR_BGR2GRA
Y))
    # The declaration of CLAHE
    # clipLimit -> Threshold for contrast limiting
    clahe = cv2.createCLAHE(clipLimit = 20)
    finalimg = np.asarray(clahe.apply(gryimg))
```

```
    finalimg = cv2.cvtColor(finalimg, cv2.COLOR_GRAY2BGR)#normalize a
nd make 3d image
    x.append(finalimg)
x = np.array(x)
```

به منظور کلاس بندی داده ها از ستون Grade (GlaS) استفاده میشود و کلاس بندی باینری صورت میگیرد.

```
y = []
for i in range(165):
  path = imgs[i].split("/",7)
  name = path[7].split('.')
  for j in range(165):
    if df['name'][j] == name[0] :
      if df[' grade (GlaS)'][j] == ' benign':
        y.append(1)
      if df[' grade (GlaS)'][j] == ' malignant':
        y.append(0)
y = np.array(y)
```

دقت شود که تمام دیتای تست و ترین دیتاست با یکدیگر مخلوط شد و در نهایت از کد زیر برای تقسیم تمام 165 داده استفاده شد:

```
from sklearn.model_selection import train_test_split
#y = to_categorical(y)
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0
.2, random_state=10)
# Checkout the Data
```

```
training data shape :  (132, 224, 224, 3) (132,)
Testing data shape :  (33, 224, 224, 3) (33,)
```

5-1.شبیه سازی ResNet50

در این بخش از ResNet50 استفاده میکنیم و جون این شبکه با دیتاست imagenet از پیش آموزش داده شده است(weights='imagenet') پس نیاز است که برای گرفتن نتیجه مطلوب تر خودمان تعدادی لایه بعد از آن استفاده کنیم که include_top=False این مورد را محقق می سازد.

```
#========================== buliding the model ======================
==========
input_shape_resnet = (224, 224, 3)
resenet_model = ResNet50(weights='imagenet',
                         input_shape=input_shape_resnet,
```

```
                            include_top=False)
#========================== unfreeze last 3 conv layers =============
===========
for layer in resenet_model.layers[0 : -11]:
    layer.trainable = False
resenet_model.summary()
#========================== adding layers ===========================
===========
inp_layer = Input(shape = (224, 224, 3))
features = resenet_model(inp_layer)
flat = Flatten()(features)
N = keras.layers.BatchNormalization()(flat)
FC1 = Dense(224, activation = 'relu')(N)
N = keras.layers.BatchNormalization()(FC1)
drop = Dropout(0.25)(N)
FC3 = Dense(100, activation = 'relu')(drop)
output = Dense(1, activation = 'relu')(FC3)
model = Model(inputs = inp_layer, outputs = output)
model.summary()
```

خروجی شبکه بصورت باینری 0 و 1 برای دو کلاس بیمار و سالم می باشد.

بخشی از اطلاعات شبکه ResNet50 و تعداد پارامترهای آن در زیر دیده میشود:

```
Model: "resnet50"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_38 (InputLayer)          [(None, 224, 224, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 230, 230, 3)  0           ['input_38[0][0]']

 conv1_conv (Conv2D)            (None, 112, 112, 64  9472        ['conv1_pad[0][0]']
                                )

 conv1_bn (BatchNormalization)  (None, 112, 112, 64  256         ['conv1_conv[0][0]']
                                )

 conv1_relu (Activation)        (None, 112, 112, 64  0           ['conv1_bn[0][0]']
                                )

 pool1_pad (ZeroPadding2D)      (None, 114, 114, 64  0           ['conv1_relu[0][0]']
                                )

 pool1_pool (MaxPooling2D)      (None, 56, 56, 64)   0           ['pool1_pad[0][0]']

 conv2_block1_1_conv (Conv2D)   (None, 56, 56, 64)   4160        ['pool1_pool[0][0]']

 conv2_block1_1_bn (BatchNormal  (None, 56, 56, 64)  256         ['conv2_block1_1_conv[0][0]']
 ization)

 conv2_block1_1_relu (Activatio  (None, 56, 56, 64)  0           ['conv2_block1_1_bn[0][0]']
 n)

 conv2_block1_2_conv (Conv2D)   (None, 56, 56, 64)   36928       ['conv2_block1_1_relu[0][0]']

 conv2_block1_2_bn (BatchNormal  (None, 56, 56, 64)  256         ['conv2_block1_2_conv[0][0]']
```

```
conv5_block3_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624     ['conv5_block3_2_relu[0][0]']

conv5_block3_3_bn (BatchNormal  (None, 7, 7, 2048)  8192        ['conv5_block3_3_conv[0][0]']
ization)

conv5_block3_add (Add)         (None, 7, 7, 2048)   0           ['conv5_block2_out[0][0]',
                                                                 'conv5_block3_3_bn[0][0]']

conv5_block3_out (Activation)  (None, 7, 7, 2048)   0           ['conv5_block3_add[0][0]']

==================================================================================================
Total params: 23,587,712
Trainable params: 4,465,664
Non-trainable params: 19,122,048
```

لایه های اضافه شده به شبکه و پارامترهای آن:

```
Model: "model_22"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_39 (InputLayer)       [(None, 224, 224, 3)]     0

 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 flatten_22 (Flatten)        (None, 100352)            0

 batch_normalization_27 (Bat  (None, 100352)           401408
 chNormalization)

 dense_66 (Dense)            (None, 224)               22479072

 batch_normalization_28 (Bat  (None, 224)              896
 chNormalization)

 dropout_22 (Dropout)        (None, 224)               0

 dense_67 (Dense)            (None, 100)               22500

 dense_68 (Dense)            (None, 1)                 101

=================================================================
Total params: 46,491,689
Trainable params: 27,168,489
Non-trainable params: 19,323,200
_________________________________________________________________
```

همانطور که مشاهده میشود 19323200 پرارامتر قابل آموزش در شبکه موجود است.

در نهایت شبکه را با استفاده از دیتای پیش پردازش شده در 3 حالت آموزش میدهیم:

پارامترهای مدل همانطور که در مقاله گفته شده است به شکل زیر انتخاب میشود:

```
loss='binary_crossentropy', optimizer=tf.optimizers.SGD(learning_rate
=0.00001), metrics=['accuracy']
```

برای نمایش نتایج که نمودار loss , Accuracy در حین آموزش و ماتریس طبقه بندی همچنین معیار های precision , f1-score روی داده های تست می باشد را با استفاده از کدهای زیر نمایش میدهیم:

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP}$$

```
rain_loss = fitting.history['loss']
val_loss = fitting.history['val_loss']
train_acc = fitting.history['accuracy']
val_acc = fitting.history['val_accuracy']
plt.subplots(figsize=(20, 10))

plt.subplot(2, 1, 1)
plt.semilogy(train_loss)
plt.semilogy(val_loss)

plt.legend(['training loss', 'validation loss'])
plt.xlabel('epochs')
plt.ylabel('loss')
plt.grid()
plt.subplot(2, 1, 2)
plt.plot(train_acc)
plt.plot(val_acc)

plt.legend(['training accuracy', 'validation accuracy'])
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.grid()
plt.show()
```

```
y_pred = model.predict(X_test)
thresh = 0.5
y_pred[y_pred >= thresh] = 1
y_pred[y_pred < thresh]  = 0
cls_report = classification_report(y_test, y_pred)

print('\n \n')
print(cls_report)
print(ConfusionMatrixDisplay.from_predictions(y_test, y_pred))
```

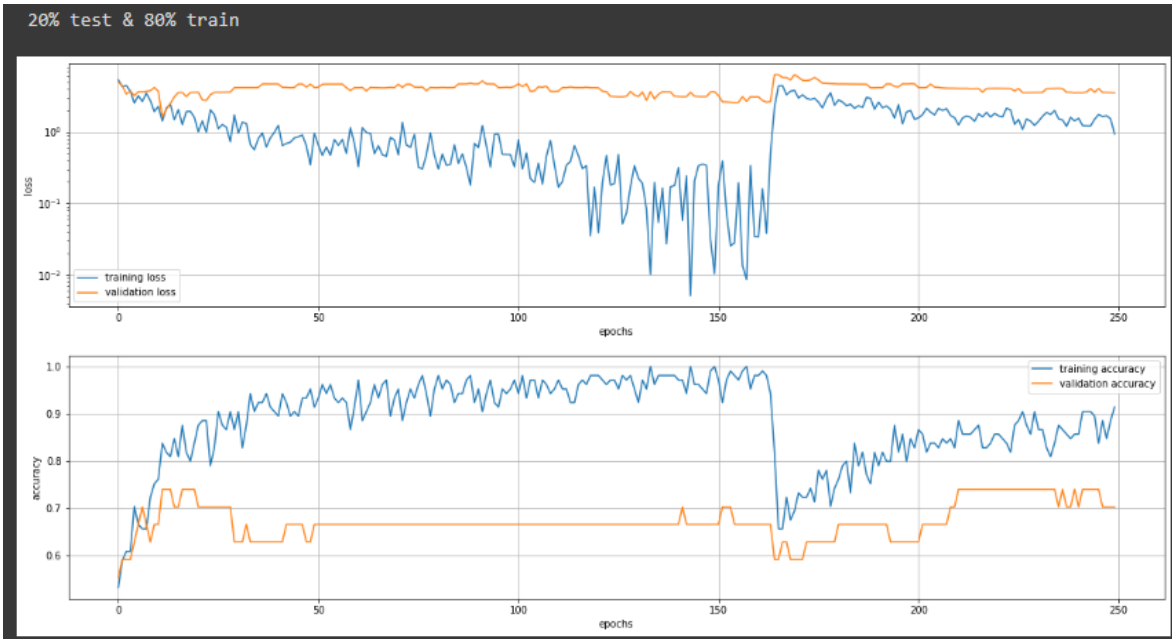
- **آموزش با 80٪ دیتای آموزشی و 20٪ دیتای تست**

Train model with 80%train & 20%test data

```
[ ]  model.compile(loss='binary_crossentropy', optimizer=tf.optimizers.SGD(learning_rate=0.00001), metrics=['accuracy'])
     fitting = model.fit(X_train, y_train,
                         batch_size=8, epochs=250, validation_split=0.2) #it's 250 epoch

     Epoch 1/250
     14/14 [==============================] - 5s 135ms/step - loss: 5.3014 - accuracy: 0.5333 - val_loss: 4.9375 - val_accuracy: 0.5556
     Epoch 2/250
     14/14 [==============================] - 1s 48ms/step - loss: 4.2692 - accuracy: 0.5905 - val_loss: 4.3980 - val_accuracy: 0.5926
     Epoch 3/250
     14/14 [==============================] - 1s 48ms/step - loss: 4.4253 - accuracy: 0.6095 - val_loss: 3.3702 - val_accuracy: 0.5926
     Epoch 4/250
     14/14 [==============================] - 1s 48ms/step - loss: 3.7008 - accuracy: 0.6095 - val_loss: 3.7037 - val_accuracy: 0.5926
     Epoch 5/250
     14/14 [==============================] - 1s 51ms/step - loss: 2.5175 - accuracy: 0.7048 - val_loss: 3.1858 - val_accuracy: 0.6296
     Epoch 6/250
     14/14 [==============================] - 1s 48ms/step - loss: 3.1909 - accuracy: 0.6667 - val_loss: 3.6214 - val_accuracy: 0.6667
     Epoch 7/250
     14/14 [==============================] - 1s 48ms/step - loss: 2.6501 - accuracy: 0.6571 - val_loss: 3.5998 - val_accuracy: 0.7037
     Epoch 8/250
     14/14 [==============================] - 1s 47ms/step - loss: 3.4470 - accuracy: 0.6571 - val_loss: 3.6706 - val_accuracy: 0.6667
```

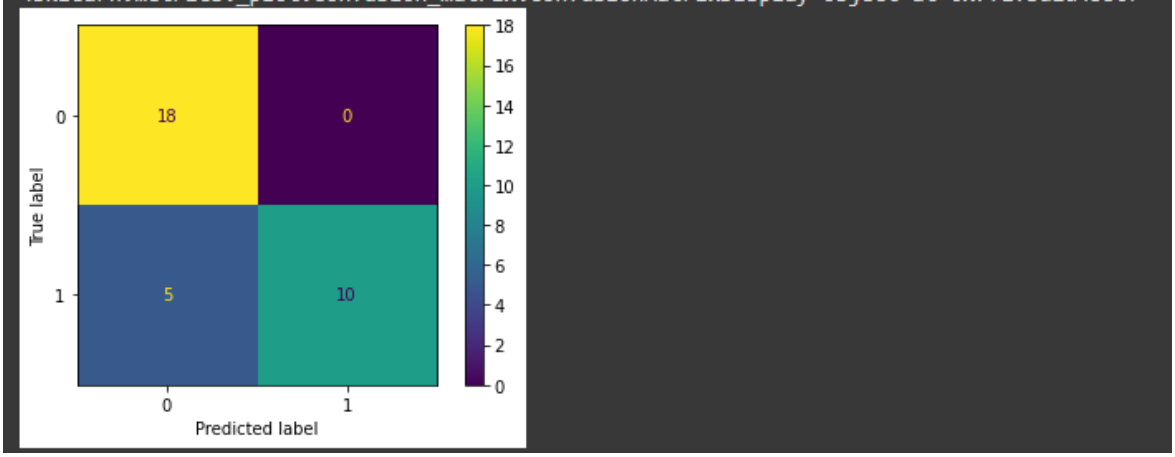
نتایج این مدل:

- **با 250 ایپاک**



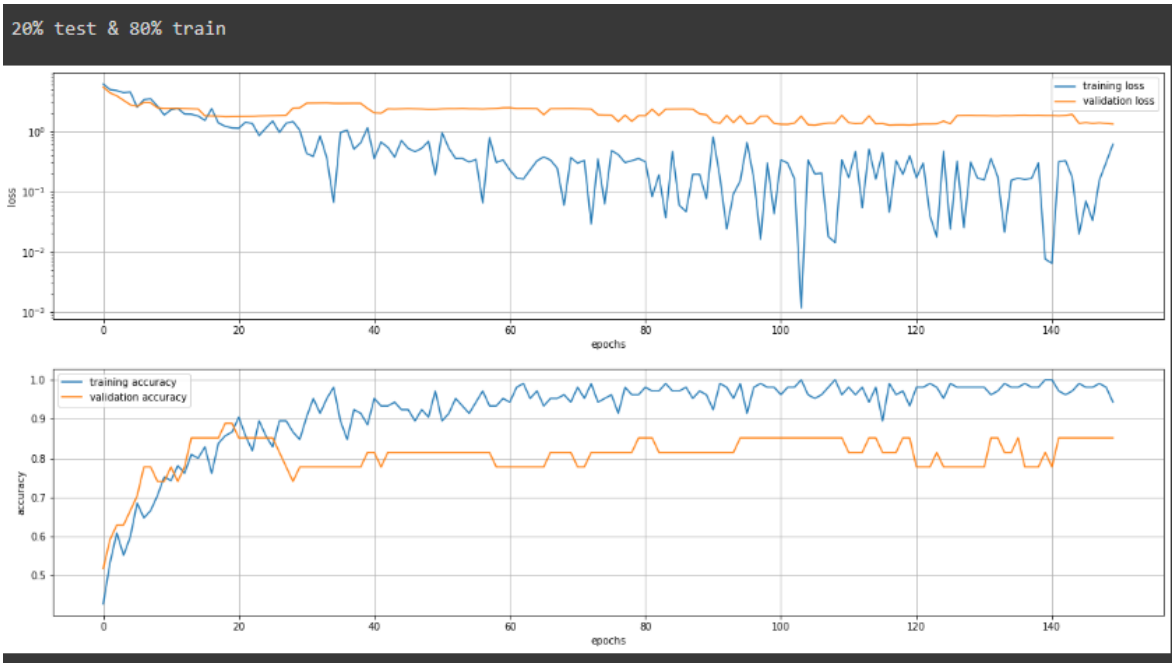
شکل 1-12:نمودار **accuracy , loss** برای حالت 80-20 با 250 ایپاک

```
20% test & 80% train

              precision    recall  f1-score   support

           0       0.78      1.00      0.88        18
           1       1.00      0.67      0.80        15

    accuracy                           0.85        33
   macro avg       0.89      0.83      0.84        33
weighted avg       0.88      0.85      0.84        33

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f1fbd1d4550>
```



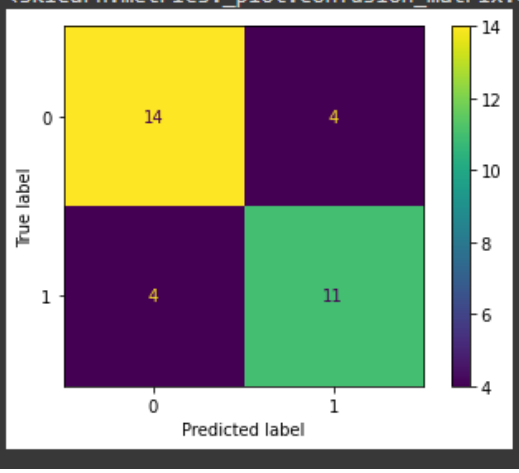
شکل 1-13: نتایج تست حالت 80-20 با 250 ایپاک

- **با 150 ایپاک**



شکل 1-14:نمودار **accuracy , loss** برای حالت 80-20 با 150 ایپاک

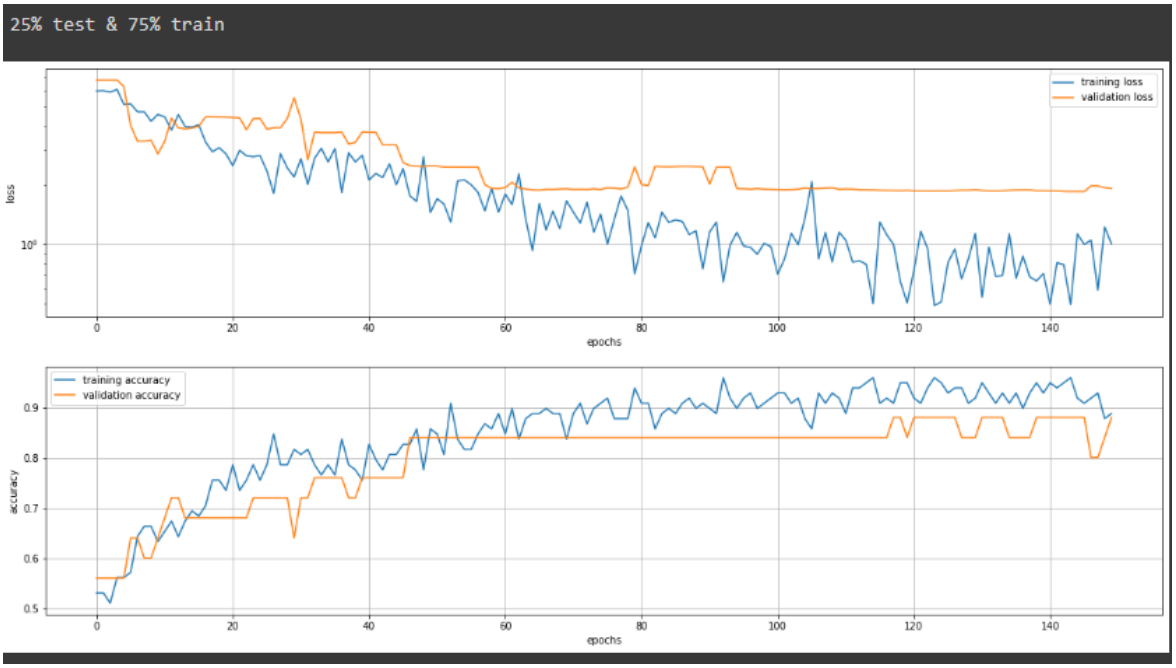
```
20% test & 80% train
              precision    recall  f1-score   support

           0       0.78      0.78      0.78        18
           1       0.73      0.73      0.73        15

    accuracy                           0.76        33
   macro avg       0.76      0.76      0.76        33
weighted avg       0.76      0.76      0.76        33

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f29cdd1cfa0>
```



شکل 1-15: نتایج تست حالت 80-20 با 150 ایپاک

همانطور که مشخص است در این بخش با 250 ایپاک به نتایج مطلوبی دست یافته ایم.

- **آموزش با 75٪ دیتای آموزشی و 25٪ دیتای تست**

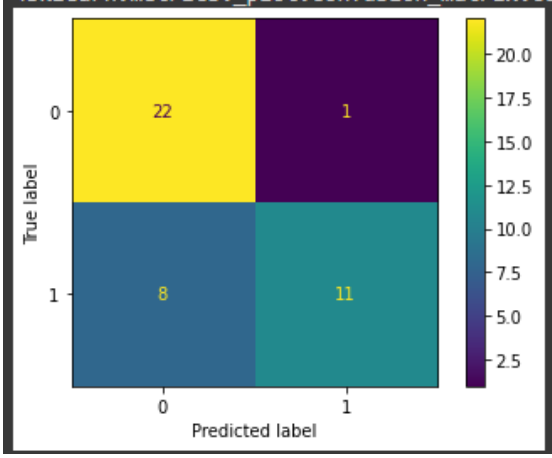


شکل 1-16:نمودار **accuracy , loss** برای حالت 75-25 با 150 ایپاک

25% test & 75% train

	precision	recall	f1-score	support
0	0.73	0.96	0.83	23
1	0.92	0.58	0.71	19
accuracy			0.79	42
macro avg	0.82	0.77	0.77	42
weighted avg	0.82	0.79	0.78	42

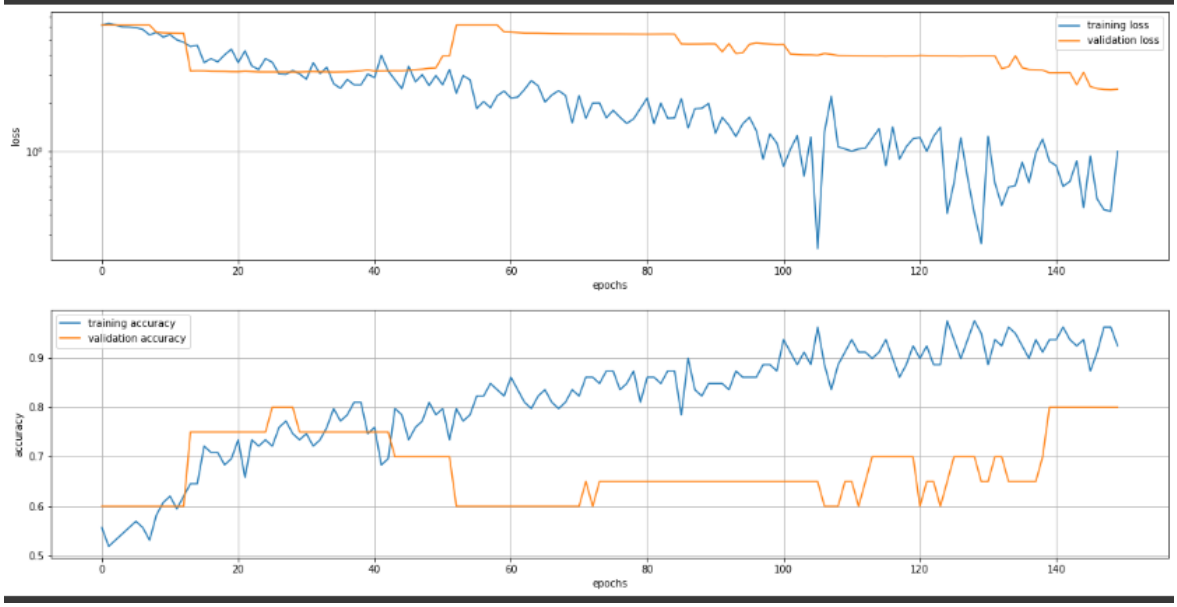
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f1d8c453490>



شکل 1-17: نتایج تست حالت 75-25

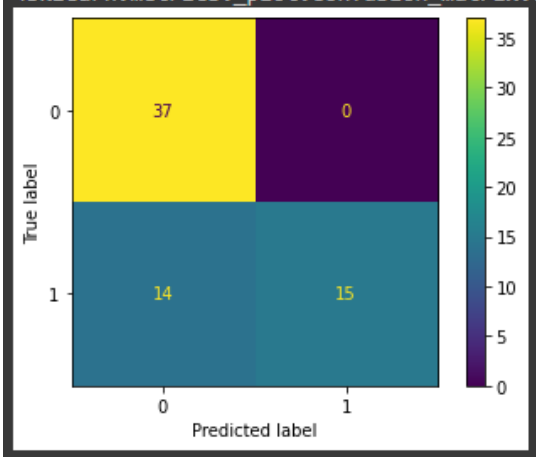
- **آموزش با 60٪ دیتای آموزشی و 40٪ دیتای تست**

40% test & 60% train



شکل 1-18:نمودار **accuracy , loss** برای حالت 60-40 با 150 ایپاک

```
40% test & 60% train

              precision    recall  f1-score   support

           0       0.73      1.00      0.84        37
           1       1.00      0.52      0.68        29

    accuracy                           0.79        66
   macro avg       0.86      0.76      0.76        66
weighted avg       0.85      0.79      0.77        66

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f1f1a0a7070>
```



شکل 1-19: نتایج تست حالت 60-40

چون تعداد داه ها کم است با کاهش داده های ترین در هر مرحله دقت مدل کاهش یافته است بطوری که در حالت آخر یعنی 60 درصد ترین و 40 درصد تست تعداد عکس هایی که نشان دهنده سرطان بوده و سالم تشخیص داده شده است به 14 عدد رسیده است در حالی که این مقدار برای حالت دوم 8 عدد و برای حالت اول 4 عدد بوده است.

پس آموزش مدل در حالت سوم خطای زیادی را تحمیل میکند چرا که تشخیص سالم بودن یک فرد بیمار نتایج فاجعه باری را میتواند به همراه داشته باشد این مورد خود را در recall عدد 1 که لیبل تصاویر سرطانی است مشاهده کرد که رفته رفته کاهش یافته است.

البته باید توجه داشت که اگه تعداد ایپاک یا تعداد داده های ورودی به مدل را زیاد کنیم مدل میتواند نتیجه ی بهتری به همراه داشته باشد.

نوسانی بودن نمودارهای دقت و loss را میتوان با تعداد کم دیتا و همچنین بی ربط بودن آن به داده های imagenet که در pretrain کردن مدل نقش داشته اند توجیه کرد.

کدهای شبیه سازی شده و را آنها در فایل موجود است.

پاسخ ۲ – آشنایی با تشخیص چهره مسدود شده

۲-۱. خلاصه‌ی ساختار شبکه

موضوع اصلی این مقاله آنالیز و بررسی دیتاست های مختلف در تشخیص چهره مسدود شده‌است. در این مقاله چندین نوع دیتاست انواع مختلف دیتاست و ترکیبی از آنها برای آموزش شبکه مورد استفاده قرار گرفته شده است. در این مقاله سه شبکه PSPNet، DeepLabv3+ و SegFormer مورد استفاده قرار گرفته است که دو شبکه اول در دسته شبکه‌های کانوولوشنی و شبکه سوم در دسته شبکه های ترانسفورمری قرار میگیرند. شبکه‌های کانولوشنی در این مسئله از بکبون(backbone) از پیش آموزش دیده Res-Net101 استفاده میکنند و شبکه‌ی ترانسفورمری از بکبون MIT-B5 بهره میبرند. در نهایت این سه شبکه برروی سه دیتاست RealOcc، COFW و RealOcc-Wild تست گرفته شده، و نتایج آن را در شکل 1.2 مشاهده می‌کنید (جدول 4 مقاله).

	Quantity	**RealOcc** (mIoU)			**COFW (Train)** (mIoU)			**RealOcc-Wild** (mIoU)		
		PSPNet	DeepLabv3+	SegFormer	PSPNet	DeepLabv3+	SegFormer	PSPNet	DeepLabv3+	SegFormer
C-Original	29,200	89.52	88.13	88.33	89.64	88.62	91.36	85.21	82.05	85.24
C-CM	29,200	96.15	96.13	97.42	91.82	92.77	**94.87**	91.33	91.01	95.16
C-WO	24,602	89.38	89.01	91.36	89.53	88.97	92.24	83.86	84.14	86.72
C-WO + C-WO-NatOcc	24,602 + 49,204	96.65	96.51	97.30	90.71	91.21	94.30	91.34	91.70	94.17
C-WO + C-WO-NatOcc-SOT	24,602 + 49,204	96.35	96.59	97.18	**92.32**	91.74	93.55	**93.26**	92.69	94.27
C-WO + C-WO-RandOcc	24,602 + 49,204	95.09	95.21	96.53	90.82	91.35	93.14	89.54	89.68	92.84
C-WO + C-WO-Mix	24,602 + 73,806	96.55	96.66	97.37	90.99	91.20	93.74	92.14	91.84	94.40
C-CM + C-WO-NatOcc	29,200 + 49,204	**97.28**	**97.33**	97.95	91.61	92.66	94.86	92.13	**93.81**	**95.43**
C-CM + C-WO-NatOcc-SOT	29,200 + 49,204	97.17	97.29	**98.02**	92.07	**92.91**	94.60	92.84	93.73	94.53

شکل 1.2 نتایج نهایی مقاله

در شکل1.2 دیتاست‌هایی که شبکه با آنها آموزش میبینید در ستون سمت چپ قرار دارند. C-WO دیتاستی‌ست که توسط خود مقاله ایجاد شده است و در آن عکس‌های افراد توسط اجسامی مسدود شده‌اند و دیتاست C-CM هم شامل عکس‌های مسدود نشده میباشد و هم مسدود شده. دیتاست C-WO-NatOcc دیتاستیست که در آن اجسامی که تصویر را مسدود کرده اند اجسام طبیعی و واقعی تری هستند (همچون دست) اما در دیتاست C-WO-RandOcc اجسامی که تصویر را مسدود کرده اند کاملا تصادفی هستند و الزامی به طبیعی بودن عکس در آن وجود ندارد.

۲-2. تفاوت بین Occlusionها

پوشاننده های مختلفی مثل دستها ، عینک و ماسک وجود دارد اما این قبیل موانع قابل استفاده در همه کاربرد ها نیستند پس یصورت رندوم اقدام به تولید پوشاننده های مصنوعی بر روی چهره میکنیم. میتوان درجه شفافیت این پوشاننده ها را بر اساس نیاز مدل انتخاب کرد.

همانگونه که در شکل1.2 مشاهده میشود، نتایج بدست آمده از شبکه هایی که با دیتاست C-WO-NatOcc آموزش دیده‌اند بهتر از شبکه‌هاییست که با C-WO-RandOcc آموزش دیده‌اند که این نشان میدهد استفاده از Occulasion های طبیعی مانند دست بهتر عمل می‌کند.

۲-3. آیا کلاس بندی کردن دادهها لزومی دارد؟

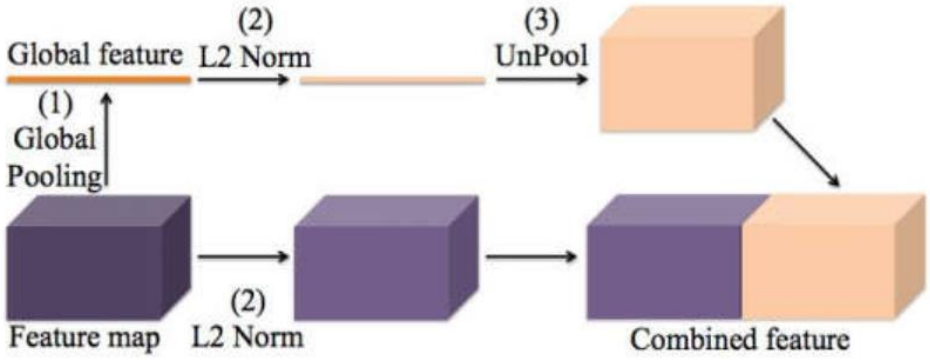
برای بررسی تاثیر استفاده از occlusion های مختلف بر روی خروجی شبکه نیاز داریم که داده ها را کلاس بندی کنیم و با استفاده از معیار mIoU که متوسط IoU های همه‌ی کلاس های موجود در دیتاست است، دقت شبکه را ارزیابی کنیم. کلاس های استفاده شده در این مقاله بصورت شکل زیر است (جدول 1 مقاله).

Class	Definition
Face	The skin of the head includes eyes, nose, mouth but excluding ears.
Face (Grey Area)	Tattoo, shadow, moustache, and beard overlapped with face, as well as skin of the bald person are considered as part of the face.
Background (Occlusions)	Non-face area and any objects such as sunglasses, shirt, hair, microphone, hands that are physically covering (overlap) the face. Words from magazines or copyright labels fall into this category as well.
Background (Grey Area)	Transparent/Translucent glasses

شکل 2.2 کلاس‌های موجود در دیتاست

۲-4. اگر intensity چهره‌ها با Occlusionهای مصنوعی متفاوت باشد،کدام شبکه ساده مطالعه شده در درس را پیشنهاد میدهید؟

متفاوت بودن intensity چهره‌ها با occlusionها باعث می‌شود که لبه‌های چهره ها با occlusionها به خوبی از هم جدا می‌شوند و تشخیص آن ساده تر میشود در نتیجه میتوان از شبکه های FCN که تماما از لایه‌های کانولوشنی هستند استفاده کرد و این شبکه‌ها با توجه به اینکه بلوک‌هایی همچون CRF ندارند، پیچیدگی محاسباتی کمتری دارند و مناسب خواهند بود. یک نمونه از شبکه‌های ساده‌ای که میتوان از آن استفاده کرد شبکه ParsNet می باشد که در تصویر زیر مشاهده می‌کنید.

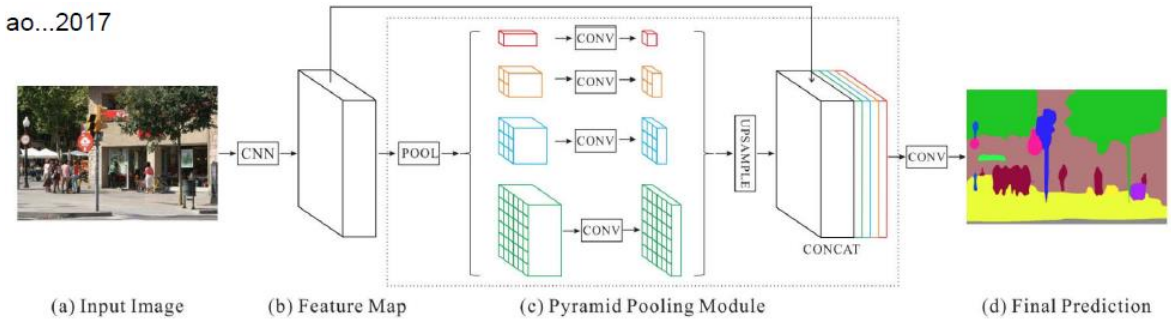


شکل 2-3 : شبکه ی **ParsNet**

۲-5. مقایسهای بین کارایی PSPNet و DeepLab

:PSPNet

در این مدل داده ها فیچر مپ از رزولوشن کم و زیاد حاصل میشود و در هم آمیخته میشود رزولوشن زیاد در بر دارنده ی اطلاعات پس زمینه تصویر و رزولوشن کم در بردارنده ی جزیئات می باشد پس میتواند نتیجه گرفت که ین سیسم تصویر را بر اساس کل تصویر و تفاوت قسمت های مختلف آن سگمنت میکند.

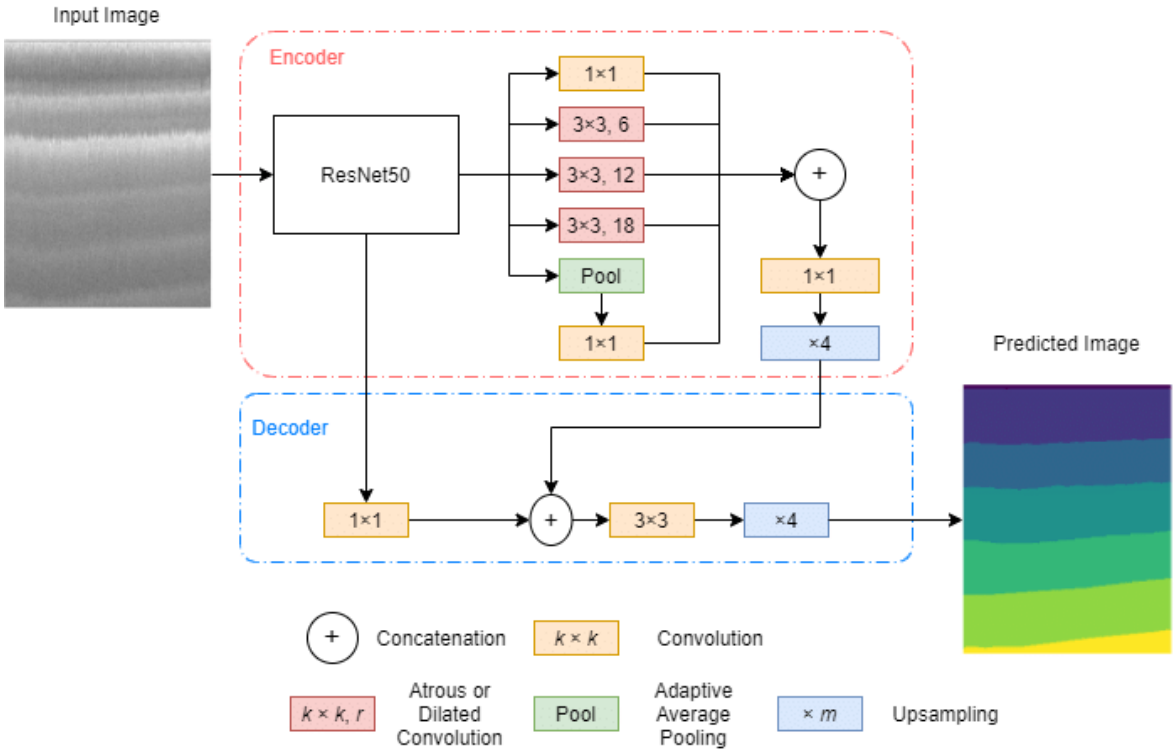


شکل 2-4:معماری **PSPNet**

:DeppLab

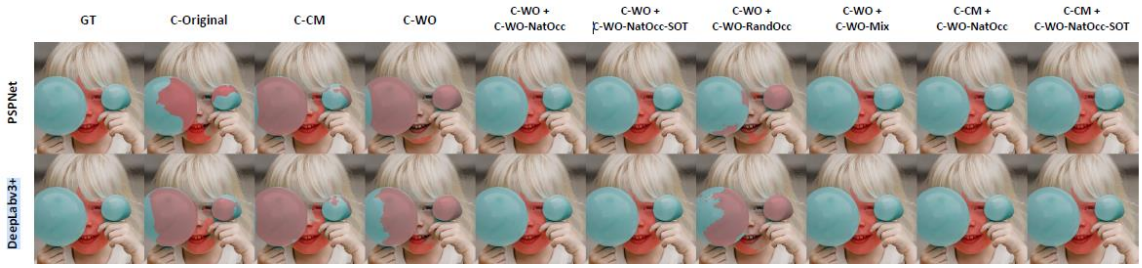
مدل «DeepLab»، شبکه‌های آموزش دیده را در طبقه‌بندی تصویر برای عمل تقسیم‌بندی معنایی با اعمال «کانولوشن بسط داده شده» با فیلترهای upsampled برای استخراج ویژگی‌های متراکم، هدف قرار می‌دهد. سپس به ادغام هرم فضایی بسط داده شده برای رمزگذاری اشیا و همچنین زمینه تصویر را در مقیاس‌های چندگانه توسعه داده میشود. برای تولید پیش‌بینی‌های دقیق و نقشه‌های تقسیم‌بندی دقیق در امتداد مرزهای شی، از شبکه‌های عصبی کانولوشنال عمیق و میدان‌های تصادفی شرطی کاملاً متصل(fully-connected) ترکیب شده استفاده میکند. تا به یک تقسیم بندی مناسب برای تصویر دست یابد.

از لحاظ دقت به یک دقت خوبی روی چندین مجموعه داده ، از جمله داده های تقسیم‌بندی معنایی PASCAL VOC 2012، PASCAL-Context، PASCALPerson-Part و Cityscapes رسیدهه شده است. از نظر سادگی نیز مدل از پشت سر هم قرار گرفتن دو ماژول بسیار خوب همچون ماژول DCNN ها وماژول CRF ها تشکیل شده است، که باعث سادگی مدل میشوند.



شکل 2-5: استفاده از **resnet50** به عنوان بکبون **Deeplab**

همانطور که در شکل زیر مشاهده میشود deeplab عملکرد دقیق تری نسبت به pspnet برای تشخیص چهرا به هنگامی که پوشاننده ای روی آن قرار دارد ،دارد.



شکل 2-6:مقایسه کارایی **PSPNet, DeepLab**

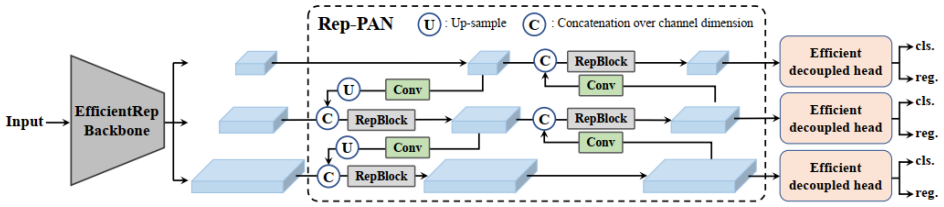
در نهایت میتوان نتیجه گرفت که PSPNet بر روی کل تصویر تمرکز میکند اما Deeplab برای شناسایی جرئیات بهتر عمل میکند.

.

پاسخ ۳ – تشخیص بلادرنگ اشیاء

۳-۱. مقدمه

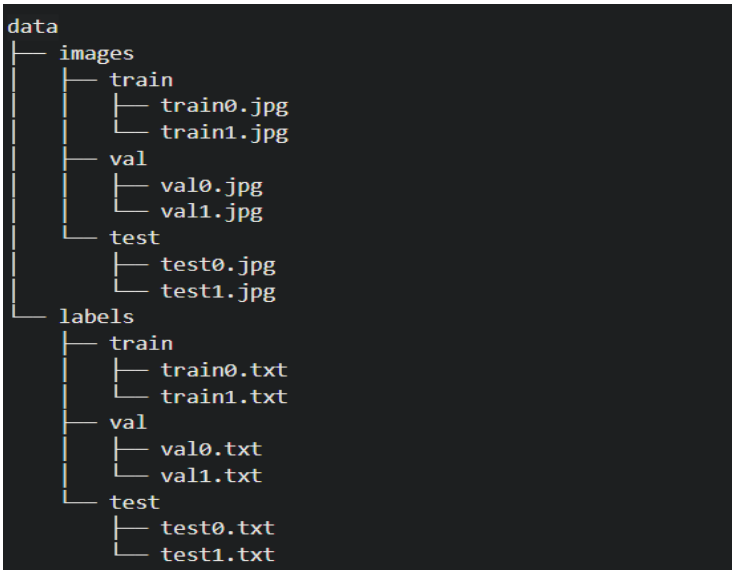
هدف این مسئله تشخیص اشیا با استفاده از YOLOv6 می‌باشد. این الگوریتم مدل‌های مختلفی دارد که با توجه به کاربردشان و حجم کار مدنظر مورد استفاده قرار می‌گیرند. YOLOv6 یک الگوریتم تشخیص شی تک مرحله‌ای است که از backbone ، neck و head تشکیل شده است. در بخش backbone عمدتا توانایی نمایش ویژگی‌ها را تعیین میکند و در عین حال مهم‌ترین بخش نیز میباشد چراکه اکثر پارامترها و محاسبات مربوط به این بخش می‌باشند. بخش neck وظیفه تجمیع ویژگی‌های فیزیکی سطح پایین با ویژگی‌های معنایی سطح بالا و سپس ساختن نقشه‌های ویژگی‌ها را دارد. بخش head از چندین لایه کانولوشنی تشکیل شده است و نتایج نهایی را با توجه به ویژگی های بخش neck پیش‌بینی می‌کند.



شکل 1.3 معماری شبکه **YOLOv6**

۳-2. شخصی‌سازی مجموعه داده جدید

در ابتدا نیاز است که یک بار با استفاده از کد !git clone https://github.com/meituan/YOLOv6 YOLOv6 را دانلود و در درایو خود ذخیره کنیم. سپس دایرکتوری را به فولدر *YOLOv6* تغییر میدهیم. در کنار این فولدر، فولدری به نام *data* ایجاد میکنیم و بصورتی که در شکل1.3 نشان داده شده است دایرکتری را در این فولدر ایجاد می‌کنیم.



شکل 2.3 ایجاد دایرکتوری مناسب برای داده ها

همانگونه مشخص است، تصاویر مربوط به آموزش و ارزیابی و تست را در فولدر مربوطه در images انتقال میدهیم و فایل txt هر دسته را نیز در فولدر مربوطه دز labels قرار می‌دهیم.

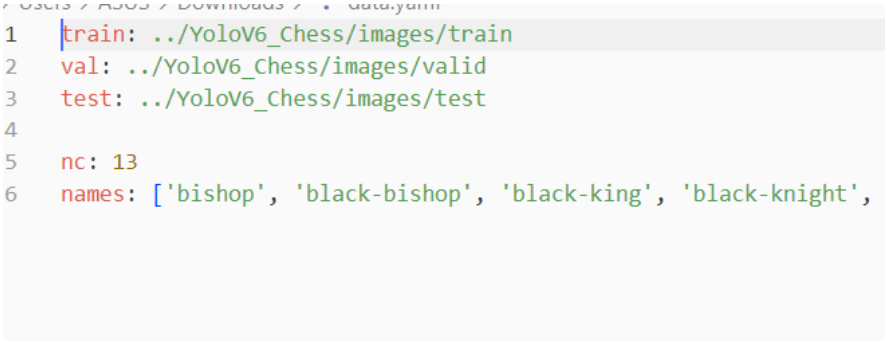
دیتاست مربوط به این مسئله تصاویری از مهره های شطرنج هستند. کلاس های این دیتاست در جدول 1.1 مشاهده میکنید.

جدول 1.3 کلاس‌های موجود در داده‌ها

White bishop	8	Bishop	1
White king	9	Black bishop	2
White knight	10	Black king	3
White pawn	11	Black knight	4
White queen	12	Black pawn	5
White rook	13	Black queen	6
		Black rook	7

۳-3. کدهای شخصی‌سازی شده YOLOv6

در ادامه نیاز است که فایلی با پسوند yaml. ایجاد کنیم که در آن مشخصات مربوط به دیتاست قرار دارد. این فایل شامل تعداد کلاس‌ها، نام کلاس‌ها و آدرس تصاویر آموزش و ارزیابی و تست می‌باشد. این فایل yaml را در فولدر YOLOv6 که دایرکتوری کولب نیز به آن تغییر یافته، ایجاد میکنیم.



شکل 3.3 محتوای فایل **yaml** ایجاد شده

در این فایل مقابل train و val و test آدرس مربوط به فولدر عکس های هریک را قرار میدهیم. سپس با توجه به اینکه از دیتاست coco استفاده نمیکنیم، is_coco را False قرار میدهیم. در مقابل nc نیز تعداد کلاس ها را که 13 میباشد قرار میدهیم در لیست names هم اسم کلاس ها را به ترتیب وارد میکنیم.

در نهایت با اجرای pip install -r requierments.txt! فایل‌های مورد نیاز YOLO را دانلود و نصب میکنیم.

برای آغاز آموزش شبکه از دستور

$$!\,python\ tools/train.py\ --batch\ 32\ --conf\ configs/yolov6s.py\ --data\ data.yaml\ --device\ 0\ --epochs\ 100$$

استفاده میکنیم.

همانگونه که مشاهده میشود، batch size برابر با 32 قرار داده شده (برای مقادیر بزرگتر، گوگل کولب ارور میداد!)

در قسمت conf هم فایل config مربوط به مدلی که میخواهیم استفاده کنیم قرار میدهیم. با توجه به اینکه داده های ما کم هستند از مدل کوچکتر که yolov6s.py میباشد استفاده میکنیم. با این انتخاب فرایند آموزش از آغاز شروع میشود ولی اگر بخواهیم ادامه‌ی وزن‌هایی که از قبل داریم را برای آموزش استفاده کنیم میتوانیم از فایل yolov6s_fintune.py استفاده کنیم، در اینصورت نیاز است که فایل مربوط به وزن های قبلی را در فولدر YOLOv6/weights قرار دهیم. با توجه به محدودیت گوگل کولب ما در این سوال ابتدا 100 ایپاک مدل را آموزش دادیم و سپس با استفاده از قابلیت finetune و استفاده از وزن های به دست آمده، 100 ایپاک دیگر هم آموزش را در یک مرحله دیگر ادامه دادیم.

```
[1]  from google.colab import drive
     drive.mount('/content/drive')

     Mounted at /content/drive

[2]  %cd /content/drive/MyDrive/NNDL/HW3/Q3/YOLOv6
     !pip install -r requirements.txt
```

شکل 4.3 لود کردن **YOLOv6**

```
!python tools/train.py --batch 32 --conf configs/yolov6s.py  --data data.yaml --device 0 --epochs 100

Using 1 GPU for training...
training args are: Namespace(batch_size=32, calib=False, check_images=False, check_labels=False, conf

Train: Checking formats of images with 2 process(es):
0 image(s) corrupted: 100% 606/606 [00:08<00:00, 69.58it/s]
Train: Checking formats of labels with 2 process(es):
606 label(s) found, 0 label(s) missing, 0 label(s) empty, 0 invalid label files: 100% 606/606 [02:54<
WARNING: ../YoloV6_Chess/labels/train/26d663ab5ffbec49f9dc8e592982cfd4_jpg.rf.0c6954a631b0faeeb2de29d
WARNING: ../YoloV6_Chess/labels/train/26d663ab5ffbec49f9dc8e592982cfd4_jpg.rf.aa2066fcdf707c6b36bcac8
WARNING: ../YoloV6_Chess/labels/train/26d663ab5ffbec49f9dc8e592982cfd4_jpg.rf.f70c87c81995f03a0de866b
Train: Final numbers of valid images: 606/ labels: 606.
187.3s for dataset initialization.
Val: Checking formats of images with 2 process(es):
0 image(s) corrupted: 100% 58/58 [00:15<00:00,  3.69it/s]
Val: Checking formats of labels with 2 process(es):
58 label(s) found, 0 label(s) missing, 0 label(s) empty, 0 invalid label files: 100% 58/58 [00:21<00:
Convert to COCO format
100% 58/58 [00:00<00:00, 25136.35it/s]
Convert to COCO format finished. Resutls saved in ../YoloV6_Chess/annotations/instances_valid.json
Val: Final numbers of valid images: 58/ labels: 58.
37.8s for dataset initialization.
```

شکل 5.3 کد آموزش شبکه

همانگونه که مشاهده میشود پس از آغاز فرایند آموزش یک فایل از اطلاعات دیتاست و داده های val به فرمت coco که json. میباشد در می آید و در فولدر annotations در فمولدر داده ها ذخیره میشود.

پس از اتمام فرایند آموزش نهایی، نتیج بصورت شکل زیر خواهند بود.

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.720
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.968
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.895
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.661
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.722
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.591
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.776
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.776
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.679
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.777
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
Results saved to runs/train/exp12
Epoch: 99 | mAP@0.5: 0.9681364411824513 | mAP@0.50:0.95: 0.719733005120343

Training completed in 0.714 hours.
```

شکل 6.3 نتایج نهایی آموزش

یکی از مهمترین معیارهایی که برای ارزیابی شبکه مورد استفاده قرار گرفته است mAP است که مخفف Mean Average Precision میباشد. Precision میزان دقت خروجی شبکه را نشان میدهد و هرچه این مقدار به 1 نزدیکتر باشد دقت شبکه در صحت لیبل گذاری تصویر ورودی بیشتر است. در حالت کلی این پارامتر بصورت زیر تعریف میشود:

$$Precision = \frac{TP}{TP + FP}$$

mAP نیز میانگین متوسط این مقدار برای کل کلاس های دیتاست است. پارامتر مهم دیگر Recall است که نشان دهنده این است که سیستم به چه اندازه توانسته اجسام را تشخیص دهد، چه درست چه غلط!

$$Recall = \frac{TP}{TP + FN}$$

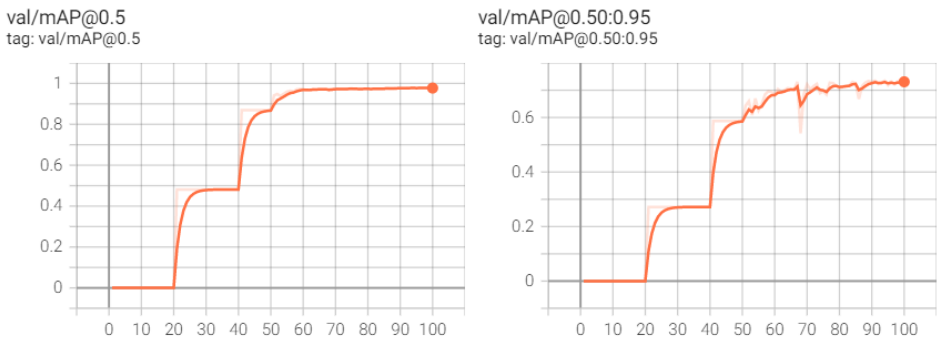
برای اینکه عملکرد یک شبکه مطلوب باشد باید هردو مقدار نزدیک به 1 باشند که مقادیر محاسبه شده برای این دو پارامتر با توجه به حجم کم دیتاست بسیار مطلوب است.

$$mAP@0.5 = 0.9681 \qquad mAP@0.5{:}\,0.95 = 0.7197$$

برای اینکه روند فرایند آموزش را بخوبی مشاهده کنیم در تصویر زیر نتیجه ایپاک بیستم قرار داده شده است.

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.272
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.481
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.286
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.351
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.279
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.257
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.610
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.632
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.611
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.633
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
Results saved to runs/train/exp11
Epoch: 20 | mAP@0.5: 0.481311151292066 | mAP@0.50:0.95: 0.2718991818929311
```

شکل 7.3 نتایج ایپاک بیستم



شکل 8.3 نمودار تغییرات **mAP** در فرایند آموزش

همانگونه که مشاهده میشود mAP به خوبی افزایش پیدا کرده است.

۳-4. خروجی segment شده نهایی

برای تست شبکه مطابق شکل زیر عمل میکنیم.

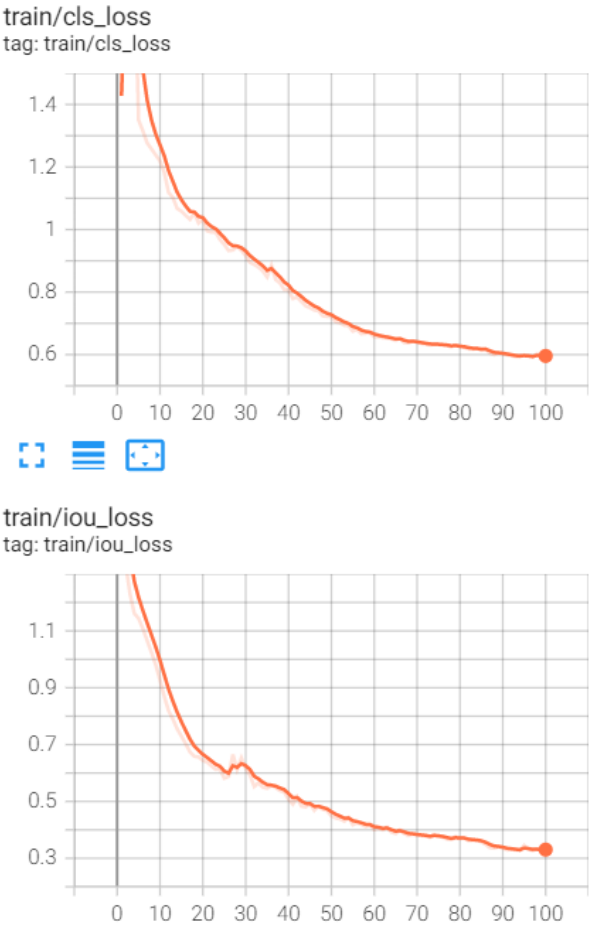
```
# infer on all images in our /test directory  runs/train/exp#/weights/best_ckpt.pt
!python tools/infer.py --yaml data.yaml --weights runs/train/exp12/weights/best_ckpt.pt --source ../YoloV6_Chess/images/test --device 0

Namespace(agnostic_nms=False, classes=None, conf_thres=0.4, device='0', half=False, hide_conf=False, hide_labels=False, img_size=[640, 640]
Save directory already existed
Loading checkpoint from runs/train/exp12/weights/best_ckpt.pt

Fusing model...
Switch model to deploy modality.
/usr/local/lib/python3.8/dist-packages/torch/functional.py:478: UserWarning: torch.meshgrid: in an upcoming release, it will be required to
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
100% 29/29 [00:21<00:00,  1.32it/s]
Results saved to runs/inference/exp
```

شکل 9.3 تست **YOLOv6**

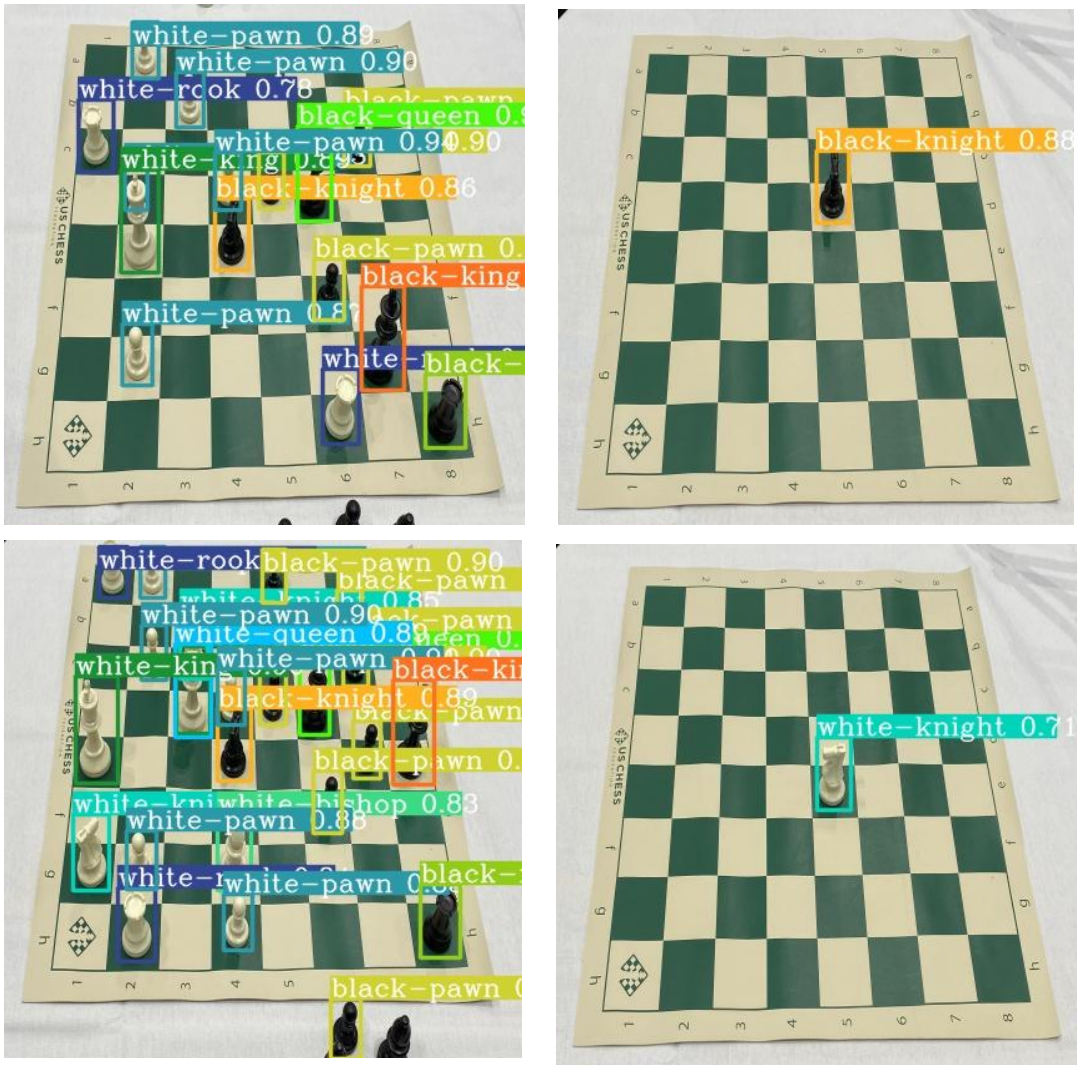
در این بخش باید در قسمت weights آدرس بهترین وزن بدست آمده از آموزش را بدهیم که در آدرس YOLOv6/runs/train/weights خواهد بود. در قسمت source هم باید محل تصاویر تست را وارد کنیم. دقت شود که در فایل yaml. باید حتما آدرس فولدر test وارد شده باشد تا خروجی ها با لیبل نمایش داده شوند.

همانگونه که میبینیم نتایج تست در فولدر YOLOv6/runs/inference/exp ذخیره میشود.

نمودار تغییرات loss هنگام آموزش شبکه، نیز در شکل10.3 قابل مشاهده است کهنشان دهنده روند مناسب کاهشی بود و iou_loss به مقدار 0.3 رسیده که مقدار مطلوبیست.



شکل 10.3 نمودارهای **loss** فرایند آموزش



شکل 11.3 تعدادی از خروجی‌های **segment** شده داده های تست

شکل 12.3 خروجی شبکه برای داده‌های متفرقه از اینترنت!