به نام خدا

دانشگاه تهران

دانشکده مهندسی

برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول

نام و نام خانوادگی	**فاطمه نائینیان – محمد عبائیانی**
شماره دانشجویی	**810198432 - 810198479**
تاریخ ارسال گزارش	**1401-09-16**

فهرست

پاسخ ۱ – آشنایی با یادگیری انتقالی Transfer Learning

1)

شماره دانشجویی اعضای گروه 810198479 و 810198432 میباشد که جمع رقم اخر برابر 11 که باقی مانده ان به عدد 4 برابر 3 میباشد بنابراین شبکه VGG-19 و مقاله سوم انتخاب می شود.

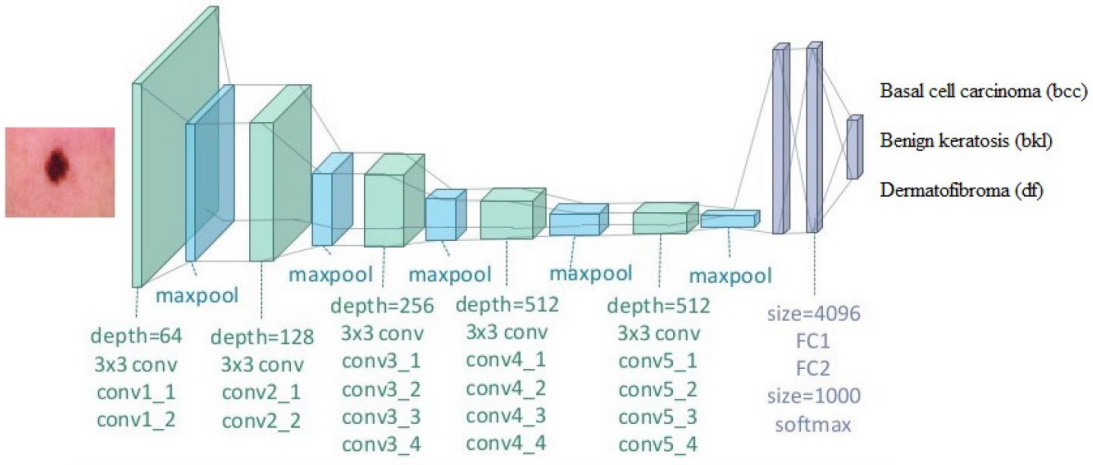
گزارش مقاله

در این مقاله به تشخیص سرطان پوست با کمک شبکه ها عصبی CNN می پردازد. سرطان پوست شایع ترین نوع سرطان است. تشخیص این سرطان در مراحل اولیه آن بسیار ارزشمند است چون سبب می شود تا بتوان آن را درمان کرد. در این راستا در سالهای اخیر تلاش های زیادی صورت گرفته تا بتوان با CNN های مبتنی بر VGG-19 با کمک دیتاست HAM10000 دقت این تشخیص را افزایش داد.

Hosny روشی ارائه داد که در آن سه نوع سرطان پوست را با 98.61% دقت تشخیص داد و Younis با روش ارائه داده توانست هفت نوع سرطان پوست را با دقت 97.07% تشخیص دهد. سپس پژوهشگر های دیگری توانستد با ارتقا دادن مدل های قبلی به نتایج جدیدی برسند.

دیتاست HAM10000 سه نوع DF و BCC و BKL وجود دارند که نوع BKL داده های غیر سرطانی هستند. به همین دلیل دیتاست بایاس دارد و بالانس نیست. به همین دلیل با عملیات Augmentation سعی می شود تا داده ها را بالانس کرد. در نهایت 1000 داده از هرکدام و سر جمع 3000 داده خواهیم داشت.

VGG-19 حاوی چندین لایه max pooling که برای استخراج ویژگی استفاده می شود است. این لایه ها حداقل به چند لایه fully connected متصل هستند. این مدل برای تشخیص سرطان پوست گسترش داده شده است. VGG-19 یکی از روش های Transfer Learning است. در این روش 20% داده ها برای تست و 20% داده ها برای validation و 60% داده ها برای آموزش استفاده می شود. مدل در 100 epochs و با batch size = 50 و learning rate = 0.01 و optimizer = Adam اجرا می شود.



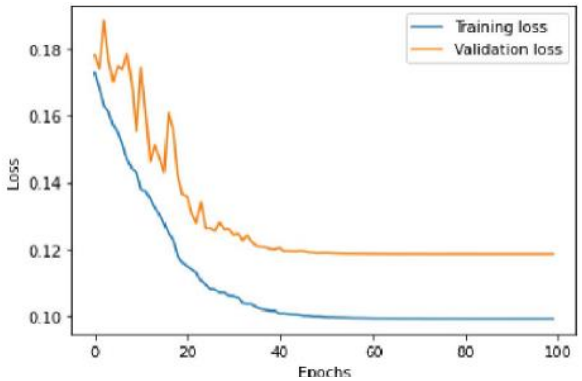
شکل 1: معماری کلی **VGG19**

بعد از آموزش شبکه، دقت شبکه بر روی داده های آموزش 0.985 و بر روی داده های تست 0.975 می شود. همچنین مقدار loss شبکه بر روی داده های آموزش 0.099 و بر روی داده های تست 0.119 می شود. با توجه به اینکه مقدار دقت آموزش و تست تفاوت خاصی ندارند پس نتیجه میگیریم شبکه Overfit نشده است. همچنین بعد از epoch 60 و 70 مقدار loss و دقت تقریبا ثابت می شود که به معنی این است که شبکه پایدار است.

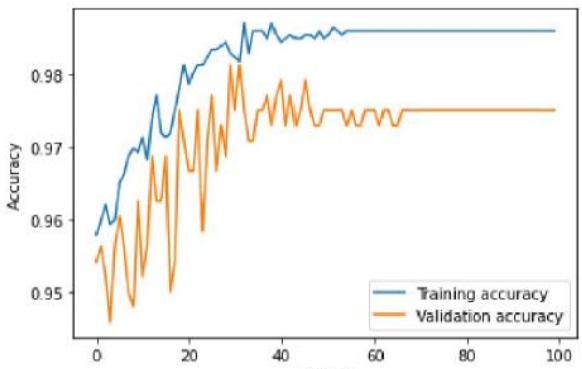
این شبکه VGG19-based CNN ابزاری قدرتمند برای تشخیص سرطان پوست به حساب می آید.

Epoch	Training		Validation	
	Accuracy	Loss	Accuracy	Loss
25	0.9823	0.1094	0.9708	0.1264
50	0.9849	0.0997	0.9750	0.1188
100	0.9859	0.0991	0.9750	0.1185

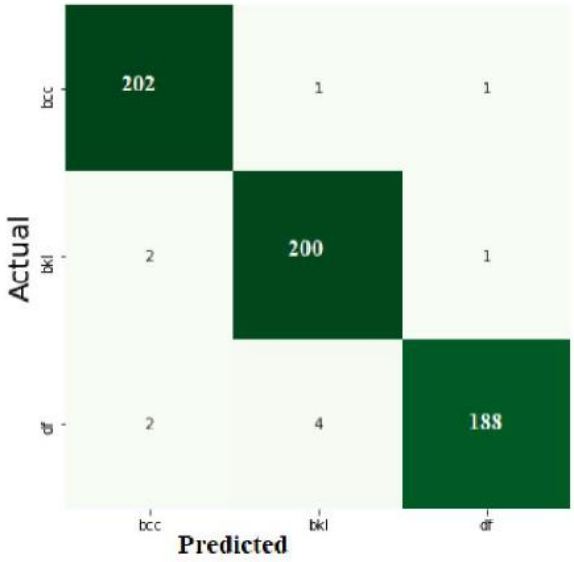
جدول 1: خلاصه نتایج **VGG19**



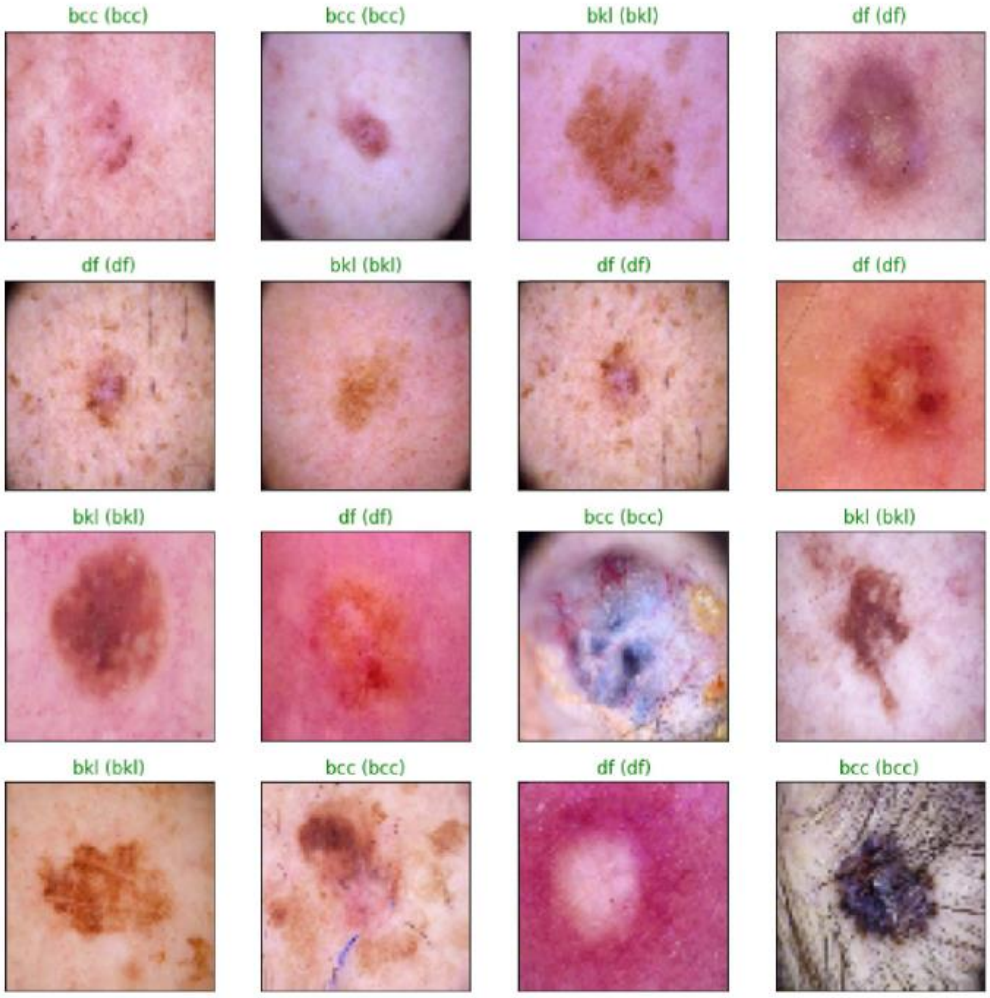
شکل 3: نمودار **loss**



شکل 2:نمودار دقت



شکل 4: **Confusion matrix of VGG19**

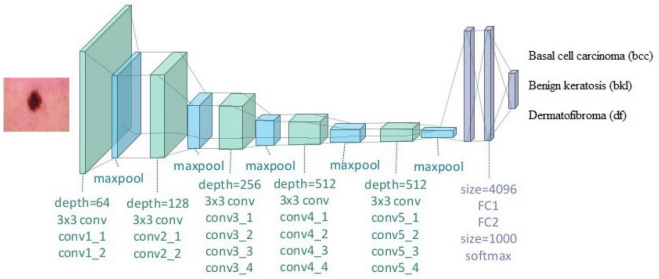


شکل 5: چند نمونه داده همراه با نتایج آن

(2

ساختار VGG-19 به شکلیست که شامل 19 لایه می شود.

ابتدا عکس هایی با سایز 64*64 به شبکه داده می شود سپس دو لایه Convolution 3*3 وجود دارد سپس خروجی انها را ترکیب کرده و از یک لایه max pooling عبور داده و سپس مجدد از دو لایه Convolution 3*3 و سپس max pool عبور میدهد. حال سه مرتبه از چهار لایه Convolution 3*3 و max pool عبور میدهد. در نهیات از دو لایه fully connected عبور میدهد و خروجی به دست می اید.



شکل 7: معماری شبکه **VGG19**

به دلیل اینکه شبکه 19 لایه دارد،تعداد پارامترهای ان بسیار زیاد می شود به همین دلیل از لحاظ وقت و حافظه پیچیدگی های زیادی خواهد داشت. این شبکه کاربرد زیادی در کلاس بندی عکس ها دارد. این شبکه بسیار بازدهی بسیار خوبی دارد. اپدیت کردن وزن ها در این شبکه با کمک error backpropagation انجام می شود. به همین دلیل تغییرات بسیار کوچکی در وزن ها اتفاق می افتد. با این حال باید از قانون زنجیره ای استفاده کرد که بسیار وقت گیر است.

VGG-19
Conv3x3 (64)
Conv3x3 (64)
MaxPool
Conv3x3 (128)
Conv3x3 (128)
MaxPool
Conv3x3 (256)
Conv3x3 (256)
Conv3x3 (256)
Conv3x3 (256)
MaxPool
Conv3x3 (512)
Conv3x3 (512)
Conv3x3 (512)
Conv3x3 (512)
MaxPool
Conv3x3 (512)
Conv3x3 (512)
Conv3x3 (512)
Conv3x3 (512)
MaxPool
Fully Connected (4096)
Fully Connected (4096)
Fully Connected (1000)
SoftMax

شکل 6: ساختار لایه های **VGG19**

همانطور که در بخش گزارش مقاله ذکر شد، دیتاست HAM10000 سه نوع DF و BCC و BKL وجود دارند که نوع BKL داده های غیر سرطانی هستند. به همین دلیل دیتاست بایاس دارد و بالانس نیست. به همین دلیل با عملیات Augmentation سعی می شود تا داده ها را بالانس کرد. این روش شامل آیینه

کردن تصاویر به صورت افقی و عمودی، تغییر روشنایی، بریدن تصویر، تغییر سایز و ترکیب این روش ها میباشد. در نهایت 1000 داده از هرکدام و سر جمع 3000 داده خواهیم داشت.

3)

باتوجه به اینکه داده های موجود در دیتاست در هر کلاس تعداد متفاوتی دارند، ممکن است کلاسی تمام case های بیماری را در خود نداشته باشد، در این صورت ممکن است مدل در صورت وجود case های جدید یا کمیاب درست عمل نکند. برای بهبود این موضوع می بایست سعی شود تمامی نمونه های موجود از هر کلاس در داده های train به میزان کافی وجود داشته باشد و همچنین اگر case جدیدی کشف شد مدل با آن case آپدیت شود.

نکته دیگری که باید به ان دقت کرد این است که هر مدل توانایی تشخیص داده هایی مشابه داده های اموزش را دارد پس اگر یک داده بی ربط به ان ورودی دهیم، مدل از پس تشخیص بر نمی اید و یا جواب های بی ربط میدهد.

4)

دیتاست مربوطه در سایت Kaggle موجود است. برای اینکه بتوانیم از دیتا ست در کولب استفاده کنیم، با دستورات زیر دیتا ست را وارد کولب میکنیم.

```
! pip install kaggle
```

شکل 8: نصب پکیج **Kaggle**

```
from google.colab import files

files.upload()
```

Choose Files kaggle.json

- **kaggle.json**(application/json) - 71 bytes, last modified: 12/4/2022 - 100% done

```
Saving kaggle.json to kaggle.json
{'kaggle.json':
b'{"username":"fatemehnaeinian","key":"53a796fd33a454499555252adf7d8362"}'}
```

```
! mkdir ~/.kaggle
```

```
! cp kaggle.json ~/.kaggle/
```

```
! chmod 600 ~/.kaggle/kaggle.json
```

```
! kaggle datasets download -d umangjpatel/ham10000-imagenet-style-dataset
```

```
Downloading ham10000-imagenet-style-dataset.zip to /content
100% 2.57G/2.58G [00:15<00:00, 192MB/s]
100% 2.58G/2.58G [00:15<00:00, 180MB/s]
```

شکل 9: لود کردن دیتاست

حال باید دیتاست را unzip کنیم.

```
! unzip /content/ham10000-imagenet-style-dataset.zip -d data
```

شکل 10: **unzip** کردن دیتا ها

خروجی کد بالا به صورت یک پوشه است که شامل 7 پوشه می شود. در مقاله از 3 نوع این سرطان ها استفاده شده است، پس ما هم پوشه آن 4 نوع دیگر را حذف میکنیم.

```
!rm -rf '/content/data/akiec'
!rm -rf '/content/data/mel'
!rm -rf '/content/data/vasc'
!rm -rf '/content/data/nv'
```

شکل 11: حذف کردن داده های بی استفاده

5)

بعد از اماده سازی داده ها، تعدادی پوشه برای جدا سازی داده های train و test و valid میسازیم. سپس با کمک ImageDataGenerator داده ها را بالانس میکنیم و همزمان به train و test و valid نیز تقسیم میکنیم. مراحل کار به صورتی است که ابتدا داده ها را بررسی میکنیم. دسته bkl نیاز با data augmentation ندارد اما دو دسته دیگر نیاز دارد تا تعداد ها زیاد شود. بنابر این ابتدا یک ImageDataGenerator برای دو دسته دیگر به روش زیر تعریف میکنیم.

```
image_generator1 = ImageDataGenerator(validation_split=0.2,
                                       rescale=1./255,
                                       shear_range = 0.6,
                                       zoom_range = 0.6,
                                       rotation_range=120,
                                       vertical_flip=True,
                                       horizontal_flip = True)
```

شکل 12: **ImageDataGenerator**

همانطور که میبینید معیار هایی برای ان تعریف کردیم تا با اعمال ان بر روی عکس های موجود، داده های جدید تولید کند. سپس با کمک این تابع، دو دسته bf وbcc را augment میکنیم.

در همین هنگام داده های test و train و validation را نیز جدا میکنیم.

بعد از عملیات data augmentation میبینیم دیتا ها بالانس شده است و تعداد تقریبا مشابهی از هر دسته دیتا داریم. در شکل زیر تعدد دیتا های هر مجموعه را میبینیم.

```
train/bcc 802
train/bkl 810
train/df 786
val/bcc 102
val/bkl 110
val/df 102
test/bcc 102
test/bkl 110
test/df 102
```

شکل 13: تعداد داده ها بعد از **data augmentation**

حال به سراغ تعریف مدل می رویم. در کتابخانه keras میتوان مدل VGG19 را پیدا کرد. ان را به عنوان base model در نظر میگیریم و پارامتر های ان را به شکل زیر تعیین میکنیم.

```
base_model = tf.keras.applications.VGG19(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=(64,64,3),
    pooling="max",
    classes=3,
    classifier_activation="relu",
)
base_model.trainable = False
base_model.summary()
```

شکل 14: مدل **VGG19** در کتابخانه **keras**

سپس در ادامه با قرار دادن چند fully connected و dropout مدل را تکمیل میکنیم تا به خروجی 3 تایی برسیم.

```
model = tf.keras.Sequential([
                            base_model,
                            keras.layers.Flatten(),
                            keras.layers.Dense(4096, activation="relu",kernel_initializer='he_uniform'),
                            keras.layers.Dropout(0.2),
                            keras.layers.Dense(1000, activation="relu",kernel_initializer='he_uniform'),
                            keras.layers.Dropout(0.2),
                            keras.layers.Dense(400, activation="relu",kernel_initializer='he_uniform'),
                            keras.layers.Dropout(0.2),
                            keras.layers.Dense(200, activation="relu",kernel_initializer='he_uniform'),
                            keras.layers.Dropout(0.2),
                            keras.layers.Dense(50, activation="relu",kernel_initializer='he_uniform'),

                            keras.layers.Dense(3,activation="softmax")

                            ])
model.summary()

model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

شکل 15: مدل نهایی با اضافه کردن **fully connected**

پارامترهای مدل به شکل زیر می شود.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 512)               20024384

 flatten_1 (Flatten)         (None, 512)               0

 dense_6 (Dense)             (None, 4096)              2101248

 dropout_4 (Dropout)         (None, 4096)              0

 dense_7 (Dense)             (None, 1000)              4097000

 dropout_5 (Dropout)         (None, 1000)              0

 dense_8 (Dense)             (None, 400)               400400

 dropout_6 (Dropout)         (None, 400)               0

 dense_9 (Dense)             (None, 200)               80200

 dropout_7 (Dropout)         (None, 200)               0

 dense_10 (Dense)            (None, 50)                10050

 dense_11 (Dense)            (None, 3)                 153

=================================================================
Total params: 26,713,435
Trainable params: 6,689,051
Non-trainable params: 20,024,384
```

شکل 16: خلاصه مدل و تعداد پارامتر ها

حال مدل را اجرا میکنیم.

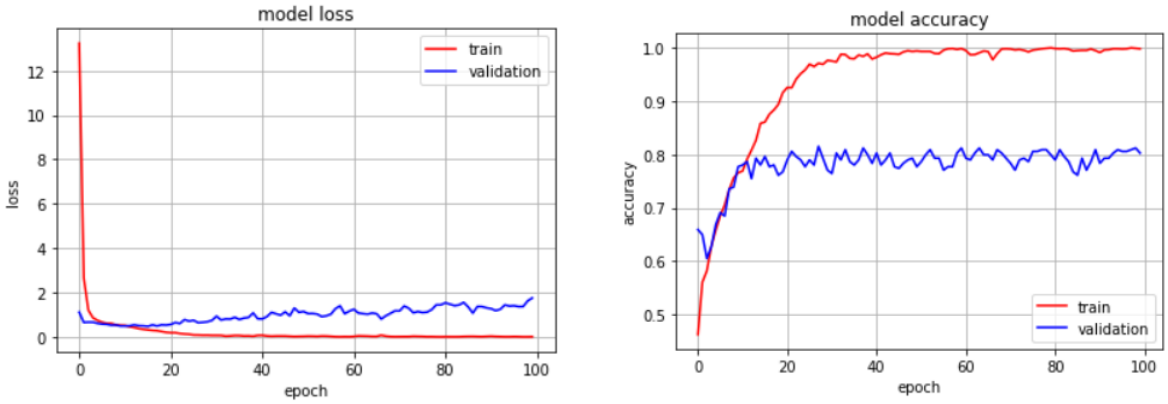
بعد از گذشت 100 epoch به دقت اموزش 0.9979 و دقت validation 0.8025 میرسیم.

4 epoch اخر را در تصویر زیر مشاهده میکنیم.

```
Epoch 97/100
24/24 [==============================] - 2s 68ms/step - loss: 0.0090 - accuracy: 0.9979 - val_loss: 1.3666 - val_accuracy: 0.8057
Epoch 98/100
24/24 [==============================] - 2s 69ms/step - loss: 0.0023 - accuracy: 0.9996 - val_loss: 1.3687 - val_accuracy: 0.8089
Epoch 99/100
24/24 [==============================] - 2s 69ms/step - loss: 0.0024 - accuracy: 0.9987 - val_loss: 1.6276 - val_accuracy: 0.8121
Epoch 100/100
24/24 [==============================] - 2s 69ms/step - loss: 0.0076 - accuracy: 0.9979 - val_loss: 1.7541 - val_accuracy: 0.8025
```

شکل 17:**4 epoch** نهایی

نمودار loss و دقت ان نیز به شکل زیر می شود.



شکل 18: نمودار دقت و **loss** مدل

حال داده های test را به مدل میدهیم. میبینیم مدل به خوبی برای داده های تست کار میکند.

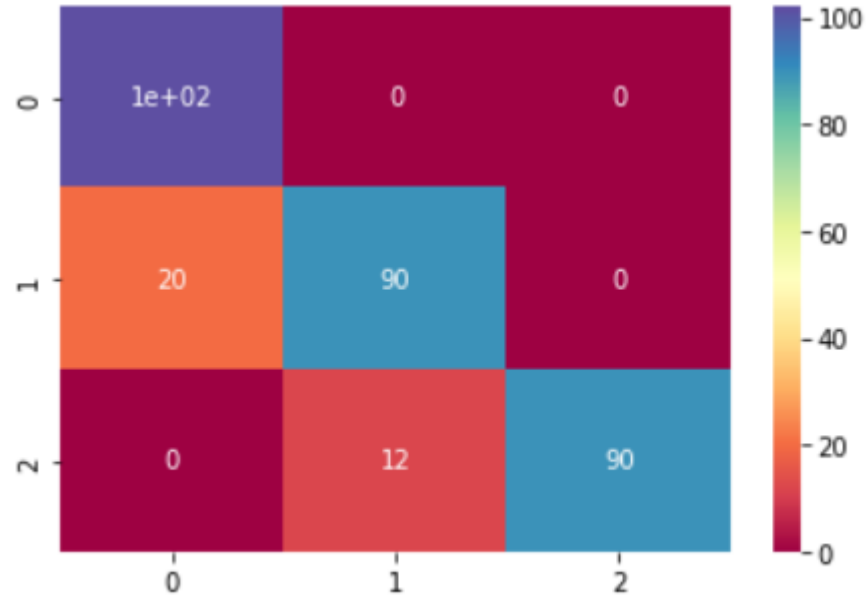
```
test loss = 2.362585
test accuracy = 0.761146
Accuracy: 0.898089
Precision: 0.906140
Recall: 0.900178
F1 score: 0.899090
```

شکل 19: نتایج تست مدل

همچنن ماتریس طبقه بندی نیز به صورت زیر است.

	precision	recall	f1-score	support
0	0.84	1.00	0.91	102
1	0.88	0.82	0.85	110
2	1.00	0.88	0.94	102
accuracy			0.90	314
macro avg	0.91	0.90	0.90	314
weighted avg	0.91	0.90	0.90	314

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f56a3363a30>
```



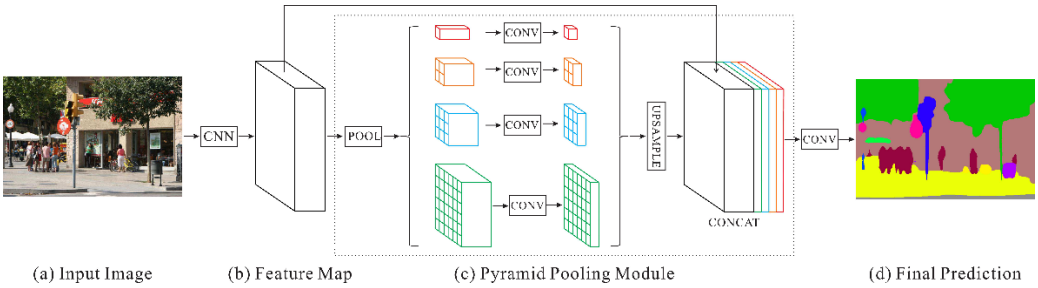
شکل 20: ماتریس طبقه بندی به ازای ورودی تست

پاسخ ۲ - آشنایی با تشخیص چهره مسدود شده

1)

در این مقاله از 3 شبکه ی PSPnet ، DeepLabv3+ و SegFormer استفاده شده. هر کدام ورودی عکس 512*512 استفاده شده و در هرکدام 400 تکرار روی داده های RealOcc برای آموزش انجام شده است.

شبکه ی PSPnet از یک ماژول Pyramid pooling برای بخش بندی داده ها استفاده میکند. این ماژول با kernel هایی با سایز های مختلف(از n*n تا 1*1) عملیات pooling را انجام داده و در نهایت همه ی خروجی ها به همراه ورودی اصلی را با هم ادغام میکند.



شکل 21: شبکه ی **PSPnet**

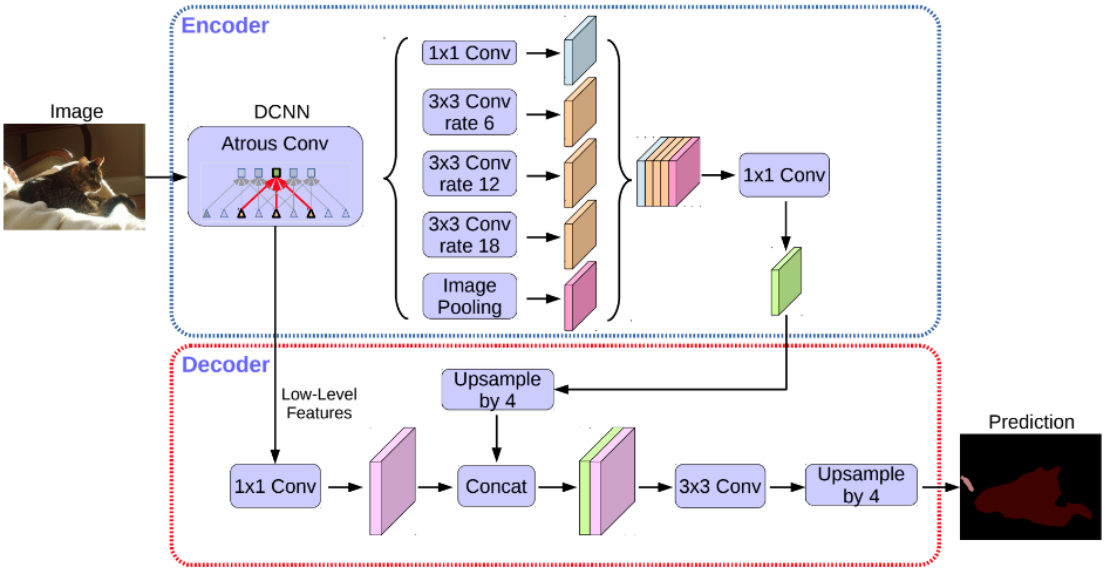
شبکه Deeplabv3+

در این شبکه در بخش encoder 4 عمل convolution و یک pooling به صورت زیر انجام میشود:

1- کانولوشن 1*1
2- کانولوشن 3*3 با stride = 6
3- کانولوشن 3*3 با stride = 12
4- کانولوشن 3*3 با stride = 18
5- یک عملیات pooling

در نهایت خروجی این عملیات ها با هم concat می شود.

در بخش decoder ورودی و خروجی encoder با هم concat شده و از یک لایه کانولوشن 3*3 عبود میکند.



شکل 22: شبکه **Deeplabv3+**

شبکه SegFormer

در این شبکه در بخش encoder 4 بلوک transformer و در بخش decoder از دو شبکه ی MLP استفاده می شود.

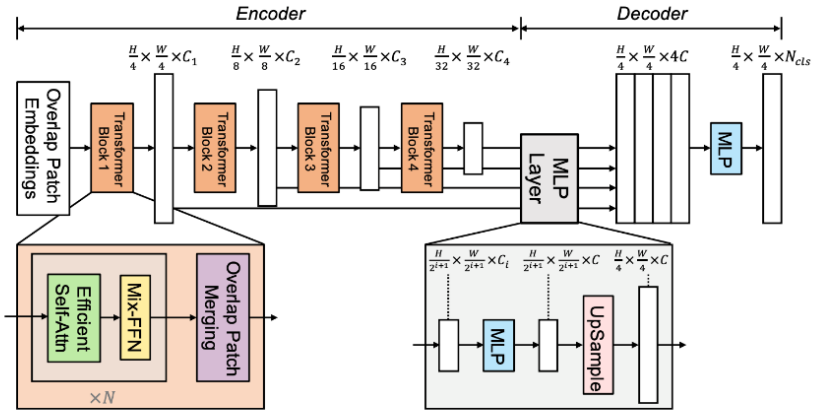


Figure 2: **The proposed SegFormer framework** consists of two main modules: A hierarchical Transformer encoder to extract coarse and fine features; and a lightweight All-MLP decoder to directly fuse these multi-level features and predict the semantic segmentation mask. "FFN" indicates feed-forward network.

شکل 23: شبکه **SegFormer**

(2

طبق خروجی های حاصل شده دقت در داده های RealOcc بالاتر بوده (که داده هایی با Occlusion طبیعی هستند)

داده های آموزش دیده روی RealOcc-wild و COFW(Train) دارای دقت کمتری هستند، اما هر کدام روی دیتای train مشخصی نسبت به دیگری برتری دارد.

بنابراین میتوان گفت که به طور کلی شدید شدن occlusion ها باعث پایین رفتن دقت خواهد شد. اما اگر در داده های آموزش نیز disturbance زیادی وجود داشته باشد در نهایت مدل پیش بینی های بهتری روی این داده ها خواهد کرد.

برای مثال مدل آموزش دیده با داده های C-WO + C-WO-NatOcc-SOT پیش بینی بهتری از C-WO ارائه میدهد.

	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 24: نتایج سه شبکه

(3

با توجه به اینکه خروجی ها در نهایت فقط نقاطی که شامل صورت افراد میشود را از دیگر نقاط جدا میکند نیازی به کلاس بندی نیست و میتوان خروجی را با توابع activator نیز تولید کرد.

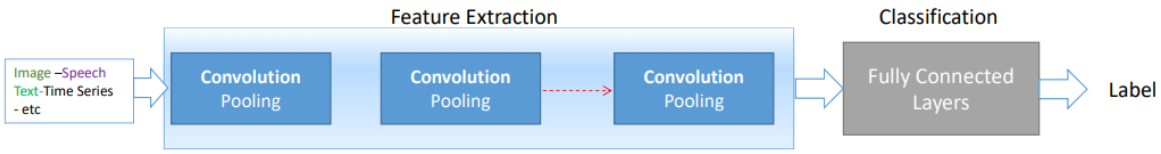
در صورت کلاس بندی تنها دو کلاس وجود خواهد داشت که نقاط شامل صورت و نقاط غیر از صورت میباشند(بکگراند) که در مقاله نیز صرفا همین دو کلاس ایجاد شده.

در صورتی که بخواهیم اجزای بیشتری را شناسایی کنیم(برای مثال بین بک گراند و موانع جلوی صورت تفکیک قائل شویم) در آن صورت نیاز به کلاس بندی خواهیم داشت.

(4

در صورت اینکه اختلاف intensity موانع با صورت ها زیاد باشد و درواقع موانع transparent باشند شکل صورت با استفاده از خطوط و حاشیه های آن قابل یادگیری است.

یکی از ساده ترین شبکه هایی که میتواند این کار را انجام دهد شبکه ی به وجود آمده از چند لایه ی convolution میباشد که به یک شبکه ی FC متصل شده و در آخرین مرحله نیز خروجی را به صورت one-hot یا توابع فعال ساز به وجود می آورد(با توجه به نوع و تعداد کلاس های خروجی).



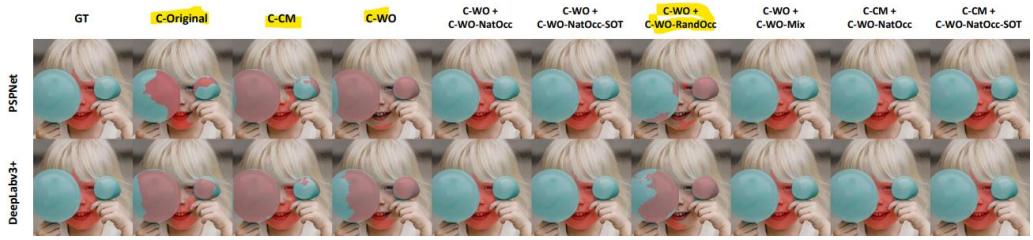
شکل 25: عملیات استخراج ویژگی

البته عملکرد این شبکه وابسته به این است که feature های اصلی صورت توسط occlusion به طور کامل از بین نروند. درصورت از بین رفتن، ممکن است شبکه نتواند به درستی پیش بینی کند.

لازم به ذکر است که عموم شبکه های پیچیده تر که با segmentation و غیره کار میکنند(مانند VGG و Alexnet و...) قابلیت انجام این مساله را حتی در صورت افزایش intensity دارا می باشند و تنها پیاده سازی آنها هزینه ی بیشتری دارد.

(5

برای این مقایسه ابتدا بخش هایی که اختلاف محسوسی با هم دارند استفاده میکنیم.



شکل 26: مقایسه نتایج دو شبکه با کمک تست ها

	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	89.51	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

جدول 2: نتایج سه شبکه

با توجه به اطلاعات بالا میتوان گفته که معماری PSPnet درصورت آموزش دیدن با داده هایی با Occlusion پایین به طور کلی بهتر عمل میکند، اما با بالا رفتن و شدید تر شدن این Occlusion ها در داده های train معماری DeepLabv3+ خروجی های بهتر و با accuracy بالاتری را ارائه میدهد.

مقایسه ی بالا بر روی نمونه هایی با دقت پایین تر انجام شده است که در عکس مثالی از خروجی های آنها قابل مشاهده می باشد.

اما به طور کلی نیز میتوان اینطور گفت که Deeplabv3+ در صورت آموزش دیدن با داده هایی با occlusion بالاتر پیش بینی بهتری میکند که این اختلاف در پیش بینی روی دیتای test با disturbance شدید تر (RealOcc-Wild) محسوس تر می باشد.

پاسخ ۳ – تشخیص بلادرنگ اشیا

1)

در این سوال یک مجموعه داده در اختیار ما قرار داده شده است. این مجموعه داده یک پوشه که تصاویر هستند و یک پوشه که label ها در ان قرار دارد. به ازای هر تصویر در مرحله train و valid یک فایل txt خواهیم داشت که شامل اطلاعاتی از جمله مختصات شی تشخیص داده شده و کلاس متعلق به شی می باشد. بدین ترتیب میتوان مدل را آموزش داد. یک فایل data.yaml نیز وجود دارد که در آن Path دیتا ها را مشخص می شود.

2)

ابتدا YOLOv6 را با کمک دستور git clone وارد نوتبوک میکنیم و کتابخانه های مورد نیاز آن را نصب میکنیم.

```
!git clone https://github.com/meituan/YOLOv6
%cd YOLOv6
!pip install -r requirements.txt
```

شکل 27: نحوه لود کردن **YOLOv6**

سپس کتابخانه torch را import میکنیم.

```
import torch
torch.cuda.is_available()
torch.cuda.get_device_name(0)
```

شکل 28: **import** کردن **torch**

حال دیتاست داده شده را به صورت فایل زیپ در colab اپلود کرده و آن را unzip میکنیم.

حال میخواهیم مدل را train کنیم. با کمک دستور زیر این امر محقق می شود.

```
!python tools/train.py --batch 32 --conf /content/YOLOv6/configs/yolov6m.py --epochs 100 --img-size 416 --data /content/YOLOv6/data.yaml --device 0
```

شکل 29: **train** کردن مدل با **YOLOv6**

در این دستور مقدار batch=32 و تعداد epoch 100 در نظر گرفته شده است و همچنین مدل مدنظر را yolo6m قرار میدهیم چون برای دیتاست در دسترس ما مناسب تر است. سپس ادرس فایل yaml را میدهیم. در این فایل ادرس داده های train و valid و test موجود است.

نتایج اخرین epoch به شکل زیر می شود.

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.705
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.966
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.875
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.695
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.704
 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 ] = 0.593
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.768
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.769
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.750
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.763
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
Results saved to runs/train/exp1
Epoch: 99 | mAP@0.5: 0.9657145490367739 | mAP@0.50:0.95: 0.7052589402090964

Training completed in 0.509 hours.
```

شکل 30: نتایج اموزش مدل **YOLOv6**

نتایج evaluation توسط داده های valid نیز به شکل زیر می شود.

```
!python tools/eval.py --data /content/YOLOv6/data.yaml --img-size 416 --weights /content/YOLOv6/runs/train/exp1/weights/best_ckpt.pt --device 0
```

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.720
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.966
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.910
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.667
 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 ] = 0.605
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.783
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.784
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.683
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.786
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
Results saved to runs/val/exp2
```

شکل 31: نتایج مدل به ازای داده های **validation**

حال میخواهیم با کمک داده های test چند خروجی را مشاهده کنیم. با کمک کد زیر عکس ها به مدل داده می شود و سپس ذخیره می شود.

```
!python tools/infer.py --weights /content/YOLOv6/runs/train/exp1/weights/best_ckpt.pt --source /content/YOLOv6/images/test --yaml /content/YOLOv6/data.yaml --device 0
```

شکل 32: تست کردن مدل

3)

حال میخواهیم خروجی test های بخش قبل را نمایش دهیم. برای نمونه عکس های زیر را داریم.

شکل 33: چند نمونه تست همراه با برچسب **segment** شده