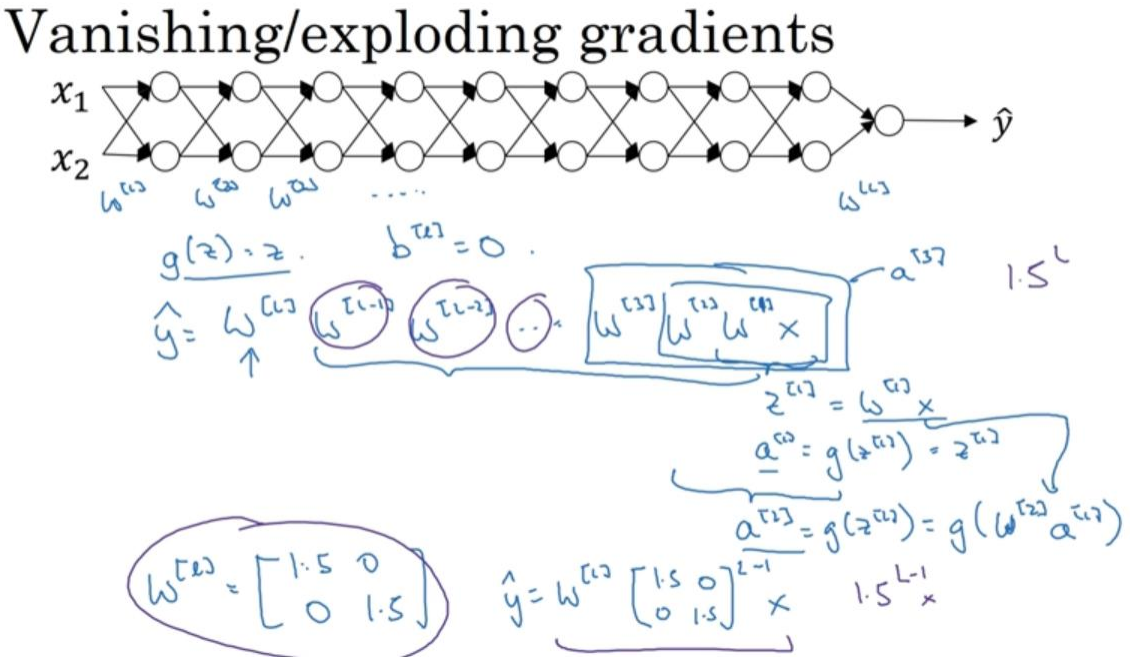
به نام خدا

علی سلطانی – 99521343

سوال 1 :

الف) به صورت کلی این دو مشکل وقتی رخ میدهد که تعداد لایه های hidden ما به شدت زیاد میشود، زیاد شدن تعداد لایه ها باعث میشود که مشتق های متوالی درهم ضرب شده و عملا تابع exp را برای گرادیان به وجود آورند، حال اگر ماتریس ضرایب ما به صورت مثال کمتر از 1 باشد باعث شده که کم کم گرادیان ما که تابعی توانی بر مبنای ضرایب بود ناپدید شود ، زیرا عدد کمتر از یک به توان زیاد کم کم به صفر میل میکند و اگر بیشتر از 1 باشد به اعدادی بسیار بسیار زیاد میل کند که باعث ناپایداری شبکه عصبی ما میشود .

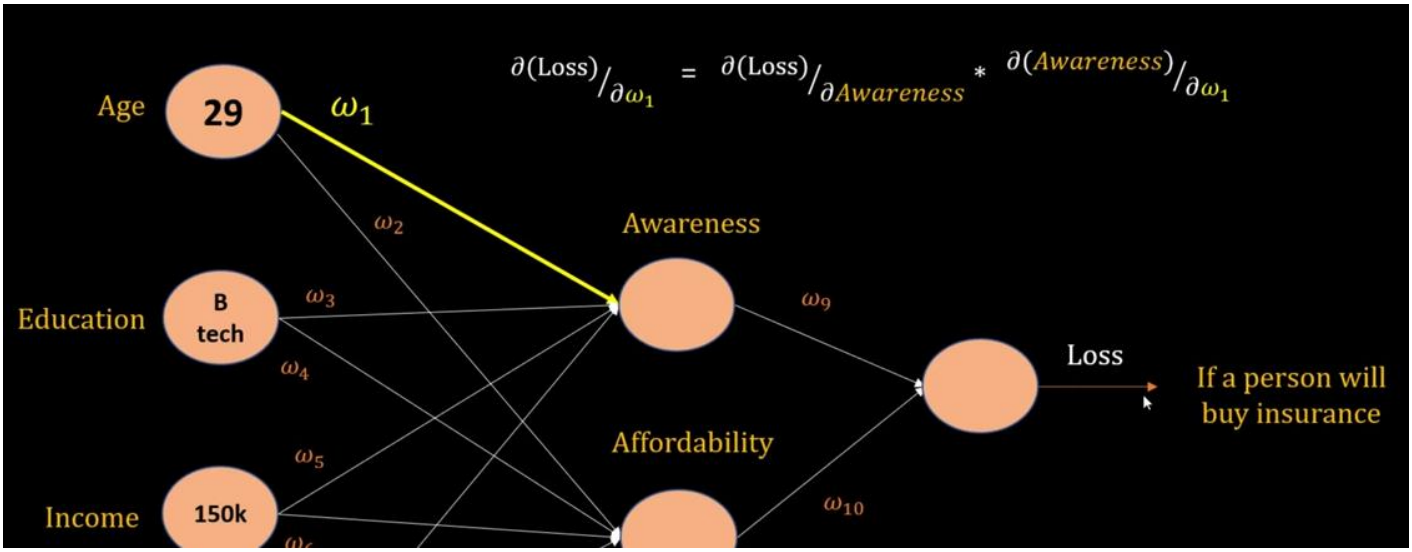
برای توضیح بیشتر میتوان به دو صورت دیگر این مشکل را بیان کرد :



اگر به عکس بالا نگاه کنیم، همان طور که توضیح داده شده اگر b را را نظر نگیریم و تابع فعال ساز(activation function) را تابع همانی مانند تابع relu به صورت تقریبی در نظر بگیریم، میتوانیم بگوییم که عملا بعد از گذشت n لایه، ضرایب ما در هم ضرب میشوند، به این صورت که ضرایب لایه اول در بعدی و همین طور الی اخر ، حال فرض کنیم که ضرایب (w ها) برای هر لایه

کمی بیشتر از 1 باشند، آن وقت داریم همان تابع توانی که در بخش قبل توضیح داده شد تشکیل میشود و ضرایب برای y^ به صورت بسیار زیادی میشوند. برای ضرایب کمتر از هم به همین منوال هست و تابع ما به سمت صفر می رود.

این مشکل را به این صورت هم میتوان توضیح داد :



فرض کنید این قسمتی از شبکه ما باشد ، حال برای بهینه کردن ضریب w1 باید مشتق ضریب بعدی نسبت به loss و همچنین مشتق w1 به ضریب بعدی را حساب کنیم تا گرادیان ما به دست آید. حال اگر تعداد لایه های ما بسیار زیاد باشد ، این مشتق ها در هم ضرب شده و تابع توانی ما را تشکیل میدهند، حال با توجه به این که این مشتق ها بیشتر از 1 یا کمتر باشند ما پدیده ناپدید شدن یا انفجار گرادیان را خواهیم داشت .

این پدیده ها باعث افزایش ارور ما در افزایش تعداد لایه ها خواهد شد، که با بررسی این نشانه ها میتوان این پدیده هارا در شبکه مشاهده کرد :

انفجار گرادیان :

- Loss به شدت بدی دارد و مدل لرن نمی کند .
- Loss به صورت Nan در می آید.
- در هر epoc مدل به شدت تغییر میکند و پایدار نیست .

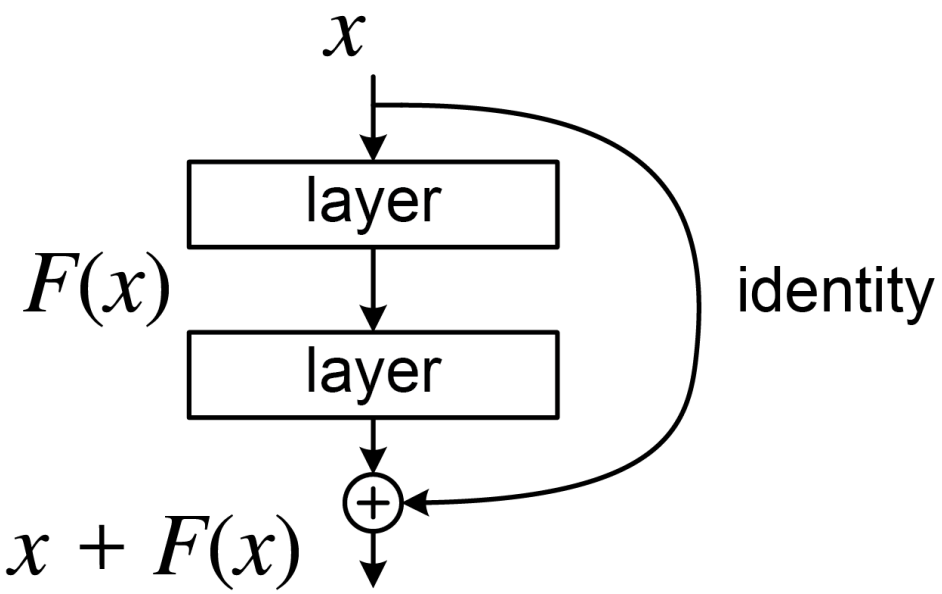
ناپدید شدن گرادیان :

- مدل به کندی رشد می کند و حتی ممکن هست به زودی یاد گرفتن آن متوقف شود .
- ضرایب مدل صفر و یا نزدیک به آن میشود .

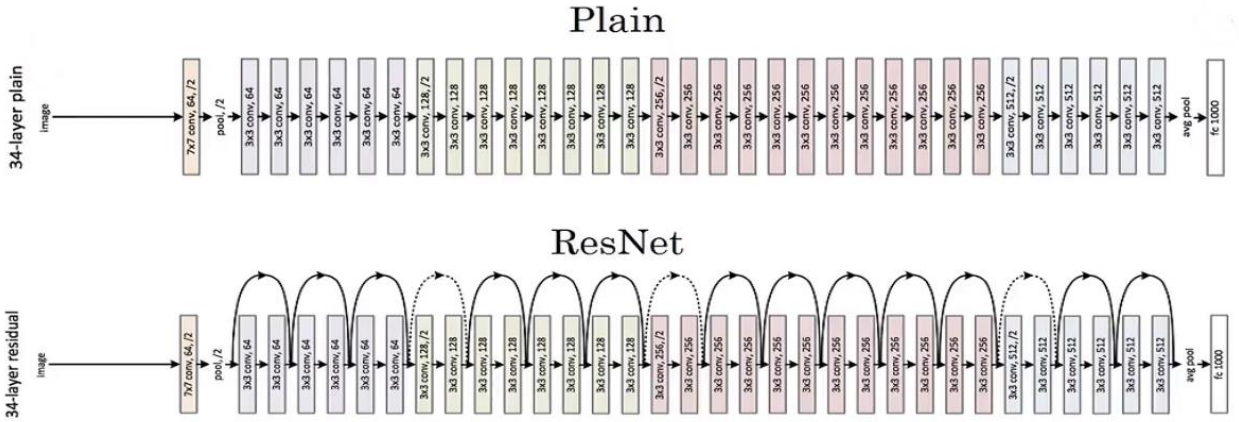
راه حل های آن :

1. می توان از تعداد لایه های کمتری استفاده کرد ، همان طور که معلوم هست مشکل از تعداد لایه های بالا به وجود می آید و با کم کردن آن باعث میشویم که صورت مسئله پاک شده و مشکل نداشته باشیم.
2. می توانیم در مشکل انفجار گرادیان از clip کردن ان استفاده کنیم به این صورت که گرادیان ما حداکثری داشته باشد و بیشتر از آن نشود .
3. یکی از مهم ترین فاکتور ها وزن های اولیه شبکه ما میباشند، که اگر به صورت مناسب انتخاب شوند این مشکل به وجود نمی آید و تا تعداد لایه های خوبی این مشکلات را نخواهیم داشت. به صورت مثال اگر واریانس وزن های ما توزیع یکنواخت داشته باشند، ضرایب به دست آمده در اطراف یک خواهند بود و از این مشکل جلوگیری خواهد شد . که نحوه مقدار دهی اولیه با توجه به تابع فعال ساز میتواند متفاوت باشد که به xavier و ... بستگی دارد .
4. می توان مقدار داده های ورودی هر لایه را پس از چند لایه انتقال داد و آن هارا با وزن ها جمع کرد . این در واقع کاری میباشد که شبکه residual انجام داده هست.

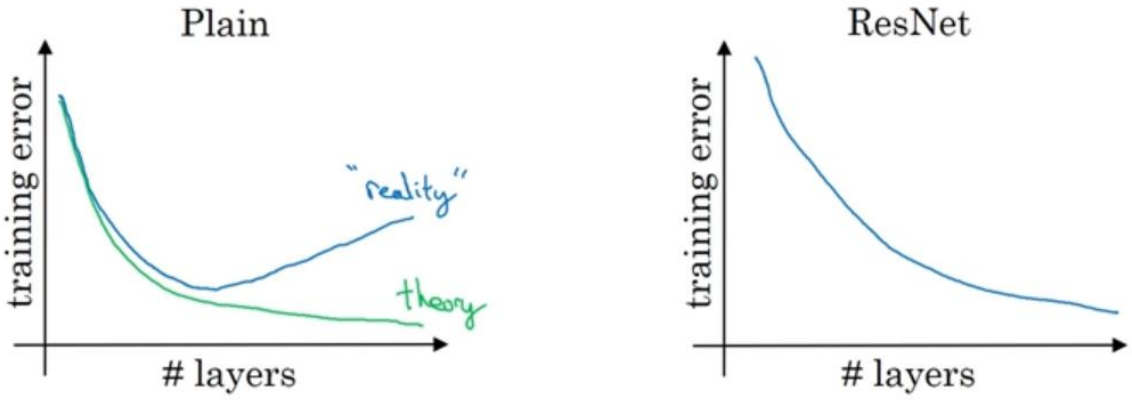
ب) مشکل ناپدید شدن گرادیان رو حل میکند ، شبکه residual که تشکیل شده از بلوک های ان میباشد ، به این صورت عمل میکند که از بلوک هایی متصل به هم تشکیل شده که بعد از هر چند لایه ضرایب دو یا سه لایه قبل را با ضرایب جدید جمع کرده و بعد به تابع فعال ساز آن انتقال میدهند . به این کار shortcut یا skip connection میگویند که باعث میشود اصطلاحا identity ضرایب حفظ شود و به صورت تابع توانی با پایه کمتر از 1 پیشرفت نکنند ؛ این اتفاق به صورت زیر رخ میدهد :



اگر در لایه های میانی ما به ضریب و بایاس صفر برسیم ، با اضافه کردن مقدار قبلی به لایه دوم قبل از تابع فعال ساز مانع از آن میشود که خروجی صفر شود و درواقع گرادیان ما که صفر شده تاثیری در مقدار ضریب خروجی بگذارد . حال اگر کل شبکه متشکل از این لوک ها باشد ، مشکل صفر شدن ضرایب و بایاس از بین میرود :



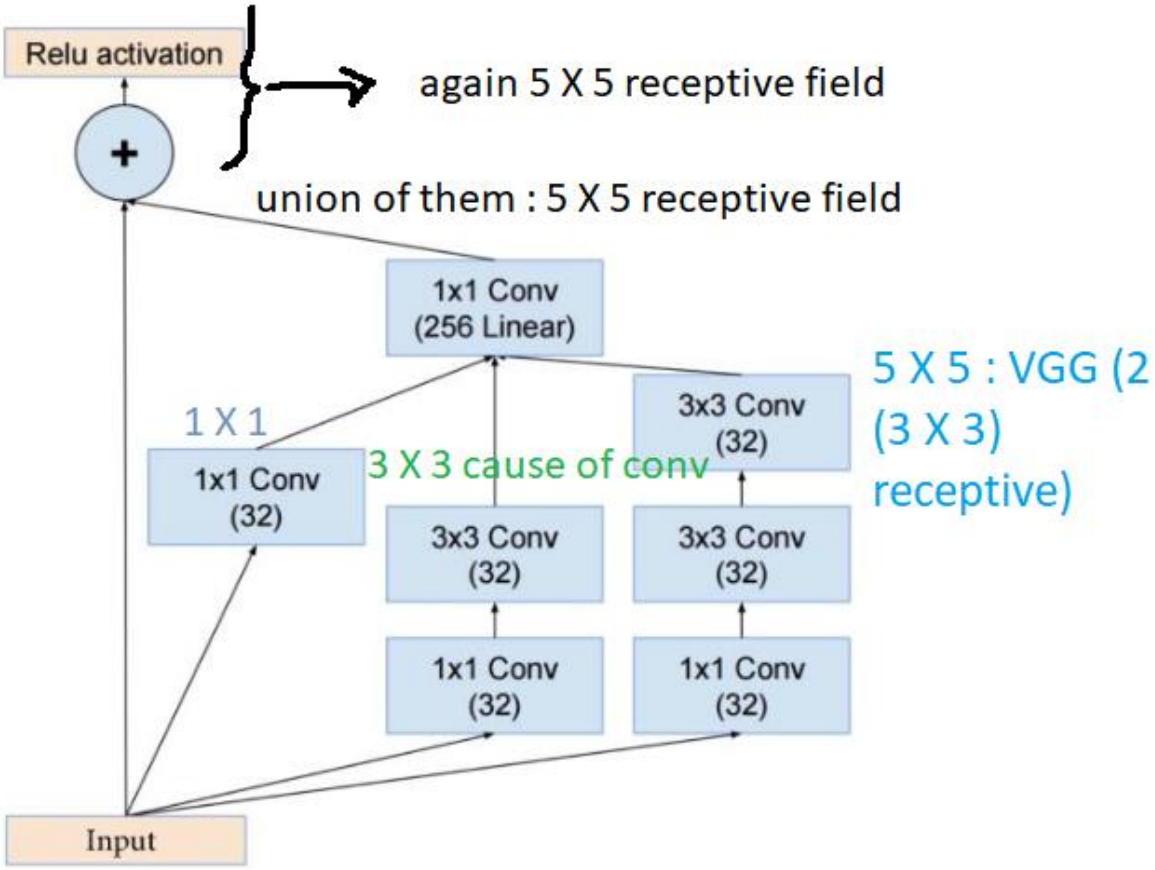
که اگر بخواهیم نتایج دو نوع شبکه بالا را بررسی کنیم داریم :



بنابراین شبکه ResNet با بهره گیری از این بلوک ها و انتقال identity و در واقع ضرایب چند لایه قبلی به لایه های بعدی از صفر شد و ناپدید شدن گرادیان جلوگیری میکند .

سوال 2 :

الف)



Receptive field : طبق تعریف ، به این معنا میباشد که هر پیکسل خروجی از چند پیکسل ورودی اثر گرفته و نشان دهنده آن میباشد ، بنابراین داریم :

شاخه سمت چپ از 1 در 1 استفاده کرده، به این معنا که صرفا هر پیکسل همان پیکسل میباشد ولی عمق آن عوض شده بنابراین هر پیکشل بعد از آن صرفا از همان پیکسل اثر گرفته هست و فیلد آن برابر با 1 در 1 میباشد، بنابراین این لایه کانولوشنی در بقیه شاخه ها هم اثری بر فیلد نمی گذارد.

همان طور که در VGG دیدیم ، ترکیب 2 کانولوشن 3 در 3 منجر به receptive به اندازه 5 در 5 میشود ، بنابراین شاخه سمت راست فیلدی 5 در 5 دارد .(کانولوشن 1 در 1 اثری بر

فیلد ندارد)
در شاخه وسط هم یک کانولوشن 1 در 1 داریم که اثری در فیلد ندارد و یک 3 در 3 که منجر به فیلد 3 در 3 میشود .
حال با کانکت کردن این 3 شاخه و اضافه کردن پدینگ، میتوانیم به این نتیجه برسیم که هر پیکسل از نتیجه پنجره 5 در 5 به دست آمده، در واقع از فیلد ها اجتماع میگیریم، به این دلیل که درست هست که یک شاخه از 3 در 3 اطراف و یکی از 5 در 5 اطراف به دست آمده، ولی در نهایت پیکسل های 5 در 5 اطراف هر پیکسل نهایتا اثر گذار هستند .
اضافه کردن خود ضرایب ورودی به نتیجه این بخش هم اثری بر فیلد نمی گذارد(اثر residual)، زیرا باز هم هر پیکسل فقط بر خود اثر میگذارد و با آن جمع میشود، بنابراین فیلد ما بیشتر نمی شود.
بنابراین در نهایت receptive field به صورت 5 در 5 خواهیم داشت.

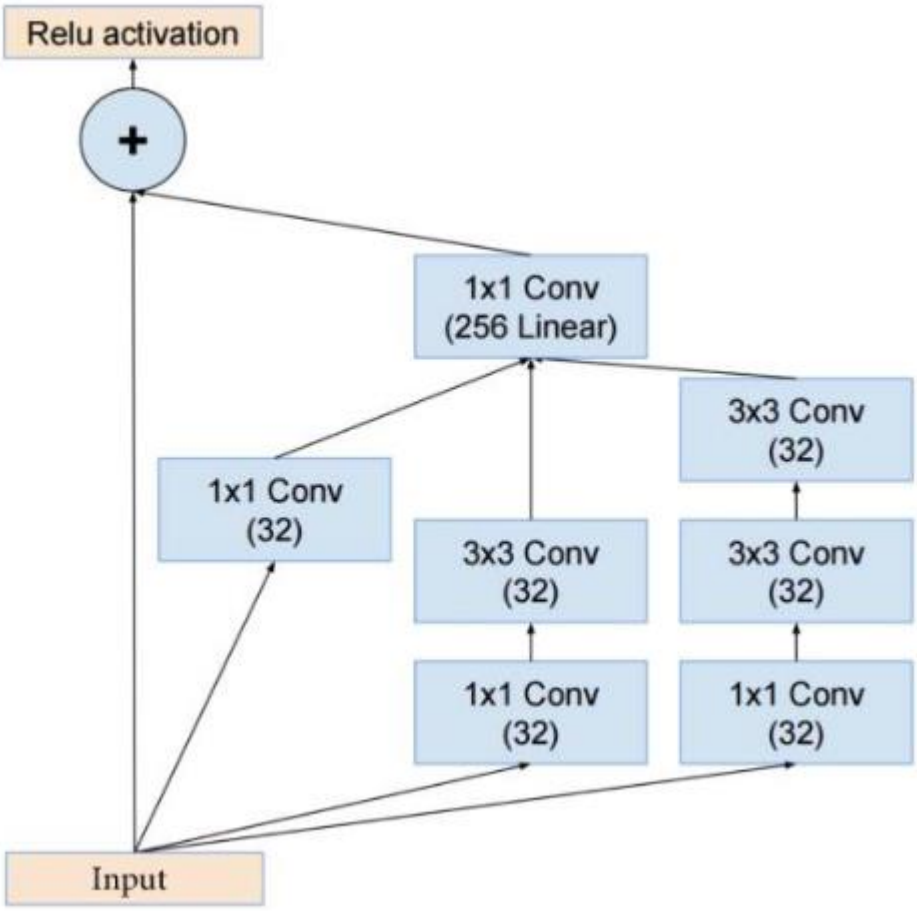
Parameters :

برای شبکه های کانولوشنی، فرمول کلی برای محاسبه پارامتر ها به شرح زیر هست :

$$((m * n * d)+1)* k)$$

که m و n ابعاد فیلتر میباشد ، d عمق هر خانه و به اضافه یک برای بایاس هر فیلتر میباشد، که اگر k فیلتر داشته باشیم ضرب در k میشود . به این دلیل این گونه هست که برای هر فیلتر ضرایب بین ورودی ها ثابت هست و درواقع ضرایب فیلتر اعمالی در نظر گرفته میشود ، بنابراین اگر فیلتر 5 در 5 داشته باشیم ، خانه های 5 در 5 اطراف آن و عمق هایشان هر کدام یک ضریب دارند، در نهایت به هر فیلتر هم یک b اضافه میشود ، حال به ازای هر فیلتر باید تمامی این ها را در نظر گرفت بنابراین ضرب در تعداد فیلتر ها شده و فرمول فوق به دست می آید .

حال برای شکل داریم :



شاخه سمت راست از 1 در یک استفاده کرده بنابراین به ازای 3 کانال ورودی داریم :

32 * (1 + (3 * 1 * 1)) ، که میشود 128 پارامتر. حال لایه بعدی از 3 در 3 استفاده کرده که میشود : 32 * (1 + (32 * 3 * 3)) که لازم به ذکر هست که عمق ما به خاطر لایه قبل به 32 واحد رسید ، نتیجه عبارت بالا میشود : 9248 پارامتر. حال در ادامه یک 32 در 3 در 3 دیگر هم داریم که تغییری در فرمول ایجاد نمی کند بنابراین آن هم 9248 پارامتر خواهد داشت .

شاخه وسط هم به عنوان ورودی از 1 در 1 استفاده کرده طبق بخش قبلی 128 پارامتر خواهد داشت . همچنین کانولوشن بعدی آن 3 در 3 در 32 + 1 در 32 هست که بازهم مثل بخش قبلی 9248 پارامتر خواهد داشت .

شاخه سمت چپ از یک 1 در 1 استفاده کرده که میشود 128 پارامتر .

حال در نهایت این سه شاخه ما کانولوشن 1 در 1 داریم که عمق ورودی آن 96 (به این علت که خروجی سه شاخه در آن کانکت میشود و پشت هم قرار میگیرند) هست و به 256 فیلتر باید مپ شود . لازم به ذکر هست که پدینگ اعمالی تغییری در تعداد پارامتر قابل اموزش نمی گذارد . بنابراین طبق فرمول داریم : 256 * (1 + (1 * 1 * 96)) که میشود : 24,832 پارامتر قابل اموزش هست . در بخش جمع هم تمامی مقادیر ورودی هم جمع شده که در نهایت این بخش اثری بر تعداد پارامتر اموزش نمی گذارد . بخش ReLU هم بخشی میباشد که صرفا تابع ReLU بر مقادیر اعمال میشود، بنابراین این بخش هم پارامتری ندارد، اگر تمامی مقادیر فوق را با هم جمع کنیم برای این بلوک داریم :

128 + 9248 + 128 + 9248 + 128 + 9248 + 24832 = 52,960 تعداد کل پارامتر های موجود این بخش هست .

ب) A)

در این بخش از کانولوشن عادی استفاده کرده بنابراین داریم :
receptive field : ترکیب دو کانولوشن عادی 3 در 3 طبق معماری VGG میدانیم که میشود 5 در 5 ، زیرا هر پیکسل در مرحله اول از 3 در 3 اطراف اثر میگیرد و در مرحله بعدی از هر 3 در 3 که هر کدام 3 در 3 بودند یک خانهدر می آید ، با توجه به این که تکراری میشود خانه ها، عملا از هر ظرف دو خانه اضافه میشود، بنابراین 5 در 5 خواهد بود ترکیب این دو ، بنابراین فیلد آن ها 5 در 5 هست .

قابل توجه هست که به علت valid بودن پدینگ، ما برای یک پیکسل از بالا و یک پیکسل از پایین عملا خروجی نخواهیم داشت و خروجی در هر مرحله 1 خانه از هر طرف کم میشود .

Parameters : طبق فرمولی که در سوال قبلی گفته شد برای این دو شبکه داریم :

(16 * (1 + (3 * 3 * 3)) = 448 پارامتر .
حال برای کانولوشن دوم داریم به ازای عمق 16 :
(32 * (1 + (16 * 3 * 3)) = 4,640 پارامتر.

B) برای این قسمت از کانولوشن محلی استفاده کرده ، به این معنا که برای هر قسمت از از تصویر فیلتر متفاوتی استفاده می کند که بنابراین به ازای هر پیکسل نهایی ما یک فیلتر داشتیم و برای هر فیلتر با توجه به اندازه و عمق و بایاس ان پارامتر اموزش داریم .

receptive field : در این بخش هم مانند بخش قبل هر پیکسل از 5 در 5 اطراف آن به دست آمده، درست هست که ممکن هست برای یک خانه دو یا سه فیلتر متفاوت در انظر گرفته شده باشد ، ولی در نهایت هر پیکسل ما از 5 پیکسل اطراف آن به دست آمده و طبق تعریف receptive field ، 5 در 5 میشود فیلد ما .

قابل توجه هست که به علت valid بودن پدینگ، ما برای یک پیکسل از بالا و یک پیکسل از پایین عملا خروجی نخواهیم داشت و خروجی در هر مرحله 1 خانه از هر طرف کم میشود .

parameter : طبق توضیحات عملکرد آن ها داریم که در نهایت به اندازه هر خانه خروجی بعد از اعمال فیلتر یک فیلتر خواهیم داشت و 16 بار این اتفاق تکرار میشود ، بنابراین با توجه به این که n در n هست و خروجی n-2 در n-2 خواهد بود داریم :

16 * n-2 * n-2 + (n-2 * n-2) * 16 * (3 * 3 * 3) خواهد بود.

حال برای قسمت بعدی هم به همین ترتیب داریم :

32 * (n-4 * n-4) + (n -4 * n-4) * 32 * (16 * 3 * 3) خواهد بود .

توضیحات محاسبات :

قسمت سمت چپ فقط ضرایب هر فیلتر بودن بایاس هست و قسمت دوم سمت راست قسمت بایاس آن میباشد . سمت چپ ما هر فیلتر به اندازه آن ضریب خواهیم داشت که برای مرحله اول عمق 3 میباشد بنابراین هر فیلتر بر 3 * 3 * 3 خانه ضریب دارد (با احتساب عمق آن ها)، حال 16 فیلتر هم داریم و این که این محلی هست ، یعنی به ازای هر خانه خروجی فیلتر جدایی زده شده بنابراین در تعداد خانه های خروجی که n-2 درn-2 باشد ضرب میشود .

در ادامه به ازای هر فیلتر یک بایاس داریم که میشود به ازای هر خانه خروجی در 16 بار فیلتر که معادل با 16 * n-2 * n-2 میباشد .

برای قسمت دومش هم به همین ترتیب میباشد .

حال اگر بخواهیم این دو را با هم قیاس کنیم به سادگی میابیم که بخش اول تعداد پارامتر

های بسیار بسیار کمتری دارد به دلیل مشترک بودن ضرایب ، ولی بخش دوم به دلیل مشترک نبودن بسیار پارامترهای بیشتری دارد ، ولی هردو receptive field یکسانی دارند، به این معنا که حالت خیلی اطلاعات بیشتری ولی به صورت محلی از همان مقدار ورودی میدهد ولی هزینه بسیار بالتری دارد و ویژگی ها فقط محلی هستند و ممکن هست با تغییر تصویر خوب نباشند برای پیشبینی مدل .

سوال 3 :

الف)

```
class_names = ("Airplane", "Automobile", "Bird", "Cat", "Deer",
               "Dog", "Frog", "Horse", "Ship", "Truck")

##################################################
################### YOUR CODES ###################
(x_train, y_train), (x_val, y_val) = cifar10.load_data()
##################################################

print('Training:', x_train.shape, y_train.shape)
print('Validation:', x_val.shape, y_val.shape)
```

با استفاده از load_data داده را لود میکنیم ، و بعد با تقسیم بر 255 عکس ها را نرمالایز کرده و با استفاده از to_categorical داده ها را به صورت one-hat لیبل بندی کرده و به صورت ارایه ای از 0 و فقط 1 یک که همان لیبل هست تبدیل میشود تا مدل بتواند آن را لرن کند .

A) در این قسمت مدل خود را می سازیم :

```
################################################
################### YOUR CODES ##################
model = keras.Sequential([
  keras.Input(shape=(32 , 32 , 3)),
  layers.Conv2D(16, kernel_size=(3, 3), padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=(2, 2)),
  layers.Conv2D(32, kernel_size=(3, 3), padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=(2, 2)),
  layers.Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=(2, 2)),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(10, activation="softmax")
])

################################################

model.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_41 (Conv2D)          (None, 32, 32, 16)        448

 max_pooling2d_35 (MaxPoolin  (None, 16, 16, 16)       0
 g2D)

 conv2d_42 (Conv2D)          (None, 16, 16, 32)        4640

 max_pooling2d_36 (MaxPoolin  (None, 8, 8, 32)         0
 g2D)

 conv2d_43 (Conv2D)          (None, 8, 8, 64)          18496

 max_pooling2d_37 (MaxPoolin  (None, 4, 4, 64)         0
 g2D)

 flatten_15 (Flatten)        (None, 1024)              0

 dense_19 (Dense)            (None, 128)               131200

 dense_20 (Dense)            (None, 10)                1290

=================================================================
Total params: 156,074
Trainable params: 156,074
Non-trainable params: 0
_________________________________________________________________
```

مدل ساخته شده متشکل از سه لایه کانولوشنی و مکس پولینگ هست و همچنین دو لایه dense جهت دسته بندی نهایی ، که در نهایت با استفاده از softmax و 10 dense لایه به این علت که دسته بندی 10 تایی هست ساخته شده و گذاشته میشود.

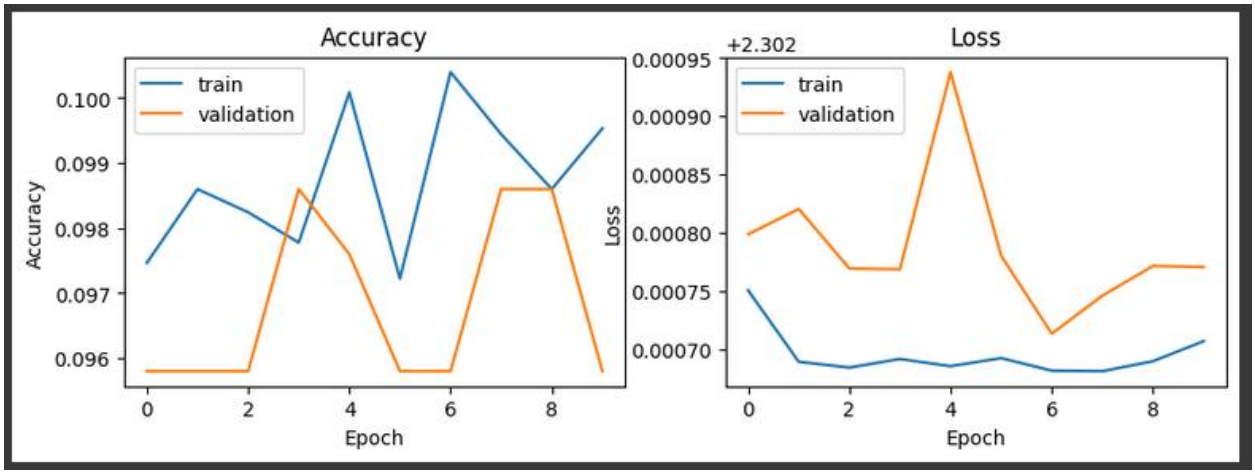
حال صرفا مدل را کامپایل کرده و در واقع فانکشن loss و نحوه اپتیمایز ان را که adam باشد ست کرده و مدل را فیت میکنیم، که در این بخش بچ سایز و تعداد epoc را ست میکنیم .

```
[62] ################################################
     ################### YOUR CODES ###################
     model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
     ################################################
```

Train your model for `10 epochs` and proper `batch_size`.

```
################################################
################### YOUR CODES ###################
history = model.fit(x_train, y_train, batch_size=128, epochs=10, validation_split=0.1)
################################################
```

نتیجه فیت شدن مدل به شرح زیر هست :



که مدل در اوایل به accuracy خوبی هم در ترین هم در ولیدیشن میرسد ، اما همان طور که انتظار می رود بعد از مدتی overfit میکند و دقت آن در دو ایپک اخر در ترین بالا می رود و شروع به حفظ داده ترین میکند، اما در ولیدیشن دقتش کم میشود.

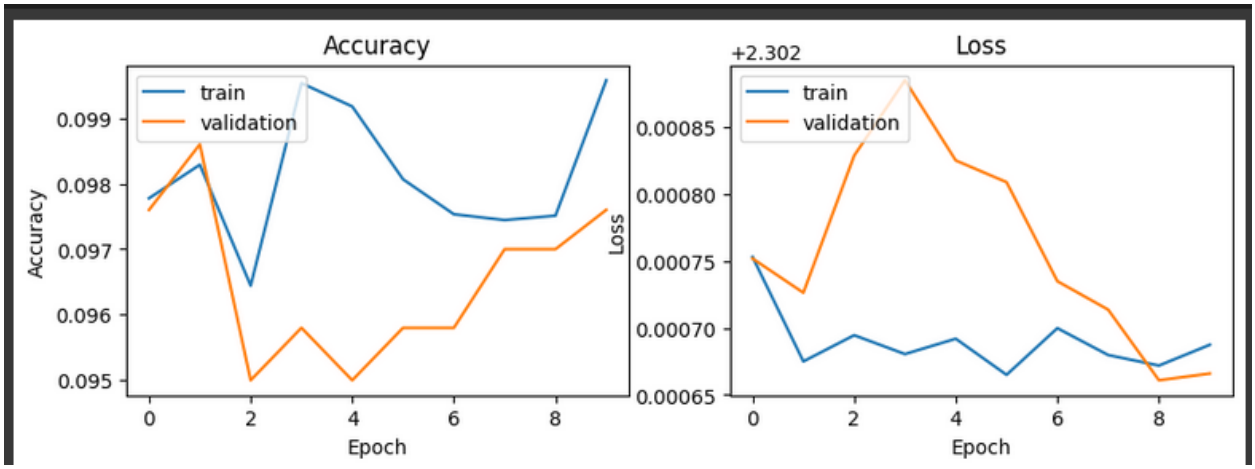
B وC) در این قسمت داده افزایی را با استفاده از قسمت preproccessing انجام میدهیم :

```
data_augmentation = keras.Sequential(
    [
        keras.layers.experimental.preprocessing.RandomFlip("horizontal"),
        keras.layers.experimental.preprocessing.RandomRotation(0.1),
        keras.layers.experimental.preprocessing.RandomZoom(0.1),
        keras.layers.experimental.preprocessing.RandomContrast(0.1),
    ]
)
model = keras.Sequential([
  keras.Input(shape=(32 , 32 , 3)),
  data_augmentation ,
  layers.Conv2D(16, kernel_size=(3, 3), padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=(2, 2)),
  layers.Conv2D(32, kernel_size=(3, 3), padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=(2, 2)),
  layers.Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=(2, 2)),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(10, activation="softmax")
])
```

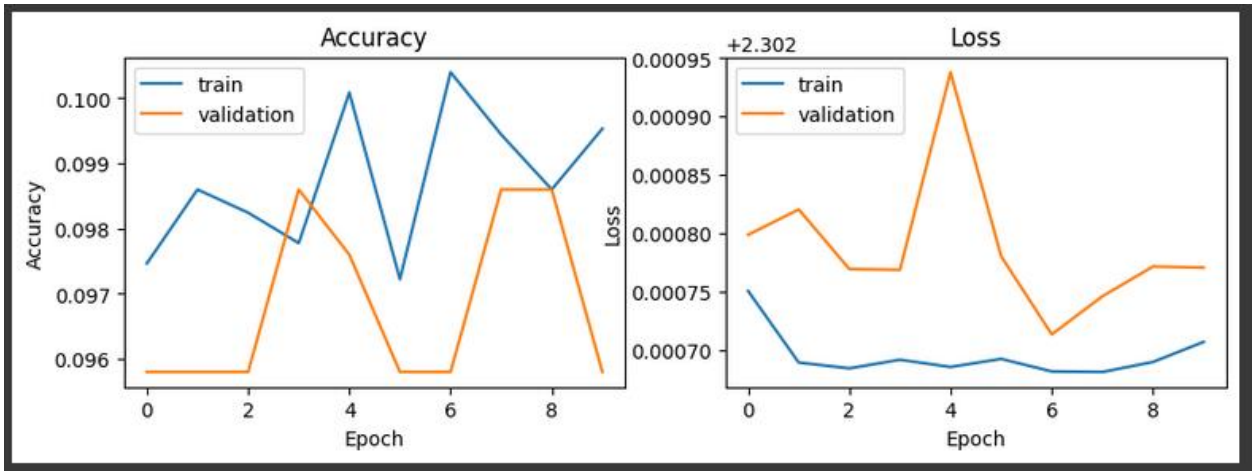
چهار تکنیک داده افزایی به کار برده شده که عبارت است از فلیپ، زوم ، روتیشن و تغییر کانترست.

حال کافی هست سطر این مدل را به مدل شبکه عصبی اضافه کنیم، که صرفا یک سطر از مدل ترتیبی ما میشود .

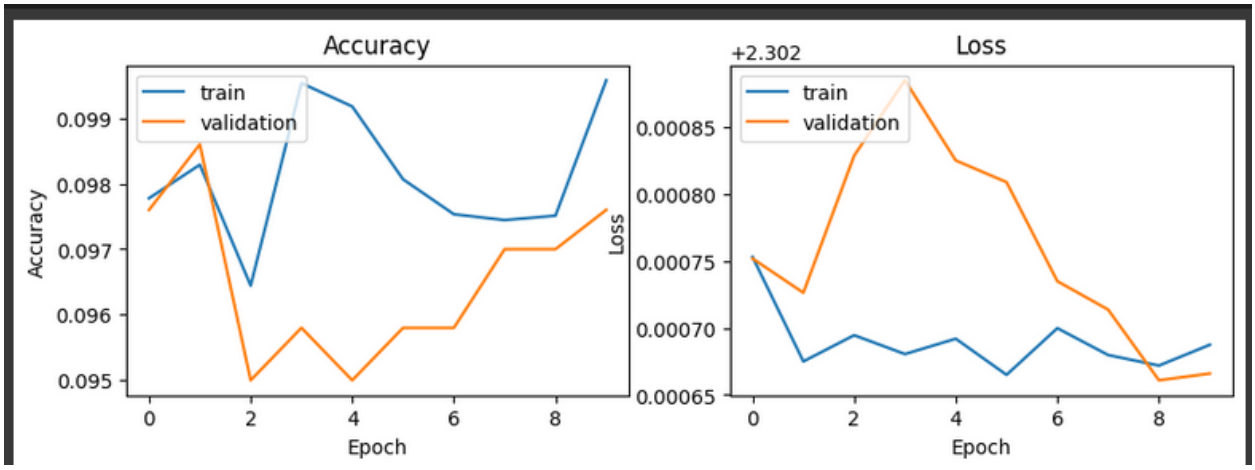
نتایج حاصل از این داده افزایی باعث میشود که مدل ما برای epoc های بالاتر هم توانایی ترین داشته باشد و overfit نشود:



همان طور که مشاهده میکنید epoc های 9 و 10 ما هم برای ولیدیشن هم برای ترین مقدار دقت ما افزایش می یابد و در واقع داده های اموزشی ما را حفظ نمی کند و سعی می کند ویژگی های را در بیارد که عمومی تر باشد بنابراین دقت و loss ما هردو بهتر شده و مشکل overfitting برطرف میشود(figure 2) ، ولی از طرفی در نمودار قبلی (figure 1) نمودار ترین هم برای دقت هم برای loss بهتر شده و لی فاصله گرفته و این دو نمودار برای ولید بدتر میشود، آن هم به دلیل حفظ داده و وپیدا کردن ویژگی هایی میباشد که جامع پذیر نیستند .



without data aug1 Figure



with augmentation2 Figure

د) کمی تلاش در ریسایز کردن انجام دادم و مدل رو ساختم ، ولی برای ولیدیشن دیتا تو فیت به مشکل خوردم .

ه)

سوال 4 :

الف) به این معنا میباشد که که از هر تعداد خانه به صورت یکی در میان یا تعداد بیشتر را اثر میدهد، برای این مورد که اطلاعات به طور مثال دو پیکسل کنار هم زیاد تفاوت خاصی ندارد، pooling هم همین کار رو میکند منتها با این تفاوت که همه اطلاعات رو از بین نمی برد، صرفا در آن پنجره مشخص شده میانگین میگیرد و یا ماکسیموم آن را در نظر میگیرد، این کار برای کم کردن پارامتر ها مخصوصا در لایه قبل dense میباشد ، زیرا این تعداد پارامتر ها باعث overfit شدن مدل می شود زیرا باعث میشود فیچر هایی استخراج شود که تفاوت کم در آن ها اثر گذار نباشد، ولی از طرفی مقداری از اطلاعات از بین می رود.

ب) 1) برای توابع میانی از Leaky ReLU استافده میکنم، زیرا باید از مشکل dead نورون ها جلوگیری کرد همچنین مدل برای یادگیری بسیار سریع تر خواهد بود و تابع فعال ساز معمولی میباشد. برای تابع نهایی میتوان از سیگوید استفاده کرد، به این علت که مسئله باینری میباشد و باید بین خوب و بد پیش بینی کرد و بسیار برای این مسئله مناسب می باشد . ولی میتوان از سافت مکس هم استفاده کرد، چون به وصرت عمومی تابع دسته بند خوبی میباشد .

2) با توجه به این که مسئله رگرسیون نیست و مسئله به صورت باینری میباشد ، میتوان از باینری کلاس انتروپی استفاده کرد که بهترین loss میباشد .

3) به صورت کلی هردو را تعریف کرده و بعد قیاس میکنیم :

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

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

طبق فرمول های بالا، پرسیژن به ما نسبت درست ها به نسبت کل پیشبینی هایی که برای کلاس کرده را میدهد، ولی ریکال نسبت درست ها به کل موجودیت هایی که در کلاس هستند را میدهد . حال ما به دنبال این هستیم تمامی معیوب ها را تشخیص دهیم ، بنابراین نسبت معیوب هایی که تشخیص میدهیم به نسبت کل معیوب ها باید زیاد باشد، بنابراین باید ریکال بیشتری داشته باشیم .

ج) 1) برای شناسایی موضوع متن زیاد شبکه های کانولوشنی خوب نیستند به این علت که برای ورودی هایی با سایز محدود طراحی شده اند و از طرفی هم برای موقعیت مکانی کنار هم اجزازی ورودی مناسب اند ، ولی در متن موقعیت مکانی یک کلمه نسبت به جمله مورد بررسی هست ، و همچنین کاربرد معنایی آن ها ، بنابراین شبکه های کانولوشنی برای این موضوع مناسب نیستند .

2) اگر داده ورودی به شکل های خاصی تبدیل شود (MFCCs) ، میتواند خوب کار کند، زیرا عملا میتواند به صورت یک بردار دو بعدی به آن نگاه کرد ، بنابراین CNN ها میتوانند در تشخیص مناسب باشند .

3) به این علت که رابطه ای در فضای spatial بین ورودی ها وجود ندارد ، احتمالا خوب کار نمیکنند، ولی برای این مورد میتوان از time series استفاده کرد که دقیق این موضوع را مورد بررسی قرار می دهند ، ب هصورت کلی چون هیچ ارث بری وجود ندارد نمی توان از CNN ها استفاده کرد ، یعنی عملا رفتار آن ها به صورت کلی شکل خاصی را پدید نمی آورد .

د) شبکه های کانولوشنی مشکلات زیادی دارند که میتوان به بعضی از آن ها اشاره کرد :

1. به شدت به موقعیت مکانی وابسته هستند، و برای داده های ورودی که رابطه مکانی بین آن ها وجود نداشته باشد خوب کار نمی کنند .
2. به دلیل محاسبات بالا در انجام کانولوشن ها میتوانند بسیار زمان گیر و هزینه بر باشند .
3. به شدت به مقدار داده ورودی برای ترین شدن وابسته هستند و حتما مقدار داده بسیار زیاد نیاز دارند تا به خوبی ترین شوند، برای همین مقدار خوبی از تکنیک ها پدید امدند.
4. با وجود تکنیک های موجود، بازهم overfitting در تعداد لایه های بالا و پیچیده وجود دارد و نمیتوان از ان چشم پوشی کرد .
5. برای داده هایی به غیر از تصویر لزوما کارکرد خوبی ندارند و نمی توان آن ها را در کنار سایر معماری ها قرار داد و استفاده کرد ، به شدت از لحاظ کارکرد پایین تر میباشند.

سوال 5 :

الف) در نوتبوک انجام شده .

ب) این دو مورد هردو فانکشنی برای محاسبه loss میباشند،

binary cross entropy تفاوت بین کلاس های پیشبینی شده و واقعیت را می سنجد ، بیشتر برای کلاس بندی کردن عکس ها و این که در یک کلاس مشخص هست یا نه استفاده میشود.

intersection over union متودی برای بررسی دقت تشخیص اشیا و segmentation کردن آن ها میباشد ، به این صورت عمل میکند که مقدار ناحیه مشخص شده به واقعیت را اندازه میگیرد ، بنابراین BCE برای دسته بندی باینری بین دو کلاس برای یک عکس میباشد ، ولی IoU برای بررسی کردن مقدار کارایی الگوریتم تشخیص اشیا میباشد .

توضیحات کد :

به صورت کلی عینا مراحل خواسته شده پیاده سازی شده ، فقط چند موردی اضافه هست :

1 – برای دانلود کردن دیتاست از gdown استفاده کردم و با استفاده از unzip آن را unzip کردم .

```
# seperate each file path and store in lists
for number in os.listdir(data_dir):
  for name in os.listdir(os.path.join (data_dir , number )):
    if name.endswith('.bmp'):
      if '_label' in name :
        labels.append(os.path.join(os.path.join (data_dir , number ), name))
      else :
        images.append( os.path.join(os.path.join (data_dir , number ), name))
# read and resize and save images
for path_image in images :
  image = Image.open(path_image)
  resized = image.resize((256 , 256))
  resized.save(os.path.join(image_root , os.path.basename(path_image).split('.')[0] + '.png' ) , 'PNG')
# do it for labeled too
for path_image in labels :
  image = Image.open(path_image)
  resized = image.resize((256 , 256))
  name =  os.path.basename(path_image).split('.')[0]
  name_without_label =  os.path.basename(name).split('_')[0]
  resized.save(os.path.join(label_root , name_without_label + '.png' ) , 'PNG')

```

با استفاده از listdir در تمامی dir ها گشتم و هر فایل bmp را جدا کردم، حال اگر label در اسم بود در لیست دیگری ذخیره کردم .

بعد بر روی هر دو لیست فور زدم و پس از ریسایز آن ها با استفاده از dir های داده شده ذخیره شان کردم . برای label ها ، label را از اسمشان حذف کردم.

حال صرفا با استفاده از فامکشن قبلی ، دو دیتا فریم ساختم که حاوی مسیر آن ها بود و یک ستون اضافه کردم برای مسیر ماسک ها .

```
[ ]  print("Train set:  ", len(os.listdir("/content/train")))
     print("Train masks:", len(os.listdir("/content/train_masks")))
```

```
Train set:   859
Train masks: 859
```

```
## Call the dataframe_creation function with apprieciate arguments to get a
## suitable dataframes for images and masks. Call the images pathes column's name "imag
## and column for masks name "mask_name"
## In the end, create a new column in df of images called "mask_path" and fill it with
df_images = dataframe_creation(image_path="/content/train", name='image_path')
df_masks = dataframe_creation(image_path="/content/train_masks", name='mask_path')
```

```
[ ]  df_images['mask_path'] = df_masks.iloc[: , -1 ]
```

```python
def data_augmentation(img, mask_img):
    """
    A function for data augmentation.
    We wanna just do some flips.
    Just make a random number, if it was greater than 0.5 do a lef_right flip
    hint:
      https://www.tensorflow.org/api_docs/python/tf/random/uniform
      https://www.tensorflow.org/api_docs/python/tf/image/flip_left_right
    Arguments:
      img: image tensor
      mask_img: mask image tensor
    return:
      img, mask_img
    """

    ################################################
    ############## YOUR CODES GO HERE ##############
    num = tf.random.uniform(shape=(),minval=0,maxval=1,dtype=tf.dtypes.float32)
    if num > 0.5 :
      img = tf.image.flip_left_right(img)
      mask_img = tf.image.flip_left_right(mask_img)
    return img, mask_img
    ################################################
```

در صورت عدد رندوم، با استفاده از تابع فلیپ ، عکس را میچرخاند کمی.

```python

def preprocessing(path, mask_path):
    '''
    Do the usual preprocessing steps for image processing algorithms
    Read image tensors. decode them, resize to img_size, cast them fo float dtype and normalize b
    Hint:
      https://www.tensorflow.org/api_docs/python/tf/io/read_file
      https://www.tensorflow.org/api_docs/python/tf/io/decode_jpeg
      https://www.tensorflow.org/api_docs/python/tf/image/resize
      https://www.tensorflow.org/api_docs/python/tf/cast
    Set channels in decoding to 3
    Arguments:
      path: image path
      mask_path: mask image path
    return:
      pre_processed image and mask image tensors
    '''
    ################################################
    ############### YOUR CODES GO HERE ###############
    image = tf.io.read_file(path)
    image_mask = tf.io.read_file(mask_path)
    decoded_image = tf.io.decode_jpeg(image , channels = 3)
    decoded_image_mask = tf.io.decode_jpeg(image_mask , channels = 3)
    resized_image = tf.image.resize(decoded_image, img_size , method='nearest')
    resized_image_mask = tf.image.resize(decoded_image_mask, img_size , method='nearest')
    casted_image = tf.cast(resized_image, tf.float32) / 255.0
    casted_image_mask = tf.cast(resized_image_mask, tf.float32) / 255.0

    return casted_image, casted_image_mask
```

عکس را دیکود کرده تا به ابجکن تنسور فلو تبدیل کنیم ، ریسایز کرده و با تقسیم بر 255 نرمالایز کرده و با فانکشن کست ، کست میکنم .

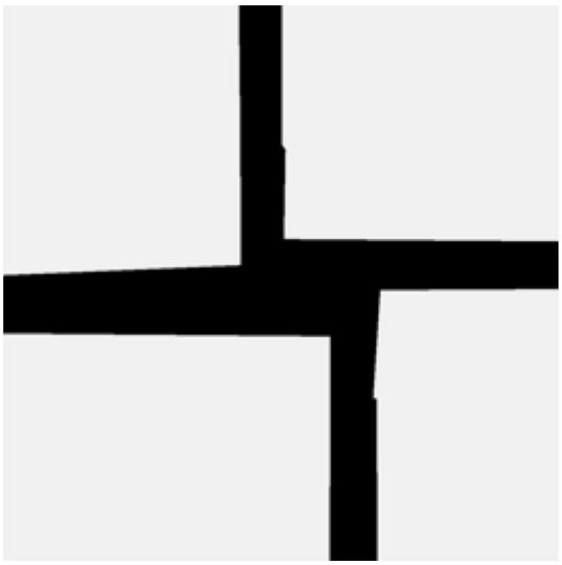
```python
def create_dataset(df, train = False):
    '''
    A function for applying preprocessing and augmentation steps.
    Augment data just in train mode.
    First make a Dataset of tensors to reach high speed and ability.
    Then apply needed steps.
    For creating dataset, please use tensorflow-tf-data-dataset-from_tensor_slices to get
      a dataset of images and correspondig masks path. use values of image_path and mask_path columns of your dataframe
    Then use map function of created ds and call above functions respectively.
    use tf.data.AUTOTUNE in map function
    Hint:
      https://www.geeksforgeeks.org/tensorflow-tf-data-dataset-from_tensor_slices/
      https://www.tensorflow.org/api_docs/python/tf/data/Dataset
    Arguments:
      df: dataframe of images with masks path.
      train: boolean for switching between train and inference mode.
    return:
      dataset
    '''
    ################################################
    ############## YOUR CODES GO HERE ##############
    tf_Dataset = tf.data.Dataset.from_tensor_slices((df["image_path"].values , df["mask_path"].values))
    preproccessed_ds = tf_Dataset.map(preprocessing, num_parallel_calls=tf.data.AUTOTUNE, deterministic=False)
    if train :
      preproccessed_ds = preproccessed_ds.map(data_augmentation, num_parallel_calls=tf.data.AUTOTUNE, deterministic=False)
    return preproccessed_ds
    ################################################
```

در این تابع به اسلایس های تنسور فلو تبدیل کرده و با استفاده از آن ها دیتا ست را میسازم، اگر هم ترین بود داده افزایی رخ میدهد .

```python
## The last step for creating a tf.data.Dataset is to batch and shuffle them
## for this please cache train dataset, then shuffle in the amount of BUFFER_SI
## batch them in the amount of BATCH_SIZE and repeat this process. Finally pref
## buffer_size=tf.data.AUTOTUNE (https://www.tensorflow.org/api_docs/python/tf/
## Hint: name of needed functions are in the description, just call
## for valid_dataset, just batch in the amount of BATCH_SIZE
################################################
############### YOUR CODES GO HERE ###############
train_dataset = train.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE).repeat()
train_dataset = train_dataset.prefetch(buffer_size=tf.data.AUTOTUNE)
valid_dataset = valid.batch(BATCH_SIZE)
################################################
```

به ترتیب کش کرده ، به اندازه بافر شافل میکنم و به اندازه بچ ، بچ کرده و این کار را تکرار میکنم، و بعد هم پرفچ می کنم با استفاده از تابع فوق. برای ولید ه مصرفا بچ میکنم.

Input Image



True Mask

نتیجه !

```
base_model = tf.keras.applications.MobileNetV2(input_shape=img_size, include_top=False)
layer_names = [
    'block_1_expand_relu',   # 64x64
    'block_3_expand_relu',   # 32x32
    'block_6_expand_relu',   # 16x16
    'block_13_expand_relu',  # 8x8
    'block_16_project',      # 4x4
]
base_model_outputs = [base_model.get_layer(name).output for name in layer_names]
down_stack = tf.keras.Model(inputs=base_model.input, outputs=base_model_outputs)

down_stack.trainable = False
```

در این قسمت با استفاده از لیست لایه ها ، خروجی شبکه را میگیرم و در لیست ذخیره میکنم، و trainable بودن آن را هم false قرار میدهم تا این قسمت فریز شود ، تاپ آن را هم با false قرار دادن اینکلود نمیکنم .

```
    initializer = tf.random_normal_initializer(0., 0.02)

    ################################################
    ############## YOUR CODES GO HERE ##############
    result = tf.keras.Sequential()
    result.add(tf.keras.layers.Conv2DTranspose(filters, size, strides=2,padding='same',kernel_initializer=initializer,use_bias=False))
    result.add(tf.keras.layers.BatchNormalization())
    # result.add(tf.keras.layers.ReLU()) # add this to have non_linear
    ################################################

    return result

up_stack = [
    upsample(512, 3),  # 4x4 -> 8x8
    upsample(256, 3),  # 8x8 -> 16x16
    upsample(128, 3),  # 16x16 -> 32x32
    upsample(64, 3),   # 32x32 -> 64x64
]
```

لایه های upsample را به صورت کلی میسازم، که تشکیل شده از کانلوشن ترنسپوز و بچ کردن آن ها .

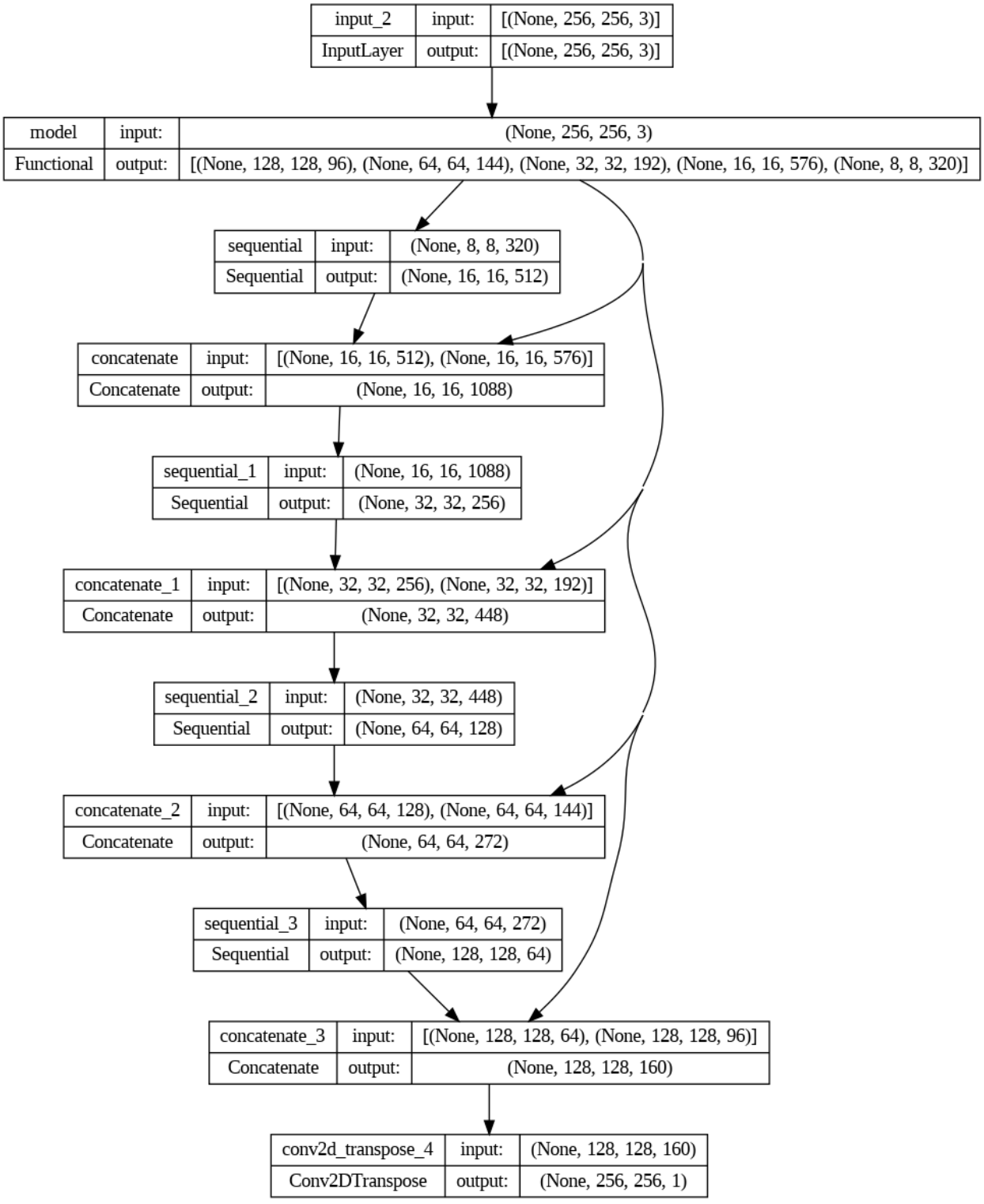
```
############## YOUR CODES GO HERE ##############
input_layer = tf.keras.layers.Input(shape=img_size)
output_encoder = down_stack(input_layer)
x = output_encoder[-1]
output_encoder = reversed(output_encoder[:-1])
for up, skip in zip(up_stack, output_encoder):
  x = up(x)
  concat = tf.keras.layers.Concatenate()
  x = concat([x, skip])

last = tf.keras.layers.Conv2DTranspose(
filters=output_channels, kernel_size=3, strides=2,
padding='same' , activation = 'sigmoid')
x = last(x)

return tf.keras.Model(inputs=input_layer, outputs=x)
```

صرفا تمامی موارد خواسته شده را انجام میدهیم. صرفا نوشتن تابع و قرار دادن هایپر پارمتر ها طبق خواسته میباشد . توضیحات مدل : هر قسمت خروجی مستقیم از downsample را با مقدار های خروجی هر لایه upsample کانکت کرده و در نهایت با استفاده از sigmoid این که یک پیکسلی در ماسک هست یا نه معلوم میشود .

```
16 OUTPUT_CLASSES = 1
17
18 model = unet_model(output_channels=OUTPUT_CLASSES)
19 model.compile(optimizer='adam',
20               loss=dice_loss,
21               metrics=['binary_accuracy' , dice_coef ])
22
23 ## plot the graph of the model with each layer's shape
24 ## hint: https://www.tensorflow.org/api_docs/python/tf/keras/utils/plot_model
25
26
27 tf.keras.utils.plot_model(model, show_shapes=True)
28
```

حال مدل را کامپایل میکنیم و هردو کال بک را در آن طبق بالا قرار میدهیم ، لاس هم همان مقدار تعریف شده قرار میدهیم . و چون خروجی ماسک هست ، تعداد چنل خروجی 1 میباشد.

خروجی مدل :

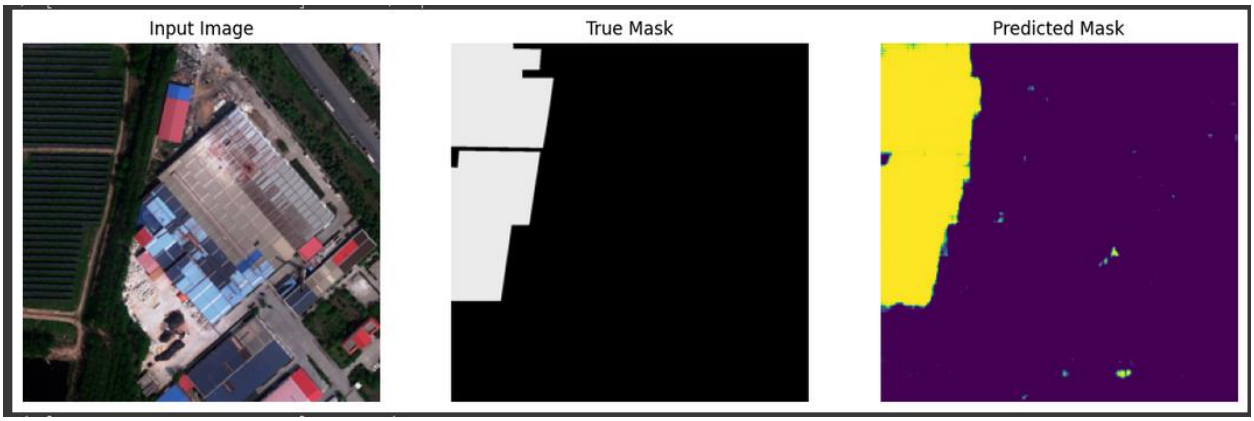


حال در نهایت مدل را ترین کرده:

```python
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=4 , restore_best_weights = True)


# Let's observe how the model improves while it is training.
# To accomplish this task, a callback function is defined below.
class DisplayCallback(tf.keras.callbacks.Callback):
    def on_epoch_begin(self, epoch, logs=None):
        if (epoch + 1) % 5 == 0:
            show_predictions(sample_image, sample_mask)

EPOCHS = 30
STEPS_PER_EPOCH = train_df.shape[0] / BATCH_SIZE
### Define the STEPS_PER_EPOCH and fit the model
### use train_dataset and EPOCHS and STEPS_PER_EPOCH for training process
### use valid_dataset for validation and introduce callbacks to the model
### save history


model_history = model.fit(train_dataset, epochs=EPOCHS,
                          steps_per_epoch=STEPS_PER_EPOCH,
                          validation_data=valid_dataset,
                          callbacks=[DisplayCallback() , callback])

```

برای این کار باید مقدار هر گام به ازای هر ایپوک را محاسبه کرد که طبق فرمول کل داده به مقدار بچ سایز میشود ، همچنین در کال بک باید دو تابع کال بک را قرار دهیم .

که نمونه خروجی ما میشود :



و در نهایت با استفاده از save تابع را دخیره میکنیم .