# تمرین ششم بینایی


## بنفشه قلی نژاد


### سوال 1)

### الف) مفهوم exploding gradiant :

موقع train یک شبکه عصبی عمیق با یادگیری مبتنی بر گرادیان ، مشتقات جزئی را با عبور از شبکه از لایه نهایی به لایه اولیه پیدا می کنیم. با استفاده از قانون زنجیره، لایه هایی عمیق تر شبکه ، ضربات ماتریس پیوسته را طی می کنند تا مشتقات خود را محاسبه کنند.در شبکه ای از n لایه پنهان، n مشتق با هم ضرب خواهند شد. اگر مشتقات بزرگ باشند، شیب به صورت تصاعدی افزایش می‌یابد تا زمانی که در نهایت منفجر شوند، و این موضوع مفهوم exploding gradiant است.

در انفجار گرادیان ، انباشته شدن مشتقات بزرگ باعث می شود که مدل بسیار ناپایدار و ناتوان در یادگیری باشد و تغییرات زیاد در وزن مدل ها، شبکه ای بسیار ناپایدار ایجاد می کند که در مقادیر شدید وزن ها به قدری بزرگ می شوند که باعث overflow مقادیر وزن NaN می شود و دیگر نمی توان آن را update کرد.

روش های تشخیص انفجار گرادیان :

- خوب نبودن مدل بر روی داده های اموزشی و lossضعیف.
- تغییرات خیلی زیاد در loss به علت ناپایداری مدل
- Loss Nan در حین آموزش
- رشد کردن وزن های مدل به صورت تعادلی و بسیار بزرگ شدن آن طی آموزش
- وزن های مدل در مرحله تمرین به NaN تبدیل می شوند.
- مشتقات ثابت

ب) مفهوم vanishing gradiant:

مفهوم ناپدید شدن گرادیان به نوعی برعکس مفهوم بالااست. یعنی در یادگیری یک شبکه عصبی اگر مشتقات کوچک باشند، شیب به صورت تصاعدی کاهش می‌یابد تا در نهایت ناپدید شود. انباشته شدن شیب های کوچک منجر به مدلی می شود که قادر به یادگیری درست نیست زیرا وزن ها و سوگیری های لایه های اولیه که تمایل به یادگیری ویژگی های اصلی از داده های ورودی (X) دارد، اپدیت نمی شوند. در بدترین حالت، گرادیان 0 خواهد بود که به نوبه خود شبکه و آموزش بیشتر را متوقف می کند.

تشخیص ناپدید شدن گرادیان :

- رشد و بهبود با سرعت خیلی کم در هنگام اموزش داده و احتمالا زود متوقف شدن (train های بعدی باعث بهبود مدل نمی شود.)
- تغییرات بیشتر در وزن‌های نزدیک‌تر به لایه خروجی و تغییر خیلی کم یا بدون تغییر در لایه‌هایی نزدیک‌تر به لایه ورودی .
- کوچک تر شدن و مقدار خیلی کمی پیدا کردن وزن های مدل در آموزش به صورت تصاعدی
- صفر شدن وزن مدل در فاز تمرین.

ب)

در سال‌های اخیر، شبکه‌های عصبی عمیق‌تر شده‌اند و شبکه‌های پیشرفته از تنها چند لایه (مانند AlexNet) به بیش از صد لایه می‌رسند.

یکی از مزایای اصلی یک شبکه بسیار عمیق این است که می تواند عملکردهای بسیار پیچیده ای را نشان دهد.
با این حال، یک مانع بزرگ برای آموزش آنها، از بین رفتن گرادیان ها است: شبکه های بسیار عمیق اغلب سیگنال گرادیان دارند که به سرعت به صفر می رسد، بنابراین نزول گرادیان به شدت کند می شود.(vanishing gradiant)
به طور خاص، در هنگام نزول گرادیان، هنگامی که از لایه نهایی به لایه اول برمی گردیم، در ماتریس وزن در هر پله ضرب می کنیم. اگر شیب ها کوچک باشند، به دلیل تعداد ضرب زیاد، گرادیان می تواند به سرعت به صفر کاهش یابد (یا در موارد نادر، به سرعت به صورت نمایی رشد کرده و برای گرفتن مقادیر بسیار بزرگ "منفجر شود").(exploding gradiant)
شبکه های معمولی مانند VGG-16 شبکه های "ساده" نامیده می شوند.

در شبکه های ساده، با افزایش تعداد لایه ها از 20 به 56 (همانطور که در زیر نشان داده شده است)، حتی پس از هزاران بار تکرار، خطای آموزش برای یک لایه 56 در مقایسه با یک شبکه 20 لایه بدتر است.وقتی شبکه‌های عمیق‌تر بتوانند شروع به همگرایی کنند، مشکل تخریب به وجود می آید: با افزایش عمق شبکه، دقت اشباع می‌شود و سپس به سرعت کاهش می‌یابد و باعث کاهش عملکرد مدل می شود

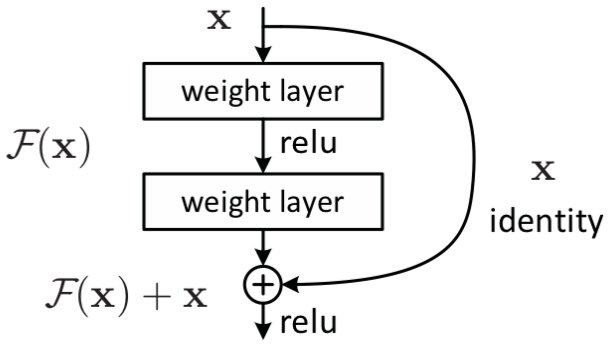
راه حل: A Residual Block:

این بلاک به جای این که به لایه ها اجازه دهد نگاشت زیربنایی را یاد بگیرند، شبکه را با نگاشت باقیمانده مطابقت می دهد. بنابراین، H(x)، نگاشت اولیه است ، و میخواهیم که شبکه متناسب شود،

F(x) := H(x)-x که H(x) := F(x) + x را نتیجه می دهد.

این روش یک میانبر یا یک skip connection اضافه می کند تا اطلاعات راحت تر از یک لایه به لایه بعدی بروند.

بنابراین، با افزودن لایه‌های جدید، به دلیل «Residual Connection» / «Skip Connection» تضمین می‌شود که عملکرد مدل کاهش نمی‌یابد.

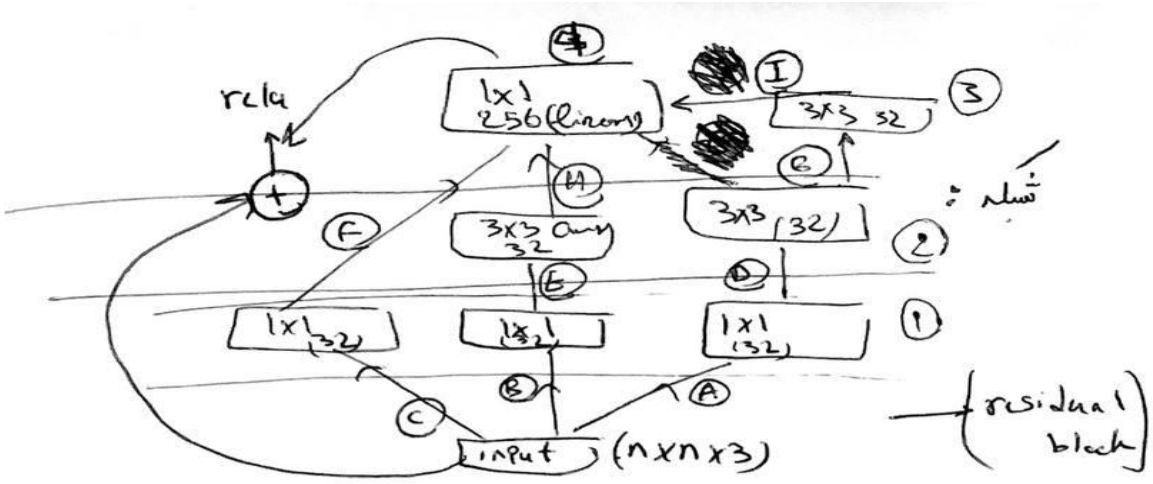



دو نوع بلوک در ResNet استفاده می شود که عمدتاً به این بستگی دارد که ابعاد ورودی/خروجی یکسان یا متفاوت باشد.

1- Identitiy block: مربوط به حالتی است که فعال‌سازی ورودی همان ابعاد فعال‌سازی خروجی را داشته باشد و عملکرد راه حل بالا را دارد.

2- زمانی که ابعاد ورودی و خروجی با هم مطابقت ندارند، می توانیم از این نوع بلوک استفاده کنیم.

سوال دوم)




(A) parameters: $((1\times1\times3)+1)\times32 = 128$, size: $n\times n\times32$

(B): parameters: $((1\times1\times3)+1)\times32 = 128$; size: $n\times n\times32$

(C): parameters: $((1\times1\times3)+1)\times32 = 128$, size: $n\times n\times32$

layer ①: A block + B block + C block: $128\times3 = 384$

(D): $\underbrace{[(3\times3\times\underbrace{32}_{\text{Prev fil number}})+1]\times32}_{\text{Parameters}} = 9248$; size: $n-2, n-2, 32$

(E) $= [(3\times3\times32)+1]\times32 = 9248$; size: $(n-2, n-2, 32)$

layer 2 → D+E: $9248\times2$

(F) (G): $[(3\times3\times32)+1]\times32 = 9248$; size: $(n-4, n-4, 32)$

layer 3 → 9248

عمق ورودی $= 32\times3 = 96$ ← هر کدام از ورودی‌ها 32 تا است، 96 $= 32\times3$

→ $96\left[(1\times1\times96)+1\right)\times256 = 24832$; size: $(n-4, n-4, 96)$

all parameters: $128\times3 + 9248\times3 + 24832 = 384 + 27744 + 2483 = 52900$

respective field:

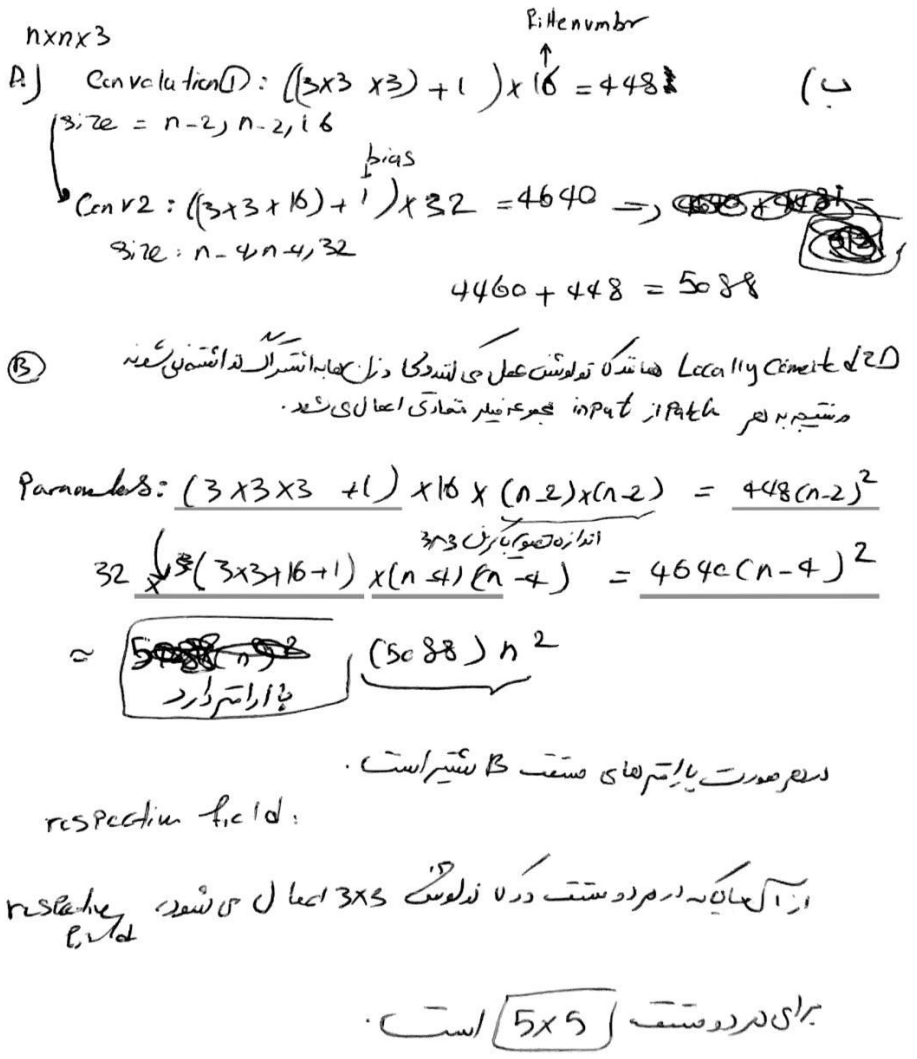
در ستت کانولوشن اول : 1x1

در ستت دوم : 1x1 ⟶ 3x3 : respective field : 3x3

در ستت سوم : 3x3 ⟶ 3x3 ⟶ 5x5
respective field

انجام در کانولوشن ستت سوم یک کانولوشن ( 5x5 ) را حاصل می شود در نتیجه
3x3
respective field در این شبکه (5x5) است .

قسمت ب)



قسمت سوم)

الف) پیاده سازی یک شبکه ساده کانولوشنی

- دریافت داده تست و آموزش.
- نرمالایز کردن داده
- تعریف مدل ( بدنه اصلی شامل دو لایه ی کانولوشنی با تعدا د 32 و 64 فیلتر)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )
```

```
conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
2D)

flatten (Flatten)           (None, 2304)              0

dropout (Dropout)           (None, 2304)              0

dense (Dense)               (None, 10)                23050

=================================================================
Total params: 42,442
Trainable params: 42,442
```

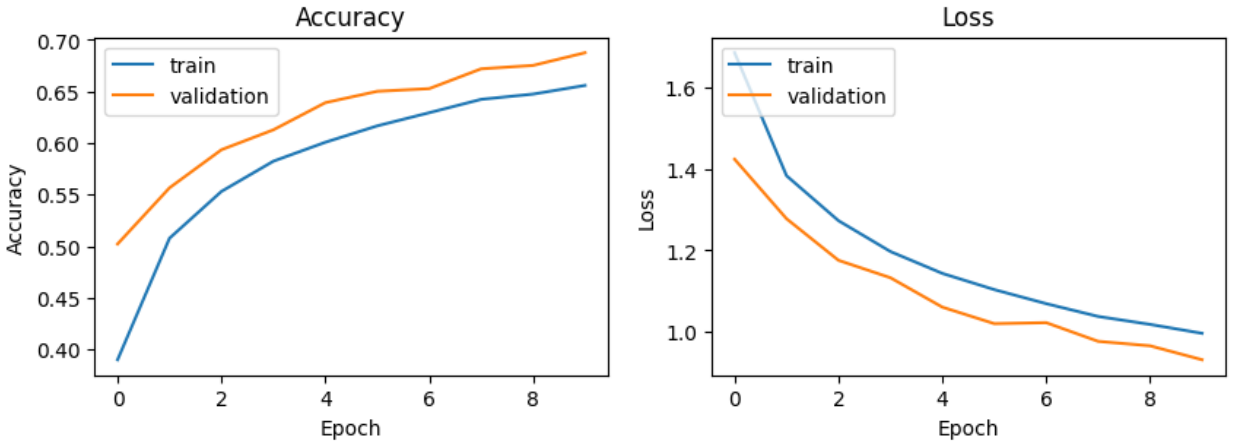
- کامپایل کردن مدل با لاس و اپتیمایز مناسب : categorical_crossentropy , adam

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

- آموزش مدل با 10 اپوک و 128batchsize

```
hist = model.fit(x_train, y_train, epochs= 10, validation_data=(x_test,y_test ), batch_size=128)
```

نتیجه آموزش :




ب)

برای داده افزایی از دو تکنیک ساده آینه ای کردن تصویر و چرخش تصویر استفاده می کنیم

می توان هم این تکنیک را روی داده ها اعمال کرد و هم در لایه های مدل تعریف شده قرار داد.

من از روش دوم استفاده کردم.

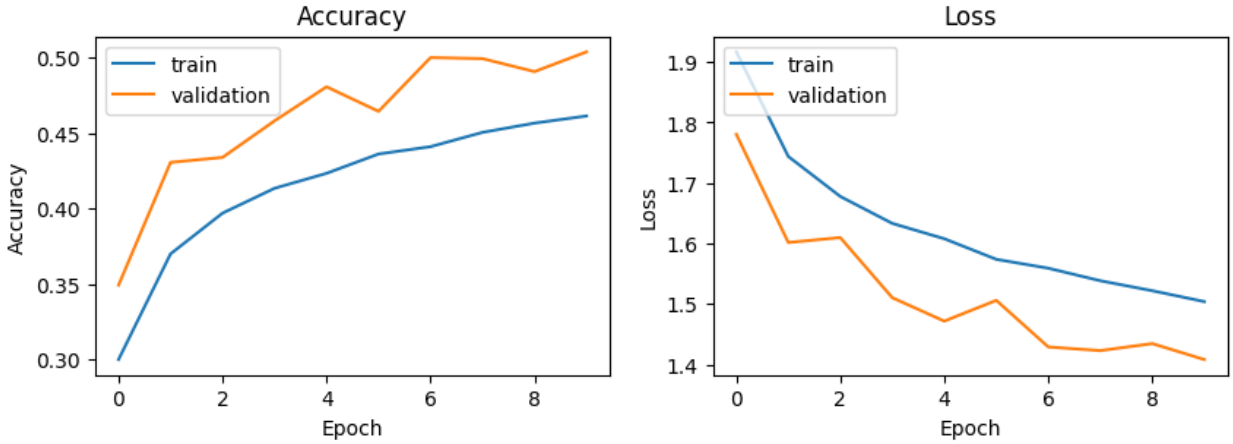
```
data_augmentation = keras.Sequential([
  layers.RandomFlip("horizontal_and_vertical"),
  layers.RandomRotation(0.2),])
```

مدل تعریف شده :

```
model = keras.Sequential(
    [
        keras.Input(shape=(32,32,3)),
```

```
    data_augmentation,
    layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dropout(0.5),
    layers.Dense(10, activation="softmax"),
])
```

حال مانند قسمت پیشین مدل را کامپایل و آموزش می دهیم و نتیجه نمودار به این شکل است.




ج) تحلیل نمودار الف و ب)

همان طور که مشاهده می شود. در حالت اول ، lossهر دوی داده های اموزش و ولیدیشن، با یک شیب ثابتی کم می شود و در نمودار accuracyو یا سرعت یادگیری نیز همین روند را داریم.

در نمودار دوم مقدار , accuracy نسبت به حالت دوم کم تر است اما در داده های ولیدیشن شیب سرعت یادگیری نسبت به داده های ترین بالا تر است و در نمودار loss با نوسانات بیشتر و با سرعت بیشتری نسبت به داده های آموزشی در حال کم شدن است.

در حالت الف همگی با یک روند ثابت پیش می رفتند و در هر دوحالت lossو accuricity داده های آموزش مقادیر بیشتری دارند.

در مورد overfitting، در هر دو قسمت رفتار داده های ولیدیشن شبیه رفتار با داده های ترین است اما در قسمت دوم و در قسمت داده افزایی رفتار و عملکردی بهتر را بر روی داده های ولیدیشن نسبت به داده های اموزش دیده می شود.

د )

انتقال داده های یادگیری :

1- استخراج داده (cifar10.loaddata())

2- Preproccess: آماده سازی داده ها با توجه به مدل ساختاری resnet50 و روش one-hot encoding

```
def preproccess(X,Y):
  X_p = keras.applications.resnet50.preprocess_input(X)
  Y_p = keras.utils.to_categorical(Y,  10)
  return X_p ,Y_p
```

3- تعریف مدل از پیش آموزش دیده ی resnet

این مدل این امکان را می دهد از وزنه هایی استفاده کنید که از قبل کالیبره شده اند تا پیش بینی کنید. در این حالت از وزن های Imagenet استفاده می کنیم و شبکه ResNet50 است. گزینه include_top=False امکان استخراج ویژگی را با حذف آخرین لایه های متراکم فراهم می کند. این اجازه می دهد تا خروجی و ورودی مدل را کنترل کنیم.

```
input_t = keras.Input(shape = (224,224,3))
res_model = keras.applications.ResNet50(include_top = False , weights = "imagenet"
, input_tensor = input_t)
```

ما در حال حاضر مقدار بسیار زیادی از پارامترها را به دلیل تعداد لایه های ResNet50 داریم، اما وزن های کالیبره شده نیز داریم. می‌توان لایه‌ها را freeze کنیم تا این مقادیر تغییر نکنند، و از این طریق در زمان و هزینه محاسباتی صرفه‌جویی کنیم

```
res_model.trainable = False
```

حال زمان این است که از این لایه از پیش آماده دیده و پارامتر های آن در مدل از پیش آموزش دیده ی خود استفاده کنیم.
از آن جایی که resnet ورودی 224 را میگیرد. یا می توانیم این تغییر را روی کل داده های اولیه اعمال کنیم و یا در مدل با تعریف لایه ی اولیه اندازه هر ورودی را به 224 * 224 تغییر بدهیم.
یعنی:

```
to_res = (224, 224)
modelp = keras.Sequential(
  [
    layers.Lambda(lambda image: tf.image.resize(image, to_res)),
    data_augmentation,
    res_model,
    layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dropout(0.5),
    layers.Dense(10, activation="softmax"),
  ]
)
```

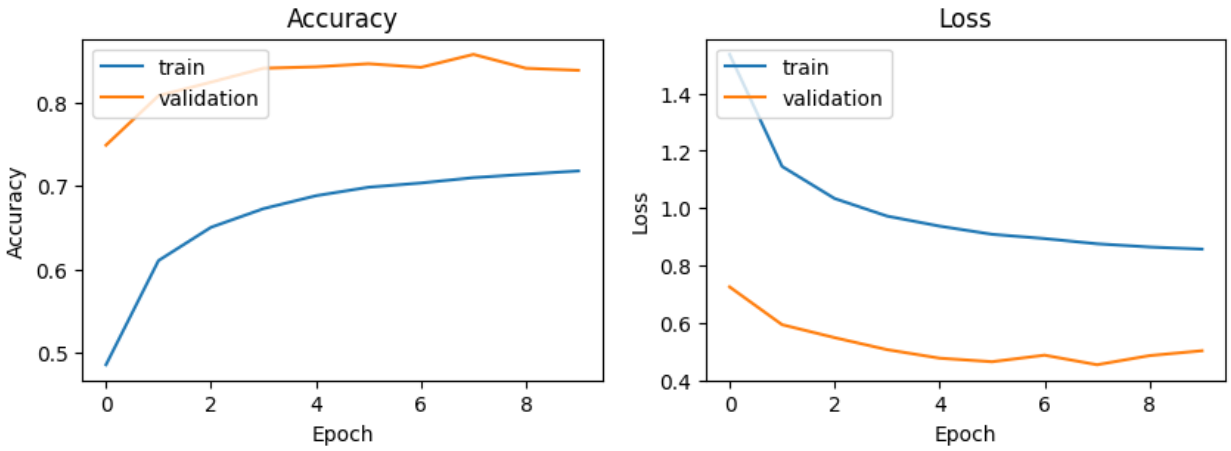
مدل ساخته شده)

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lambda (Lambda)             (None, 224, 224, 3)       0

 sequential_1 (Sequential)   (None, None, None, 3)     0

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

 conv2d_4 (Conv2D)           (None, 5, 5, 64)          1179712

 max_pooling2d_4 (MaxPooling  (None, 2, 2, 64)         0
 2D)

 flatten_2 (Flatten)         (None, 256)               0
```

```
 dropout_2 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 10)                2570

=================================================================
Total params: 24,769,994
Trainable params: 1,182,282
Non-trainable params: 23,587,712
```

---

در نهایت همانند قسمت های قبلی مدل را کامپایل و  با 10 اپوک آموزش می دهیم. نتیجه)




همان طور که دیده می شود. سرعت رشد و کاهش ضرر و سرعت اموزش  نسبت به قسمت های قبل بیشتر  شده اما میزان سرعت یادگیری در این فسمت هم برای داده های ولیدیشن و هم داده های آموزشی نسبت به حالت های قبل بیشتر است.

در این نمودار ها نیز کمی overfitting مشاهده می کنیم چرا که رفتار داده های آموزش و ولیدیشن متفاوت است.

د) در این قسمت نیز همانند قسمت قبل عمل می کنیم .برای شبکه ی resnet include_top را فالس می کنیم و  اندازه ورودی را 3 * 32 * 32 می دهیم. سپس 4  لایه اول آن را جدا می کنیم تا در مدل خود به صورت جدا از آن لایه ها استفاده کنیم.

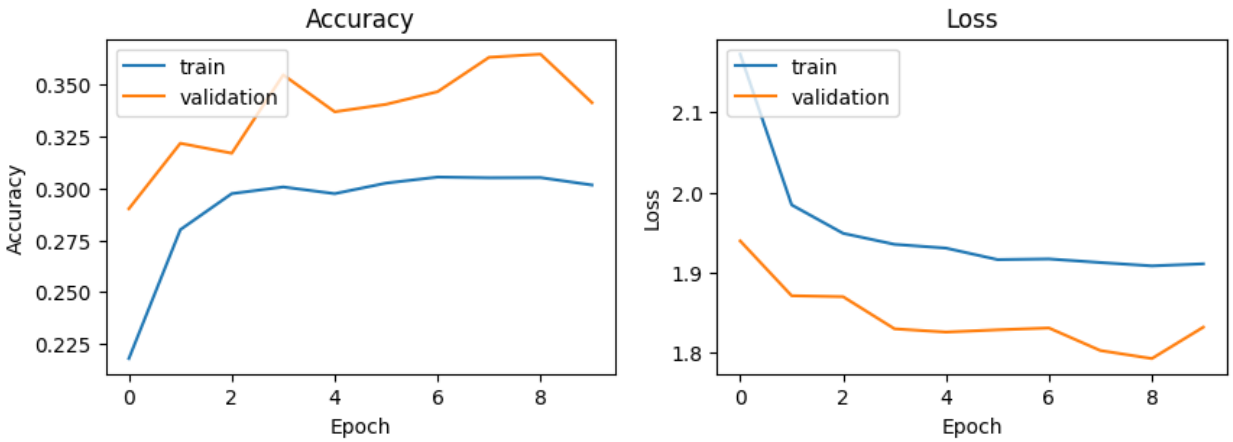
```
input_t = keras.Input(shape = (32,32,3))  ;res_model = keras.applications.ResNet50(include_top = False , weights = "imagenet" , input_tensor = input_t)

modelp = keras.Sequential(
  [reslayers[0], #input
    data_augmentation,
    reslayers[1] ,
    reslayers[2] ,
    reslayers[3],
```

```
    layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dropout(0.5),
    layers.Dense(10, activation="softmax"),
  ])
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 32, 32, 3)         0

 conv1_pad (ZeroPadding2D)   (None, 38, 38, 3)         0

 conv1_conv (Conv2D)         (None, 16, 16, 64)        9472

 conv1_bn (BatchNormalizatio  (None, 16, 16, 64)       256
 n)

 conv2d (Conv2D)             (None, 14, 14, 64)        36928

 max_pooling2d (MaxPooling2D  (None, 7, 7, 64)         0

 flatten (Flatten)           (None, 3136)              0

 dropout (Dropout)           (None, 3136)              0

 dense (Dense)               (None, 10)                31370
=================================================================
Total params: 78,026
Trainable params: 68,298
Non-trainable params: 9,728
```

در نتیجه  نیازی به ریسایز داده نخواهیم داشت.

نتیجه)




مدل نسبت به حالت اول سریع تر ترین می شود چرا که شبکه اول عمیق تر است. اما نتیجه مطلوب نیست و به نوعی در هیچ کدام از داده ها به خوبی آموزش ندیده اند.

سوال 4)

الف)

Stride یا گام ، تعداد قدم‌هایی است که فیلتر در طول و عرض ورودی حرکت می‌کند. یعنی مشخص می‌کند که فیلتر چقدر در هر مرحله جابه‌جا می‌شود.

وقتی از stride بزرگتر از ۱ استفاده می‌شود، فیلتر با یک فاصله بیشتر از یک واحد در طول و عرض ورودی حرکت می‌کند. در نتیجه، خروجی این لایه کوچکتر از اندازه ورودی خواهد بود.

تفاوت stride با pooling این است که pooling، اطلاعاتی را که در ورودی وجود دارد خلاصه می‌کند. اما stride، عملگری است که فیلتر را به طول و عرض ورودی (برای مثال دو تا یکی یا هر مقداری که ست شود) حرکت می‌دهد و اطلاعات را با استفاده از فیلتر استخراج می‌کند.

برای مثال دو روش ماکس پولینگ و اوریج پولینگ داریم که اطلاعات به دست اورده را با ماکس گرفتن یا میانگین گرفتن جدا می کند

تاثیر بر روی عملکرد:

- **کاهش ابعاد خروجی:** استفاده از stride بزرگتر از ۱ منجر به کاهش اندازه خروجی و تعداد پارامتر ها می‌شود
- **کاهش اطلاعات مورد بررسی:** با استفاده از stride بزرگتر، فیلتر در هر قدم اطلاعات کمتری از ورودی مشاهده می‌کند. این ممکن است باعث از دست رفتن برخی جزئیات ویژگی‌های مهم شود.

به طور کلی، استفاده از stride در شبکه‌های عصبی می تواند محاسبات را ساده تر و سرعت پردازش را افزایش دهد.

ب) 1

لایه های میانی: برای این لایه می توانیم از فعال ساز های نظیر relU ، Lrlu و یا Prlu اسسفاده کنیم. این فعال ساز یک فعال ساز غیر خطی است که به صورت max(0,x) عمل میکند. و همجنین می تواند نیز مشکلاتی نظیر vanishing gradiant را حل کند و برای مسایل دسته بندی مناسب و استفاده به صرفه ای دارد.

لایه ی اخر: اگر دسته بندی ما چند کلاسه باشد ( برای مثال خود طبقه سالم و معیوب نیز دسته بندی داشته باشند) فعال ساز softmax l مناسب ترین فعال ساز است. در صورتی که طبقه بندی ما دو کلاسه باشد، فعالساز sigmoidنیز می تواند مناسب باشد.

ب2) می توانیم از تابع خطای cross-entropy استفاده کنیم. این تایع خطا :

- این تابع برای مسائل چند کلاسه مناسب است. بین دقت و قطعیت تعادل را برقرار می کند. در صورتی که طبقه بندی ما وزن دار باشد می توانیم از weighted cross entropy استفاده کنیم

ب3) برای این که دقت را افزایش یدهیم، باید دقت شبکه در تشخیص محصول معیوب زیاد باشد. در نتیجه اگر دقت بالا برود، احتمال اینکه اشیای معیوب بهتر تشخیص داده بشوند و محصولات معیوب کم تر به دست مشتری برسد و محصولات سالم باشند، از طریق precision این امر میسر خواهد شد.

Precisionتعداد محصولات معیوب را کم می کند زیرا این مدل تنها زمانی که مطمئن است که محصول معیوب است، آن را تشخیص می دهد.

ج)

ج–الف) شبکه های کانولوشنی برای تشخیص متن و در حوزه nlp لزوما کاربرد خوبی ندارد و عمدتا از شبکه های rnn در این حوزه استفاده می شود.

این الگو ها وابستگی های متنی را بیشتر درک می کنند و شبکه های کانولوشنی بیشتر وابستگی مکانی دارند.

ج–ب) می توان از شبکه های کانولوشنی در تشخیص صوت استفاده کرد. صوت نسبت به زمان تغییر می کند و با این شبکه ها می توان این تغییرات را به خوبی درک کرد و می تواند الگو های تغییرات را به خوبی درک کند.به طور کلی از طریق ویژگی های زمانی و مکانی و تغییرات صوت در فرکانس آن به خوبی تغییرات آن را یاد گرفته و تشخیص دهند.

ج–پ)

برای تشخیص ویژگی های رفتاری انسانی، شبکه ی عصبی کانولوشنی نمی تواند به خوبی عمل کند چراکه باید ارتباطات انسانی را نیز درک کند و همچنین داده های خود جدول را بررسی کند.

د)

- . احتیاج به مقدار زیادی داده: شبکه‌های عصبی کانولوشنی برای دست‌یابی به عملکرد بهتر و عملکرد مناسب، نیاز به مجموعه‌ای از داده‌های آموزشی بزرگ دارند. و اگر نباشد شبکه ممکن است با مشکلاتی مانند بیش‌برازش (overfitting) روبرو شود و عملکرد آن در داده‌های جدید کاهش یابد.
- پیچیدگی محاسباتی: شبکه‌های عصبی کانولوشنی معمولاً دارای ساختار پیچیده‌ای هستند و برای آموزش و استفاده از آنها، نیاز به قدرت محاسباتی بالا و منابع سخت‌افزاری مناسب است. آموزش شبکه‌های بزرگ می‌تواند زمان‌بر و هزینه‌بر باشد و نیاز به منابع پردازشی قدرتمند داشته باشد.
- . تحلیل تفسیرپذیری: یکی از مشکلات شبکه‌های عصبی کانولوشنی، عدم توانایی تفسیر و تحلیل دقیق فرایند تصمیم‌گیری درون شبکه است. این به این معنی است که ممکن است مشکلی در توضیح علت تصمیمات شبکه به وجود بیاید و درک دقیق از دلایل و فرآیند تصمیم‌گیری شبکه مشکل باشد.

به طور کلی، در استفاده از شبکه‌های عصبی کانولوشنی برای تحلیل داده‌های جدولی، باید سازوکارها و روش‌های دیگری را بررسی کنیم که بهترین عملکرد را در این نوع مسائل ارائه دهد. به طور کلی این شبکه ها در وابستگی های مکانی و تشخیص الگو ها در تصویر بسیار قدرتمندند اما در تحلیل داده ها یا پردارش متن ضعیف تر عمل میکنند.

سوال 5)

الف) 80 درصد خواسته های کد پیاده سازی شده است که در این قسمت به ظور خلاصه شرح می کنیم و کل کد در نوت بوک قرار داردد.

- بخش دسته بندی داده ها و تبدیل آن ها به دیتا ست های مناسب.

1- جدا کردن ایمیج ها و ماسک ها از یک دیگر، تبدیل آن ها به فرمت تصویر و ذخیره سازی آن ها به صورت جدا با تغییر اندازه.
2- تبدیل کردن این داده ها به یک دیتا فریم با دسته بندی id، image_path , mask_path
3- دیکود کردن تمام داده های دیتا فریم و تبدیل آن ها به دیتا ست . همچنین نرمال کردن داده ها(بین 0–1 ) و داده افزایی آینه ای با کمک عدد رندم .
4- نمابش تصاورر جدا شده

- تعریف مدل )

1- تعریف مدل mobilenetv2 به عنوان backbone و استفاده از برخی از لایه های این مدل.
2- Upsampling: تعریف دیکودر برای مدل خواسته شده و استفاده از یک لایه ی کانولوشنی.
3- تعریف اصلی مدل و استفاده از خروجی backbone. در هر مرحله اخرین لایه اوت پووت را یه لیست up_stack می دهد و خروجی آن را با هر یک از لایه های خروجی بک بوت ترکیب می کند. در نهایت خروجی آن به عنوان خروجی مدل ما معرفی می شود.
ساختار مدل اصلی در اسلاید بعدی است

- کامپایل مدل ) با کمک اپتیمایزر ادم و تابع خطای doss function مدل را کامپایل می کنیم و مرحله اخر نیز مرحله آموزش و predictioinاست که کامل پیاده سازی نشده است.

ب)

BCE Loss (Binary Cross-Entropy Loss) و IoU Loss (Intersection over Union Loss) دو تابع هزینه متداول استفاده شده در مسائل دسته‌بندی دقیق (Semantic Segmentation) هستند. این دو تابع هزینه برای اندازه‌گیری تفاوت بین ماسک‌ها (معرفی شده در زیر) و خروجی شبکه عصبی استفاده می‌شوند.

• BCE Loss: تابع هزینه Binary Cross-Entropy Loss یک معیار برای اندازه‌گیری تفاوت بین دو توزیع است. در Semantic Segmentation، هر پیکسل تصویر ورودی به یک دسته تعلق می‌گیرد یا نه (دسته مورد نظر یک باشد و سایر دسته‌ها صفر). BCE Loss برای هر پیکسل، احتمال دسته‌بندی شده توسط ماسک و خروجی شبکه را محاسبه می‌کند و تفاوت بین آنها را اندازه‌گیری می‌کند. این تابع هزینه به شبکه کمک می‌کند تا احتمال دسته‌بندی هر پیکسل را بهبود دهد و به دسته‌بندی دقیق‌تری برسد.

• IoU Loss: تابع هزینه Intersection over Union Loss یک معیار برای اندازه‌گیری همپوشانی بین دو ماسک است. IoU (Intersection over Union) به عنوان نسبت منطقه‌ی همپوشانی بین دو ماسک به مجموع منطقه‌ی هر دو ماسک استفاده می‌شود. IoU Loss برای هر پیکسل، مقدار IoU بین ماسک و خروجی شبکه را محاسبه می‌کند و تفاوت بین آنها را اندازه‌گیری می‌کند. این تابع هزینه به شبکه کمک می‌کند تا همپوشانی ماسک تولید شده توسط شبکه با ماسک مورد نظر بهبود یابد.

به طور خلاصه، BCE Loss معیاری برای اندازه‌گیری تفاوت بین احتمالات دسته‌بندی شده توسط ماسک و خروجی شبکه است، در حالی که IoU Loss معیاری برای اندازه‌گیری همپوشانی بین ماسک تولید شده توسط شبکه و ماسک مورد نظر است. هر دو تابع هزینه به شبکه کمک می‌کنند تا دقت و صحت نتایج را در Semantic Segmentation افزایش دهند.