HW6

(1

در بحث بهینه سازی وزن های مدل که در اخر هر epoch انجام میشود از حاصل ضرب مشتق های جزیی استفاده میکنیم که از اخرین لایه مشتق ها را تا لایه ابتدایی به دست اورده (backpropagation)و در هم ضرب میکنیم (chain role) که اگر شبکه ما عمق زیادی داشته باشد شامل تعداد زیادی ضرب ماتریسی مشتق ها خواهد بود اگر مقدار مشتق ها کوچک باشد به صورت exponential کوچک میشود و در لایه های دور تر از لایه اخر یعنی لایه های اول وزن ها به سمت صفر میرود و vanishing gradient رخ میدهد و یادگیری از لایه های ابتدایی که ورودی ما است دیگر رخ نمیدهد

اگر برعکس مشتق ها بزرگ باشد به مرور چون در هم ضرب میشوند بسیار مقدار نهایی لایه های اول بزرگ میشوند و اگر خیلی بزرگ شود overflow رخ داده و به سمت not a number=nan شدن وزن ها میرود و exploding gradient رخ میدهد و مدل ناپایدار و ناکارامد میشود

در هر دو حالت اگر از یک مقداری بزرگ شدن یا کم شدن مشتق ها زیاد تر شود یادگیری متوقف میشود

در حالت vanishing gradient:

وزن ها به مرور صفر میشوند و خیلی اهسته یاد میگیرد تغییرات لایه های اخر بیشتر است

در حالت exploding gradient:

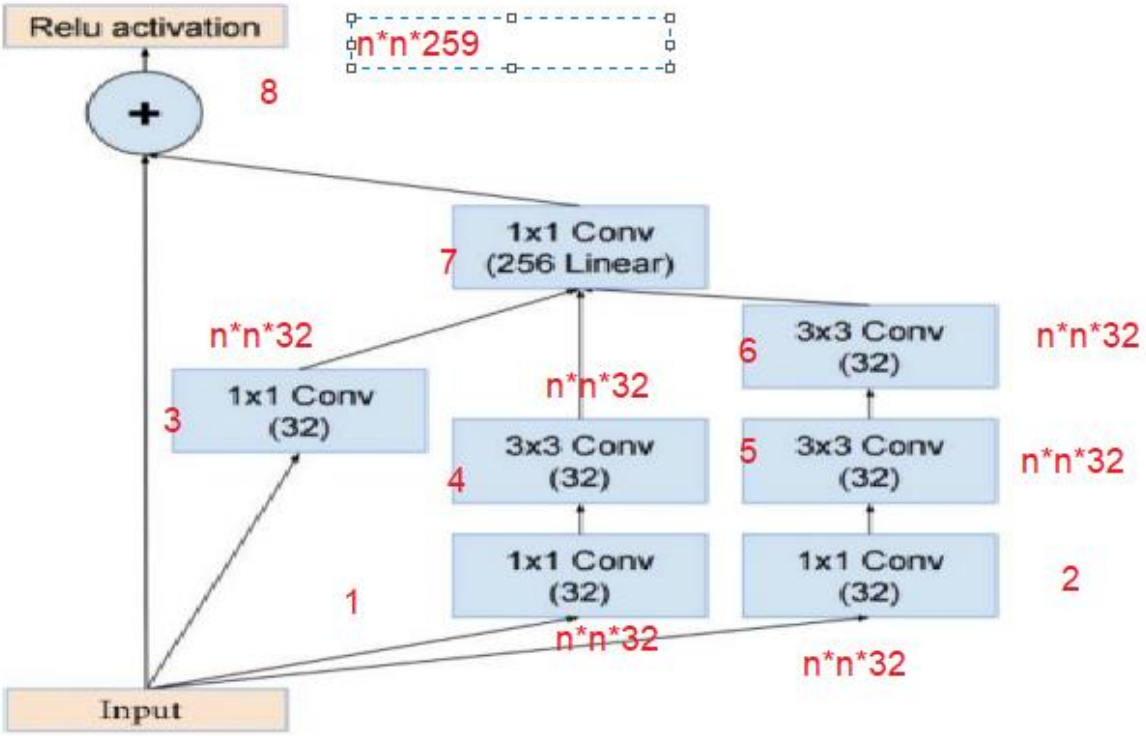
تغییرات مدل خیلی زیاد است و توانایی یادگیری ندارد و به مرور وزن ها nan میشود و حتی عملا روی دیتا اموزشی هم عملکرد خوبی ندارد

ب)

مدل های قبل از ResNet زمانی که شبکه از یک حدی عمیق تر میشد دچار vanishing مشدند در دیتا امئزشی و تست یادگیری کندی داشتند

در مدل های قبل که هنگام اپدیت وزن ها از backpropagation ,gradient decsent برای اپدیت وزن ها از اخر به اول حداقل کردن loss function استفاده میشد که در بخش قبل گفتیم که این موضوع باعث vanishing میشود در مدل ResNet این موضوع با skip connection حل میکند در واقع یک نگاشت identity میگذارد که هیچ کاری انحام نمیدهد و باعث میشود که مشتق دیتا های لایه های اول را بی تاثیر نکند

(2



تعداد unit لایه صفر 3 تا است زیرا 3 کانال r,g,b دارد که به ازای هز یک یک unit داریم در مرحله بعد ما 32 نورون داریم که به ازای هر یک ازunit های لایه قبل ما یک کرنل 1*1 داریم یعنی یک ضریب و یک bais داریم 128=(1+1*3)32

در لایه دوم هم مشابه لایه یک است و همچنین لایه 3 اما در لایه 4و5 ما در هر unit به ازای هریک از 32 unit مرحله قبل ما یک کرنل 3*3 یعنی 9 تا ضریب و یک bais داریم که 9248=(1+9*32)*32 در لایه 7 هم به در هر یک از 256 unit به ازای

هر یک از96 نورون مرحله قبل یک کرنل 1*1 یعنی 1 ضریب دارد و یک bais

0=>0

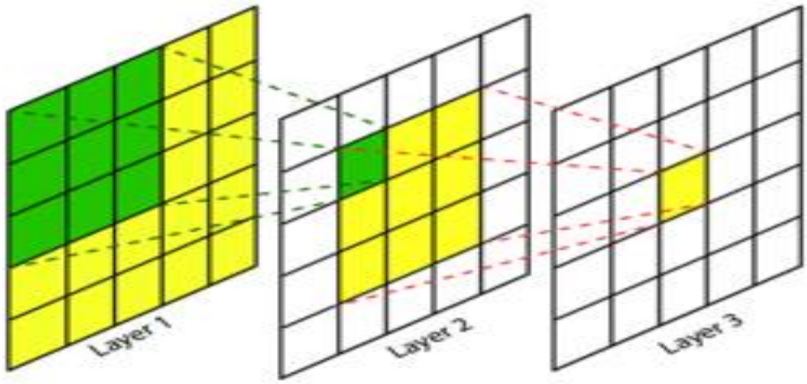
3و2و1=> 128

6و5و4=>9248

7=>256*97= 24832

اخر=>0

RF:



ماکسیمم سایز کرنل لایه های متصل قبل+ (سایز کرنل -1)

درلایه های 1و2و3 سایز کرنل 1*1 است یعنی معادل یک پیکسل از تصویر اولیه است در لایه های 4و5 هم کرنل ما 3*3 است یعنی هر پیکسل معادل 9 پیکسل از لایه قبل یا یک پنجره 3*3 است و لایه قبل هم معادل یک پیکسل از تصویر اولیه است یعنی اسن دو لایه معادل 9 پیکسل یا یک پنجره 3*3 از لایه اول اند

لایه 6 هم معادل 9 پیکسل از لایه قبل یا یک پنجره 3*3 است که چون لایه قبل ان معادل 9 پیکسل یا یک پنجره 3*3 از تصویر اول است پس در کل معادل یک پنجره 5*5 از تصویر اولیه است یعنی RF 5*5 است لایه 7 معادل ماکسیمم RF لایه های قبل یعنی 5*5 است

لایه اخر هم ماکسیمم لایه 0 و 7 است یعنی 5*5

ب)

A)
Conv2D(filters=16, kernel_size=(3, 3), padding='valid')
Conv2D(filters=32, kernel_size=(3, 3) , padding='valid')

B)
LocallyConnected2D(filters=16, kernel_size=(3, 3) , padding='valid')
LocallyConnected2D (filters=32, kernel_size=(3, 3) , padding='valid')

در لایه اول بخش A ما 16 نورون داریم که در هر یک به ازای هر یک از 3 unit لایه قبل یک کرنل 3*3 دارد که هر یک از انها یک bias دارند 16(9*3+1) و در لایه بعد ما 32 نورون داریم که به 16 تای قبلی وصل اند و یعنی 16 تا کرنل دارند که هریک 3*3 اند و یک bias دارد در کل 32*(16*9+1)

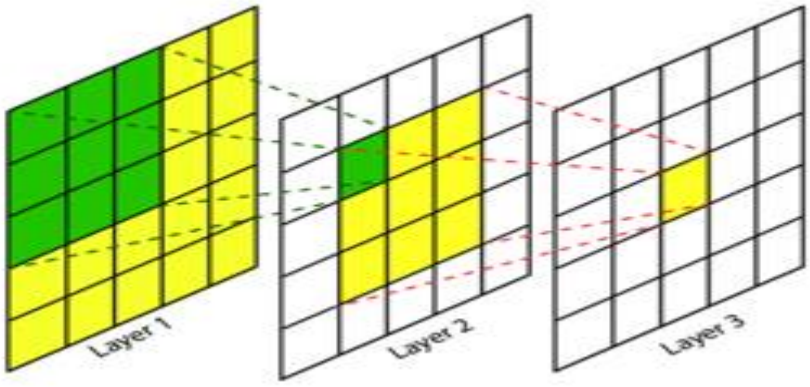
در مورد B در لایه اول ما 16 unit داریم که هر یک چون کرنل ما 3*3, local است (n-2)(n-2) نورون دارد که هر یک دارای 3 کرنل 3*3 اند و یک bias در مجموع: 16*(n-2)(n-2)(3*9+1)

در لایه دوم این بخش 32 تا unit داریم که هر یک چون تصویر ورودی (n-2)(n-2) است و کرنل ما 3*3 است دارای (n-4)(n-4

نورون است و هر یک 16 کرنل برای هر یک از unitهای متصل قبل دارند که هر یک 3*3 اند و یک bias دارند که در مجموع :

32*(n-4)(n-4)*(16*9+1)

RF:
در هر دو لایه اول یک پنجره 3*3 یا 9 پیکسل از تصویر اولیه را میخواهد و لایه دوم هم یک پنجره 3*3 از لایه قبل خود که خودش یک پنجره 3*3 از تصویر اول را شامل میشود را میخواهد که یعنی یک پنجره 5*5 از تصویر ورودی میخواهد



(3

در این سوال ابتدا دیتاست را load میکنیم :

```
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)
```

```
Training: (50000, 32, 32, 3) (50000, 1)
Validation: (10000, 32, 32, 3) (10000, 1)
```

همانطور که مشاهده میکنیم این دیتاست دارای 10 کلاس است و دیتا ورودی ات 50000 تا تصویر 32 در 32 با 3 کانال است

سپس با استفاده از تابع to_categorical مقادیر label ها را one hat میکنیم و همچنین normalize میکنیم تا اعداد کانال های ورودی بین صفر و یک بیاید

```
[11] ##################################################
     ################### YOUR CODES ###################
     num_classes=10
     y_train = keras.utils.to_categorical(y_train, num_classes)
     y_val = keras.utils.to_categorical(y_val, num_classes)
     x_train = (x_train).astype(np.float32) / 255
     x_val = (x_val).astype(np.float32) / 255
     ##################################################
```

سپس به سراغ ساخت مدل میرویم

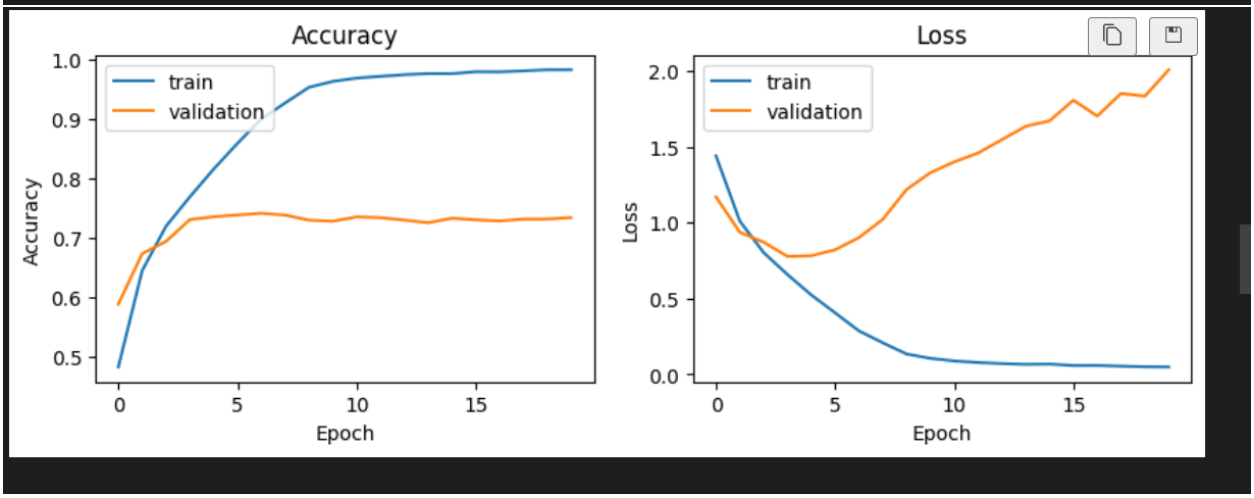
```
model = keras.Sequential()
model.add(keras.layers.Input(shape=x_train[0].shape))
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu',  padding='same'))
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
model.add(keras.layers.MaxPool2D())

model.add(keras.layers.Conv2D(64, (3, 3), activation='relu',  padding='same'))
model.add(keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add(keras.layers.MaxPool2D())

model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(units=512, activation='relu'))
model.add(keras.layers.Dense(units=num_classes, activation='softmax'))

model.summary()
```

با توجه به نمونه هایی که برای حل این سوال دیدم و ازمایش کردن این مقادیر به این نتیجه رسیدم که قرار دادن padding = same

و داشتن دو مرحله که هر یک شامل دو لایه با فیلتر های یکسان توان دو اند بهتر است اعداد فیلتر ها را از RESNET الگو برداری کردم پدینگ هم به دلیل این است که با pooling خیلی سایز گوچک نشود

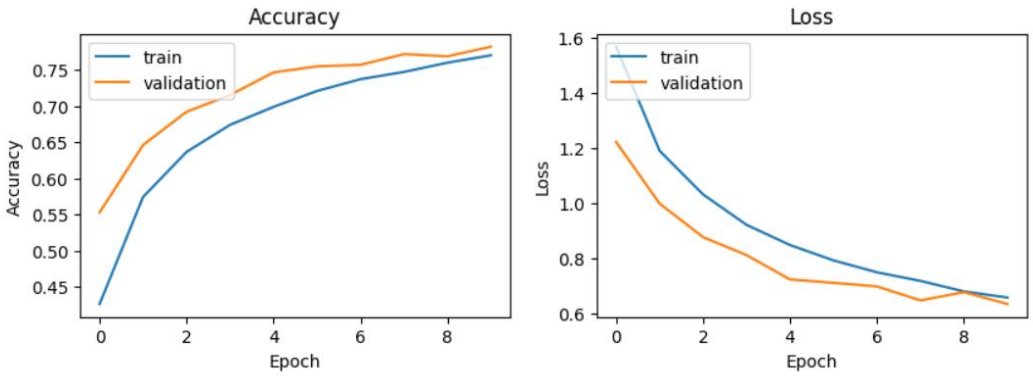
اما باز هم نتیجه درست نبود و overfit شدم

```
====] - 7s 9ms/step - loss: 1.4391 - accuracy: 0.4828 - val_loss: 1.1686 - val_acc
====] - 4s 8ms/step - loss: 1.0095 - accuracy: 0.6454 - val_loss: 0.9357 - val_acc
====] - 4s 8ms/step - loss: 0.8018 - accuracy: 0.7199 - val_loss: 0.8704 - val_acc
====] - 4s 8ms/step - loss: 0.6572 - accuracy: 0.7695 - val_loss: 0.7769 - val_acc
====] - 4s 9ms/step - loss: 0.5228 - accuracy: 0.8163 - val_loss: 0.7810 - val_acc
====] - 4s 8ms/step - loss: 0.4049 - accuracy: 0.8597 - val_loss: 0.8203 - val_acc
====] - 5s 10ms/step - loss: 0.2860 - accuracy: 0.9008 - val_loss: 0.9010 - val_ac
====] - 4s 9ms/step - loss: 0.2077 - accuracy: 0.9277 - val_loss: 1.0206 - val_acc
====] - 4s 8ms/step - loss: 0.1334 - accuracy: 0.9541 - val_loss: 1.2181 - val_acc
====] - 4s 8ms/step - loss: 0.1051 - accuracy: 0.9640 - val_loss: 1.3283 - val_acc
====] - 4s 9ms/step - loss: 0.0879 - accuracy: 0.9696 - val_loss: 1.3998 - val_acc
====] - 4s 8ms/step - loss: 0.0776 - accuracy: 0.9725 - val_loss: 1.4577 - val_acc
```



سپس متوجه شدم که برای جلوگیری از overfitting باید از لایه dropout استفاده کرد که بخشی از نورون های لایه قبل رو نادیده میگیرد و از overfitting جلوگیری میکند

```
model = keras.Sequential()
model.add(keras.layers.Input(shape=x_train[0].shape))
model.add(keras.layers.Conv2D(64, (3, 3), activation='relu',  padding='same'))
model.add(keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add(keras.layers.MaxPool2D())
model.add(Dropout(0.3))
model.add(keras.layers.Conv2D(128, (3, 3), activation='relu',  padding='same'))
model.add(keras.layers.Conv2D(128, (3, 3), activation='relu'))
model.add(keras.layers.MaxPool2D())
model.add(Dropout(0.6))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(units=512, activation='relu'))
model.add(Dropout(0.3))
model.add(keras.layers.Dense(units=num_classes, activation='softmax'))

model.summary()
```




```
Epoch 1/10
500/500 [==============================] - 12s 21ms/step - loss: 1.5679 - accuracy: 0.4264 - val_loss: 1.2218 - val_accuracy: 0.5529
Epoch 2/10
500/500 [==============================] - 10s 19ms/step - loss: 1.1901 - accuracy: 0.5744 - val_loss: 0.9984 - val_accuracy: 0.6462
Epoch 3/10
500/500 [==============================] - 10s 20ms/step - loss: 1.0315 - accuracy: 0.6366 - val_loss: 0.8766 - val_accuracy: 0.6921
Epoch 4/10
500/500 [==============================] - 10s 19ms/step - loss: 0.9211 - accuracy: 0.6740 - val_loss: 0.8112 - val_accuracy: 0.7153
Epoch 5/10
500/500 [==============================] - 10s 20ms/step - loss: 0.8479 - accuracy: 0.6989 - val_loss: 0.7230 - val_accuracy: 0.7465
Epoch 6/10
500/500 [==============================] - 10s 20ms/step - loss: 0.7921 - accuracy: 0.7208 - val_loss: 0.7107 - val_accuracy: 0.7549
Epoch 7/10
500/500 [==============================] - 10s 19ms/step - loss: 0.7488 - accuracy: 0.7372 - val_loss: 0.6976 - val_accuracy: 0.7570
Epoch 8/10
500/500 [==============================] - 10s 20ms/step - loss: 0.7176 - accuracy: 0.7472 - val_loss: 0.6469 - val_accuracy: 0.7718
Epoch 9/10
500/500 [==============================] - 10s 20ms/step - loss: 0.6795 - accuracy: 0.7600 - val_loss: 0.6769 - val_accuracy: 0.7688
Epoch 10/10
500/500 [==============================] - 10s 20ms/step - loss: 0.6573 - accuracy: 0.7702 - val_loss: 0.6339 - val_accuracy: 0.7821
```

دیگر overfit نشده

ب)

من برای این سوال چند مدل دیتا جنریتور تعریف کردم که اشتباه بود در اخر به مدل زیر که از کلاس های اماده keras استفاده میکند مواردی که به صورت ورودی میدهیم تبدیل هایی است که دیتا های جدید را میسازد

```
# Data augmentation
datagen= ImageDataGenerator(vertical_flip=True, rotation_range= 30, validation_split=0.2,fill_mode ='nearest')

datagen.fit(x_train)

new_train = datagen.flow(x_train, y_train, batch_size =100)
new_val= datagen.flow(x_val, y_val, batch_size =100)

# datagen.fit(x_train)
# Generate augmented images
# augmented_images = datagen.flow(x_train, y_train, batch_size=9, shuffle=False)

# # Plot augmented images
```

پس از تعریف یک بار باید تابع fit را برای x_train صدا بزنیم (علت سینتکسی دارد)

Datagen.flow() هم در واقع با گرفتن ورودی ها و label های متناسب دیتا های جدید را batch batch تولید و با همان label ها ذخیره میکند

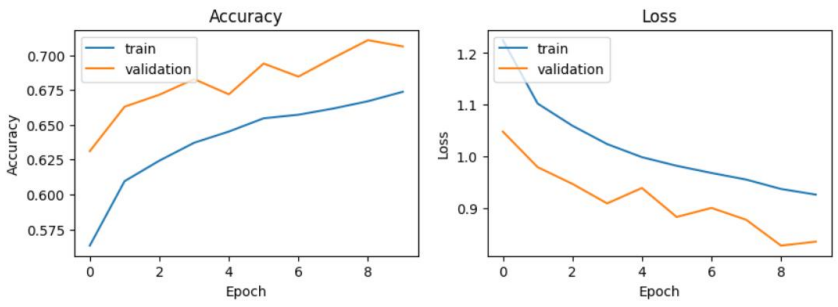
در اخر مانند الگو ورودی تابع fit انرا صدا میزنیم

```
# Train the model with augmented data
history = model.fit(new_train,

                    epochs=10,
                    validation_data=new_val,
                    shuffle=True)
```

New_train -> x_train, y_train

New_val -> (x_val, y_val)

```
plot_acc_loss(history)
```



Over_fit نشده یعنی در دقت ان در دیتا های تست و validation به مرور زیاد شده

ج)

از نظر underfitting در ابتدا که مدل هنوز چیزی یاد نگرفته هردو underfit اند یعنی دقت پایینی در دیتا test , validation دارند

و به مرور زمان دقت در هر دو بالا میرود و دیتا تست را فقط حفظ نمیکند که روی تست دقت بالا و روی validation دقت کمی داشته باشد پس overfit نمیشود

د)

یک لایه resize اضافه میکنیم بعد از خواندن ورودی اضافه میکنیم بعد مدل ResNet را لود میکنیم و وزن های ان را freeze

میکنیم زیرا ورودی ما دیتاست کوچکی است در اخر هم برای کلاس بندی دیتا یک لایه flatten,Dense قرار میدهیم تا ورودی یک بعدی شود و بعد با لایه dense کلاس بندی شود در واقع تنها لایه در حال یادگیری همین لایه است

```
modelres = keras.Sequential()
modelres.add(layers.Input(shape=x_train[0].shape))
modelres.add(layers.Resizing(224,224))
resnetmodel = tf.keras.applications.ResNet50(
    include_top=False,
    weights="imagenet",
)

#train nakon va freeze show chizi ke yad gerefte ra emal mikonad
resnetmodel.trainable = False
modelres.add(resnetmodel)
modelres.add(layers.Flatten())
modelres.add(layers.Dense(10,activation= 'softmax'))
modelres.summary()
```

ه)

ابتدا لایه های مدل را میبینیم و چون گفته شده که resize نکنیم پس input_shape میدهیم

```
resnetmodel=tf.keras.applications.ResNet101(
    include_top=False,
    weights="imagenet",
    input_shape=(32,32,3),
)
resnetmodel.trainable = False
```

```
Downloading data from https://storage.googleapis.com/te
171446536/171446536 [==============================] -
```

```
resnetmodel.summary(expand_nested=True)
```

سپس در لایه ها تا لایه مدنظر پیش میرویم و انرا ذخیره میکنیم و میدانیم شامل لایه های قبل نیز هست

```
for layer in resnetmodel.layers:
  if layer.name == 'conv3_block4_out' :
    layercorrect= layer
    break
```

```
flatlayer= layers.Flatten()(layercorrect.output)
middleDense= layers.Dense(1024, activation = 'relu')(flatlayer)
resultlayer= layers.Dense(10, activation = 'softmax')(middleDense)
model_resnet_new = keras.Model(inputs= [resnetmodel.input], outputs = [resultlayer])
model_resnet_new.summary()
```

مدل را مجدد میسازیم

4)

الف)

Stride در واقع تعداد پرش برحسب پیکسل برای محاسبه بعدی را معلوم میکند اگر برابر 1 باشد انگار پرشی نداریم و یکی یکی جلو میرویم و اگر برابر 2 باشد گویا یکی در میان جلو میرویم

در pooling که 3 مدل average, max , min دارد ما تصویر را گویا ناحیه بندی میکنیم و به جا ان ناحیه در تصویر جدید یک عدد بر حسب اینکه چه مدلی از pooling را انتخاب کردیم میگذاریم مثلا میانگین را جایگزین میکنیم

در stride ما گویا محاسبات لایه را برای پیکسل های کمتری انجام میدهیم که دقت ما را پایین تر می اورد ولی در pooling بعد از اتمام محاسبات ما یک لایه pooling داریم که ابعاد را کاهش میدهد و این لایه یادگیری ندارد

هر دو باعث کاهش ابعاد تصویر و کاهش احتمال overfit شدن میشود زیرا از انجایی که در stride ما پرش داریم و در pooling

ما یک محاسبه برای کل ناحیه داریم و عملا تمام پیکسل ها را نداریم سبب میشود که مدل ما نتواند تصویر های تست را حفظ کند و در نتیجه از overfitting جلوگیری میکند

تفاوت در اینجا است که در stride ما یکسری پیکسل را دور میریزیم ولی درpooling سعی داریم کل پیکسل های ان ناحیه در نتیجه اثرگذار باشند بنابراین اطلاعات کمتری از دست میرود مدل یادگیری بهتری خواهد داشت

ب)

در لایه های میانی از تابع فعال سازی relu استفاده میکنیم زیرا ساده است و ما با گرادیان ان اشنا هستیم و مشکلی در backpropagation ایجاد نمیکند همچنین از رشد نمایی در محسبات توابع ضرر و بهینه سازی جلوگیری میکند بنابراین از Exploding , vanishing جلوگیری میکند همچنین اگربخوهیم در سمت منفی ها مشتق صفر نشود از leacky relu میتوان استفاده کرد که کمی محاسبات در ان پیچیده تر است

در مرحله اخر ما یک classification داریم که میدانیم اگر binary باشد یعنی دو کلاس داشتیم از sigmoid, softmax میتوان استفاده کرد که اینجا هم همین است (اگر چند کلاس داشتیم باید از softmax استفاده گکنیم که برای چند کلاس کارامد است و یک احتمال بین 0و1 به ما برمیگرداند که مجموعا یک اند)

2)

بهترین تابع loss: crros entropy برای مسایلclassification استفاده میشود که از لگاریتم احتمال برای به دست اوردن مقدار خطا استفاده میکند

ID	Actual	Predicted probabilities	Corrected Probabilities	Log
ID6	1	0.94	0.94	-0.0268721464
ID1	1	0.90	0.90	-0.0457574906
ID7	1	0.78	0.78	-0.1079053973
ID8	0	0.56	0.44	-0.3565473235
ID2	0	0.51	0.49	-0.30980392

$$logloss = -\frac{1}{N}\sum_{i}^{N}\sum_{j}^{M} y_{ij}\log(p_{ij})$$

- N is the number of rows
- M is the number of classes

پیشبینی های دارای اختلاف بیشتر دارای لگاریتم بزرگتر و بلعکس تاثیر کمتری دارد

3)

Precision:

برابر مقدار پیش بینی است که ما درست پیش بینی کردیم که محصول سالم است به کل پیش بینی هایی که گفتیم محصول سالم است

Recall:

برابر مقدار پیش بینی هاییاست که گفتیم محصول سالم است و درست بوده به کل محصولات سالم

بنابراین precision برای ما بهترین معیار خواهد بود زیرا fasle precision یعنی میزان محصولاتی که ما در بین پیشبینی های خود اشتباه کردیم و به دست مشتری رسیده که باید مینیمم شود بنابراین باید precision بیشینه باشد

ج)

شبکه عصبی کانولوشنال (CNN یا ConvNet) یک معماری شبکه برای یادگیری عمیق است که مستقیماً از داده ها یاد می گیرد. CNN ها به ویژه برای یافتن الگوهایی در تصاویر برای تشخیص اشیا، کلاس ها و دسته ها مفید هستند آنها همچنین می توانند برای کلاس بندی تصویر و ویدیو موثر باشند

Recurrent neural networks ویژگی های متوالی داده ها را تشخیص می دهند و از الگوها برای پیش بینی سناریوی احتمالی بعدی استفاده می کنند. RNN ها در یادگیری عمیق و در توسعه مدل هایی استفاده می شوند که فعالیت نورون ها را در مغز انسان شبیه سازی می کنند.

بنابراین به جز مورد پردازش متن برای تمام موارد دیگراز cnn میتوان استفاده کرد

د)

1)

یک ConvNet به یک مجموعه داده بزرگ برای پردازش و آموزش شبکه عصبی نیاز دارد.

2) اگر CNN چندین لایه داشته باشد وhyperparameter زیادی دارد و کند است

3) نیاز به تکرار زیادی برای رسیدن به hyperparameter های درست دارد

4) اگر خوب اموزش ببیند باز هم اگر اشیا دچار اسکیل یا چرخش شوند به خوبی تشخیص نمیدهد یا اگر تصویر دارای نویز باشد

5)

با بررسی نام فایل ها متوجه میشویم به شکل زیر میتوان دیتا تست واموزشی را جدا کرد:

```
inflating: ./ss_dataset/3/365.bmp
inflating: ./ss_dataset/3/317_label.bmp
inflating: ./ss_dataset/3/147.bmp
inflating: ./ss_dataset/3/281.bmp
inflating: ./ss_dataset/3/228_label.bmp
inflating: ./ss_dataset/3/383_label.bmp
inflating: ./ss_dataset/3/376.bmp
inflating: ./ss_dataset/3/5_label.bmp
inflating: ./ss_dataset/3/486.bmp
inflating: ./ss_dataset/3/333.bmp
inflating: ./ss_dataset/3/580_label.bmp
inflating: ./ss_dataset/3/594_label.bmp
inflating: ./ss_dataset/3/458.bmp
inflating: ./ss_dataset/3/278_label.bmp
inflating: ./ss_dataset/3/126_label.bmp
inflating: ./ss_dataset/3/295_label.bmp
inflating: ./ss_dataset/3/68_label.bmp
inflating: ./ss_dataset/3/67.bmp
inflating: ./ss_dataset/3/89_label.bmp
```

```
for (dirpath, dirname, filenames) in os.walk(data_dir):
  for filename in filenames:
    Image = filename.split('.')[0]
    if 'label' in Image:
      labels.append(dirpath + f'/{filename}')
    else:
      images.append(dirpath + f'/{filename}')
```

سپس برای اینکه فرمت عکس ها را تشخیص دهیم به شکل زیر نام انها را استخراج و به فرمت png ذخیره میکنیم

```
for Image_path in images:
  FileNamelast = Image_path.split('/')[-1]
  FileName=FileNamelast.split('.')[0]
  Image = Image.open(Image_path)
  Image = Image.resize((256, 256))
  dirName = Image_path.split('/')[-2]
  Image.save(image_root + '/' + dirName + '_' + FileName + '.png', 'png')

for labelPath in labels:
  FileNameWithlabel = labelPath.split('/')[-1].split('.')[0]
  FileName=FileNameWithlabel.replace('_label', '')
  Image = Image.open(labelPath)
  Image = Image.resize((256, 256))
  directoryName = labelPath.split('/')[-2]
  Image.save(label_root + '/' + directoryName + '_' + FileName + '.png', 'png')
```

چون دیتاست دارای فولدر هایی است که در انها نام عکس ها یکسان است پس باید به اسم عکس ها نام فولدر را هم اضافه کنیم

سپس نیز dataframe با دو فیلد name,id طبق خواسته سوال ایجاد میکنیم

Index در این دیتا فریم معادل key در دیکشنری است و دیتا ها با ان شناخته میشوند

```
for (directorypath, directoryname, filenames) in os.walk(image_path):
  for fileName in filenames:
    resultName=directorypath + '/' + fileName
    imagePathes.append(resultName)
    imageNames.append(fileName.split('.')[0])

df = {'id': imageNames, name: imagePathes}
df = pd.DataFrame(df)
df.set_index('id')
```

در اخر برای agumentation یک عدد رندوم تولید و اگر بزرگتزر از 0.5 بود flip میکنیم

Preprocessing هم برای دیتا train دو کار resize,normalize و در دیتا label هم تنها سایز را درست میکنیم زیرا باید مقدار پیکسل پنل خورشیدی بیشتری باشد

```
img = tf.io.read_file(path)
img = tf.io.decode_jpeg(img, channels=3)
img = tf.image.resize(img, size)
img = tf.cast(img, tf.float32)/255.0

LabelImage = tf.io.read_file(LabelPath)
LabelImage = tf.io.decode_jpeg(LabelImage, channels=3)
LabelImage = tf.image.resize(LabelImage, size)
LabelImage = LabelImage[:, :, 0]
LabelImage = tf.math.sign(LabelImage)
```

یک thereshold میگذاریم تا همه را صفر و یک کنیم اگر بزرگتر از 128 بود یک وگرنه صفر که با تابع sign این کار را انجام میدهیم اگر اخرین بیت یک باشد یعنی از 128 بزرگتر وگرنه کوچک تر است

دیتاست را مانند توضیحات میسلزیم و در اخر بخشی از دیتا برای validation میگذاریم

```
DataFarameTrain, valid_df = train_test_split(DataFarameTrain, test_size=0.2)
train = create_dataset(DataFarameTrain, True)
valid = create_dataset(valid_df)
```

یک مدل موبایل نت میسازیم و لایه های انرا به مدل back میدهیم و قابلیت اموزش انرا متوقف میکنیم

```
size = (256, 256)
################################################
############## YOUR CODES GO HERE ##############
back = tf.keras.applications.MobileNetV2(size + (3,), include_top=False, weights='imagenet')
namesoflayers = ['block_1_expand_relu', 'block_3_expand_relu', 'block_6_expand_relu',
                 'block_13_expand_relu', 'block_16_project']
outputL = [back.get_layer(name).output for name in namesoflayers]
back = tf.keras.Model(inputs=back.input, outputs=outputL)
back.trainable = False
################################################
```

مدل جدید را طبق توضیحات میسازیم

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

################################################
############## YOUR CODES GO HERE ##############
newmodel = tf.keras.Sequential()
newmodel.add(tf.keras.layers.Conv2DTranspose(filters=filters, kernel_size=size, strides=(2, 2),
                                             padding='same', kernel_initializer=initializer, use_bias=False))
newmodel.add(tf.keras.layers.BatchNormalization())


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

return newmodel
```

سپس در این قسمت برای ساخت مدل :Unet

```
inputs = tf.keras.layers.Input(shape=size + (3,))
back_outputs = back(inputs)
temp = back_outputs[-1]
reversed_encoder_back = reversed(back_outputs[0:-1])
for dec, back in zip(up_stack, reversed_encoder_back):
    temp = dec(temp)
    temp = tf.keras.layers.concatenate([temp, back])

last_classifier = tf.keras.layers.Conv2DTranspose(output_channels, 3, strides=2, activation='sigmoid', padding='same')
temp = last_classifier(temp)
model = tf.keras.Model(inputs=inputs, outputs=x)
return model
################################################
```

ما ابتدا یک encoder داریم که همان back است و خروجی هر مرحله انرا ذخیره میکنیم

و مراحل decoder را هم در up_stack داریم تنها کاری که باید بکنیم این است که خروجی مراحل encoder را به تک تک مراحل decoder بدهیم و سپس با خروجی ان سمت concat کنیم دلیل reverse هم به این علت است که خروجی ها را از اخر به اول میخواهیم

```
newmodel = unet_model(1)
newmodel.compile(optimizer='adam',
                 loss=dice_loss,
                 metrics=['binary_accuracy', dice_coef])
```

مدل را ساختیم

```
earlystop = tf.keras.callbacks.EarlyStopping(patience=4, restore_best_weights=True)
```

این call back که کلا در حین اجرا پراسس را انجام میدهند می اید میبیند اگر تا 4 گام جلو تر مقدار val_loss کاهش نیافت فرایند را متوقف میکند