

مقدمه :

در این تکلیف میبایستی روی دیتاست Derma MNIST شبکه عصبی کانولوشنی را پیاده سازی می کردیم. این شبکه با استفاده از معماری معروف VGG پیاده سازی شده است و از روی آن الهام گرفته شده است. در ادامه نیز یک مدل غیر خطی نیز با keras functional API پیاده سازی شده است که در ادامه این گزارش به همه این موارد پرداخته خواهد شد.

درک دیتاست Derma MNIST:

دیتاست موجود از لینک

<https://zenodo.org/record/4269852/files/dermamnist.npz?download=1>

گرفته شده است که می توان در google colab با استفاده از کامند کنترلی wget آن را در notebook پروژه دانلود کرد. داده های دریافتی به ۳ بخش train، validation و test تقسیم می شوند. بچ train شامل ۷۰۰۷ عدد سمپل دیتا عکس رنگی که ماتریس فیچر هایمان (X) را تشکیل می دهند، و شامل لیبل های این سمپل ها (y) که اعدادی در بازه ۰ الی ۶ می باشند. در واقع دیتاست مسئله شامل ۷ کلاس می باشد. سائز اولیه هر کدام از سمپل ها یا همان عکس ها ۲۸ در ۲۸ می باشند که برای کارایی بالاتر و دادن این سمپل ها به شبکه کانولوشنی، آن ها را در مرحله پیش پردازش resize می کنیم. داده های validation و test نیز به ترتیب شامل ۱۰۰۳ و ۲۰۰۵ عدد سمپل می باشند که پیش پردازش ها را نیز روی آن ها اعمال میکنیم.

در ادامه برای درک از توزیع دادگان برای هر کلاس در مسئله در هر کدام از سه بچ گفته شده، را نمایش میدهم (شکل ۱)

```
train_labels :
Label counts of (0): 228
Label counts of (1): 359
Label counts of (2): 769
Label counts of (3): 80
Label counts of (4): 779
Label counts of (5): 4693
Label counts of (6): 99

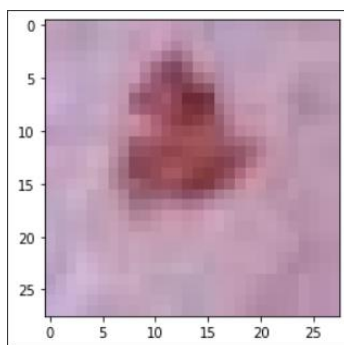
val_labels :
Label counts of (0): 33
Label counts of (1): 52
Label counts of (2): 110
Label counts of (3): 12
Label counts of (4): 111
Label counts of (5): 671
Label counts of (6): 14

test_labels :
Label counts of (0): 66
Label counts of (1): 103
Label counts of (2): 220
Label counts of (3): 23
Label counts of (4): 223
Label counts of (5): 1341
Label counts of (6): 29
```

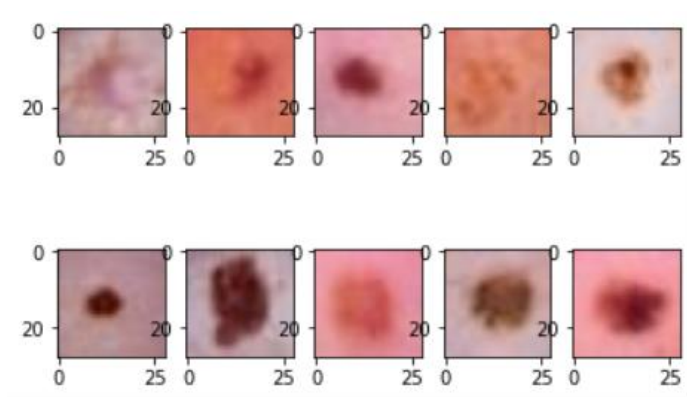
شکل ۱

همانطور که در **شکل ۱** مشخص می باشد، توزیع دادگان در کلاس های مسئله کمی غیریکنواخت می باشد به عنوان مثال در ۳ بیج آموزش و ارزیابی و تست تعداد داده های کلاس شماره ۵ خیلی بیشتر است نسبت به کلاس های دیگر در دیتاست برای این کار می شود از resampling استفاده کرد اما با این روش تغییر در دقت مدل ها صورت نگرفت و حتی بدتر شد به همین دلیل از این تکنیک صرف نظر شد و در کد اضافه نشد.

برای اطمینان بیشتر از صحت و درک بیشتر از سمپل های دیتاست (بیج آموزشی)، آن ها را visualize می کنیم (**شکل ۲** و **شکل ۳**):



شکل ۲



شکل ۳

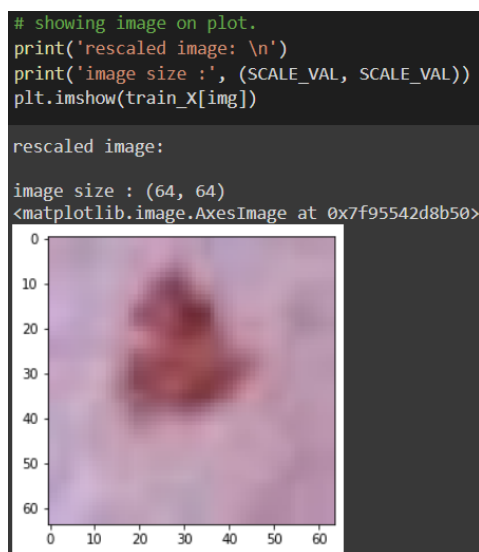
همانطور که مشخص دادگان این مسئله در رابطه با نوعی بیماری پوستی هستند که به کمک مدل های CNN می توانیم به حل این مسئله کمک کنیم.

در ادامه به سراغ پیش پردازش این داده های تصویری میرویم.

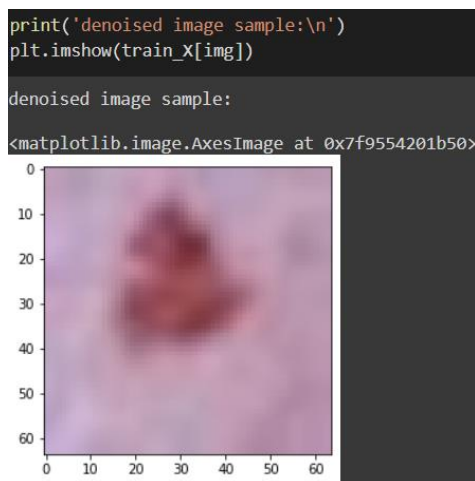
پیش پردازش داده ها:

در این مسئله از ۵ پیش پردازش داده ها استفاده شد که شامل:

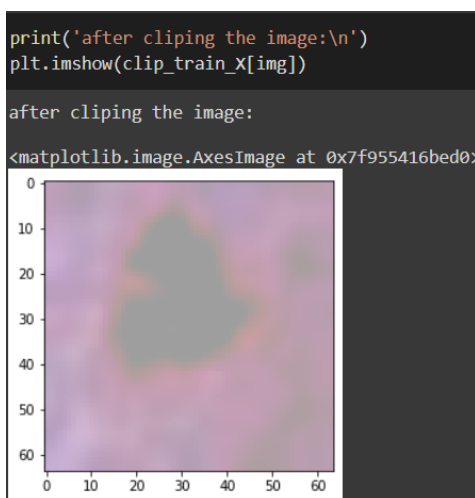
۱. نرمالایز کردن دیتا پیکسل تصاویر
۲. oneHotEncode برای لیبیل ها تا بتوان با softmax و معیار خطا categorical_crossentropy کار کرد.
۳. Rescale کردن تصاویر به توان های ۲ که این جا به ۶۴ در ۶۴ تبدیل شده اند تا بتوان راحتتر با معماری VGG کار کرد. (شکل ۴)
۴. از بین بردن نویز denoise داده های تصاویر به کمک فیلتر GaussianBlur در کتابخانه openCv انجام گرفته است به کمک این فیلتر می توان تصاویر را کمی smooth تر کرد و داده های نویز را تا حد امکان کاهش داد تا شبکه های CNN محاسبات با خطای نسبتا کمتری داشته باشند. (شکل ۵)
۵. Clip کردن تصاویری که مقدار پیکسل هایشان از میانگین کمتر هستند. این کار کمی به توزیع مقادیر پیکسل ها در تصویر ها برای تشخیص شیء مورد نظر مثلا یک غده سرطانی کمک میکند ولی این پیش پردازش روی داده ها اعمال نشدند زیرا تا حدی accuracy را کاهش می دادن و نحوه کارکرد آن در notebook با یک مثال آورده شده است. (شکل ۶)



شکل ۴



شکل ۵



شکل ۶

توجه شود که این پیش پردازش ها روی داده های train، validation و test انجام گرفته اند.

توجه شود که مدل های پیاده سازی شده بدون پیش پردازش ها نیز تست شده اند که در آخر گزارش نمودار های learning curve آن ها برای loss و accuracy آورده شده اند.

برای ران کردن بخش غیر پیش پردازش داده ها، تمامی پیش پردازش ها غیر از توابع rescale و oneHotEncode کامنت شدند و بلاک non-preprocessing از حالت کامنت در آمدند.

حال در ادامه به سراغ مدل های CNN می رویم که برای مسئله پیاده شده اند.

مدل ها و تنظیم کردن پارامترهای آن ها:

در این پروژه برای تنظیم کردن پارامترها و تست کردن حالات متفاوت ۷ مدل به وسیله داده های train آموزش داده شده اند و با داده های validation تست شده اند. علاوه بر ۷ مدلی که مدل Sequential هستند، یک مدل غیرخطی به کمک functional API طراحی شده است که مربوط به بخش امتیازی می باشد.

پایه مدل ها ۲ مدل cnn_model_1 و cnn_model_2 می باشد که به ترتیب پارامترهایشان در مدل های بعدی ارتقا یافته است. cnn_model_1 به طور کلی شامل ۵ لایه کانولوشنی می باشد که فقط تعداد بخش هایی لایه ایی که ۵ بخش است (لایه های pooling و conv2D) از VGG الهام گرفته شده است. (شکل ۷)

cnn_model_2 کاملاً شبیه VGG_16 می باشد. به طور کلی شامل ۱۳ لایه کانولوشن می باشد که در ۲ بخش اول هر کدام شامل ۲ لایه کانولوشنی و یک لایه maxPooling می باشد و در ۳ لایه آخر هر کدام شامل ۳ لایه کانولوشنی و یک لایه maxPooling می باشد و در انتها نیز در بخش fully connected نیز که به عنوان classification بین کلاس ها استفاده می شود، یک لایه با تابع فعالسازی relu می باشد و در نهایت لایه آخر یا همان output نیز شامل ۷ نرون با تابع فعال سازی softmax می باشد. (شکل ۸)

```

FILTER_SIZE = (3, 3)
POOLING_SIZE = (2, 2)
NUM_CLASSES = train_y.shape[1]

cnn_model_1 = Sequential()

#----- conv layer 1 -----
cnn_model_1.add(Conv2D(8, FILTER_SIZE, padding='same', activation='relu', input_shape=(SCALE_VAL, SCALE_VAL, 3)))
cnn_model_1.add(MaxPool2D(POOLING_SIZE, strides=(2,2))) # output to next layer => 32, 32, 8
#----- conv layer 2 -----
cnn_model_1.add(Conv2D(16, FILTER_SIZE, padding='same', activation='relu'))
cnn_model_1.add(MaxPool2D(POOLING_SIZE, strides=(2,2))) # output to next layer => 16, 16, 16
#----- conv layer 3 -----
cnn_model_1.add(Conv2D(32, FILTER_SIZE, padding='same', activation='relu'))
cnn_model_1.add(MaxPool2D(POOLING_SIZE, strides=(2,2))) # output to next layer => 8, 8, 32
#----- conv layer 4 -----
cnn_model_1.add(Conv2D(64, FILTER_SIZE, padding='same', activation='relu'))
cnn_model_1.add(MaxPool2D(POOLING_SIZE, strides=(2,2))) # output to next layer => 4, 4, 64
#----- conv layer 5 -----
cnn_model_1.add(Conv2D(128, FILTER_SIZE, padding='same', activation='relu'))
cnn_model_1.add(MaxPool2D(POOLING_SIZE, strides=(2,2))) # output to next layer => 2, 2, 128
#----- FC layer -----

cnn_model_1.add(Flatten())
cnn_model_1.add(Dense(512, activation='relu'))
cnn_model_1.add(Dense(NUM_CLASSES, activation='softmax'))
cnn_model_1.summary()

```

شکل ۷

دقت شود که به ترتیب تغییرات رو مدل با پارامترهای مختلف و افزودن regularization و dropout در مدل های بعدی انجام میگیرند. این ۲ مدل صرفاً ۲ مدل پایه برای شروع تست و تنظیم پارامترهای اولیه می باشند.

```

cnn_model_2 = Sequential()
#----- conv layer 1 -----
cnn_model_2.add(Conv2D(8, FILTER_SIZE, padding='same', activation='relu', input_shape=(SCALE_VAL, SCALE_VAL, 3)))
cnn_model_2.add(Conv2D(8, FILTER_SIZE, padding='same', activation='relu'))
cnn_model_2.add(MaxPool2D(POOLING_SIZE, strides=(2,2))) # output to next layer => 32, 32, 8
#----- conv layer 2 -----
cnn_model_2.add(Conv2D(16, FILTER_SIZE, padding='same', activation='relu'))
cnn_model_2.add(Conv2D(16, FILTER_SIZE, padding='same', activation='relu'))
cnn_model_2.add(MaxPool2D(POOLING_SIZE, strides=(2,2))) # output to next layer => 16, 16, 16
#----- conv layer 3 -----
cnn_model_2.add(Conv2D(32, FILTER_SIZE, padding='same', activation='relu'))
cnn_model_2.add(Conv2D(32, FILTER_SIZE, padding='same', activation='relu'))
cnn_model_2.add(Conv2D(32, FILTER_SIZE, padding='same', activation='relu'))
cnn_model_2.add(MaxPool2D(POOLING_SIZE, strides=(2,2)))
#----- conv layer 4 -----
cnn_model_2.add(Conv2D(64, FILTER_SIZE, padding='same', activation='relu'))
cnn_model_2.add(Conv2D(64, FILTER_SIZE, padding='same', activation='relu'))
cnn_model_2.add(Conv2D(64, FILTER_SIZE, padding='same', activation='relu'))
cnn_model_2.add(MaxPool2D(POOLING_SIZE, strides=(2,2)))
#----- conv layer 5 -----
cnn_model_2.add(Conv2D(128, FILTER_SIZE, padding='same', activation='relu'))
cnn_model_2.add(Conv2D(128, FILTER_SIZE, padding='same', activation='relu'))
cnn_model_2.add(Conv2D(128, FILTER_SIZE, padding='same', activation='relu'))
cnn_model_2.add(MaxPool2D(POOLING_SIZE, strides=(2,2)))
#----- FC layer -----
cnn_model_2.add(Flatten())
cnn_model_2.add(Dense(512, activation='relu'))
cnn_model_2.add(Dense(NUM_CLASSES, activation='softmax'))
cnn_model_2.summary()

```

شکل ۸

در کامنت ها در **شکل ۷** مشخص است که در هر لایه ورودی به لایه بعدی یه ابعادی دارد به عنوان مثال در همین شکل بعد بخش ۵ ام یعنی آخرین لایه maxpooling ورودی که به flatten داده می شود $128 \times 2 \times 2$ می باشد که این مورد در summery مدل `cnn_model1` (**شکل ۹**) مشخص است، برای بقیه مدل ها نیز به همین شکل است.

Model: "sequential"		
Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 64, 64, 8)	224
max_pooling2d (MaxPooling2D)	(None, 32, 32, 8)	0
conv2d_1 (Conv2D)	(None, 32, 32, 16)	1168
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 16)	0
conv2d_2 (Conv2D)	(None, 16, 16, 32)	4640
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 32)	0
conv2d_3 (Conv2D)	(None, 8, 8, 64)	18496
max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 64)	0
conv2d_4 (Conv2D)	(None, 4, 4, 128)	73856
max_pooling2d_4 (MaxPooling2D)	(None, 2, 2, 128)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 512)	262656
dense_1 (Dense)	(None, 7)	3591
=====		
Total params: 364,631		
Trainable params: 364,631		
Non-trainable params: 0		

شکل ۹

همانطور که در **شکل ۹** مشخص است ورودی لایه flatten یا همان shape آن در ستون دوم summery برابر $512 = 2 * 2 * 128$ می باشد.

نحوه بدست آوردن تعداد وزن ها یا تعداد پارامتر ها نیز به این شکل می باشد:

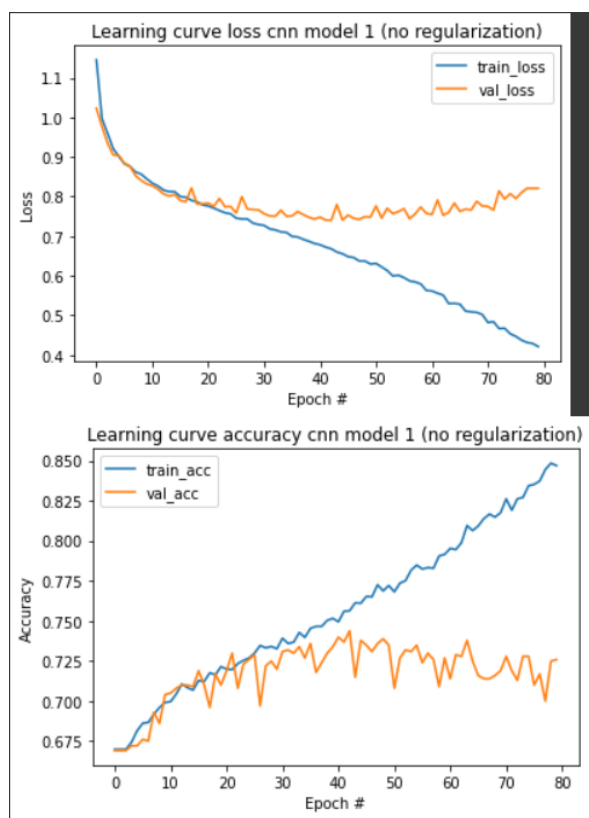
Calculating parameters (weights) in each layer:

- Parameters = (FxF * number of channels + bias-term) * depth
for example in first conv2d layer we have: $(3 * 3 * 3 + 1) * 8 = 224$

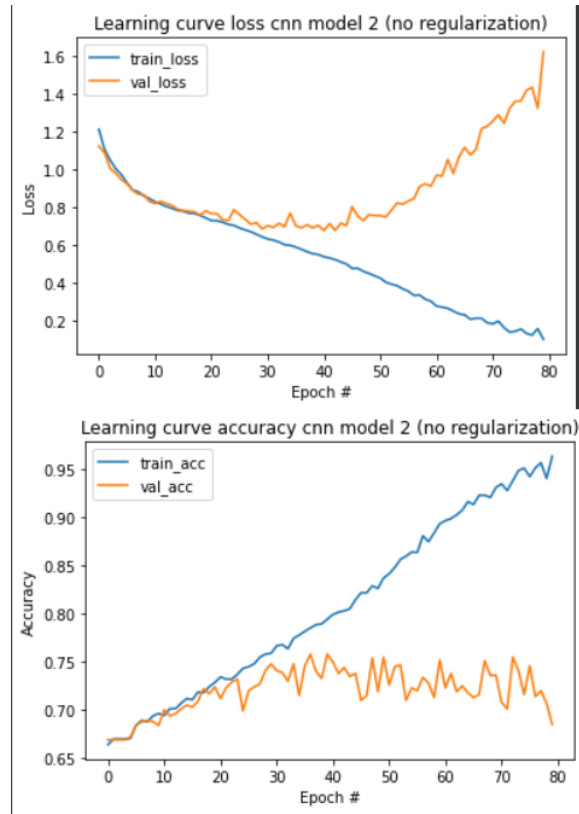
که این فرمول در notebook آورده شده است.

حال در ادامه به سراغ آموزش مدل ها و ارزیابی مدل ها به وسیله داده های ارزیابی و آموزشی می رویم.

با استفاده از نمودار learning curve برای مدل ها آن ها را تحلیل می کنیم:



شکل ۱۰



شکل ۱۱

با توجه به نمودار های ۲ مدل پایه (شکل ۱۰ و شکل ۱۱)، متوجه می شویم ۲ مدل به شدت overfit شده اند و در epoch حدودا ۴۰ برای cnn_model_1 و epoch حدودا ۳۵ برای cnn_model_2 شاهد overfit شدن مدل ها هستیم.

در این شرایط مدل خیلی نمی تواند داده های جدید را سامان دهی کند و کلاس بندی کند (generalize کند) هرچند که داده های آموزشی را میتواند به خوبی یادبگیرد.

در مرحله به دلیل رخدادن overfit ما از regularization و dropout استفاده می کنیم.

در شکل ۱۲ که به cnn_model_1، regularization و dropout اضافه شده است را مشاهده می کنیم. در این مسئله از ریگولاریزیشن L2 استفاده شده است. در واقع با اضافه کردن L2 یک هزینه اضافی به loss function می شود که نمی گذارد مدل وزن های بیش از حد زیادی را بدست آورد که این خود سبب جلوگیری از overfitting می شود. و اینکه L2 مجموع توان ۲ وزن ها را در نظر می گیرد و به همین دلیل وزن های بزرگ فقط کوچک می شوند و وزن های کوچک کوچکتر از حد نمی شوند. در مدل ها فقط در لایه آخر از dropout استفاده شده است زیرا مدل دقت کمی پیدا میکرد که در شکل ۱۳ مشخص است.


```

cnn_model_3 = Sequential()

#----- conv layer 1 -----
cnn_model_3.add(Conv2D(8, FILTER_SIZE, padding='same', activation='relu', input_shape=(SCALE_VAL, SCALE_VAL, 3), kernel_regularizer= l2(0.001)))
cnn_model_3.add(MaxPool2D(POOLING_SIZE, strides=(2,2))) # output to next layer => 32, 32, 8
# cnn_model_3.add(Dropout(0.2))

#----- conv layer 2 -----
cnn_model_3.add(Conv2D(16, FILTER_SIZE, padding='same', activation='relu', kernel_regularizer= l2(0.001)))
cnn_model_3.add(MaxPool2D(POOLING_SIZE, strides=(2,2))) # output to next layer => 16, 16, 16
# cnn_model_3.add(Dropout(0.2))

#----- conv layer 3 -----
cnn_model_3.add(Conv2D(32, FILTER_SIZE, padding='same', activation='relu', kernel_regularizer= l2(0.001)))
cnn_model_3.add(MaxPool2D(POOLING_SIZE, strides=(2,2)))
# cnn_model_3.add(Dropout(0.2))

#----- conv layer 4 -----
cnn_model_3.add(Conv2D(64, FILTER_SIZE, padding='same', activation='relu', kernel_regularizer= l2(0.001)))
cnn_model_3.add(MaxPool2D(POOLING_SIZE, strides=(2,2)))
# cnn_model_3.add(Dropout(0.2))

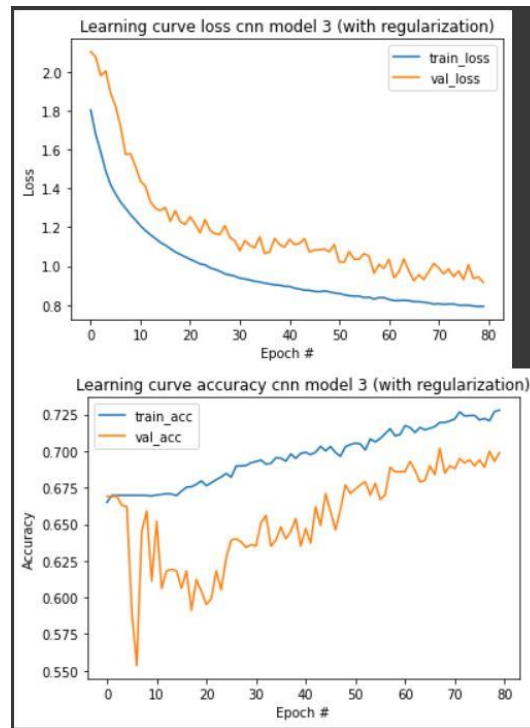
#----- conv layer 5 -----
cnn_model_3.add(Conv2D(128, FILTER_SIZE, padding='same', activation='relu', kernel_regularizer= l2(0.001)))
cnn_model_3.add(MaxPool2D(POOLING_SIZE, strides=(2,2)))
# cnn_model_3.add(Dropout(0.2))

#----- FC layer -----

cnn_model_3.add(Flatten())
cnn_model_3.add(Dense(512, activation='relu', kernel_regularizer= l2(0.001)))
cnn_model_3.add(Dropout(0.3))
cnn_model_3.add(Dense(NUM_CLASSES, activation='softmax'))
cnn_model_3.summary()

```

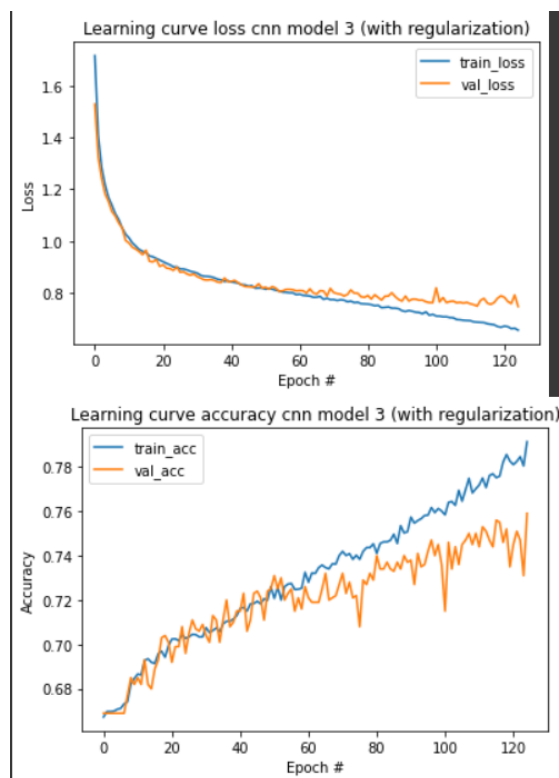
شکل ۱۲



شکل ۱۳

نمودار رسم شده **شکل ۱۳** با برداشتن کامنت های dropout در **شکل ۱۲** برای بخش های کانولوشنی بدست آمده است.

در شکل ۱۴ که مربوط به شکل ۱۲ می باشد، نشان میدهد که از overfitting به مقدار قابل توجهی کاسته شده است.



شکل ۱۴

در cnn_model_4 نیز که همان مدل پایه cnn_model_2 می باشد که به آن regularization و dropout استفاده شده، کاهش میزان overfitting به شکل قابل توجهی مشهود است. (شکل ۱۵)

توجه شود که در ۲ مدل cnn_model_3 و cnn_model_4 میزان epoch نیز برای هر کدام از ۲ مدل که همان مدل های cnn_model_1 و cnn_model_2 بودند نیز تغییر کرده اند که به ترتیب ۱۲۵ و ۱۴۵ می باشد.

دقت ها برای مدل cnn_model_3:

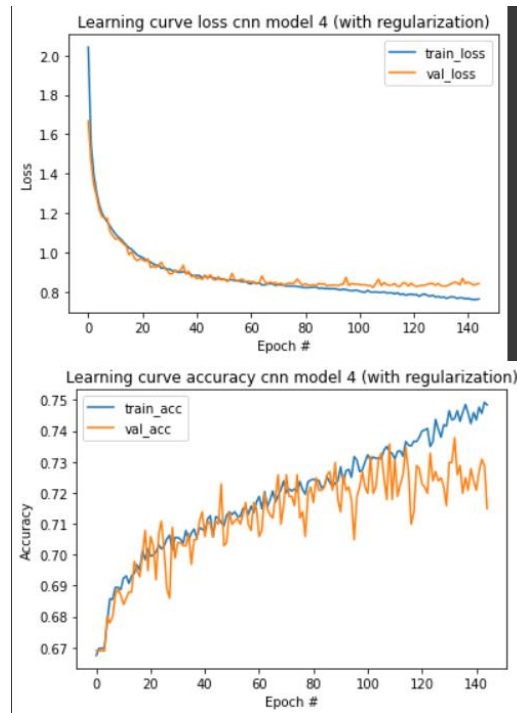
Train_acc: 0.7909

Validation_acc: 0.7587

دقت ها برای مدل cnn_model_4:

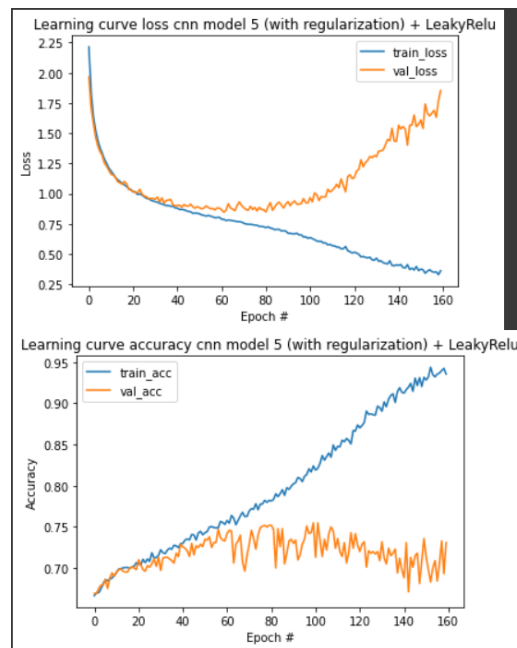
Train_acc: 0.7484

Validation_acc: 0.7149

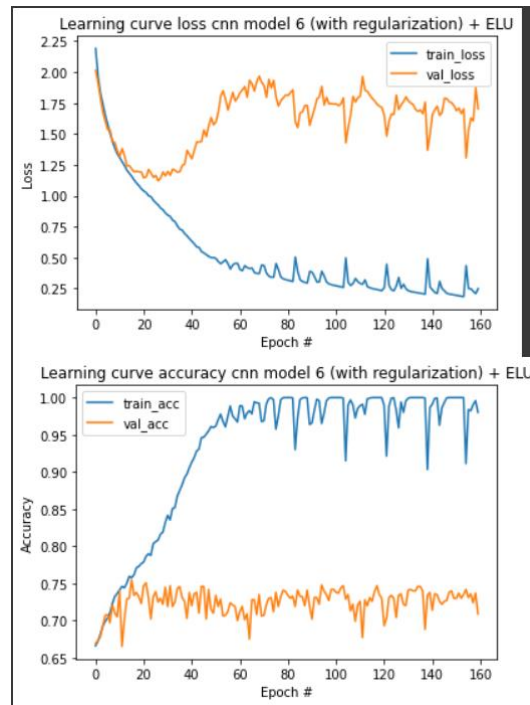


شکل ۱۵

در ادامه نیز توابع فعالسازی LeakyRelu و ELU بر روی مدل cnn_model_4 تست شده اند.



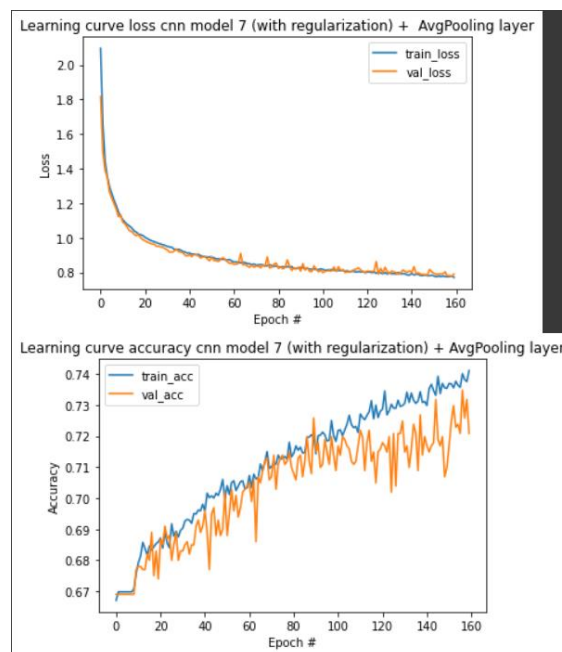
شکل ۱۶



شکل ۱۷

همانطور که در **شکل ۱۶** و **شکل ۱۷** مشخص است مدل ها به شدت overfit شده اند.

در ادامه نیز روی مدل cnn_model_4 (مدل هفتم) به جای لایه maxpool از لایه Average pooling استفاده می کنیم.



شکل ۱۸

همانطور که در **شکل ۱۸** مشاهده می شود عملکرد مدل cnn_model_7 نیز به نسبت خوب است و اصلا overfit نیست حتی می توانستیم میزان epoch را نیز در این مدل افزایش دهیم ولی چون این پارامتر را ثابت گرفتیم.

ولی با توجه به لایه maxpooling و خطاهای مدل cnn_model_3 و cnn_model_4، که خطا کمتر است ما لایه maxpooling را در نظر میگیریم. به طور کلی نیز چون تسک ما پیدا کردن یک شی خاص مثل غده سرطانی در یک پس زمینه روشن می باشد، بهتر است که برای افزایش دقت مدل و کاهش خطای مدل، از روش maxpooling استفاده کنیم تا بتوان پیکسل های با مقدار بالاتر (مربوط به شی خاص در تصویر) که پر ارزش تر هم می باشند نسبت به پیکسل های دیگر در تصویر، استخراج کنیم. ولی به طور کلی Average pooling صرفا به صورت smooth ترکیبی از پیکسل های کم ارزش و پر ارزش را تولید می کند که ممکن است کمی در دقت و خطا مدل تاثیر گذار باشد.

دقت ها برای مدل cnn_model_5:

Train_acc: 0.9355

Validation_acc: 0.7308

دقت ها برای مدل cnn_model_6:

Train_acc: 0.9802

Validation_acc: 0.7089

دقت ها برای مدل cnn_model_7:

Train_acc: 0.7411

Validation_acc: 0.7208

در ادامه به مدل غیر خطی پیاده شده به وسیله functional API می پردازیم:

مدل ساخته شده شامل ۲ شاخه feature extractor می باشد که ورودی را به لایه های کانولوشنی ۲ شاخه می دهیم و در نهایت در یک لایه مشترک آن ها را با یکدیگر merge می کنیم و در نهایت لایه حاصله را به fully connected می دهیم که تسک classification را برای ما انجام دهد. این موارد در **شکل ۱۹** و **شکل ۲۰** مشخص می باشند.

همچنین ساختار و معماری این CNN در **شکل ۲۱** قابل مشاهده می باشد.

Non linear model using functional API:

```
[ ] from tensorflow.keras.utils import plot_model
    from tensorflow.keras.layers import Input
    from tensorflow.keras.layers import concatenate
    from tensorflow.keras.models import Model
    from keras.layers.pooling import MaxPooling2D
```

شکل ۱۹ کتاب خانه ها و ماژول های مورد استفاده در کراس

```
visible = Input(shape = (SCALE_VAL, SCALE_VAL, 3))

# first feature extractor-----
conv1_1 = Conv2D(32, FILTER_SIZE, activation='relu', kernel_regularizer = l2(0.001))(visible)
conv1_2 = Conv2D(64, FILTER_SIZE, activation='relu', kernel_regularizer = l2(0.001))(conv1_1)
pool1 = MaxPool2D(Pooling_size, strides=(2,2))(conv1_2)
dp1 = Dropout(0.3)(pool1)
flat1 = Flatten()(dp1)

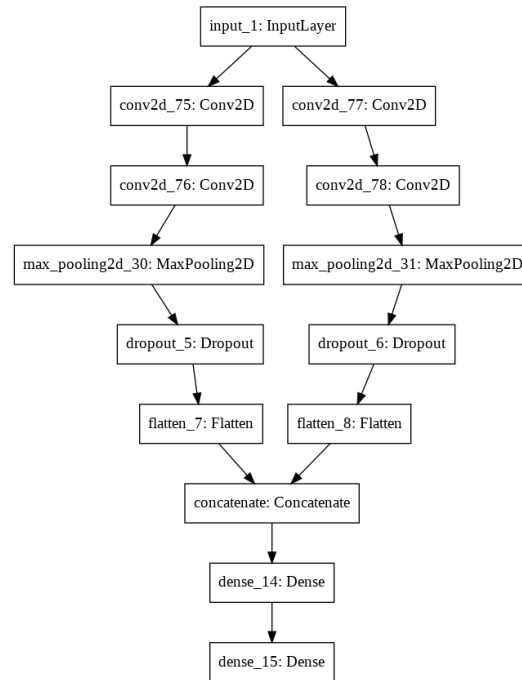
# sec feature extractor-----
conv2_1 = Conv2D(16, FILTER_SIZE, activation='relu', kernel_regularizer = l2(0.001))(visible)
conv2_2 = Conv2D(32, FILTER_SIZE, activation='relu', kernel_regularizer = l2(0.001))(conv2_1)
pool2 = MaxPool2D(Pooling_size, strides=(2,2))(conv2_2)
dp2 = Dropout(0.3)(pool2)
flat2 = Flatten()(dp2)

# merge feature extractors-----
merge = concatenate([flat1, flat2])

# classification part-----
hidden1 = Dense(512, activation = 'relu', kernel_regularizer = l2(0.001))(merge)
cnn_model_nonlinear = (Dropout(0.3))(hidden1)
# prediction output
output = Dense(NUM_CLASSES, activation = 'softmax')(hidden1)
cnn_model_nonlinear = Model(inputs = visible, outputs = output)

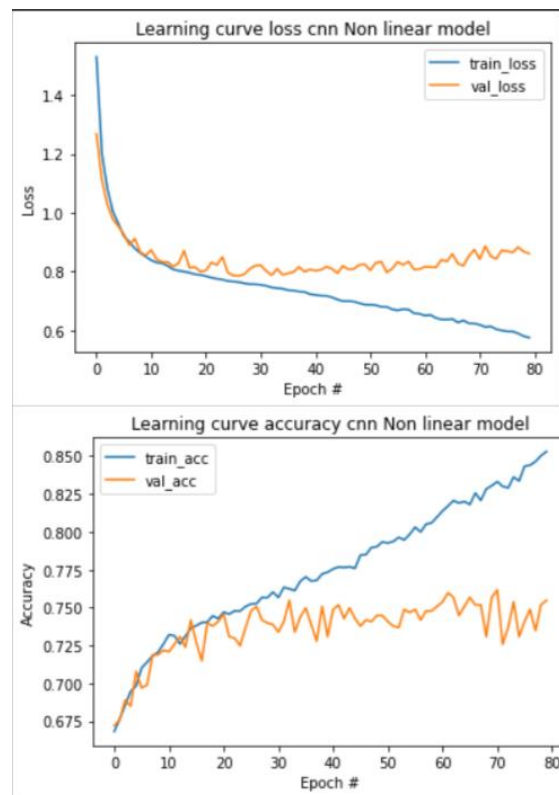
cnn_model_nonlinear.summary()
plot_model(cnn_model_nonlinear, to_file='shared_input_layer.png')
```

شکل ۲۰



شکل ۲۱

حال در ادامه نیز learning curve این مدل غیر خطی را رسم میکنیم:



دقت های مدل غیر خطی:

Train_acc: 0.8530

Validation_acc: 0.7547

توجه شود که تغییر مقادیر کرنل ها و stride صرفا روی shape خروجی برای لایه بعد تاثیر گذار است و این موارد تست شدن و بهترین حالت کرنل های 3×3 و stride برابر ۱ برای لایه های کانولوشنی و برای لایه های pooling سائز کرنل 2×2 و stride=2 می باشند.

خروجی هر لایه برای لایه های کانولوشنی برابر است با:

The output of the CONV layer is then $W_{output} \times H_{output} \times D_{output}$, where:

- $W_{output} = ((W_{input} - F + 2P) / S) + 1$
- $H_{output} = ((H_{input} - F + 2P) / S) + 1$
- $D_{output} = K$

که W عرض و H ارتفاع و D عمق تصویر خروجی از لایه کانولوشنی می باشد.

برای لایه های pooling نیز خواهیم داشت:

Applying the POOL operation yields an output volume of size

$W_{output} \times H_{output} \times D_{output}$, where:

- $W_{output} = ((W_{input} - F) / S) + 1$
- $H_{output} = ((H_{input} - F) / S) + 1$
- $D_{output} = D_{input}$

در نهایت در ادامه یکسری ارزیابی روی مدل های برتر انجام میدهم. مدل های انتخابی مدل های شماره ۳ و ۵ و مدل غیر خطی می باشد.

ارزیابی روی ۳ مدل:

همانطور که گفته شد به ترتیب برای ۳ مدل ابتدا ماتریس confusion رسم می شود و سپس متریک های recall و precision و accuracy برای ۷ کلاس در مسئله رسم میشوند. (شکل ۲۲ و شکل ۲۳)

```
Confusion matrix for cnn model 1:
[[ 7.  9.  7.  0.  4.  6.  0.]
 [ 1. 33.  5.  0.  1. 12.  0.]
 [ 3.  7. 48.  0.  8. 44.  0.]
 [ 3.  7.  0.  0.  0.  2.  0.]
 [ 4.  1.  7.  0. 39. 60.  0.]
 [ 1.  7. 18.  0. 17. 628.  0.]
 [ 1.  2.  0.  0.  1.  4.  6.]]

Confusion matrix for cnn_model_5:
[[ 10.  9.  6.  0.  2.  6.  0.]
 [ 7. 26.  6.  3.  2.  8.  0.]
 [ 9.  9. 38.  2.  5. 47.  0.]
 [ 3.  6.  0.  1.  0.  2.  0.]
 [ 4.  1. 16.  0. 28. 62.  0.]
 [ 1. 11. 22.  0. 14. 623.  0.]
 [ 1.  3.  0.  0.  0.  3.  7.]]

Confusion matrix for cnn model 3:
[[ 8. 10.  4.  0.  5.  6.  0.]
 [ 6. 33.  4.  0.  1.  8.  0.]
 [ 1. 13. 44.  0. 15. 37.  0.]
 [ 5.  5.  0.  0.  0.  2.  0.]
 [ 4.  0.  3.  1. 48. 55.  0.]
 [ 3.  6. 14.  0. 28. 619.  1.]
 [ 1.  1.  1.  0.  1.  5.  5.]]
```

شکل ۲۲

```
for model 1:
on label 0, acc=96.11, precision=35.00, recall=21.21
on label 1, acc=94.82, precision=50.00, recall=63.46
on label 2, acc=90.13, precision=56.47, recall=43.64
on label 3, acc=98.80, precision=nan, recall=0.00
on label 4, acc=89.73, precision=55.71, recall=35.14
on label 5, acc=82.95, precision=83.07, recall=93.59
on label 6, acc=99.20, precision=100.00, recall=42.86

for model 2:
on label 0, acc=95.21, precision=28.57, recall=30.30
on label 1, acc=93.52, precision=40.00, recall=50.00
on label 2, acc=87.84, precision=43.18, recall=34.55
on label 3, acc=98.40, precision=16.67, recall=8.33
on label 4, acc=89.43, precision=54.90, recall=25.23
on label 5, acc=82.45, precision=82.96, recall=92.85
on label 6, acc=99.30, precision=100.00, recall=50.00

for model 3:
on label 0, acc=95.51, precision=28.57, recall=24.24
on label 1, acc=94.62, precision=48.53, recall=63.46
on label 2, acc=90.83, precision=62.86, recall=40.00
on label 3, acc=98.70, precision=0.00, recall=0.00
on label 4, acc=88.73, precision=48.98, recall=43.24
on label 5, acc=83.55, precision=84.56, recall=92.25
on label 6, acc=99.00, precision=83.33, recall=35.71
```

شکل ۲۳

در ادامه نیز خطا و دقت validation را برای انتخاب مدل نهایی نشان می دهیم:

```
[ ] score_model1, score_model2, score_model3 = model_loss_acc(cnn_model_3, val_X, val_y), model_loss_acc(cnn_model_5, val_X, val_y), model_loss_acc(cnn_model_nonlinear, val_X, val_y)

print("model 1 (val-loss, val-acc): ", (score_model1[0], score_model1[1])) # for cnn_model_3
print("model 2 (val-loss, val-acc): ", (score_model2[0], score_model2[1])) # for cnn_model_5
print("model 3 (va-loss, val-acc): ", (score_model3[0], score_model3[1])) # for cnn_model_nonlinear

model 1 (val-loss, val-acc): (0.7473375797271729, 0.7587238550186157)
model 2 (val-loss, val-acc): (1.8533053398132324, 0.7308076024055481)
model 3 (va-loss, val-acc): (0.861431360244751, 0.7547357678413391)
```

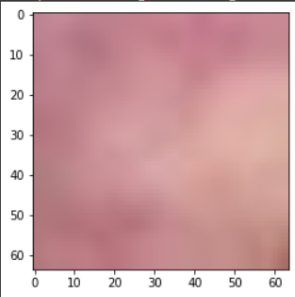
توجه شود که توابع استفاده شده در این بخش در Basic metrics functions که در notebook مشخص می باشد پیاده سازی شده اند.

در نهایت با توجه به معیار ها، متوجه می شویم که مدل شماره ۱ که در واقع همان cnn_model_3 می باشد بهتر است.

حتی جالب است که غیر بالانس بودن دیتاست در این جا نیز خودش را نمایش می دهد با توجه به **شکل ۲۲** و **شکل ۲۳** کلاس ۵ خیلی بهتر تشخیص داده شده است و recall این کلاس از بقیه کلاس ها بیشتر است.

در ادامه نیز یک سمپل تستی از دیتا های val را نیز با مدل برای امتحان کردن تابع predict استفاده شده است:

```
img = 102
plt.imshow(val_X[img]) # testing sample that we want to test trained model with predict function.

<matplotlib.image.AxesImage at 0x7f9460a80e10>


[ ] # (cnn_model_3.predict(val_X[img].reshape(1, 64, 64, 3))).argmax(axis=1)
y = val_X[img].reshape(1, 64, 64, 3)
label = val_y[img].argmax(axis=0)
predict = (cnn_model_3.predict(y)).argmax(axis=1)
print('predicted: ', predict, '\nlabel: ', label)

predicted: [2]
label: 2
```

در ادامه با توجه به نتایج مدل cnn_model_3 را انتخاب می کنیم برای ارزیابی روی دادگان تست

ارزیابی مدل نهایی با دادگان تست:

در ابتدا داده های val و train را با یکدیگر می چسبانیم:

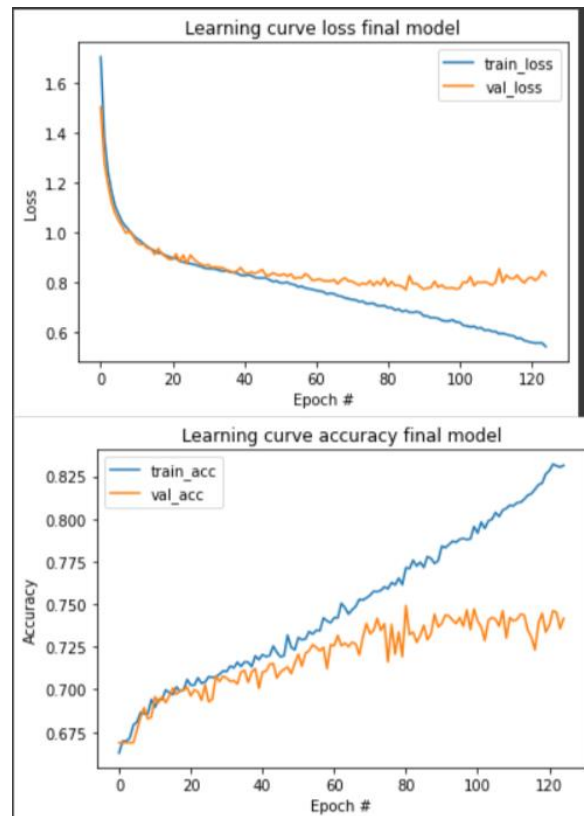
```
Final evaluation on test data:

[ ] all_train_X = np.concatenate( (train_X, val_X) )
    all_train_y = np.concatenate( (train_y, val_y) )

[ ] print('all x shape: ', all_train_X.shape)
    print('\nall y shape: ', all_train_y.shape)

all X shape: (8010, 64, 64, 3)
all y shape: (8010, 7)
```

و در نهایت مدل را با all_train_X و all_train_y فیت میکنیم. در ادامه learning curve را نمایش میدهیم:



دقت های مدل نهایی:

Train_acc: 0.8313Validation_acc: 0.7416

متریک های مدل نهایی به ازای هر کلاس:

```
[ ] cm_final = create_confusionMatrix(y_pred_final, test_y)

print('Confusion matrix for final cnn model:\n', cm_final)

Confusion matrix for final cnn model:
[[2.700e+01 1.500e+01 1.300e+01 1.000e+00 4.000e+00 6.000e+00 0.000e+00]
 [1.000e+01 4.600e+01 2.200e+01 0.000e+00 4.000e+00 2.000e+01 1.000e+00]
 [9.000e+00 7.000e+00 1.280e+02 0.000e+00 2.100e+01 5.500e+01 0.000e+00]
 [3.000e+00 5.000e+00 5.000e+00 3.000e+00 1.000e+00 6.000e+00 0.000e+00]
 [6.000e+00 6.000e+00 2.800e+01 0.000e+00 7.100e+01 1.100e+02 2.000e+00]
 [3.000e+00 2.200e+01 5.400e+01 0.000e+00 5.800e+01 1.201e+03 3.000e+00]
 [0.000e+00 8.000e+00 1.000e+00 0.000e+00 3.000e+00 6.000e+00 1.100e+01]]

[ ] acc, per, recall = calculate_metrics(y_pred_final, test_y, test_x)
for j in range(test_y.shape[1]):
    print('on label {}, acc={:0.2f}, precision={:0.2f}, recall={:0.2f}'.format(j, acc[j], per[j], recall[j]))

on label 0, acc=96.51, precision=46.55, recall=40.91
on label 1, acc=94.01, precision=42.20, recall=44.66
on label 2, acc=89.28, precision=51.00, recall=58.18
on label 3, acc=98.95, precision=75.00, recall=13.04
on label 4, acc=87.88, precision=43.83, recall=31.84
on label 5, acc=82.89, precision=85.54, recall=89.56
on label 6, acc=98.80, precision=64.71, recall=37.93
```

خطا و دقت test مدل نهایی:

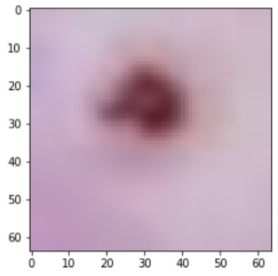
```
[ ] score_finalModel = model_loss_acc(final_model, test_x, test_y)

print("final model (val-loss, val-acc): ", (score_finalModel[0], score_finalModel[1])) # for final model

final model (val-loss, val-acc): (0.8285632729530334, 0.741645872592926)
```

تست تابع predict برای مدل نهایی:

```
[ ] img = 102
plt.imshow(test_X[img]) # testing sample that we want to test trained model with predict function.

<matplotlib.image.AxesImage at 0x7f94601b1150>


[ ] y = test_X[img].reshape(1, 64, 64, 3)
label = test_y[img].argmax(axis=0)
predict = (final_model.predict(y)).argmax(axis=1)
print('predicted: ', predict, ' \nlabel: ', label)

predicted: [5]
label: 5
```

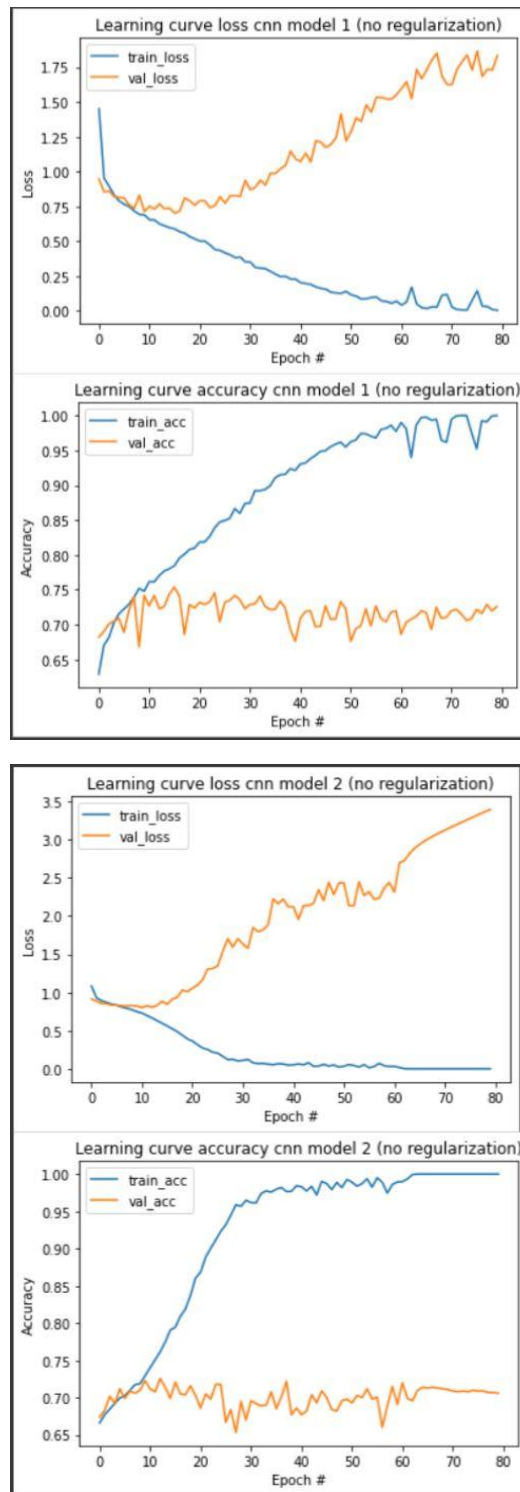
در ادامه نیز مدل نهایی را save می کنیم.

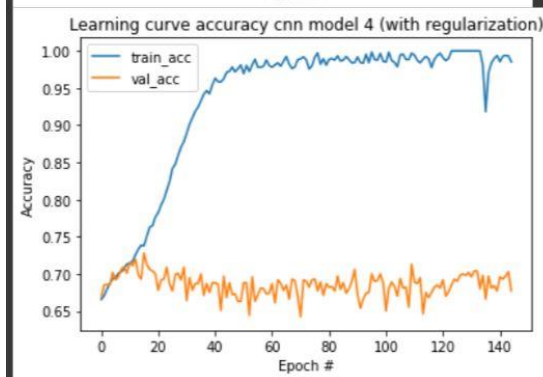
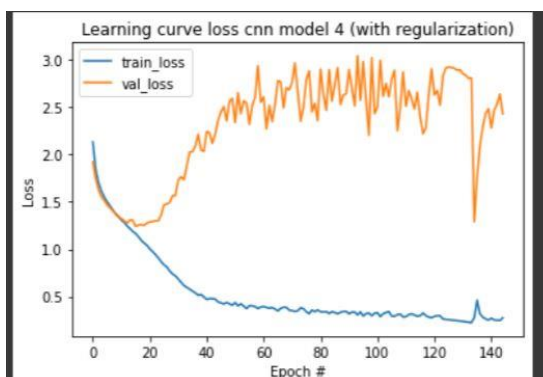
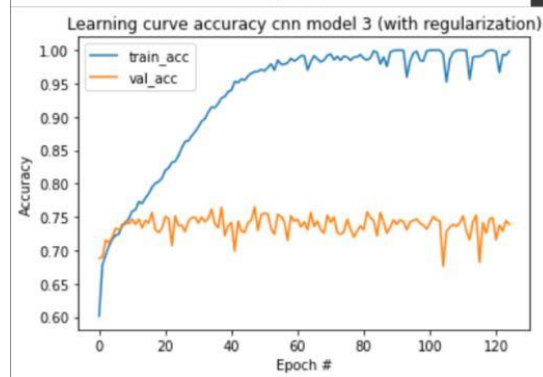
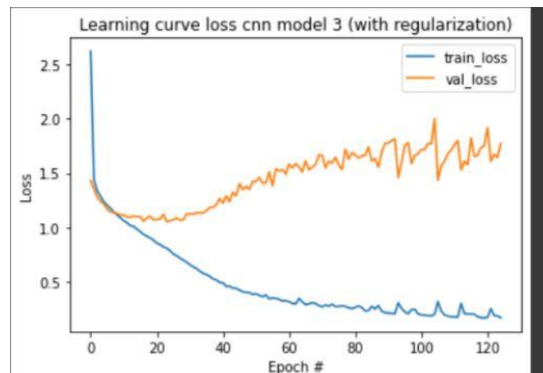
در ادامه نتایج مدل ها برای موقعی که preprocessing نداشته باشیم آورده شده است.

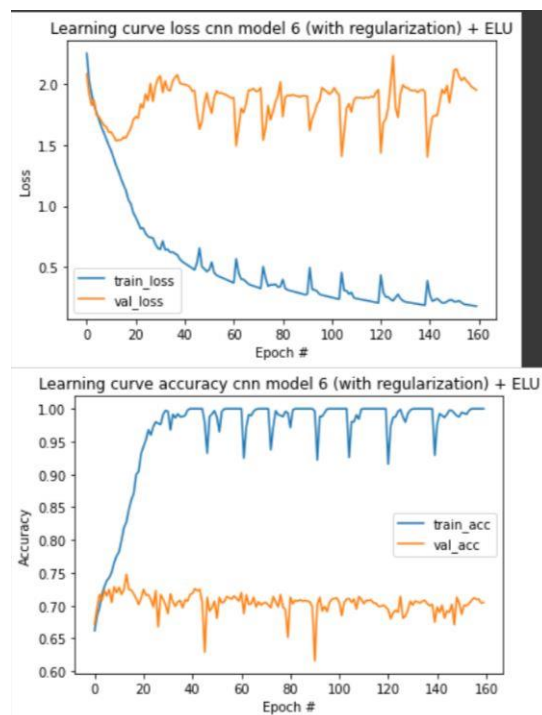
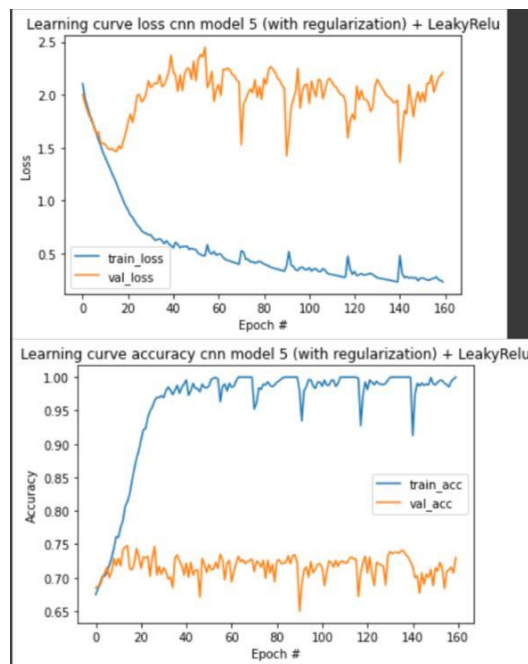
مدل های بدون پیش پردازش:

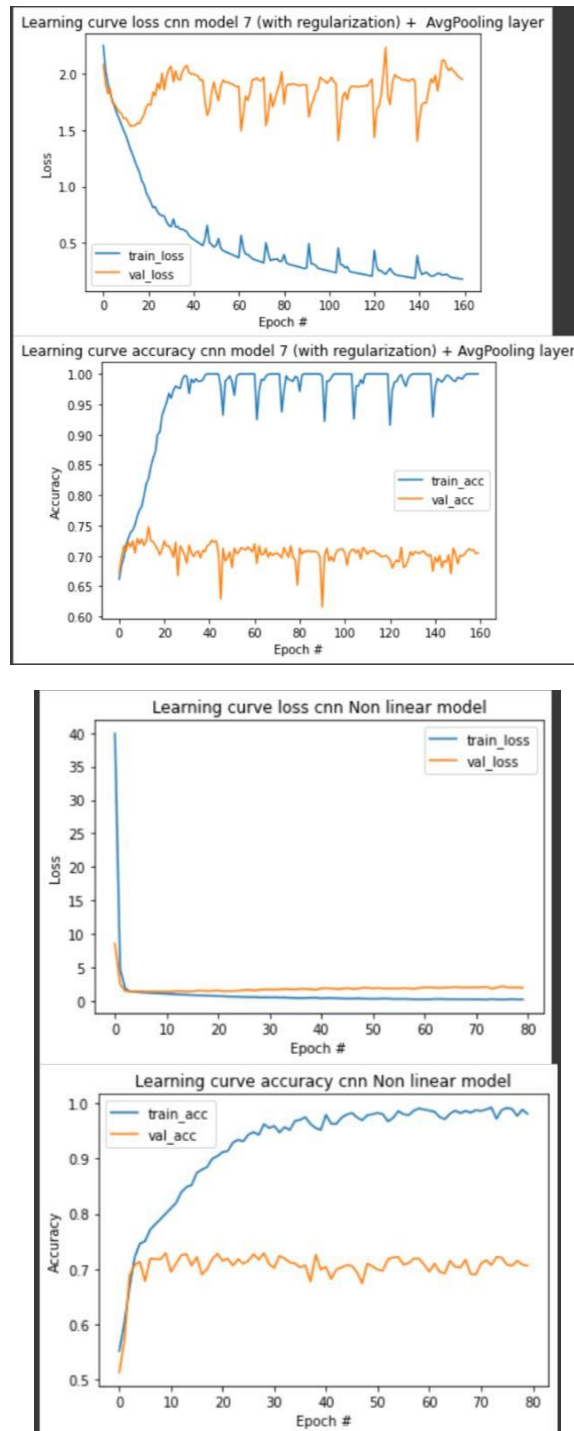
برای اینکه معماری ها بر پایه سایز rescale شده تصاویر به اندازه 64×64 پیاده شده بودند، این پیش پردازش انجام شده است ولی نرمالایز کردن و حذف نویز روی داده های انجام نگرفته است.

در ادامه نمودار های learning curve مدل ها بدون پیش پردازش آورده شده اند:









همانطور که در نمودار learning curve ۸ مدل مشخص است که با همان پارامترها، تمامی مدل ها به شکل قابل توجهی overfit شده اند. پس انجام پیش پردازش داده ها عملی حیاتی در این مسئله می باشد.

در ادامه نیز ماتریس confusion برای مدل ۳ مانند مدل ۲ برتر در بخش قبل آورده شده اند که پیش پردازش روی داده ها انجام نگرفته است:

```
Confusion matrix for cnn model 1:
[[ 10.  4.  6.  0.  7.  6.  0.]
 [ 3. 29.  3.  0.  4. 13.  0.]
 [ 3. 14. 36.  0.  6. 50.  1.]
 [ 1.  4.  0.  3.  0.  4.  0.]
 [ 1.  0.  3.  0. 53. 53.  1.]
 [ 3.  8. 17.  0. 40. 602.  1.]
 [ 1.  1.  0.  0.  1.  3.  8.]]

Confusion matrix for cnn_model_5:
[[ 8.  8.  3.  1.  5.  8.  0.]
 [ 5. 19. 11.  0.  5. 11.  1.]
 [ 6. 11. 44.  1.  8. 40.  0.]
 [ 2.  2.  0.  4.  0.  4.  0.]
 [ 3.  1.  6.  0. 38. 63.  0.]
 [ 3.  8. 18.  1. 27. 611.  3.]
 [ 1.  1.  0.  0.  0.  4.  8.]]

Confusion matrix for cnn model 3:
[[ 8.  5.  9.  1.  6.  4.  0.]
 [ 4. 15. 13.  2.  7. 11.  0.]
 [ 4.  5. 43.  1.  8. 49.  0.]
 [ 1.  4.  0.  4.  0.  3.  0.]
 [ 2.  4. 13.  4. 35. 53.  0.]
 [ 1.  8. 26.  1. 37. 597.  1.]
 [ 0.  1.  1.  0.  1.  5.  6.]]
```

✓ لینک colab نوت بوک پروژه :

<https://colab.research.google.com/drive/1tRYAsVvrUeLOtiLowS8hbD911LeBaa0w?usp=sharing>

✓ فایل ارسالی شامل notebook با اسم Derma_MNIST_CNN.ipynb و فایل pdf گزارش می باشد.