Farshad Borjalizade

HW3 – CNN with keras

Image Processing

University of Tehran

Jun 2021

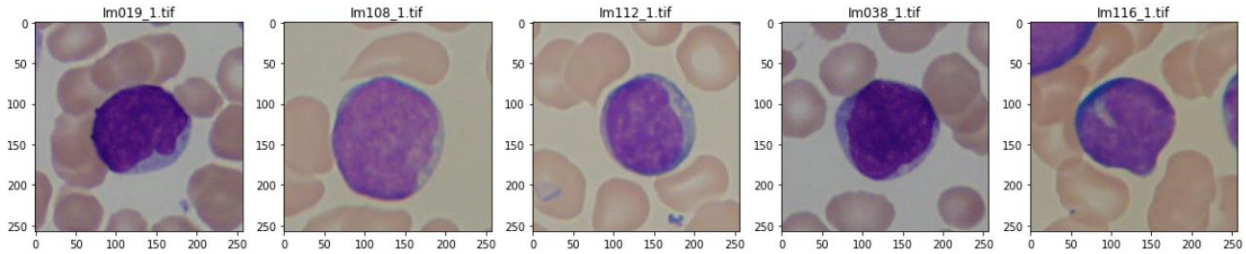
فهرست

چکیده

شبکه های عصبی امروزه نقش های بسیار مهمی را در زمینه یادگیری ماشین و پردازش تصویر و پردازش زبان های طبیعی و ... ایفا می کنند. در همین راستا قصد داریم که در زمینه دسته بندی تصاویر از این موضع بهره ببریم، در این تمرین می خواهیم به وسیله تصاویرسلول های سالم (blast cell) و سلول های غیر سالم (not blast cell) را به کمک شبکه های عصبی دسته بندی کنیم در مورد مجموعه داده به طور مفصل در بخش های بعدی توضیح خواهد داده شد و همین طور برای آموزش داده های موجود از شبکه های CNN (convolution neural network) استفاده می کنیم که آن را به وسیله کتابخانه های Keras, Tensorflow پیاده سازی کرده ایم.

مجموعه داده

مجموعه دادگانی که در این تمرین در اختیار داریم به دو پوشه مجزای سلول های سالم و سلول های ناسالم تقسیم بندی شده اند، در مجموع ۲۶۰ تصویر در ۲ کلاس مختلف قابل شناسایی است در تصویر پایین تعدادی از تصاویر سلول ها که به صورت تصادفی انتخاب شده اند را مشاهده می کنید.



سلول های سالم

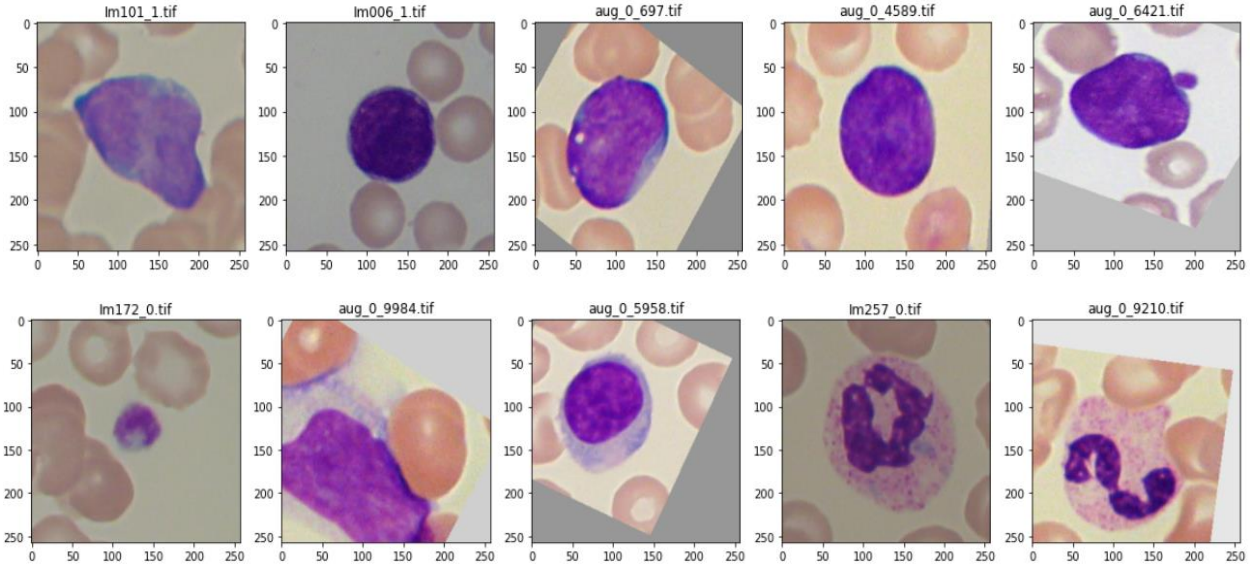
از آنجایی که تعداد تصاویر مربوط به سلول ها به اندازه کافی برای آموزش کامل و صحیح شبکه مناسب نیست به همین خاطر از Image augmentation استفاده میکنیم این تکنیک بدین صورت است که با تغییرات جزئی از جمله rotation, shifting, shear, zoom, flip در تصاویر موجود در مجموعه داده فعلی تصاویر بیشتری را تولید میکند تا شبکه به خوبی با داده های مختلف آموزش ببیند در واقع با این کار چند هدف را دنبال می کنیم یکی اینکه تعمیم پذیری مدل خود برای تصاویر سلول های خارج از مجموعه داده را افزایش می دهیم دوم اینکه تعداد داده های مجموعه آموزش هم به حد قابل قبولی افزایش می یابد که خود به تنهایی می تواند کمک بسیاری به آموزش خوب شبکه می کند در تصویر پایین مشاهده می کنید که چگونه از این تکنیک بهره گرفته شده است.

```
datagen2 = ImageDataGenerator(
              rotation_range=45,
              width_shift_range=0.2,
              height_shift_range=0.2,
              shear_range=0.2,
              zoom_range=0.2,
              horizontal_flip=True,
              fill_mode='constant', cval=125)
```

```
[ ]  for class_folder in folder_list:
         image_list = os.listdir(os.path.join(dataset_dir, class_folder))

         for image_name in image_list:
             current_image_path = os.path.join(dataset_dir, class_folder, image_name)
             image = io.imread(current_image_path)
             image = image.reshape((1, ) + image.shape)
             j = 0
             for batch in datagen2.flow(image, batch_size=16,
                                        save_to_dir=os.path.join(dataset_dir, class_folder),
                                        save_prefix='aug',
                                        save_format='tif'):
                 j += 1
                 if j > 3:
                     break
```

با انجام این کار مجموعه داده های ما ۱۲۶۸ تصویر افزایش پیدا می کند که حد قابل قبولی برای دسته بندی دو کلاسه به حساب می آید. در تصاویر زیر سلول های سالم و ناسالم را به صورت تصادفی و بعد از اعمال تکنیک data augmentation **مشاهده می کنید.**




```
Found 1268 images belonging to 2 classes.
```

از آنجایی که داده ها را ترتیب از روی پوشه های مربوط به کلاس ها خواندیم نظم داده ها را برهم می زنیم تا مدل به صورت تصادفی با داده های روبه رو شود فقط نکته ای که قابل اهمیت است باید توجه داشت که برچسب های تصاویر برهم نخورد و به همراه تصویر مربوط به خود در زمان برهم زدن نظم تصاویر جابه جا شود. این کار را به وسیله قطعه کد زیر انجام داده ایم.

```
prev_state = np.random.get_state()
np.random.shuffle(my_data)
np.random.set_state(prev_state)
np.random.shuffle(my_labels)
```

حال می توان با خیال راحت مجموعه داده را به داده های test, train تقسیم کرد

```
train_x, train_y = my_data[:1020], my_labels[:1020]
```

```
val_x, val_y = my_data[1020:], my_labels[1020:]
```

بدین ترتیب ۱۰۲۰ داده برای آموزش و ۲۴۸ داده برای ارزیابی مدل در اختیار داریم.

برای اینکه برچسب های داده های را به شکل منظمی تبدیل کنیم که بتوان آن را برای شبکه قابل فهم کرد آن ها را به برچسب های ۰ و ۱ تبدیل و داده ها را نرمال می کنیم.

```
train_y = to_categorical(train_y, num_classes=2)
train_y.shape, train_y.dtype
```

```
((1020, 2), dtype('float32'))
```

```
train_x = train_x.astype('float32') / 255.0
train_x.shape, train_x.dtype
```

```
((1020, 257, 257, 3), dtype('float32'))
```

پیاده سازی شبکه

همانظور که در مقدمه گفته شده برای آموزش و دسته بندی تصاویر قصد داریم از شبکه های CNN به وسیله کتاب خانه های Keras, Tensorflow استفاده کنیم پس ابتدا کتابخانه های لازم را وارد می کنیم.

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPool2D, Flatten, GlobalAveragePooling2D, Dropout, ReLU, ZeroPadding2D
```

برای شروع، ابتدا ورودی شبکه که به اندازه تصاویر هستند ۳*۲۵۷*۲۵۷ به ورودی می دهیم و این ورودی را به اولین لایه Conv به اندازه کرنل ۳*۳ به همراه ۲۵۶ فیلتر و strides=2 و padding=same می دهیم و از تابع فعال ساز ReLU برای داده ها و در نهایت با یک MaxPool با اندازه ۲*۲ و strides=2 به همراه padding=(0*0) را اعمال می کنیم بدین صورت بلوک اول ما از لایه ها به اتمام می رسد و همین طور با استفاده از تکنیک هایی شبکه را درست می کنیم. پیاده سازی و خلاصه ای از مدل در تصاویر زیر آورده شده است.

```
input_layer = Input(shape=(257, 257, 3))

#Block 1
x = Conv2D(filters=256, kernel_size=(3, 3), strides=2, padding='same')(input_layer)
x = ReLU()(x)
x = ZeroPadding2D(padding=(0, 0))(x)
x = MaxPool2D(pool_size=(2, 2), strides=2 , padding='valid')(x)

#Block 2
x = ZeroPadding2D(padding=(2, 2))(x)
x = Conv2D(filters=128, kernel_size=(3, 3), strides=1, padding='valid')(x)
x = ReLU()(x)
x = MaxPool2D(pool_size=(2, 2), strides=2 , padding='valid')(x)

#Block 3
x = ZeroPadding2D(padding=(3, 3))(x)
x = Conv2D(filters=96, kernel_size=(5, 5), strides=1, padding='valid')(x)
x = ReLU()(x)

#Block 4
x = ZeroPadding2D(padding=(2, 2))(x)
x = Conv2D(filters=96, kernel_size=(5, 5), strides=1, padding='valid')(x)
x = ReLU()(x)

#Block 5
x = ZeroPadding2D(padding=(2, 2))(x)
x = Conv2D(filters=32, kernel_size=(3, 3), strides=1, padding='valid')(x)
x = ReLU()(x)
x = MaxPool2D(pool_size=(3, 3), strides=2 , padding='valid')(x)

#Block 6
x = GlobalAveragePooling2D()(x)

x = Dense(units=1000, activation='relu')(x)
x = ReLU()(x)
x = Dropout(0.5)(x)

#Output layer
predictions_layer = Dense(units=2, activation='softmax')(x)

my_cell_classifier_model = Model(inputs=input_layer, outputs=predictions_layer)
```

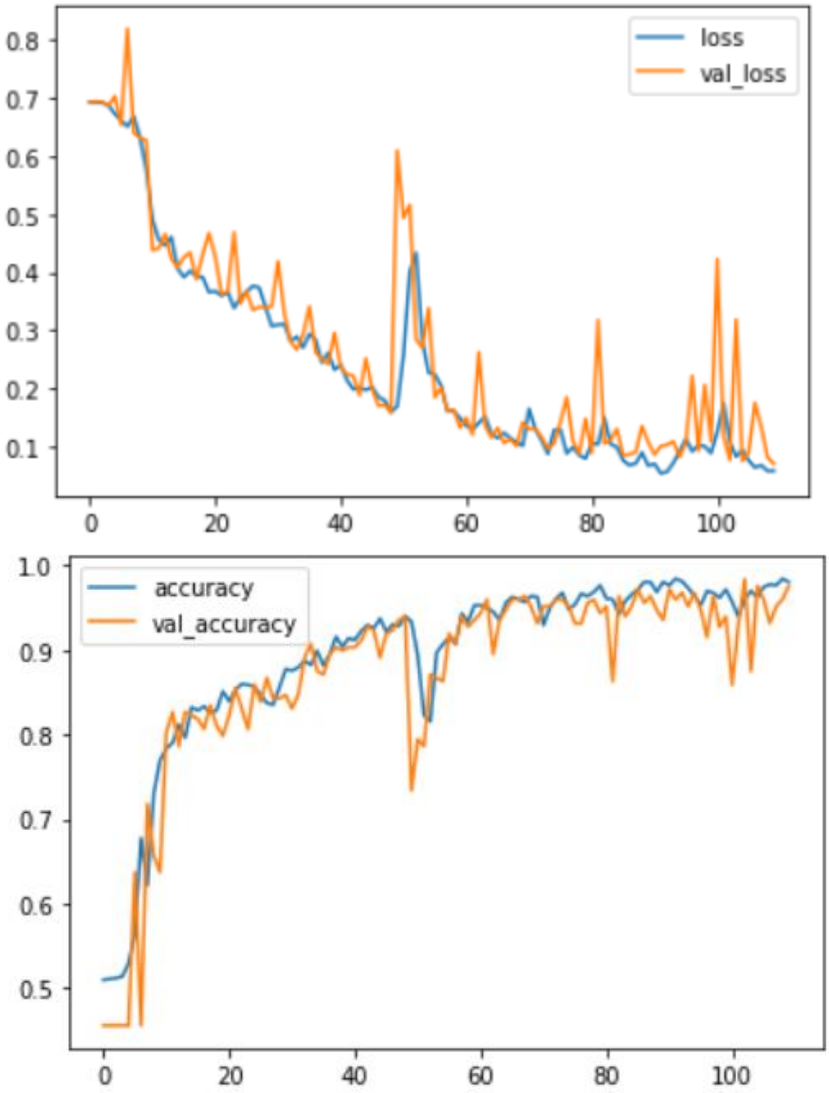
```
_________________________________________________________________
input_2 (InputLayer)         [(None, 257, 257, 3)]     0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 129, 129, 256)     7168
_________________________________________________________________
re_lu_6 (ReLU)               (None, 129, 129, 256)     0
_________________________________________________________________
zero_padding2d_5 (ZeroPaddin (None, 129, 129, 256)     0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 64, 64, 256)       0
_________________________________________________________________
zero_padding2d_6 (ZeroPaddin (None, 68, 68, 256)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 66, 66, 128)       295040
_________________________________________________________________
re_lu_7 (ReLU)               (None, 66, 66, 128)       0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 33, 33, 128)       0
_________________________________________________________________
zero_padding2d_7 (ZeroPaddin (None, 39, 39, 128)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 35, 35, 96)        307296
_________________________________________________________________
re_lu_8 (ReLU)               (None, 35, 35, 96)        0
_________________________________________________________________
zero_padding2d_8 (ZeroPaddin (None, 39, 39, 96)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 35, 35, 96)        230496
_________________________________________________________________
re_lu_9 (ReLU)               (None, 35, 35, 96)        0
_________________________________________________________________
zero_padding2d_9 (ZeroPaddin (None, 39, 39, 96)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 37, 37, 32)        27680
_________________________________________________________________
re_lu_10 (ReLU)              (None, 37, 37, 32)        0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 18, 18, 32)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              33000
_________________________________________________________________
re_lu_11 (ReLU)              (None, 1000)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1000)              0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 2002
=================================================================
Total params: 902,682
Trainable params: 902,682
Non-trainable params: 0
```

همان طور که مشخص است تعداد پارامترهایی که در این مدل باید یاد گرفته شود تقریبا ۹۰۰ هزار عدد است که به نسبت شبکه های امروزی عدد کمی محسوب می شود برای آموزش مدل از بهینه ساز Adam به همراه learning_rate=0.0001 و تابع خطای binary_crossentropy استفاده می کنیم. مدل را با batch_size=100 به همراه ۱۰۰ epoch برای آموزش آماده می کنیم، نتیجه چند اجرا ی آخر را مشاهده می کنید.

```
my_cell_classifier_model.fit(x=train_x, y=train_y,
                             epochs=110,
                             batch_size=100,
                             shuffle=True,
                             validation_data=(val_x, val_y))
```

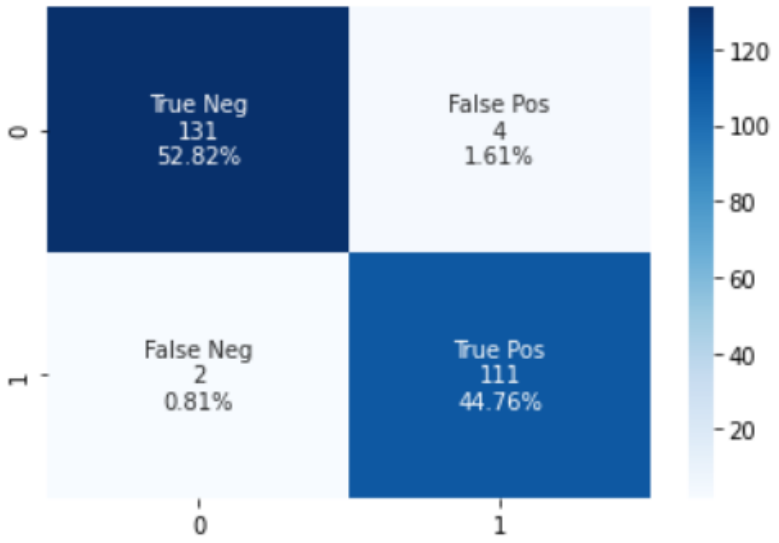
```
Epoch 100/110
11/11 [==============================] - 3s 310ms/step - loss: 0.0886 - accuracy: 0.9716 - val_loss: 0.1091 - val_accuracy: 0.9395
Epoch 101/110
11/11 [==============================] - 3s 302ms/step - loss: 0.1254 - accuracy: 0.9569 - val_loss: 0.4229 - val_accuracy: 0.8589
Epoch 102/110
11/11 [==============================] - 3s 306ms/step - loss: 0.1745 - accuracy: 0.9392 - val_loss: 0.1182 - val_accuracy: 0.9315
Epoch 103/110
11/11 [==============================] - 3s 308ms/step - loss: 0.1077 - accuracy: 0.9608 - val_loss: 0.0766 - val_accuracy: 0.9839
Epoch 104/110
11/11 [==============================] - 3s 311ms/step - loss: 0.0826 - accuracy: 0.9696 - val_loss: 0.3187 - val_accuracy: 0.8750
Epoch 105/110
11/11 [==============================] - 3s 312ms/step - loss: 0.0943 - accuracy: 0.9627 - val_loss: 0.0752 - val_accuracy: 0.9758
Epoch 106/110
11/11 [==============================] - 3s 310ms/step - loss: 0.0760 - accuracy: 0.9745 - val_loss: 0.0895 - val_accuracy: 0.9597
Epoch 107/110
11/11 [==============================] - 3s 308ms/step - loss: 0.0639 - accuracy: 0.9775 - val_loss: 0.1752 - val_accuracy: 0.9315
Epoch 108/110
11/11 [==============================] - 3s 310ms/step - loss: 0.0678 - accuracy: 0.9765 - val_loss: 0.1341 - val_accuracy: 0.9516
Epoch 109/110
11/11 [==============================] - 3s 303ms/step - loss: 0.0581 - accuracy: 0.9843 - val_loss: 0.0816 - val_accuracy: 0.9597
Epoch 110/110
11/11 [==============================] - 3s 310ms/step - loss: 0.0583 - accuracy: 0.9804 - val_loss: 0.0703 - val_accuracy: 0.9758
<tensorflow.python.keras.callbacks.History at 0x7efe9a4fc890>
```

آخرین دقتی که از مدل بر روی داده های آموزش گرفته ایم به ٪۹۸ و یر روی داده تست ٪۹۷ است که نشان دهنده این است که مدل به خوبی آموزش دیده است و تعمیم پذیری مناسبی هم با توجه به اختلاف ناچیز داده آموزش و تست دارد. در نمودارهای زیر می توان نحوه آموزش مدل را به صورت کامل مشاهده کرد.



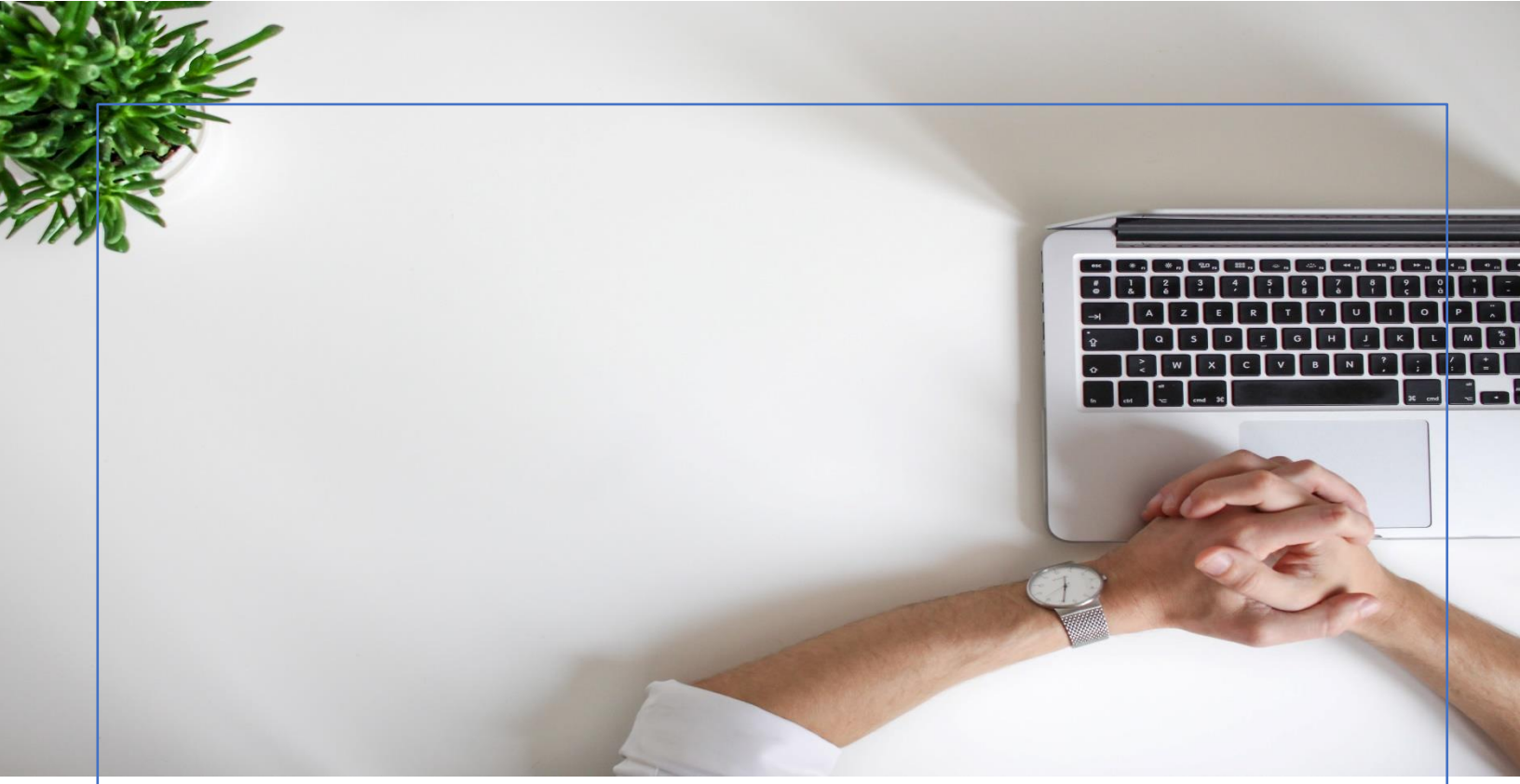
حال برای ارزیابی بیشتر و دقیق تر confusion matrix **را به صورت گرافیکی به نمایش می گذاریم.**

	precision	recall	f1-score	support
0	0.98	0.97	0.98	135
1	0.97	0.98	0.97	113
accuracy			0.98	248
macro avg	0.98	0.98	0.98	248
weighted avg	0.98	0.98	0.98	248



همان طور که از آمار مشخص است مدل به طور کامل و مناسب آموزش دیده است و به خوبی برای داده های دیده نشده عمل می کند.

در واقع ۱۱۱ مورد را به درستی سالم تشخیص داده است، ۱۳۱ مورد را به درستی ناسالم، ۲ مورد را به غلط ناسالم و ۴ مورد را به غلط سالم تشخیص داده است.



نتیجه گیری

برداشـتی که از این تمرین می توان داشـت این اسـت که با افزایش تعداد پارامترها و لایه های شـبکه نمی توان به طور کلی انتظار داشـت که عملکرد شـبکه بهتر شـود چرا که همین تمرین را بر رو شـــبکه معروف Alexnet که تقریبا ۲۴ میلیون پارامتر دارد پیاده ســـازی کردیم و نتیجه گرفته شده به مراتب کمتر از مدل فعلی بود.

پس می بایســـتی در اســـتفاده از لایه ها و پارامترها دقت کافی را به خرج داد و با آزمون و خطا و کاهش و افزایش تعداد لایه ها و تغییر اندازه فیلترها و امتحان کردن بهینه سـازهای مختلف شبکه را آموزش داد تا به نتیجه دلخواه رسید.