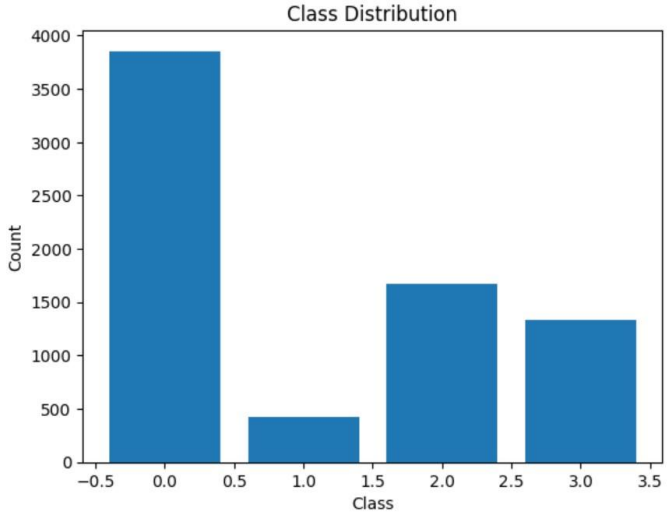
من مسئله ۴ کلاسه انجیر را با استفاده از مدل پیش اموزش دیده Densenet121 و هم چنین با ابزارهای hyperparametersearch و center loss اموزش دادم که هر کدام را شرح خواهم داد.

در ابتدا distribution چهار کلاس را در دیتاست train رسم کردم که اگر نیاز بود اقداماتی برای imbalance بودن دیتاست انجام بدهم:



همانطور که معلوم است کمی دیتاست imbalance هست که برای ان از دو روش resampling و وزن دهی استفاده کردم که در قسمت hyperparametersearch توضیح میدم.

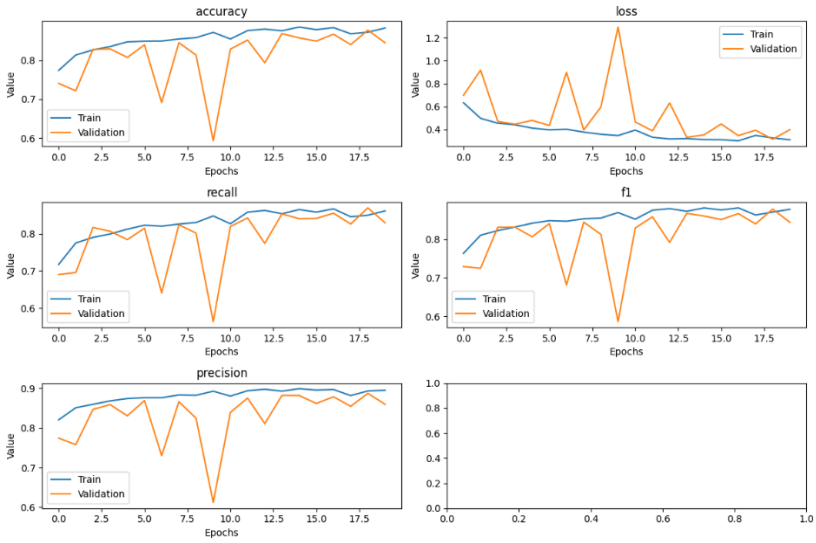
مرحله اول:

در ابتدا تصمیم رفتم بدون استفاده از بهینه کردن هایپرپارامترها با استفاده از augmentation و مدل پیش اموزش دیده Densenet121 دیتاست را train کنم که بتوانم در ادامه مقایسه انجام بدهم.

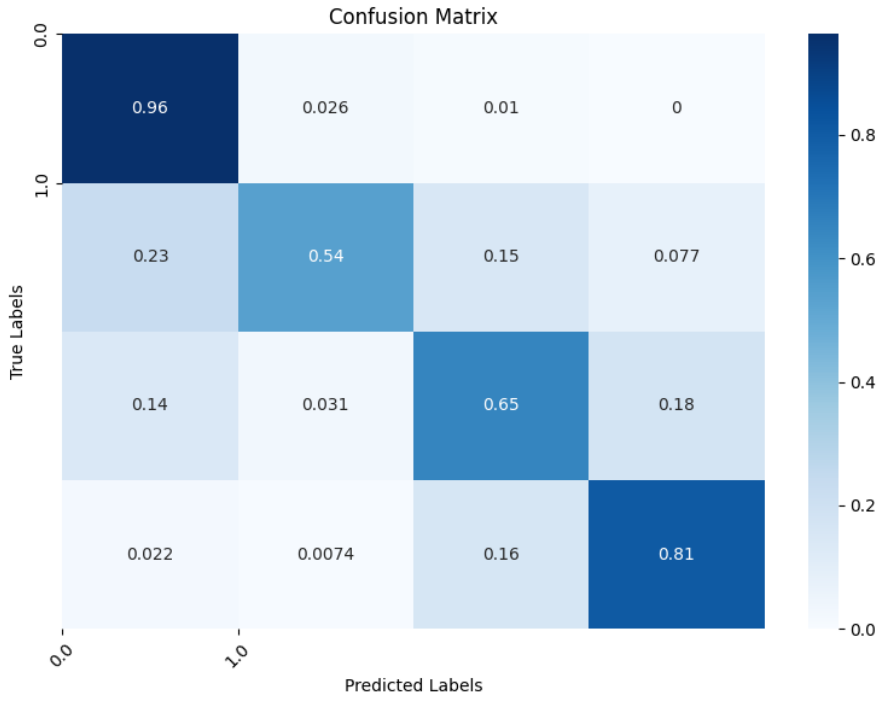
```
shapeimg=100
OldModel = DenseNet121(include_top=False,weights='imagenet',input_shape=(100, 100, 3))
OldModel.trainable=False

# Add Global Average Pooling layer
x = OldModel.output
x = GlobalAveragePooling2D()(x)
# Add a fully-connected layer
x = Dense(256, activation='relu')(x)
```

مطابق بالا اول پارامتر trainable را false گذاشتم وبعد یک لایه GlobalAveragepooling اضافه کردم و یک لایه Dense با تعداد نورون های ۲۵۶ تا و در نهایت لایه Dense نهایی که تعداد نورون های ان برابر با ۴

بود را قرار دادم در ابتدا به یزان ۱۰ epoch اموزش دادم و بعد با نرخ یادگیری بسیار کم مدل را finetune کردم نمودار های validation,train به صورت زیر شد:



که در نهایت confusion matrix برای دیتاست test به صورت زیر شد:



از نظر من تفاوت زیادی که در پیش بینی درصد لیبل های انجیر دارد یکی بحث imbalance بودن دیتاست هست و دیگری لیبل های دو و سه انجیر به هم شباهت زیادی داشتند.

مرحله دوم:

روی پارامترهای augmentation با استفاده از ابزار GPyOpt سرچ زدم تا پارامترهای بهینه را پیدا کنم.

ابزار GPyOpt:

GPyOpt یک کتابخانه محاسباتی برای بهینه‌سازی بر پایه گاوسی (Gaussian) است که در زمینه مسائل بهینه‌سازی تابعی استفاده می‌شود. این کتابخانه به صورت یک پکیج در زبان برنامه‌نویسی پایتون (Python) قابل استفاده است.

GPyOpt بر اساس روش‌های بهینه‌سازی بر پایه گاوسی مانند Bayesian Optimization عمل می‌کند. این روش‌ها بر اساس استدلال برای تخمین تابع هدف و استفاده از اطلاعات جمع‌آوری شده در هر مرحله از بهینه‌سازی عمل می‌کنند. با استفاده از این روش‌ها، می‌توان به صورت موثری تابع هدف را بهینه کرد، حتی در صورتی که تابع هدف پیچیده و ناهمگن باشد یا در دسترس بودن اطلاعات مربوط به آن محدود باشد.

GPyOpt امکانات متنوعی برای تنظیم و کنترل بهینه‌سازی را فراهم می‌کند. این شامل مواردی مانند تعیین محدوده‌های جستجو، تنظیمات الگوریتم‌های بهینه‌سازی، تعیین تعداد نقاط نمونه‌برداری و معیارهای ارزیابی است. همچنین، GPyOpt قابلیت توسعه و سفارشی‌سازی با استفاده از توابع پایتون را نیز داراست.با استفاده از GPyOpt، می‌توانید به طور موثر بهینه‌سازی تابع هدف خود را انجام دهید و نقاط بهینه را در فضای جستجو پیدا کنید. این کتابخانه معمولا در مسائلی که تابع هدف بازگشتی و پیچیده استفاده می‌شود.

تابع GPyOpt.methods.BayesianOptimizationدرماژول GPyOpt.methodsکتابخانه GPyOpt استفاده می‌شود و یک نمونه از بهینه‌سازی بیزی با استفاده از روش‌های بیزی GPy را ایجاد می‌کند. این تابع پارامترهای زیر را دریافت می‌کند:

f:تابع هدف بهینه‌سازی که باید توسط کاربر تعریف شود. این تابع باید ویژگی‌هایی که قرار است بهینه شوند را دریافت کرده و یک معیار عملکرد (مثلاً تابع هزینه) را برگرداند.

domain:لیستی از دیکشنری‌ها که هر دیکشنری شامل اطلاعات پارامترهای بهینه‌سازی است. هر دیکشنری باید شامل سه کلید باشد name :که نام پارامتر را مشخص می‌کند، typeکه نوع پارامتر را مشخص می‌کند (می‌تواند پیوسته یا گسسته باشد) و domainکه محدوده‌ی مجاز برای مقادیر پارامتر را مشخص می‌کند.

initial_design_numdata:تعداد نقاط اولیه برای طراحی اولیه. این نقاط به صورت تصادفی در دامنه پارامترها انتخاب می‌شوند و برای شروع فرایند بهینه‌سازی استفاده می‌شوند.

acquisition_type:نوع تابع اکتساب (acquisition) که برای انتخاب نقطه بعدی برای ارزیابی استفاده می‌شود. مقدار پیش‌فرض آن 'LCB' است که بهینه‌سازی بیشینه حاشیه‌ای (Lower Confidence Bound) را انجام می‌دهد. دیگر مقادیر ممکن شامل 'EI' (Entropy Search) و 'MPI' (Maximum Probability of Improvement)هستند.

```
# set parameters
bounds = [{'name': 'rr',  'type': 'continuous', 'domain': (10, 50)},
          {'name': 'wsr', 'type': 'continuous', 'domain': (0.1, 0.4)},
          {'name': 'hsr', 'type': 'continuous', 'domain': (0.1, 0.4)},
          {'name': 'sr',  'type': 'continuous', 'domain': (0.1, 0.4)},
          {'name': 'zr',  'type': 'continuous', 'domain': (0.1, 0.4)}]

# initialization
myBopt = GPyOpt.methods.BayesianOptimization(f=image_data_generator_opt_f,
                                             domain=bounds,
                                             initial_design_numdata=7,
                                             acquisition_type='LCB')
```

من بر روی هایپر پارامترهای rotation_range, width_shift_range, hight_shift_range, shear_range, zoom_range سرچ زدم و نتایج به صورت زیر شد:

```
rotation_range       ; 29.779146887598596
width_shift_range    ; 0.1
hight_shift_range    ; 0.4
shear_range          ; 0.4
zoom_range           ; 0.1
val_loss             ; 0.4809447228908539
Optimum parameters are saved!
```

مرحله سوم:

در این مرحله تصمیم گرفتم از loss به نام center loss که البته باید به صورت custom زده بشه استفاده کنم در ابتدا توضیحی درباره center loss میدهم.

Center loss، یک روش برای این هست که ویژگی های هر کلاس به هم نزدیک بشوند. این روش در حوزه تشخیص چهره و دسته‌بندی تصاویر استفاده می‌شود و هدف آن ایجاد مراکزی است که نماینده‌ی هر کلاس باشند و فاصله‌ی این مراکز با نمونه‌های هر کلاس را کاهش دهند.

فرآیند آموزش با استفاده از Center loss به این صورت است که در هر مرحله از آموزش، مراکز کلاس‌ها بروزرسانی می‌شوند تا نمونه‌های هر کلاس به آن مرکز نزدیک‌تر شوند. برای محاسبه مرکز هر کلاس، میانگین ویژگی‌های نمونه‌های همان کلاس را محاسبه می‌کنیم.در این روش، علاوه بر تابع هدف دسته‌بندی سنتی (مانند تابع هزینه‌ی cross-entropy)، تابع هدف دیگری به نام Center loss نیز استفاده می‌شود. تابع هدف Center loss در واقع فاصله‌ی میانگین ویژگی‌های یک نمونه از مرکز کلاس مربوطه را کمینه می‌کند. این تابع هدف معمولاً به صورت یک تابع مربع فاصله (squared Euclidean distance) تعریف می‌شود.در واقع تابع ضرری که تعریف می کنیم به این صورت هست جمع cross-entropy در یک ضریبی به نام لاندا در center loss.این ضریب لاندا اهمیت زیادی داره اگر لاندا ۰ باشد که تابع ضرر ما دقیقا همان cross entropy میشود و اگر برابربا ۱ باشد ممکن هست همه مرکزها رو هم بیفتند پس میتوان این هایپرپارامتر لاندا را به عنوان یکی از پارامترهای سرچ در نظر گرفت.

کد center loss به صورت زیر است:

```
def call(self, y_true, y_pred):
    """
        y_ture : same teacher signal as for classification (1-hot vector)
                 shape = (batch_size, num_classes)
        y_pred : output of features in the middle layer of the model
                 shape = (batch_size, feature_dims)
    """
    labels = tf.argmax(y_true, axis=-1)
    centers_batch = tf.gather(self.centers, labels)
    diff = centers_batch - y_pred
    loss = tf.reduce_mean(tf.square(diff))
    unique_label, unique_idx, unique_count = tf.unique_with_counts(labels)
    appear_times = tf.gather(unique_count, unique_idx)
    appear_times = tf.reshape(appear_times, [-1, 1])
    diff = diff / tf.cast((1 + appear_times), tf.float32)
    diff = self.alpha * diff
    self.centers = tf.compat.v1.scatter_sub(self.centers, labels, diff)
    return loss
```

مرحله چهارم:

در این مرحله تصمیم گرفتم سه هایپرپارامتر را با استفاده از kerastunner بهینه کنم که ان سه پارامتر به صورت زیر است:

۱)پارامتر تعداد نورون ها در لایه ما قبل Dense نهایی که به عنوان بردار ویژگی در center loss استفاده میشه.

۲)پارامتر لاندا که در مرحله سوم ذکر شد.

۳) learning rate

در ابتدا توضیحی درباره kerastunner میدهم.

Keras Tuner از چندین روش برای جستجو در فضای هایپرپارامترها استفاده می‌کند. برخی از این روش‌ها عبارتند از:

۱. Random Search: در این روش، هایپرپارامترها به صورت تصادفی انتخاب می‌شوند. مزیت این روش این است که به طور موثر فضای هایپرپارامترها را بررسی می‌کند و می‌تواند به نتایج خوبی برسد. این روش به خصوص برای مسائلی که فضای هایپرپارامترها بزرگ است مناسب است.

۲. Grid Search: در این روش، فضای هایپرپارامترها به صورت گرید تعریف می‌شود و همه‌ی ترکیب‌های ممکن از هایپرپارامترها بررسی می‌شوند. این روش مناسب برای فضای هایپرپارامترهای کوچک‌تر است که بتوان همه‌ی ترکیب‌های ممکن را بررسی کرد.

۳. Hyperband: این روش به صورت مرحله‌ای هایپرپارامترها را بررسی می‌کند. در هر مرحله، مدل‌هایی با هایپرپارامترهای مختلف آموزش داده می‌شوند و عملکرد آنها ارزیابی می‌شود. هایپرپارامترهایی که عملکرد بهتری دارند، در مراحل بعدی بیشتر در نظر گرفته می‌شوند و فرآیند بهینه‌سازی ادامه می‌یابد. این روش به طور موثر با هزینه‌ی محدودی از نظر منابع محاسباتی به بهینه‌سازی هایپرپارامترها می‌پردازد.

۴.Bayesian: در این روش، از مدل‌های محتملیت‌گرا (probabilistic models) برای مدل‌سازی تابع هدف استفاده می‌شود. این مدل‌ها تلاش می‌کنند تا توزیع برازش بهتری از تابع هدف را در فضای هایپرپارامترها مدل کنند. با استفاده از این توزیع برازش، می‌توان به طور هوشمندانه‌تری هایپرپارامترها را جستجو کرد و بهترین مجموعه‌ی هایپرپارامترها را بر اساس آن انتخاب کرد.

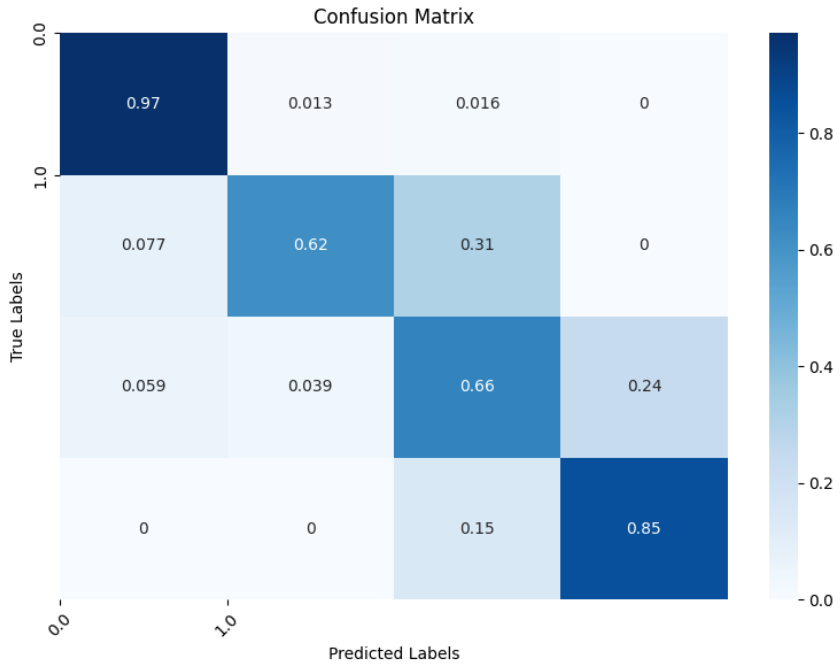
من با استفاده از BayesianOptimization سرچ کردم.

و نتایج بهینه پس از سرچ به صورت زیر شد:

```
[ ]  best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]
     print(best_hps.get('units'))

     print(best_hps.get('lr'))

     print(best_hps.get('landa'))
```

```
11
0.002032498205514228
0.1
```

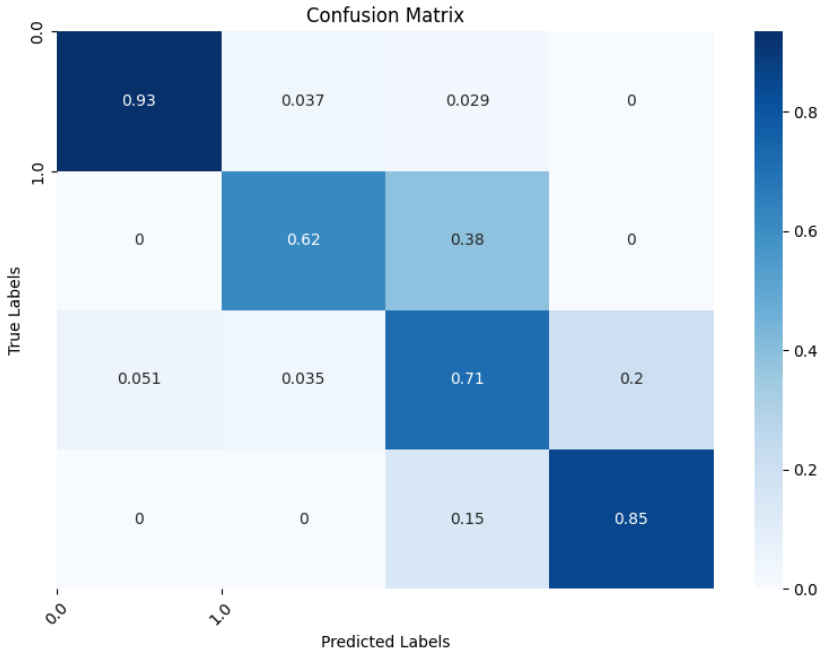
و سپس مدل را finetune کردم و confusion matrix ان به صورت زیر شد:



مشاهده میکنید نتایج نسبت به حالت قبل از سرچ بهتر شده است ولی هنوز راضی کننده نیست.

مرحله پنجم:

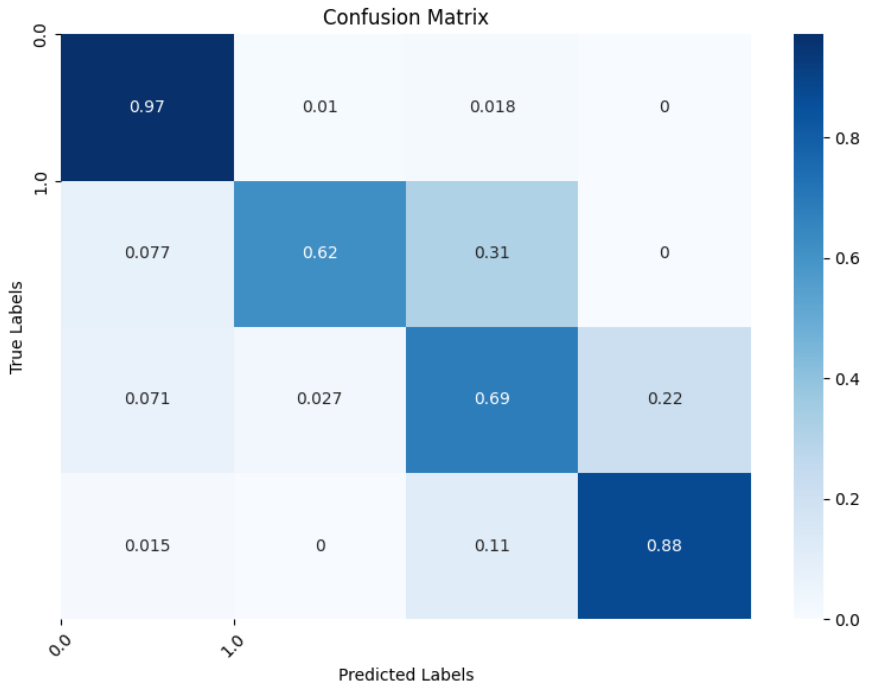
تصمیم گرفتم برای مقابله imbalance بودن دیتاست وزن اضافه کنم و یکبار دیگه با وزن های مرحله چهارم مدل را train کنم. واین وزن دهی بر روی loss function تاثیر میذاره.و وزن ها را با توجه به نسبت تعداد هر لیبل گذاشتم. البته این هایپرپارامترهای وزن دهی قابل سرچ کردن بود ولی انجام ندادم. پس از وزن دهی نتایج به صورت زیر شد:



فقط نتایج در لیبل ۳ بهتر شده است به همین علت این روش را کنار گذاشتم.

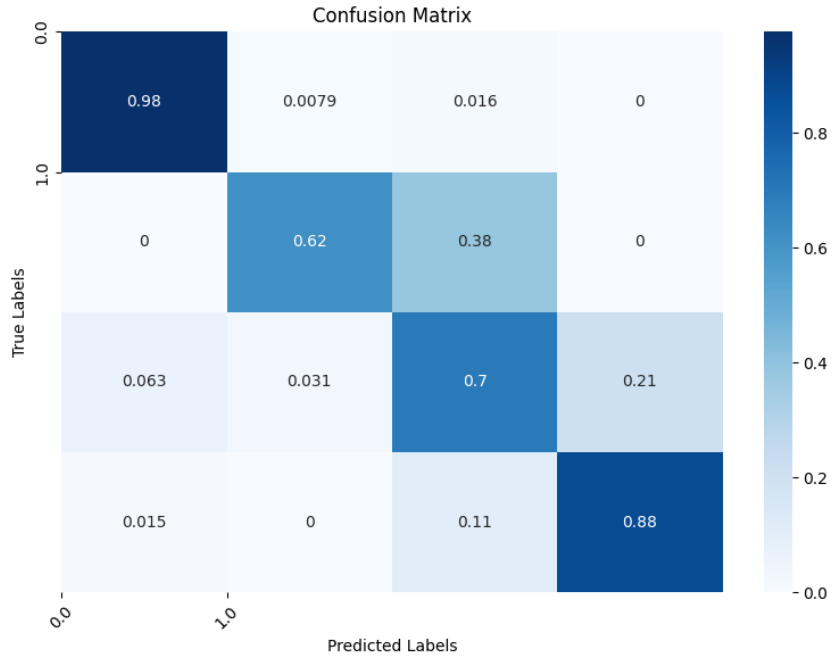
مرحله ششم:

تصمیم گرفتم این بار برای مقابله با نامتعادل بودن دیتاست از روش resample استفاده کنم.و پس از ان نتایج به صورت زیر شد:

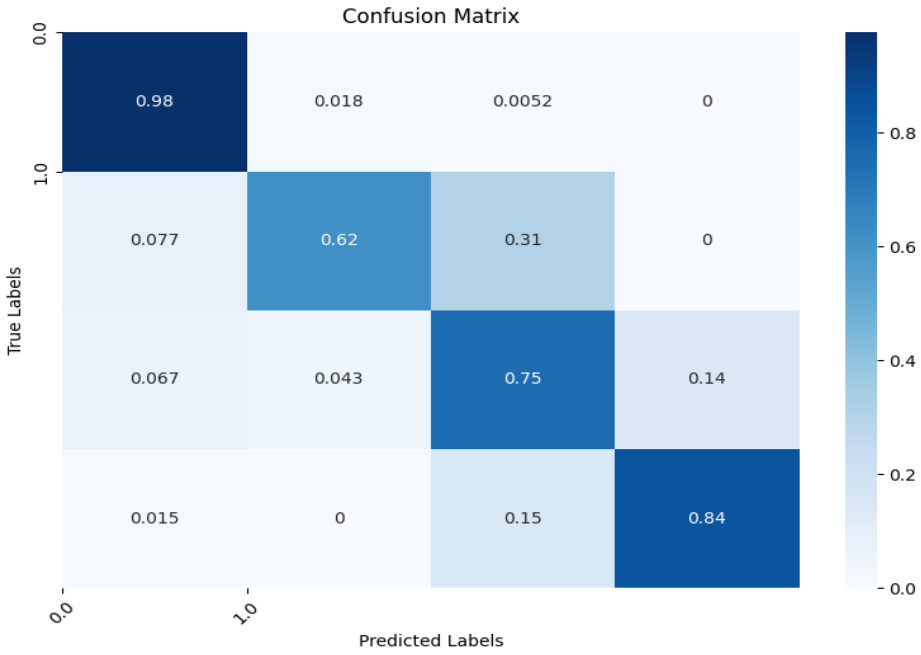


مشاهده میکنید نسبت به روش وزن دهی،resampling روش بهتری هست و نتایج بهتر شده است.

و پس از این مراحل تصمیم گرفتم برای جلوگیری از overfit شدن یک لایه dropout اضافه کنم یکبار با نرخ ۰.۵ و بار دیگر با نرخ ۰.۸ نتایج برای نرخ ۰.۵ به صورت زیر شد:



و با نرخ ۰.۸ نتایج به صورت زیر شد:



مرحله هفتم:

این بار تصمیم گرفتم با استفاده از optuna هایپرپارامترها را بهینه کنم.

Optuna یک کتابخانه برای بهینه‌سازی هایپرپارامتر است که بر روی مسائل یادگیری ماشین و بهبود عملکرد الگوریتم‌ها تمرکز دارد. این کتابخانه از الگوریتم‌های بهینه‌سازی ترکیبی (مانند تکنیک‌های نمونه‌برداری تصادفی و الگوریتم‌های جستجوی تکاملی) برای جستجوی فضای هیپرپارامتر استفاده می‌کند.

روش اصلی بهینه‌سازی در Optuna الگوریتم تکاملی TPE (Tree-structured Parzen Estimator) است. این الگوریتم از روش‌های Bayesian Optimization استفاده می‌کند و با استفاده از تخمین‌گر Parzen، مدلی احتمالی برای تابع هدف (مثلاً دقت یک مدل یادگیری ماشین) را تخمین می‌زند. سپس با استفاده از این مدل احتمالی، نقاطی در فضای هیپرپارامتر را بررسی می‌کند تا برترین نقطه را پیدا کند.

استفاده از Optuna شامل مراحل زیر است:

۱. تعریف فضای هایپرپارمتر: شما باید فضای هایپرپارمتر را تعریف کنید. برای هر هایپرپارمتر ، محدوده مقادیر ممکن را تعیین کنید. ممکن است یک هیپرپارامتر عدد صحیح یا حقیقی باشد.

۲. تعریف تابع هدف: شما باید تابع هدف را تعریف کنید که بر اساس هایپرپارمتر ، عملکرد مدل را ارزیابی می‌کند. معمولاً در مسائل یادگیری ماشین، تابع هدف مرتبط با دقت یا خطا است.

۳. اجرای بهینه‌سازی: با استفاده از تابع `study.optimize`، بهینه‌سازی را شروع کنید. Optuna الگوریتم TPE را بر روی فضای هیپرپارامترها اجرا می‌کند و تلاش می‌کند برترین مقادیر هایپرپارمتر را پیدا کند.

۴. دریافت بهترین هایپرپارمتر: پس از اتمام بهینه‌سازی، شما می‌توانید بهترین مقادیر هایپرپارمتر را با استفاده از `study.best_params` دریافت کنید.

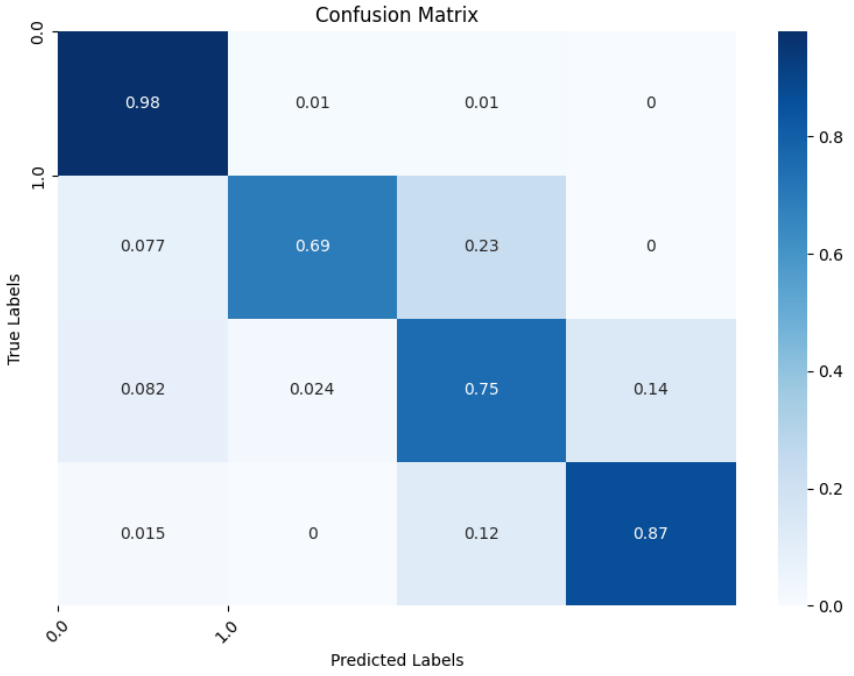
۵. استفاده از بهترین هایپرپارمتر: با استفاده از بهترین هیپرپارامترها، می‌توانید یک مدل یادگیری ماشین با عملکرد بهتر ایجاد کنید.

به طور کلی، Optuna یک روش قدرتمند برای جستجاز فضای هایپرپارمتر استفاده می‌کند و با استفاده از الگوریتم‌های بهینه‌سازی ترکیبی، بهترین مقادیر هایپرپارمتر را پیدا می‌کند. بهینه‌سازی به صورت ترکیبی از الگوریتم‌های نمونه‌برداری تصادفی و الگوریتم‌های جستجوی تکاملی انجام می‌شود.

من سه پارامتر مرحله چهارم به علاوه نرخ dropout را با استفاده از optuna سرچ کردم و مقادیر پارمترها به صورت زیر شد:

```
Best trial:
  Value: 0.8181194067001343
  Params:
    dropout_rate: 0.2
    units: 10
    lr: 0.0017097368533760024
    landa: 0.6
```

و پس finetune کردن مدل پیش اموزش دیده نتایج به صورت زیر شد:



بین kerastuner و optuna برای مسئله انجیر ۴ کلاسه با optuna نتیجه بهتری گرفتم و هم چنین راحتی بیشتری برای کار با optuna داشتم.

جمع بندی نتایج برای داده های تست:

	accuracy	Average core diameter	macro avg f1_score
مدل پیش اموزش دیده Densenet121 بدون سرچ	0.8282	0.7404	0.7
سرچ هایپرپارامترها با استفاده از kerastunner	0.8448	0.7755	0.72
وزن دادن برای بالانس کردن دیتاست	0.8435	0.77915	0.71
بالانس کردن با استفاده از resample	0.8562	0.78696	0.75
اضافه کردن drop out با نرخ ۰.۵	0.8625	0.7947	0.74
اضافه کردن drop out با نرخ 0.8	0.8727	0.7947	0.74
سرچ هایپرپارامترها با استفاده از optuna	0.8676	0.7953	0.76
سرچ با optuna و تغییر دادن batchsize	0.8816	0.8229	0.79