به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین دوم

پرسش ۱	نام و نام خانوادگی	آرین فیروزی
	شماره دانشجویی	810100196
پرسش ۲	نام و نام خانوادگی	آرمان مجیدی
	شماره دانشجویی	810100205
	مهلت ارسال پاسخ	۱۴۰۱.۰۸.۲۹

فهرست

شکل‌ها

پرسش 1

پرسش 2

جدول‌ها

پرسش 1

پرسش 2

پرسش ۱. تشخیص ضایعه سرطانی با استفاده از CNN

۱-۱. معرفی مقاله

در این سوال مدل مربوط به مقاله ی داده شده در محیط jupyter notebook و با استفاده از دادگان پیشنهاد شده در صورت سوال تولید شده و سپس به تحلیل و بهبود مدل میپردازیم.

۱-۲. پیش پردازش تصاویر

تصاویر مربوط به دادگان مقاله از سایت Kaggle و با استفاده از کتابخانه ی مربوط به آن دانلود و در آدرس data/. ذخیره میشود. در مقاله ی منبع، از روش های نرمال سازی، تغییر اندازه ی تصویر و داده افزایی استفاده شده است که به ترتیب به کوچکتر شدن فضای عکس ها و آسانتر شدن فرایند آموزش، کاهش بعد تصاویر و سریعتر شدن هر epoch در آموزش و تست، و بیشتر شدن انواع تصویر مشاهده شده در مدل کمک میکند. در بخش پیش پردازش ما هر سه روش را پیاده سازی کردیم و علاوه بر آن داده ها را بالانس کردیم که در ادامه توضیح آن آورده شده است. روش های دیگری که برای پیش پردازش ممکن بود مفید باشد اعمال شد ولی به علت اینکه تاثیر مثبت زیادی دیده نشد، در نسخه نهایی اعمال نشد. از جمله این روش ها میتوان به سیاه و سفید کردن عکس ها (که گرچه باعث افزایش سرعت آموزش شد، ولی دقت مدل را پایین آورد و چون سرعت آموزش bottleneck این تمرین نبود از آن صرف نظر کردیم) و استاندارد سازی (که باتوجه با اینکه داده های outlier زیادی در دادگان نبود و همچنین contrast غده سرطانی با پوست عادی را کمتر میکرد اعمال نشد) اشاره کرد.

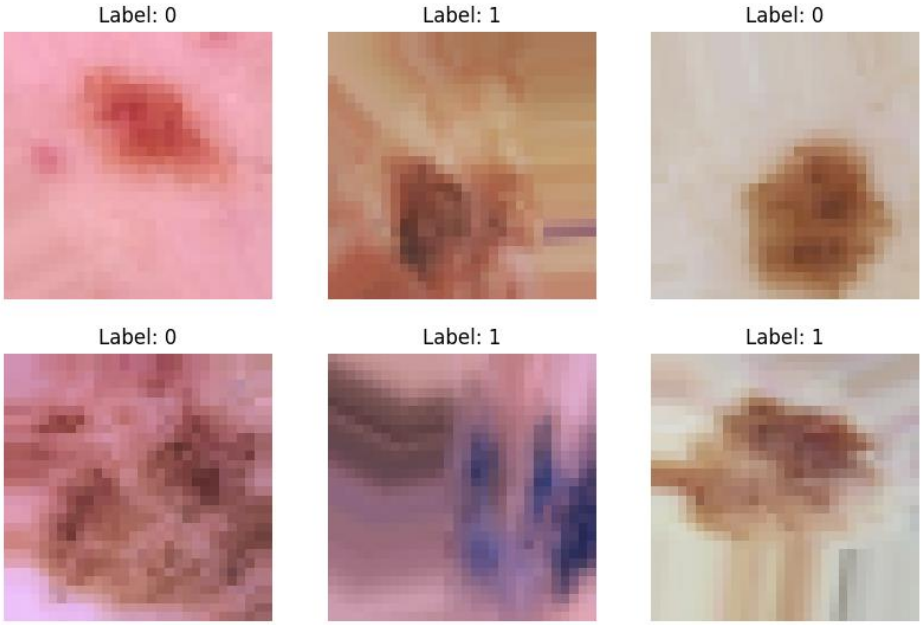
برای پیش پردازش، ابتدا با استفاده از تابع tf.keras.utils.image_dataset_from_directory از کتابخانه ی tensorflow.keras دادگان از پوشه ی ذخیره شده خوانده میشوند. این تابع به صورت خودکار داده ها را به کلاس های مختلف تقسیم میکند و با استفاده از پارامتر های ورودی، سایز عکس ها را ۳۰x۳۰ تنظیم کرده و به صورت رندم ترتیب عکس ها را جابجا میکنیم تا هنگام آموزش مدل به صورت رندم عکس ها را ببیند.

در گام بعدی با چاپ کردن تعداد عکس ها در کلاس های مختلف متوجه میشویم که هم تعداد عکس ها (که معمولا حدود یک دهم تعداد پارامتر ها باید باشد) کمتر از حد معقول است و هم تعداد عکس های یک کلاس بیشتر از دیگری است و داده ها بالانس نیستند. بالانس نبودن داده ها، میتواند باعث آموزش بایاس دار مدل شود، به صورت که مدل به یک کلاس بیشتر از دیگری ارجحیت دهد، و کم بودن

تعداد مثال های دیده شده باعث میشود که مدل کمتر آموزش ببیند و همچنین باعث افزایش احتمال overfit شدن یک مدل بر روی داده های محدود آموزش شود. برای حل این مشکل از داده افزایی کمک میگیریم. داده افزایی با استفاده از ImageDataGenerator از کتابخانه ی tensorflow انجام گرفته و در آن از روش های چرخش، زوم، flip عمودی و افقی، تغییر زاویه دید و افزایش و کاهش طول و عرض تصاویر استفاده کردیم که به صورت رندم بر روی داده ها اعمال میشود که با توجه به دادگان موجود و ذات عکس های پزشکی، به داده هایی که در دنیای واقعی ممکن است ببینیم نزدیک است. در این بخش تعداد داده ها به حدود سه برابر مقدار اولیه افزایش پیدا کرده است.

بعد از augment کردن داده ها، داده ها را به صورت رندم به سه دسته ی آموزش، ارزیابی و صحت سنجی (validation) تقسیم میکنیم و داده های آموزش را دوباره بالانس میکنیم تا آموزش مدل biased نشود. این تقسیم کردن به صورت ۸۰٪ برای آموزش، ۱۰٪ برای ارزیابی و ۱۰٪ برای صحت سنجی انجام گرفته است.

در آخرین مرحله ی پیش پردازش، داده ها را نرمال سازی میکنیم تا مقدار عددی رنگ ها به بازه ی ۱ تا ۰ تغییر پیدا کند. تعدادی از تصاویر بعد از پیش پردازش در تصویر ۱-۱ قابل مشاهده هستند.



شکل ۱.۱. تصاویر بعد از اعمال پیش پردازش

۱-۳. داده افزایی

با توجه به این که در کد نوشته شده داده افزایی را همراه با پیش پردازش انجام دادیم، مطالب این بخش در پاراگراف سوم بخش پیش پردازش توضیح داده شد.

۱-۳. پیاده سازی

۱-۳-۱. مدل مقاله

در این بخش تلاش کردیم مدل ارائه شده تا حد امکان نزدیک به مدل استفاده شده در مقاله باشد. برای پیاده سازی از کتابخانه tensorflow.keras استفاده کردیم و مدل پیشنهادی مقاله را پیاده کردیم. مدل پیاده سازی شده تنها در دو بخش با مقاله تفاوت دارد: اول اینکه در مقاله از sparse categorical crossentropy استفاده شده که با توجه به اینکه دادگان ما تنها در دو کلاس بودند، از اکتیویشن binary crossentropy استفاده کردیم و دوم اینکه با توجه به تفاوتی که ذکر شد، لایه آخر مدل تنها یک نورون لازم داشت و این تغییر نیز اعمال شد. ساختار مدل در شکل ۱.۲ نشان داده شده است.

Layer (type)	Output Shape	Param #
conv2d_130 (Conv2D)	(None, 28, 28, 16)	448
max_pooling2d_130 (MaxPooling2D)	(None, 14, 14, 16)	0
conv2d_131 (Conv2D)	(None, 14, 14, 32)	4,640
max_pooling2d_131 (MaxPooling2D)	(None, 7, 7, 32)	0
conv2d_132 (Conv2D)	(None, 7, 7, 64)	18,496
max_pooling2d_132 (MaxPooling2D)	(None, 4, 4, 64)	0
conv2d_133 (Conv2D)	(None, 4, 4, 128)	73,856
max_pooling2d_133 (MaxPooling2D)	(None, 2, 2, 128)	0
flatten_29 (Flatten)	(None, 512)	0
dense_93 (Dense)	(None, 64)	32,832
dense_94 (Dense)	(None, 32)	2,080
dense_95 (Dense)	(None, 1)	33

Total params: 132,385 (517.13 KB)

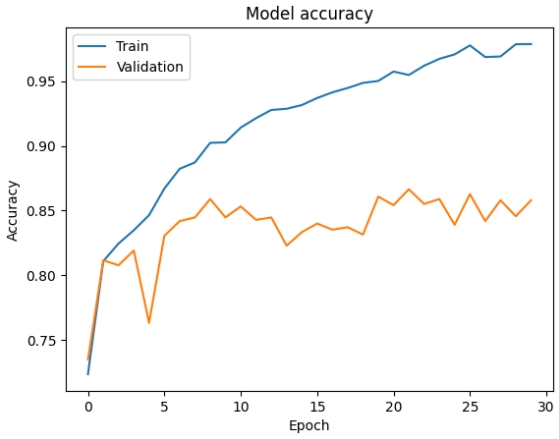
شکل ۱.۲. ساختار مدل پیشنهادی مقاله

برای آموزش مدل، با پیروی از مقاله دل را در ۳۰ epoch آموزش دادیم و اندازه ی هر batch را ۱۲۸قرار دادیم. برای تامین داده ها از دیتالودر tenorflow استفاده کردیم که در بخش ۱-۱ توضیح مربوطه داده شد. مزیت این روش درگیری کمتر با فرمت ورودی و آسانتر شدن برخی از پیش پردازش ها (مثل resize) میباشد. همچنین استفاده از دیتالودر مربوط به کتابخانه ی مدل باعث میشود در صورت به روز رسانی کتابخانه مجبور به تغییر در کد نباشیم و کد maintainable تر باشد، ولی از طرف دیگر استفاده از این دیتالودر ها به مطالعه ی آن دیتالودر نیاز دارد و همچنین انعطاف پذیری این روش نسبت به روش

دستی کمتر است، چرا که دیتاست خروجی از فرمت خاصی پیروی میکند که ممکن است به ما اجازه ی دسترسی سطوح پایینتر (مثل مقدار عددی تصاویر) را ندهد.

نتیجه نهایی آموزش این مدل در شکل ۱.۳ و ۱.۴ نمایش داده شده اند. با بررسی این نتایج میتوان مشاهده کرد که مدل overfit شده است چرا که پس از تعدادی epoch دقت آموزش افزایش می یابد ولی دقت صحت سنجی ثابت است، همچنین loss صحت سنجی افزایش پیدا کرده است.

شکل ۱.۳. مقدار loss مدل پیشنهادی مقاله



شکل ۱.۴. مقدار دقت مدل پیشنهادی مقاله

۱-۳-۲. مدل اصلاح شده

با توجه به نتایج آموزش مدل قبلی، میتوان نتیجه گرفت که مدل overfit شده است. برای جلوگیری از این اتفاق، میتوانیم از لایه های dropout و batchNorm استفاده کنیم. هر دو لایه میتوانند گزینه ی خوبی برای این کار باشند، ولی در این مدل ما با توجه به این که داده ها از قبل نرمال شده بودند، از لایه dropout استفاده کردیم که پیاده سازی و استفاده ی آسانتری دارد. همچنین لایه های کانوولوشنی مدل مقاله بیش از حد بزرگ بودند که برای دیتاست ما مزیتی ایجاد نمیکرد و فقط پیچیدگی

Layer (type)	Output Shape	Param #
conv2d_138 (Conv2D)	(None, 28, 28, 16)	448
max_pooling2d_138 (MaxPooling2D)	(None, 14, 14, 16)	0
conv2d_139 (Conv2D)	(None, 14, 14, 32)	4,640
max_pooling2d_139 (MaxPooling2D)	(None, 7, 7, 32)	0
conv2d_140 (Conv2D)	(None, 7, 7, 64)	18,496
max_pooling2d_140 (MaxPooling2D)	(None, 4, 4, 64)	0
dropout_48 (Dropout)	(None, 4, 4, 64)	0
conv2d_141 (Conv2D)	(None, 4, 4, 64)	36,928
max_pooling2d_141 (MaxPooling2D)	(None, 2, 2, 64)	0
dropout_49 (Dropout)	(None, 2, 2, 64)	0
flatten_31 (Flatten)	(None, 256)	0
dense_99 (Dense)	(None, 32)	8,224
dense_100 (Dense)	(None, 8)	264
dense_101 (Dense)	(None, 1)	9

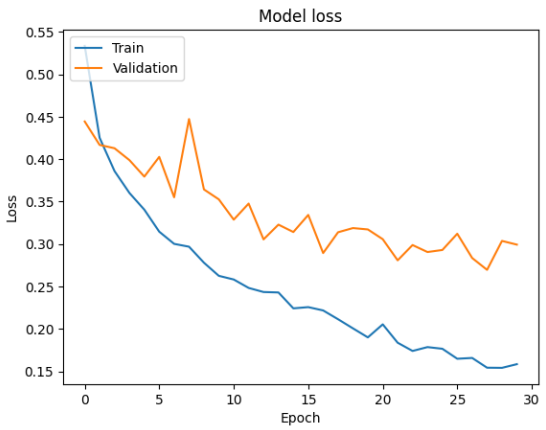
Total params: 69,009 (269.57 KB)

شکل ۱.۵. ساختار مدل اصلاح شده

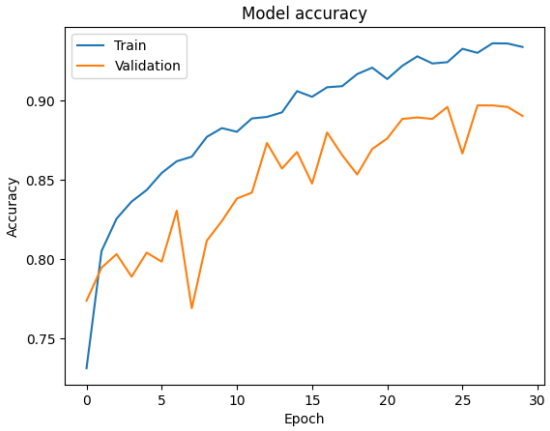
آموزش را بالا میبرد، در نتیجه تعداد نورون های لایه آخر کانوولوشنی را کمتر کردیم و به تبع آن لایه های fully connected هم کوچکتر شدند و پارامتر های مدل به حدود نصف مدل قبلی کاهش پیدا کرد و دقت مدل بیشتر شد. ساختار این مدل در شکل ۱.۵ نشان داده شده است.

در آموزش این بخش نیز مانند بخش قبلی، تقریبا از مقاله پیروی شده، با این تفاوت که مقدار batch به علت کوچکتر بودن تعداد دادگان، ۳۲ تنظیم شده است.

نتیجه نهایی آموزش این مدل در شکل ۱.۶ و ۱.۷ نمایش داده شده اند. با بررسی این نتایج میتوان مشاهده کرد که این بار مدل overfit نشده و مقدار خطا و دقت دادگان آموزش و صحت سنجی تقریبا به یک اندازه تغییر میکند که خروجی دلخواه ماست.



شکل ۱.۶. مقدار loss مدل پیشنهادی مقاله



شکل ۱.۷. مقدار دقت مدل پیشنهادی مقاله

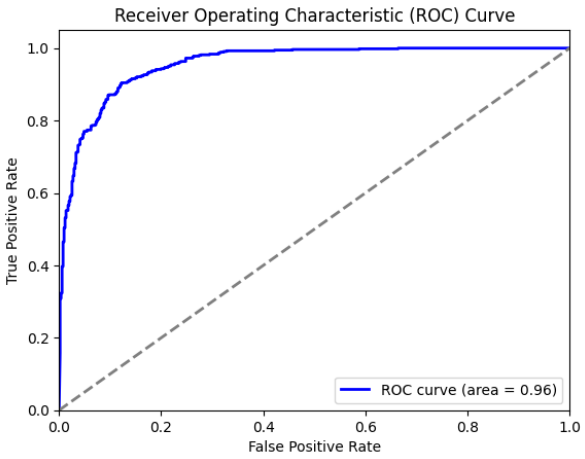
۱-۵. تحلیل نتایج

در این بخش نتایج با معیار ها مختلف ارزیابی شده اند. نمودار ها loss و دقت برای مدل ها در بخش پیاده سازی رسم شده اند و چون ارزیابی بر روی مدل آموزش داده شده انجام میگیرد، نمیتوان نمودار آن را رسم کرد و به جای آن از معیار های دیگر استفاده میکنیم.

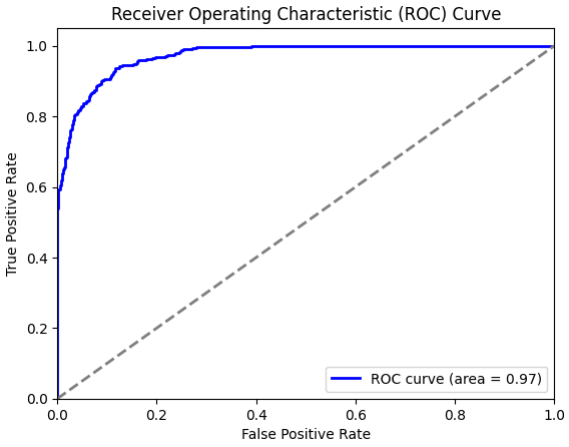
۱-۵-۱. نمودار ROC

این نمودار با توجه به پیشبینی های صحیح مدل (true positive و false positive) در بازه های مختلف رسم میشود. مساحت زیر نمودار برای مدل رندم ۰.۵ است و هرچقدر این مقدار بیشتر باشد، مدل پیشبینی های دقیق تری دارد. با مقایسه این نمودار برای مدل مقاله قبل و بعد اصلاح مشاهده میکنیم که

مدل اصلاح شده منحنی صعودی تری تولید میکند و مساحت زیر آن بیشتر است، در نتیجه پیشبینی های بهتری ارائه میدهد. این نمودار در اشکال ۱.۸ و ۱.۹ قابل مشاهده است.



شکل ۱.۸. نمودار ROC مدل پیشنهادی مقاله

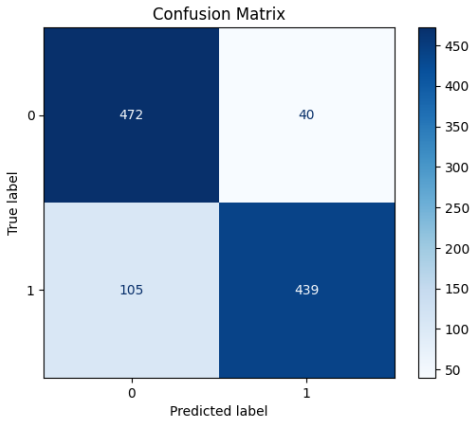


شکل ۱.۹. نمودار ROC مدل اصلاح شده

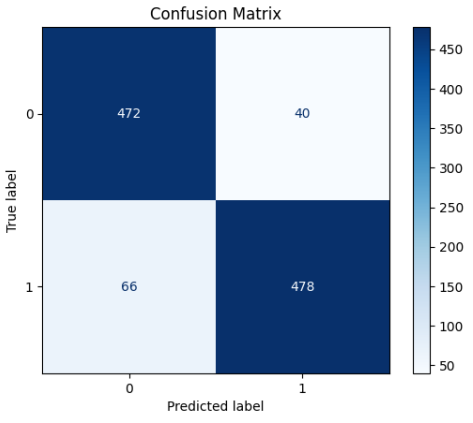
۱-۵-۲. ماتریس آشفتگی

ماتریس آشفتگی ماتریسی است که در آن مقدار پیشبینی های درست و اشتباه هر کلاس نشان داده میشود، به صورتی که هر ستون مقدار پیشبینی شده و هر سطر مقدار واقعی کلاس را نشان میدهد. هر چقدر ماتریس به ماتریس قطری نزدیکتر باشد، مدل عملکرد بهتری داشته است.

با مقایسه ی ماتریس های آشفتگی متوجه میشویم که مدل قبل از اصلاح در تشخیص کلاس یک مشکل داشت ولی پس از اعمال تغییرات تشخیص معتبر تری انجام داده است. این ماتریس برای دادگان ارزیابی در اشکال ۱.۱۰ و ۱.۱۱ قابل مشاهده است.



شکل ۱.۱۰. ماتریس آشفتگی مدل پیشنهادی مقاله



شکل ۱.۱۱. ماتریس آشفتگی مدل اصلاح شده

۱-۵-۳. معیار های دیگر

برای ارزیابی بیشتر مدل ها، از معیار های precision، recall، f1 score و دقت استفاده شده است. سه معیار اول برای هر کلاس تعریف شده اند و معیار دقت برای کل دادگان ارزیابی است. دلیل استفاده از معیارf1 این است که دو معیار اول در شرایطی که مدل تنها یک خروجی بدهد(یا نسبت به یک خروجی biased باشد) با هم تفاوت دارند و نمیتوانیم دقت واقعی مدل را با آن معیارها ارزیابی کنیم. تعاریف معیارها در ذیل آورده شده:

- Precision : این معیار دقت مدل را با استفاده از نسبت مقادیری که درست پیشبینی شده اند به کل مقادیری که در آن کلاس وجود داشتند به دست می آید.
- Recall: این معیار حساسیت مدل را با استفاده از نسبت مقادیری که درست پیشبینی شده اند به کل مقادیری که به عنوان آن کلاس پیشبینی شده به دست می آید.
- F1-score: این معیار به صورت نسبت دوبرابر ضرب معیار های بالا تقسیم بر جمعشان به دست می آید و هدف آن ارائه یک معیار واحد برای ارزیابی حساسیت و precision است.
- Accuracy: تعداد تمام پیش بینی های درست تقسیم بر کل داده ها

برای محاسبه این معیار ها از تابع classification_report کتابخانه ی scikit-learn استفاده کرده ایم و نتایج آن در جدول ۱.۱ قابل مشاهده است. با مقایسه این جدول توجه میشویم که precision کلاس اول و recall کلاس دوم در مدل اصلاح شده بهتر استُ به این معنی که تعدادی از عکسهای مربوط به کلاس دوم، به اشتباه مربوط به کلاس اول تشخیص داده میشدند که در مدل اصلاح شده بهتر شده است. همچنین

با مشاهده f1 scoreها در میابیم که همانطور که در تعاریف ذکر شد، این معیار نسبت به دو معیار گفته شده قابل اطمینان تر است و نتایج هر دو معیار را برای ارزیابی استفاده میکند، در نتیجه با یک نگاه ساده به این معیار میتوان دریافت که هر مدل چگونه عمل کرده است.

جدول ۱.۱ مقایسه ی مدل مقاله و مدل اصلاح شده

		Precision	Recall	F1-Score	Accuracy
مدل مقاله	کلاس اول	۰.۸۲	۰.۹۲	۰.۸۷	۰.۸۶
	کلاس دوم	۰.۹۲	۰.۸۱	۰.۸۶	
مدل اصلاح شده	کلاس اول	۰.۸۸	۰.۹۲	۰.۹۰	۰.۹۰
	کلاس دوم	۰.۹۲	۰.۸۸	۰.۹۰	

۱-۷. مدل عمیق تر

در این بخش تلاش کردیم با ارائه یک مدل عمیق تر، نتایج بهتری نسبت به مل اصلاح شده به دست آوریم که در ادامه روش پیاده سازی و نتایج آن شرح داده شده اند.

۱-۷-۱. پیاده سازی

برای بهتر کردن نتیجه ی مدل، یکی از روش های ساده، افزایش تعداد لایه ها است. به این منظور، یک لایه کانوولوشنی و یک لایه fully connected به مدل اضافه کردیم و تعداد نورون های لایه های آخر کانوولوشنی را افزایش دادیم. ساختار این مدل در شکل ۱.۱۲ نشان داده شده است.

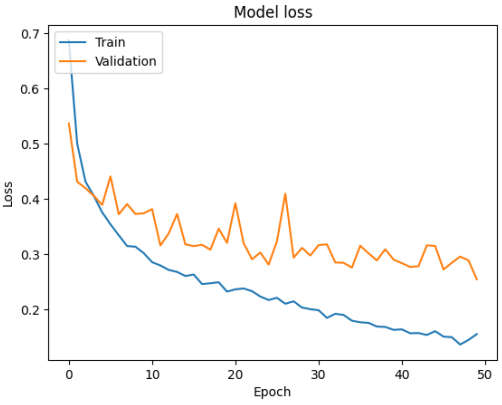
برای آموزش این مدل، بر خلاف مدل های قبلی از epoch۳۰ استفاده شده است چون مدل لایه هایی بیشتر دارد و

Layer (type)	Output Shape	Param #
conv2d_162 (Conv2D)	(None, 28, 28, 16)	448
max_pooling2d_162 (MaxPooling2D)	(None, 14, 14, 16)	0
conv2d_163 (Conv2D)	(None, 14, 14, 32)	4,640
max_pooling2d_163 (MaxPooling2D)	(None, 7, 7, 32)	0
conv2d_164 (Conv2D)	(None, 7, 7, 64)	18,496
max_pooling2d_164 (MaxPooling2D)	(None, 4, 4, 64)	0
dropout_62 (Dropout)	(None, 4, 4, 64)	0
conv2d_165 (Conv2D)	(None, 4, 4, 64)	36,928
max_pooling2d_165 (MaxPooling2D)	(None, 2, 2, 64)	0
dropout_63 (Dropout)	(None, 2, 2, 64)	0
conv2d_166 (Conv2D)	(None, 2, 2, 64)	36,928
max_pooling2d_166 (MaxPooling2D)	(None, 1, 1, 64)	0
dropout_64 (Dropout)	(None, 1, 1, 64)	0
flatten_36 (Flatten)	(None, 64)	0
dense_118 (Dense)	(None, 64)	4,160
dense_119 (Dense)	(None, 32)	2,080
dense_120 (Dense)	(None, 8)	264
dense_121 (Dense)	(None, 1)	9

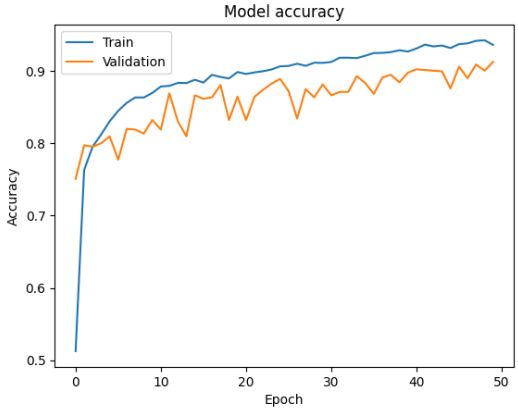
Total params: 103,953 (406.07 KB)

شکل ۱.۱۲. ساختار مدل عمیق تر

سرعت همگرایی آن کمتر از مدل های قبلی است. نتایج حاصل از از مدل در شکل های ۱.۱۳ و ۱.۱۴ قابل مشاهده است. نتایج نشان میدهد که در طول فرایند آموزش روند دقت صعودی و روند خطا نزولی میباشد و به علت وجود لایه های dropout دچار overfit نشده ایم.



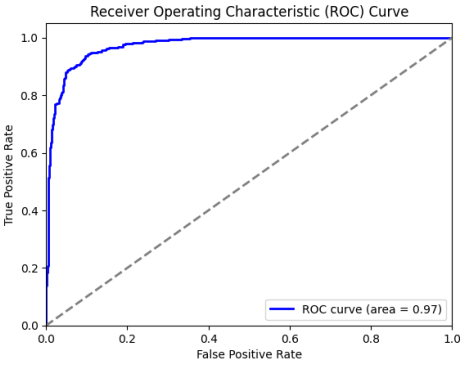
شکل ۱.۱۳. مقدار loss مدل عمیق تر



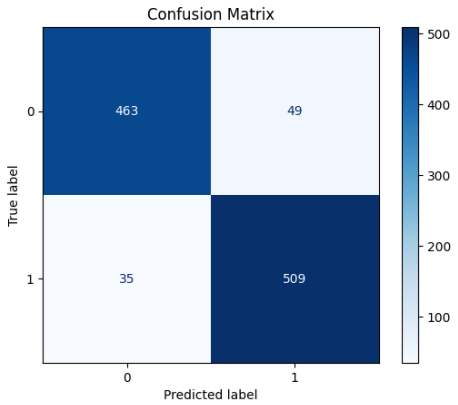
شکل ۱.۱۴. مقدار دقت مدل عمیق تر

۱-۷-۲. تحلیل نتایج و مقایسه

در این بخش نیز مانند بخش های قبلی نمودار ROC و ماتریکس آشفتگی برای مدل رسم شده است که به ترتیب در اشکال ۱.۱۵ و ۱.۱۶ قابل مشاهده هستند. نمودار ROC این مدل فرق چندانی با مدل اصلاح شده ندارد ولی ماتریکس آشفتگی نشان میدهد که تشخیص اشتباه کلاس دوم به عنوان کلاس اول که تفاوت مدل های مقاله و اصلاح شده بود بهتر شده است.



شکل ۱.۱۵. نمودار ROC مدل عمیق تر



شکل ۱.۱۶. ماتریس آشفتگی عمیق تر

برای ارزیابی بیشتر مدل و مقایسه آن با مدل های قبلی معیار های مورد بررسی در مدل ها را برای این مدل نیز ارزیابی کرده و در جدول ۱.۲ نشان دادیم. همانطور که در جدول قابل مشاهده است، مدل عمیق تر از مدل اصلاح شده بهتر عمل کرده که این عملکرد به علت لایه های بیشتر و توانایی مدل کردن مسائل پیچیده تر در این مدل است. در عوض، سرعت همگرایی این مدل پایین تر بوده و شاید tradeoff بین دقت و سرعت بیشتر از اندازه مطلوب باشد، چون تعداد پارامتر ها بیشتر شده ولی دقت تنها ۲ درصد افزایش پیدا کرده است.

جدول ۱.۲ مقایسه ی مدل ها

		Precision	Recall	F1-Score	Accuracy
مدل مقاله	کلاس اول	۰.۸۲	۰.۹۲	۰.۸۷	۰.۸۶
	کلاس دوم	۰.۹۲	۰.۸۱	۰.۸۶	
مدل اصلاح شده	کلاس اول	۰.۸۸	۰.۹۲	۰.۹۰	۰.۹۰
	کلاس دوم	۰.۹۲	۰.۸۸	۰.۹۰	
مدل عمیق تر	کلاس اول	۰.۹۳	۰.۹۰	۰.۹۲	۰.۹۲
	کلاس دوم	۰.۹۱	۰.۹۴	۰.۹۲	

پرسش ۲ – تشخیص بیماری‌های برگ لوبیا با شبکه‌های عصبی

2-۱. پیش‌پردازش تصاویر

در این قسمت ابتدا داده‌ها را از سایت Kaggle دانلود می‌کنیم و سپس آدرس آن را در متغیر data_path قرار می‌دهیم. سپس طبق مقاله سایز تصاویر را برابر با (224, 244, 3) قرار می‌دهیم. batch_size را نیز مشابه با مقاله برابر با 32 قرار می‌دهیم. حال، با استفاده از یکی از toolbox های keras (tf.keras.preprocessing.image_dataset_from_directory) عکس‌ها را لود می‌کنیم. با توجه به این که در خود Kaggle عکس ها به سه دسته train, validation, test دسته‌بندی کرده‌است، نیازی به انجام این کار نداریم. بعد، با تابع dataset_to_numpy() که خودمان تعریف کرده‌ایم، عکس‌ها را به آرایه numpy تبدیل می‌کنیم تا متغیرهای مربوط به train, validation, test تولید شوند.

استفاده از این پیش پردازش دلایل متفاوتی دارد که در پایین برخی از آن ها آمده است:

- **کاهش محاسبات پردازشی:** در پیش‌پردازش، با کوچک ساختن ابعاد عکس محاسبات پردازشی را کاهش داده‌ایم؛ در واقع، برخی کاهش ابعاد باعث کاهش اطلاعات موجود در تصویر به حد قابل قبولی می‌شوند اما با انتخاب ابعاد مناسب، می‌توان از کاهش اطلاعات قابل توجه جلوگیری کرد.
- **هم‌اندازه شدن ابعاد تمامی تصاویر:** در شبکه‌های عصبی، ما توقع داریم که ابعاد ورودی یکسان باشد؛ در نتیجه، با یک پیش‌پردازش ابعاد تصویر را یکی می‌کنیم تا توانایی دادن آن را به مدل داشته‌باشیم.
- **مدیریت نویزهای تصویر:** با اعمال پیش‌پردازش به دیتاست، می‌توانیم نویزهای موجود در تصویر را کاهش و در صورت امکان، به طور کامل حذف کنیم.

2-2. پیاده‌سازی

2-2-1. انتخاب مدل‌ها

ابتدا مدل MobileNetV2 را بررسی می‌نماییم. این مدل مشابه MobileNet می‌باشد، با این تفاوت که در این مدل از بلوک‌های inverted residuals همراه با bottlenecking features استفاده شده‌است.

این مدل در حالت اصلی خود عکس‌هایی با سایز (224, 224, 3) می‌گیرد؛ همچنین، ورودی این مدل نباید سایزی کمتر از سایز (32, 32, 3) داشته‌باشد. روش پیشنهادی و استفاده‌شده در مقاله، transfer learning می‌باشد. همچنین این مدل به خودی خود در لایه آخر، 1000 نورون دارد در صورتی که مطلوب ما 3 نورون می‌باشد. در نتیجه آرگومان include_top را غیرفعال می‌کنیم و همچنین به علت استفاده از روش transfer learning، وزن‌های مدل MobileNetV2 را freeze می‌کنیم. این مدل با تصاویر ImageNet آموزش داده شده‌است.

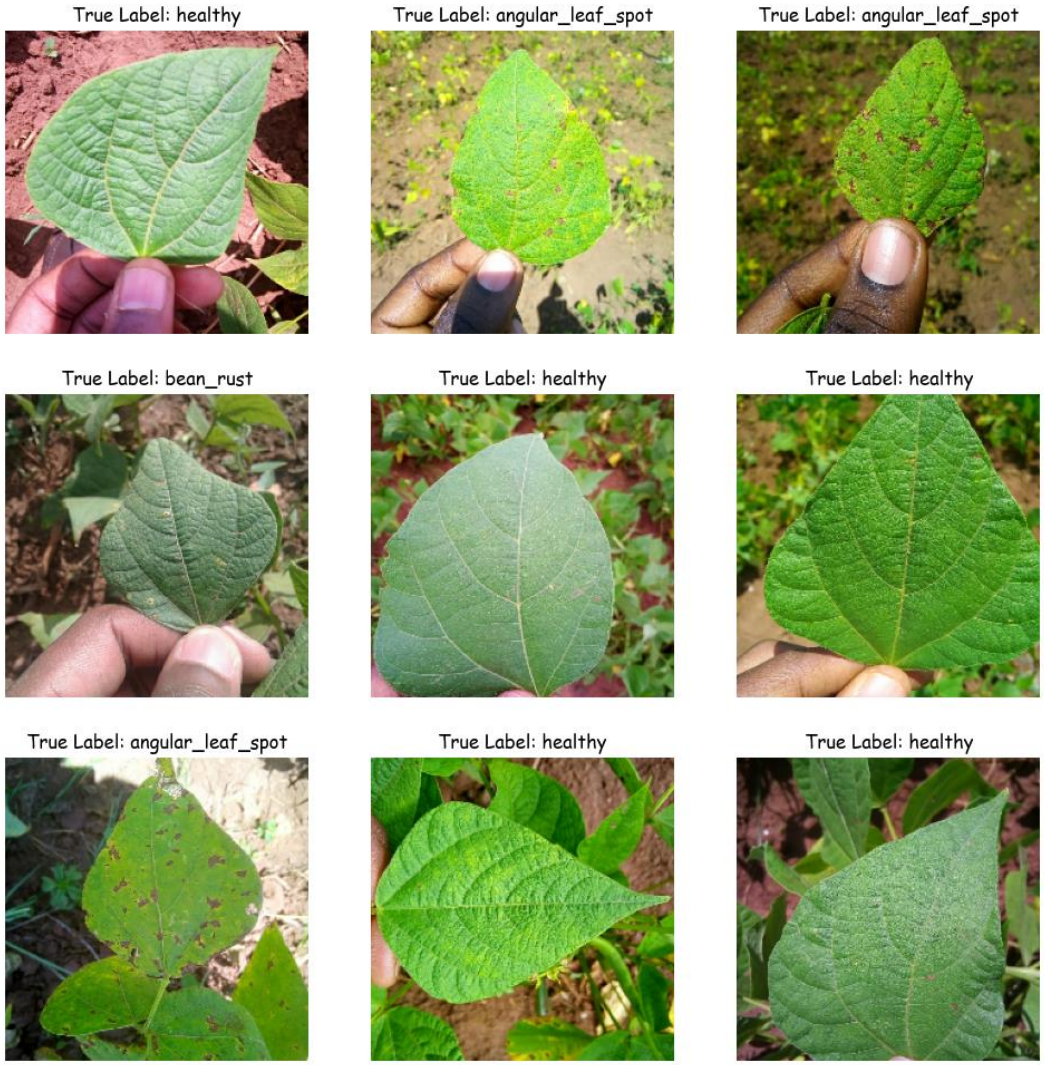
حال به بررسی مدل EfficientNetB6 می‌پردازیم. این مدل در حالت اصلی خود عکس‌هایی با سایز (1, 1, 1) می‌گیرد. روش پیشنهادی و استفاده‌شده در مقاله، transfer learning می‌باشد. همچنین این مدل به خودی خود در لایه آخر، 1000 نورون دارد در صورتی که مطلوب ما 3 نورون می‌باشد. در نتیجه آرگومان include_top را غیرفعال می‌کنیم و همچنین به علت استفاده از روش transfer learning، وزن‌های مدل EfficientNetB6 را freeze می‌کنیم. این مدل با تصاویر ImageNet آموزش داده شده‌است.

در انتها، به بررسی مدل NasNet می‌پردازیم. این مدل در دو صورت NasNetLarge و NasNetMoible وجود دارد. مدل NasNetLarge در حالت اصلی خود عکس‌هایی با سایز (1, 1, 1) می‌گیرد اما مدل NasNetMobile همان سایز عکس خودمان یعنی (224, 224, 3) را می‌پذیرد. به علت محاسبات طولانی و سنگین NasNetLarge، طبق پیشنهاد دستیار آموزشی از مدل NasNetMobile استفاده کردم. روش پیشنهادی و استفاده‌شده در مقاله، transfer learning می‌باشد. همچنین این مدل به خودی خود در لایه آخر، 1000 نورون دارد در صورتی که مطلوب ما 3 نورون می‌باشد. در نتیجه آرگومان include_top را غیرفعال می‌کنیم و همچنین به علت استفاده از روش transfer learning، وزن‌های مدل NasNetMobile را freeze می‌کنیم. این مدل با تصاویر ImageNet آموزش داده شده‌است.

به طور کلی می‌توان گفت که یادگیری انتقالی یا transfer learning، شیوه‌ایست که در آن وزن‌های مدل pre-trained را freeze می‌کنیم و سپس مطابق با اهداف خود، یک تعداد لایه به مدل اضافه می‌کنیم و آن ها را آموزش می‌دهیم.

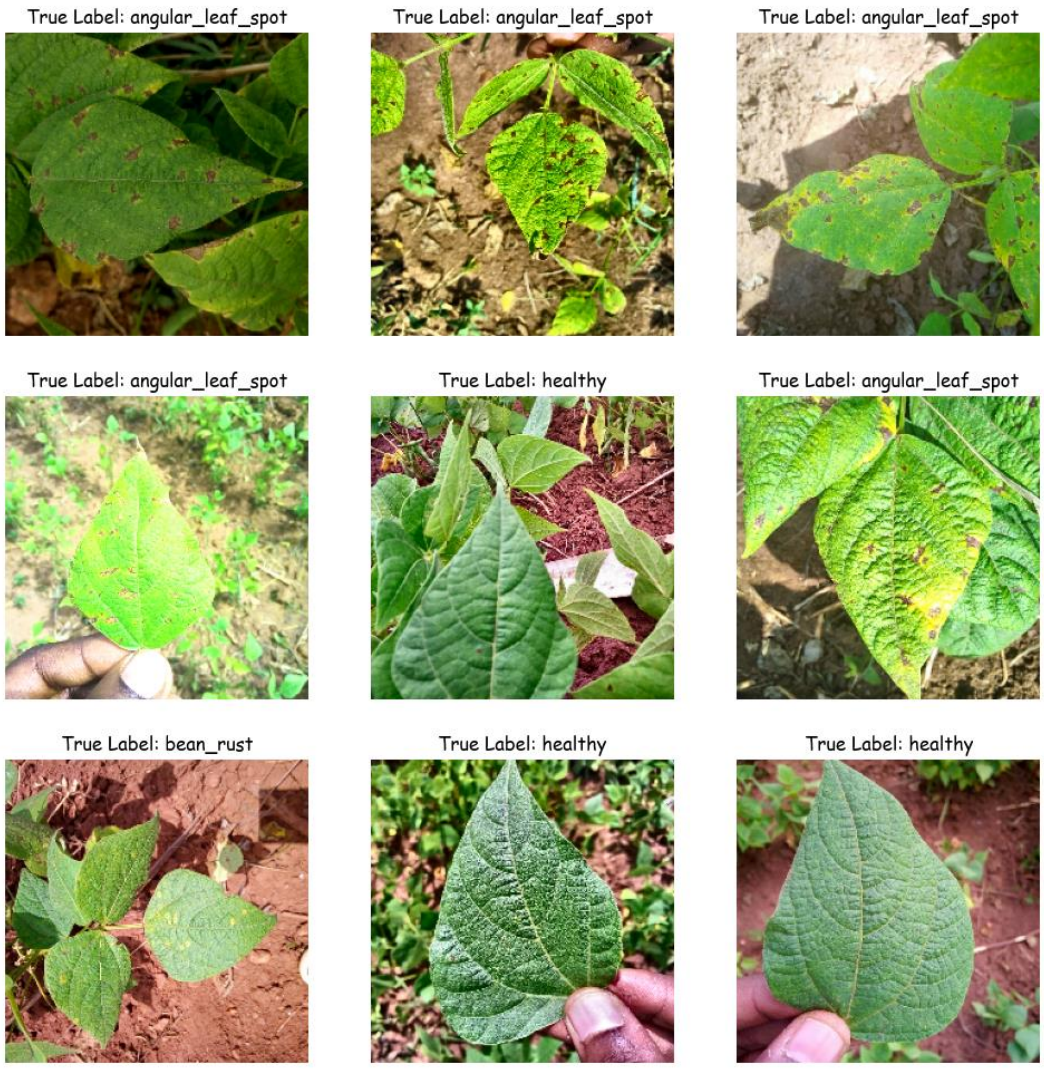
2-2-2. تقویت داده

برای تقویت داده، مطابق فایل تمرین از کتابخانه Albumentations استفاده می‌کنیم. شکل 1، تصاویر دیتاست (بدون تقویت داده) را نشان می‌دهد.



شکل 1. عکس‌های تصادفی از دیتاست (بدون تقویت داده)

سپس، با استفاده از تکنیک‌های تقویت داده مانند چرخش تصادفی 90 درجه‌ای عکس، آینه‌ای کردن عکس، تغییر تصادفی روشنایی و کنتراست عکس، تغییر رنگ و اشتباع عکس و همچنین همسانسازی عکس، اقدام به augment کردن عکس‌ها می‌کنیم. شکل 2، مثالی از تصاویر اصلی به همراه تصاویر تقویت‌شده را نشان می‌دهد.



شکل 2. عکس‌های تصادفی از دیتاست (همراه با تقویت داده)

2-2-3. تقویت داده

مطابق آن چه در گام 2-2-1 گفته‌شد، سایز ورودی به هر مدل را می‌دانیم؛ اما مقاله برای تمامی مدل‌ها از عکس‌هایی با سایز (224, 224, 3) استفاده کرده‌است. در نتیجه ما باید به نحوی مطابق خواسته‌های خودمان و با سایز دلخواه و مطلوبمان عکس را به مدل بدهیم. برای این کار، در داخل مدل‌های pre-trained، آرگومان include_top را برابر با False قرار می‌دهیم تا سایز عکس مطابق خواسته ما باشد.

به طور کلی برای تنظیم اندازه سایز دو حالت داریم. در حالت اول مطلوب ما عکسی با سایز کوچکتر از عکس خودمان است. در این صورت، عکس را downsample می‌کنیم. حالت دوم ما عکس مطلوب عکسی با سایز بزرگتر است که در این صورت باید upsample و interpolation صورت گیرد.

اصولا هر مدلی یک سایز ویژه و خاص خود را دارد و انتخاب آن سایز ویژه می‌تواند اینگونه تعبیر شود که در فضا و قیود همان مدل شبیه‌سازی خود را انجام می‌دهیم.

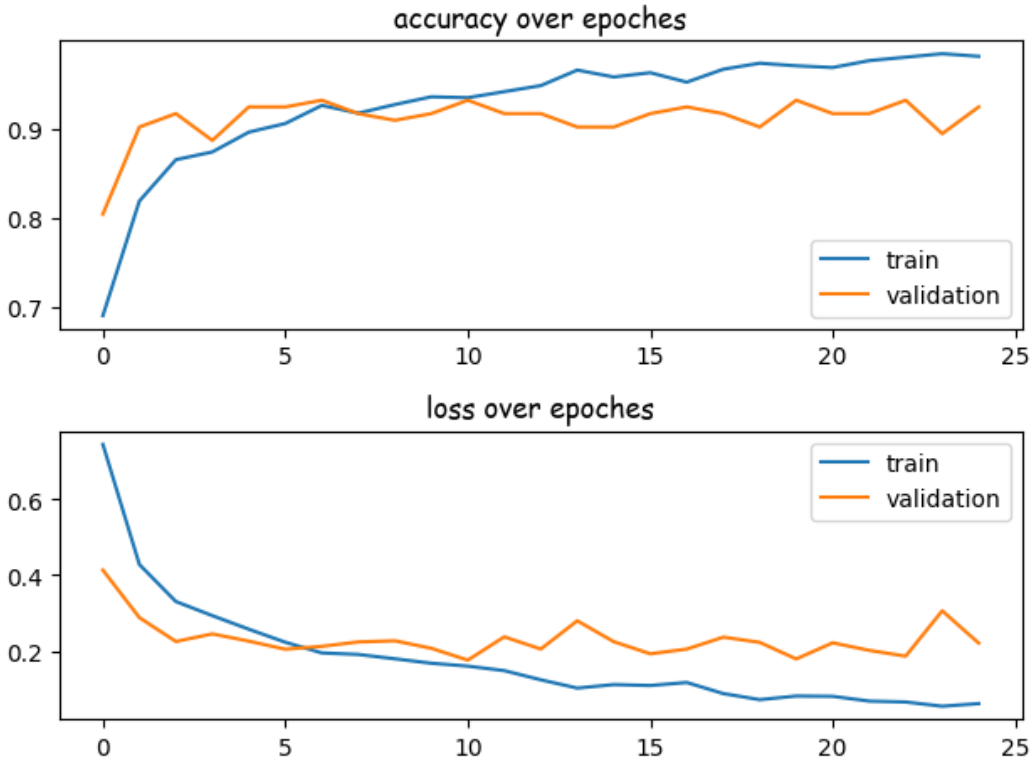
همچنین، فرض و مشاهدات من این را منتج شد که برای دو مدل MobileNetV2, NasNet، مقادیر عکس‌ها را به بازه [0, 1] مپ کنم، اما در مدل EfficientNetB6 مقادیر عکس‌ها را در همان بازه [0, 255] نگه دارم (علت عدم تغییر بازه مدل EfficientNetB6 را توسط شهود خود کم بودن سرعت همگرا شدن شناسایی کردم).

2-2-4. بهینه‌سازها

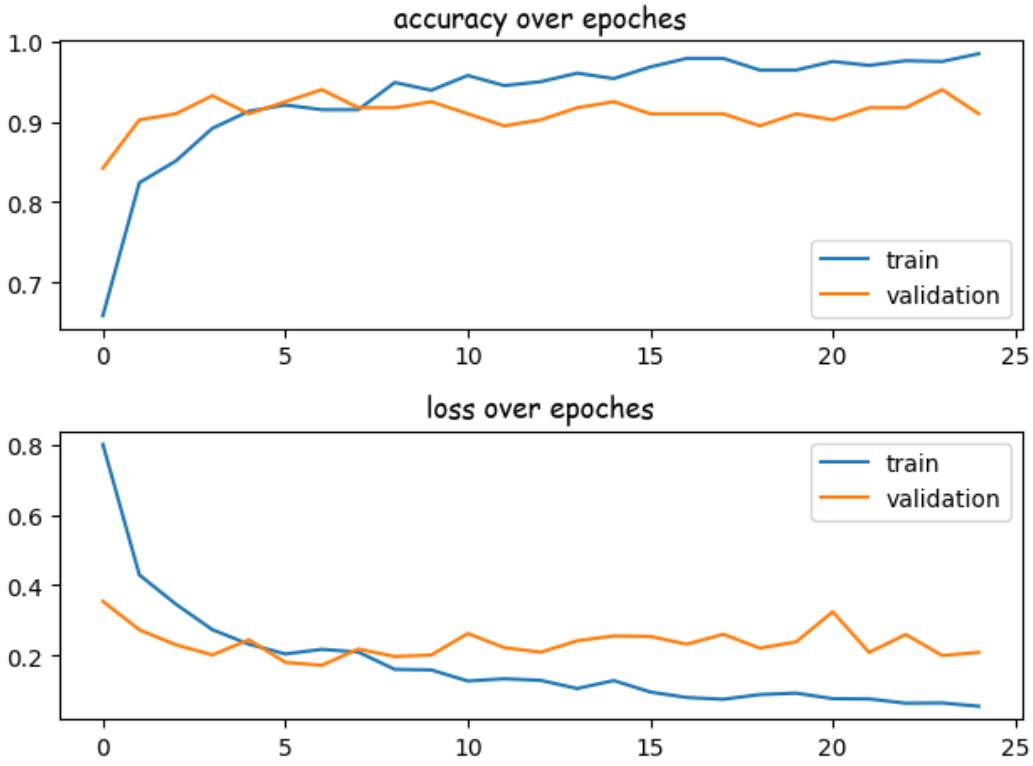
نتایج بدست آمده از هر بهینه‌ساز در قسمت 2-3 نشان داده‌شده است.

2-2-5. آموزش مدل

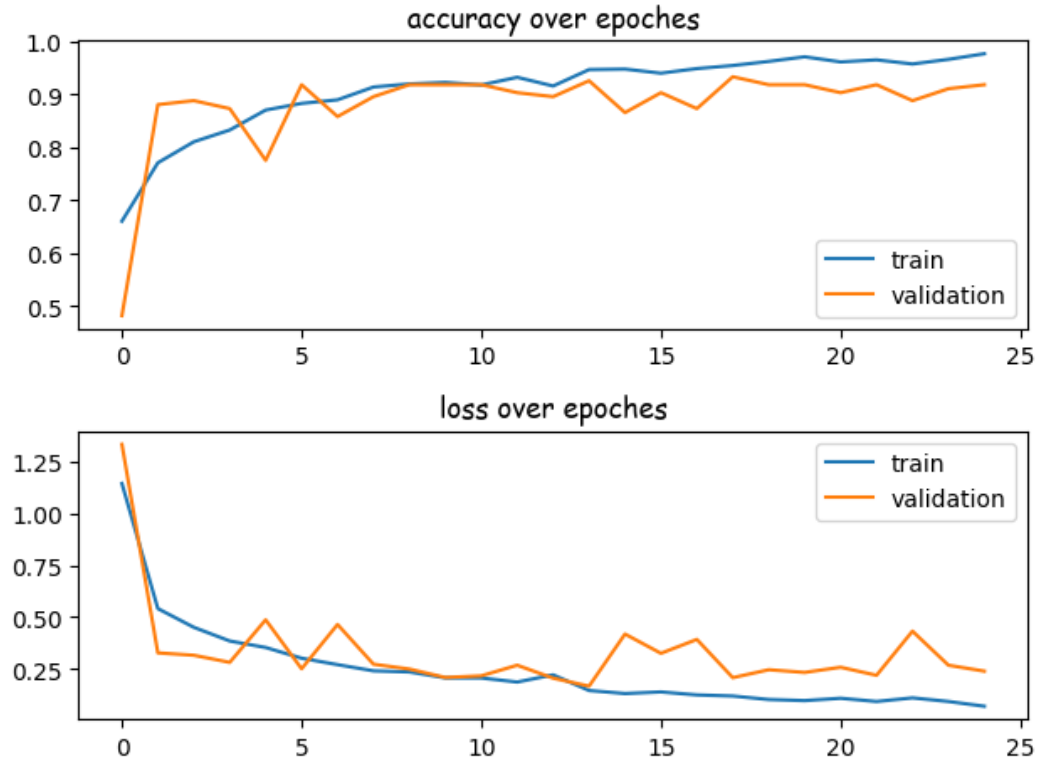
- **مدل MobileNetV2**: این مدل با 3 بهینه‌ساز Adam, Nadam, RMSprop آموزش داده می‌شود، تابع دقت و هزینه آموزش با این 3 بهینه‌ساز در شکل 3، 4 و 5 آمده‌است.
  - طبق نتایج بدست آمده در طی 25 ایپاک، مشاهده می‌کنیم که در آموزش مدل با بهینه‌ساز Adam، مقدار دقت در ایپاک 25 ام روی داده train برابر با 98.24% و روی داده validation برابر با 92.48% می‌باشد. بهترین نتیجه نیز متعلق به ایپاک 23 ام می‌باشد که دقت روی داده train آن برابر با 98.11% و روی داده validation برابر با 93.23% می‌باشد.
  - طبق نتایج بدست آمده در طی 25 ایپاک، مشاهده می‌کنیم که در آموزش مدل با بهینه‌ساز Nadam، مقدار دقت در ایپاک 25 ام روی داده train برابر با 98.65% و روی داده validation برابر با 90.98% می‌باشد. بهترین نتیجه نیز متعلق به ایپاک 24 ام می‌باشد که دقت روی داده train آن برابر با 97.69% و روی داده validation برابر با 93.98% می‌باشد.
  - طبق نتایج بدست آمده در طی 25 ایپاک، مشاهده می‌کنیم که در آموزش مدل با بهینه‌ساز RMSprop، مقدار دقت در ایپاک 25 ام روی داده train برابر با 97.12% و روی داده validation برابر با 91.73% می‌باشد. بهترین نتیجه نیز متعلق به ایپاک 18 ام می‌باشد که دقت روی داده train آن برابر با 94.75% و روی داده validation برابر با 93.23% می‌باشد.



شکل 3. آموزش مدل **MobileNetV2** با بهینه‌ساز **Adam**



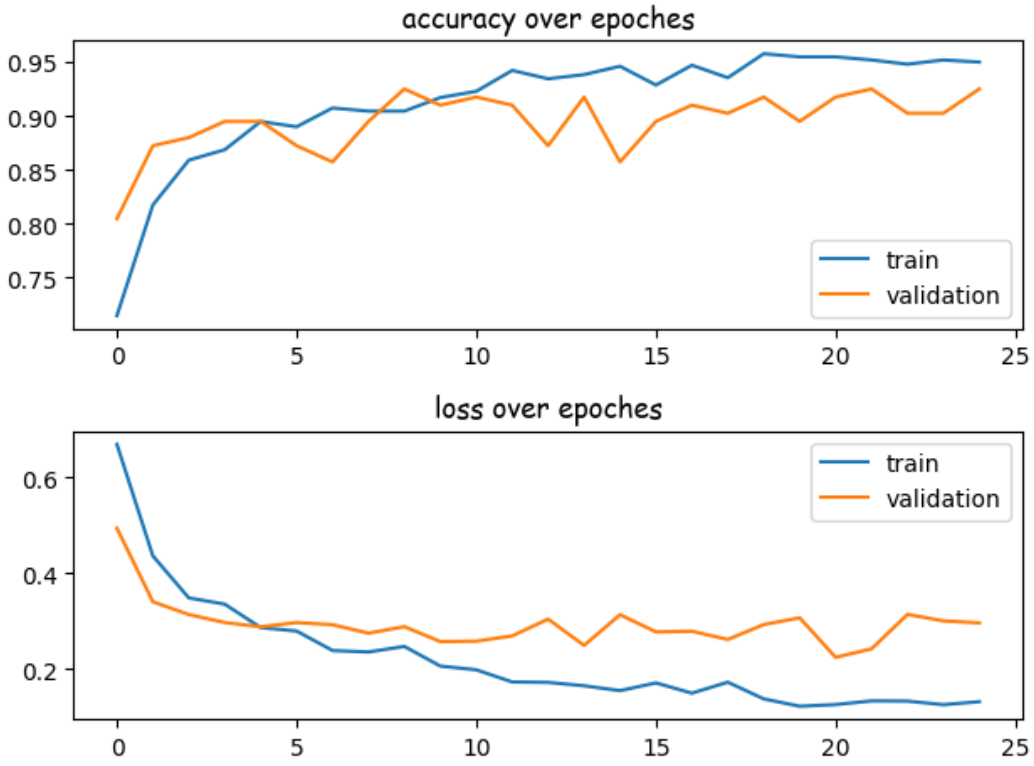
شکل 4. آموزش مدل **MobileNetV2** با بهینه‌ساز **Nadam**



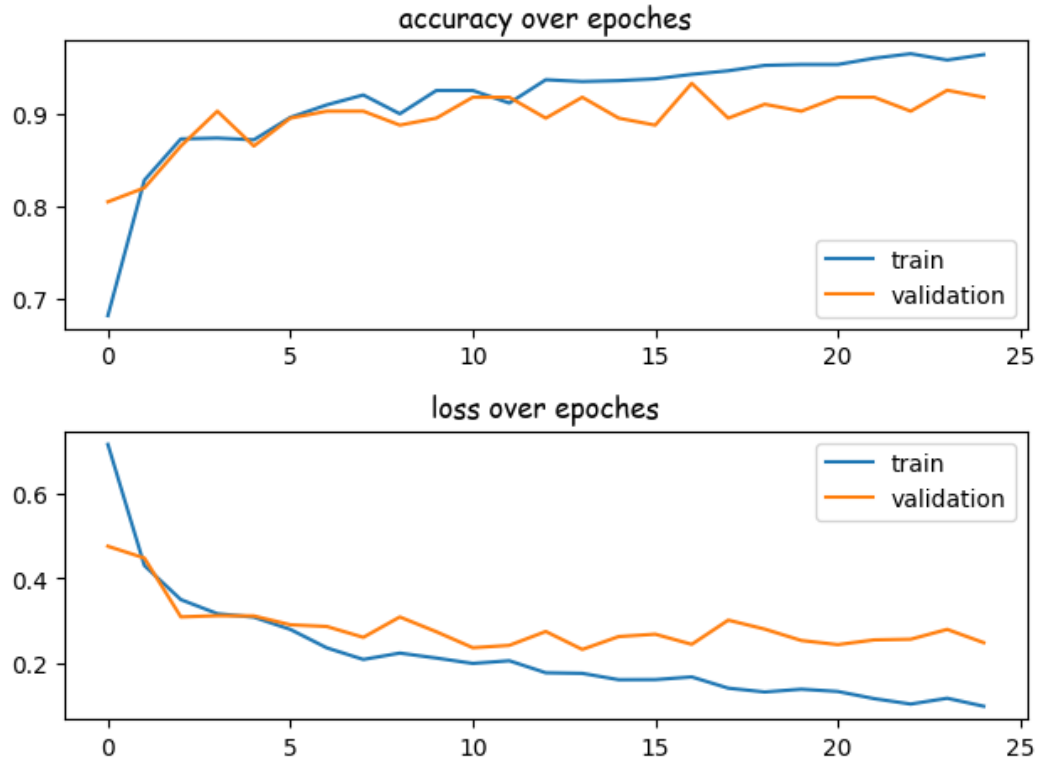
شکل 5. آموزش مدل **MobileNetV2** با بهینه‌ساز **RMSprop**

- **مدل EfficientNetB6**: این مدل با 3 بهینه‌ساز Adam, Nadam, RMSprop آموزش داده می‌شود، تابع دقت و هزینه آموزش با این 3 بهینه‌ساز در شکل 6، 7 و 8 آمده‌است.
  - طبق نتایج بدست آمده در طی 25 ایپاک، مشاهده می‌کنیم که در آموزش مدل با بهینه‌ساز Adam، مقدار دقت در ایپاک 25 ام روی داده train برابر با 94.42% و روی داده validation برابر با 92.48% می‌باشد. بهترین نتیجه نیز متعلق به ایپاک 22 ام می‌باشد که دقت روی داده train آن برابر با 95.25% و روی داده validation برابر با 92.48% می‌باشد.
  - طبق نتایج بدست آمده در طی 25 ایپاک، مشاهده می‌کنیم که در آموزش مدل با بهینه‌ساز Nadam، مقدار دقت در ایپاک 25 ام روی داده train برابر با 96.13% و روی داده validation برابر با 91.73% می‌باشد. بهترین نتیجه نیز متعلق به ایپاک 17 ام می‌باشد که دقت روی داده train آن برابر با 94.75% و روی داده validation برابر با 93.23% می‌باشد.
  - طبق نتایج بدست آمده در طی 25 ایپاک، مشاهده می‌کنیم که در آموزش مدل با بهینه‌ساز RMSprop، مقدار دقت در ایپاک 25 ام روی داده train برابر با 95.44% و

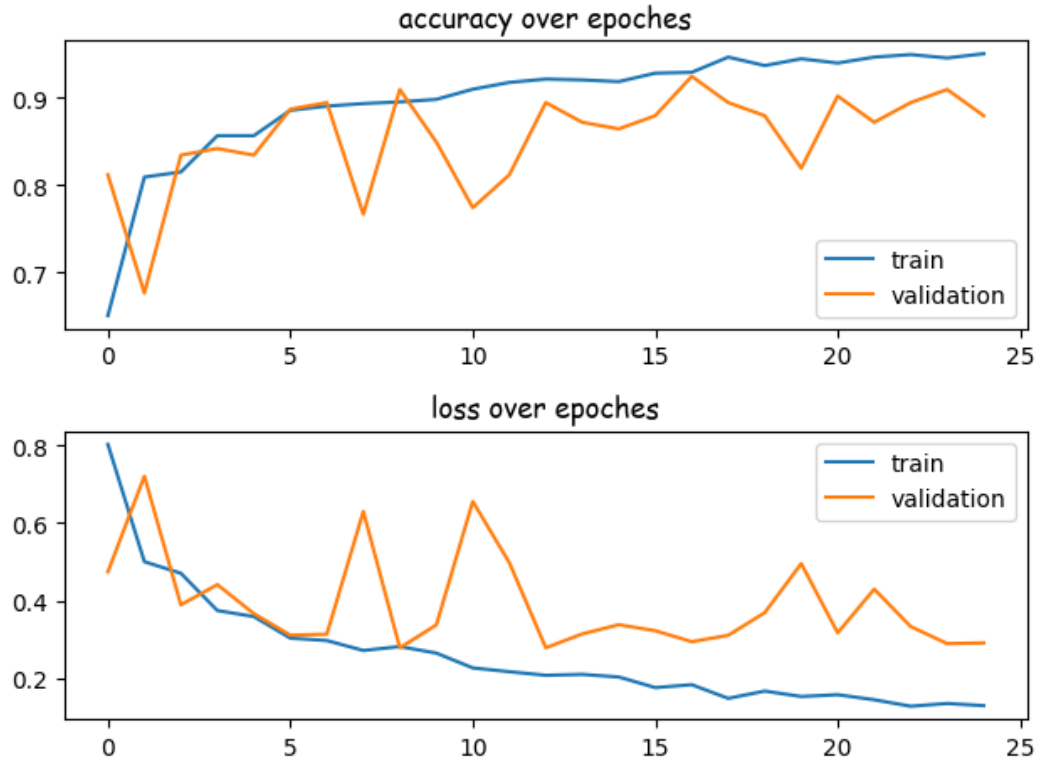
روی داده validation برابر با 87.97% می‌باشد. بهترین نتیجه نیز متعلق به ایپاک 17 ام می‌باشد که دقت روی داده train آن برابر با 93.67% و روی داده validation برابر با 92.48% می‌باشد.



شکل 6. آموزش مدل **EfficientNetB6** با بهینه‌ساز **Adam**

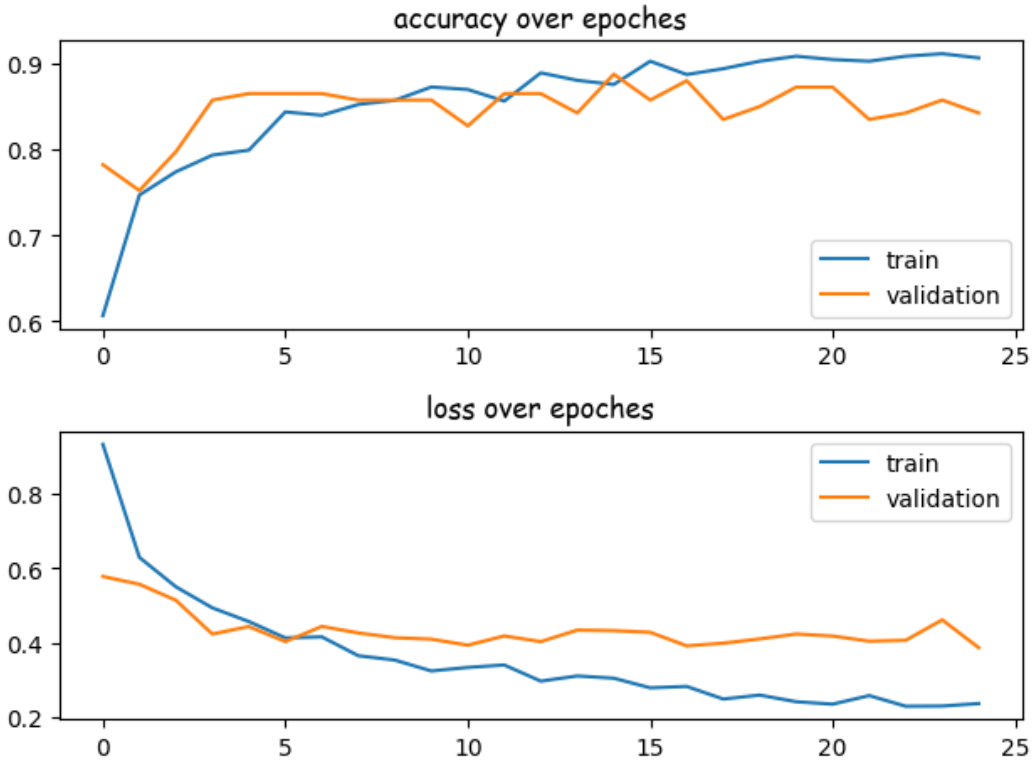


شکل 7. آموزش مدل **EfficientNetB6** با بهینه‌ساز **Nadam**

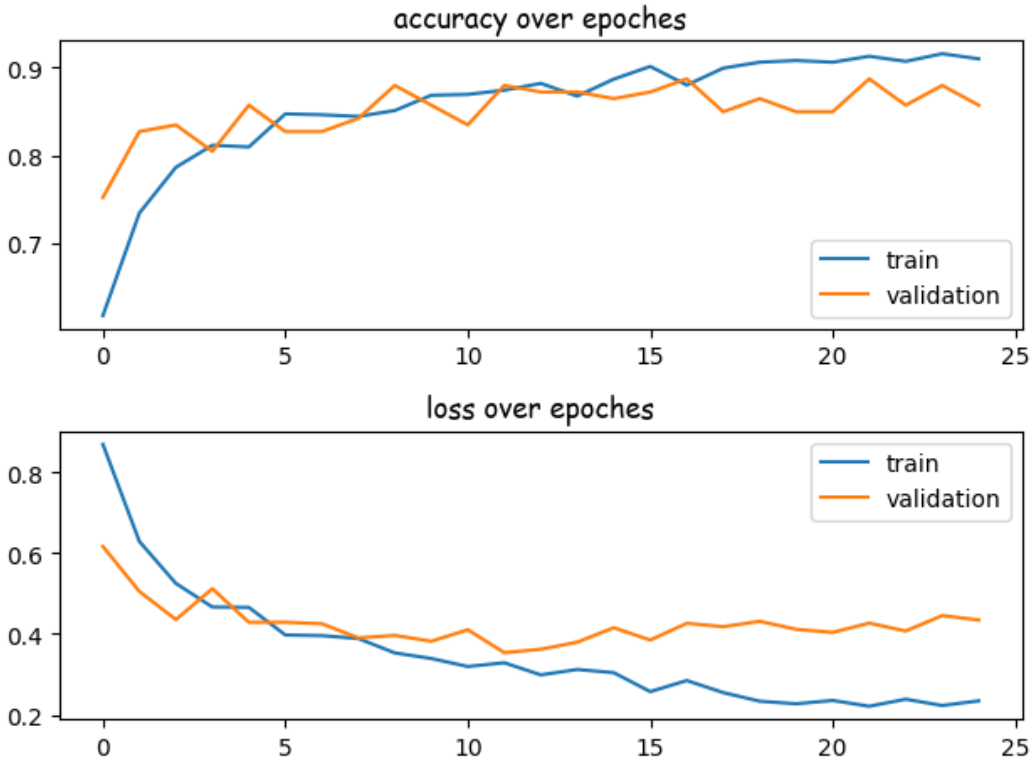


شکل 8. آموزش مدل **EfficientNetB6** با بهینه‌ساز **RMSprop**

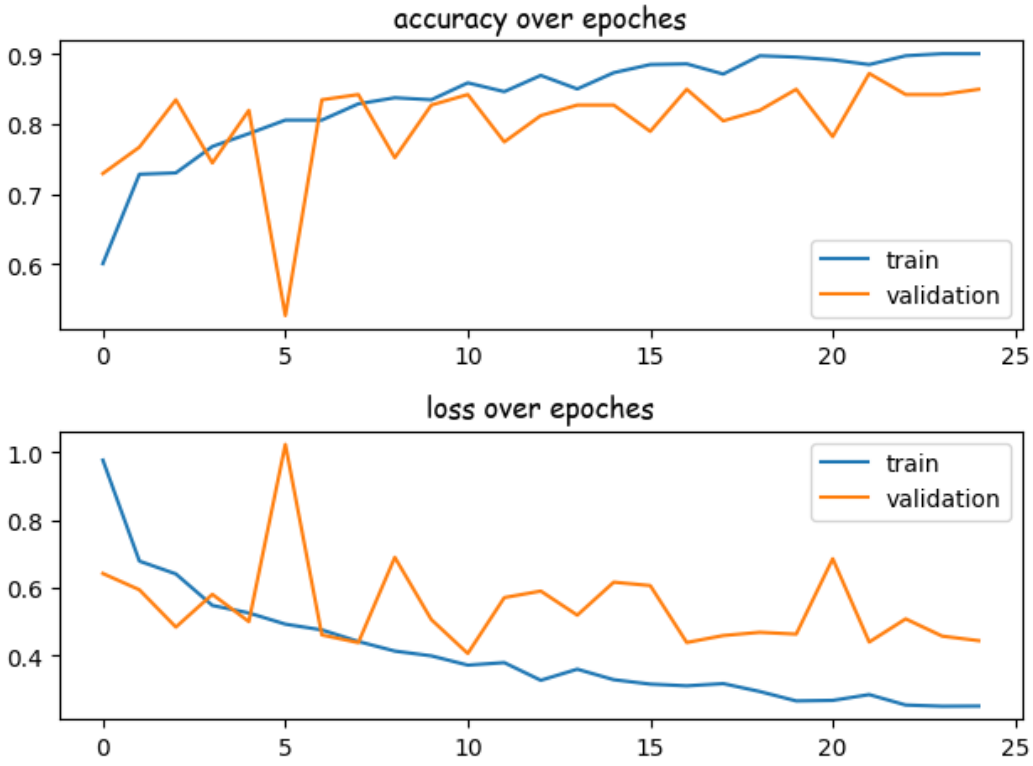
- **مدل NasNetMobile**: این مدل با 3 بهینه‌ساز Adam, Nadam, RMSprop آموزش داده می‌شود، تابع دقت و هزینه آموزش با این 3 بهینه‌ساز در شکل 9، 10 و 11 آمده‌است.
  - طبق نتایج بدست آمده در طی 25 ایپاک، مشاهده می‌کنیم که در آموزش مدل با بهینه‌ساز Adam، مقدار دقت در ایپاک 25 ام روی داده train برابر با 91.48% و روی داده validation برابر با 84.21% می‌باشد. بهترین نتیجه نیز متعلق به ایپاک 15 ام می‌باشد که دقت روی داده train آن برابر با 86.74% و روی داده validation برابر با 88.72% می‌باشد.
  - طبق نتایج بدست آمده در طی 25 ایپاک، مشاهده می‌کنیم که در آموزش مدل با بهینه‌ساز Nadam، مقدار دقت در ایپاک 25 ام روی داده train برابر با 90.89% و روی داده validation برابر با 85.71% می‌باشد. بهترین نتیجه نیز متعلق به ایپاک 22 ام می‌باشد که دقت روی داده train آن برابر با 91.91% و روی داده validation برابر با 88.72% می‌باشد.
  - طبق نتایج بدست آمده در طی 25 ایپاک، مشاهده می‌کنیم که در آموزش مدل با بهینه‌ساز RMSprop، مقدار دقت در ایپاک 25 ام روی داده train برابر با 90.00% و روی داده validation برابر با 84.96% می‌باشد. بهترین نتیجه نیز متعلق به ایپاک 22 ام می‌باشد که دقت روی داده train آن برابر با 87.94% و روی داده validation برابر با 87.22% می‌باشد.



شکل 9. آموزش مدل **NasNetMobile** با بهینه‌ساز **Adam**



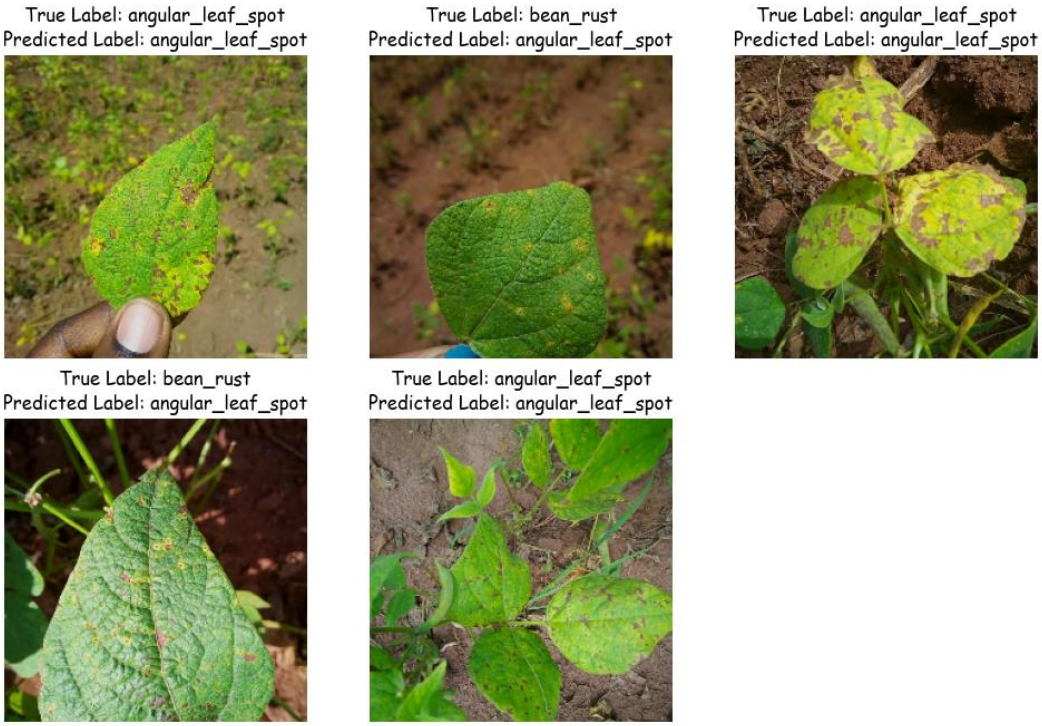
شکل 10. آموزش مدل **NasNetMobile** با بهینه‌ساز **Nadam**



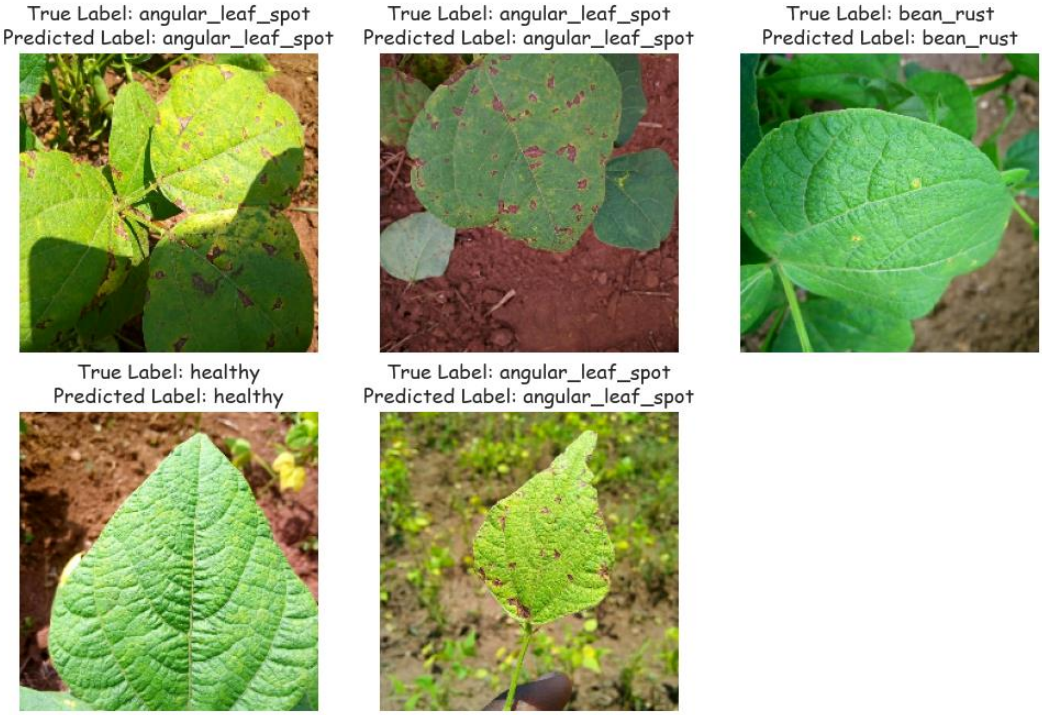
شکل 11. آموزش مدل **NasNetMobile** با بهینه‌ساز **RMSprop**

3-2. تحلیل نتایج

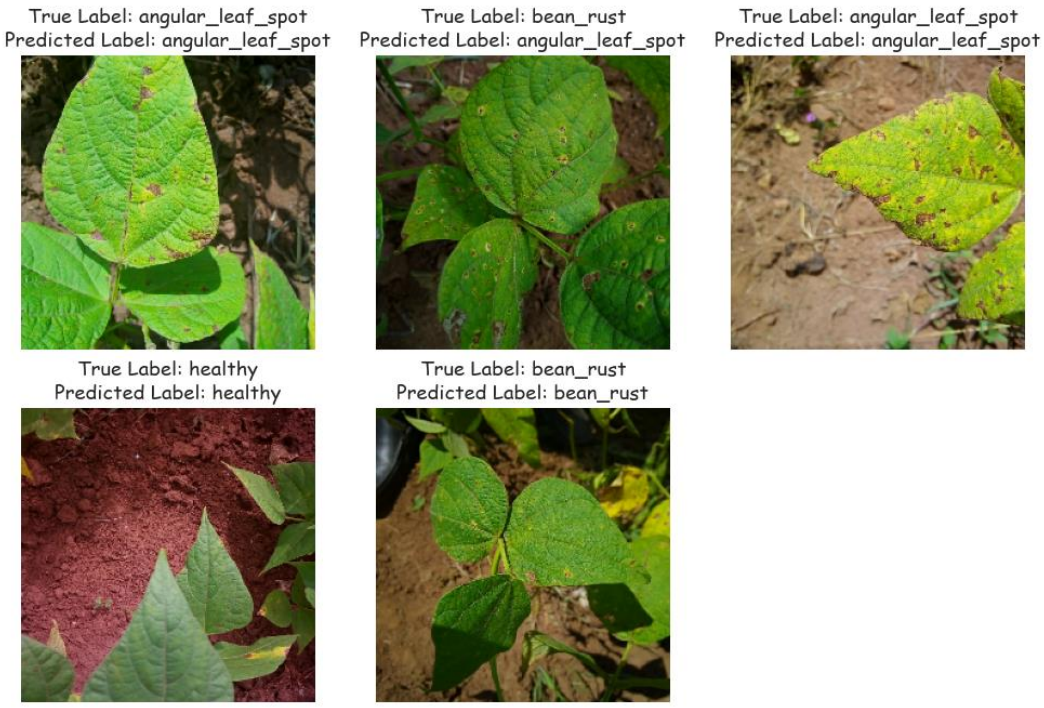
به دلیل این که در این 9 حالت، 3 مدل ما تعریف شده‌اند و تنها با بهینه‌سازهای متفاوتی کامپایل شده‌اند، از هر مدل، مدلی که با بهینه‌ساز RMSprop آموزش دیده‌اند را لود کرده و داده‌های تست را توسط آن مدل‌ها بررسی می‌کنیم. شکل 12، 13 و 14 نشان‌دهنده 5 تصویر تصادفی به همراه برچسب پیش‌بینی و برچسب اصلی آن ها برای این 3 مدل گفته‌شده را نشان می‌دهد.



شکل 12. **5** تصویر تصادفی از داده‌های تست به همراه برچسب آن‌ها برای مدل **MobileNetV2**



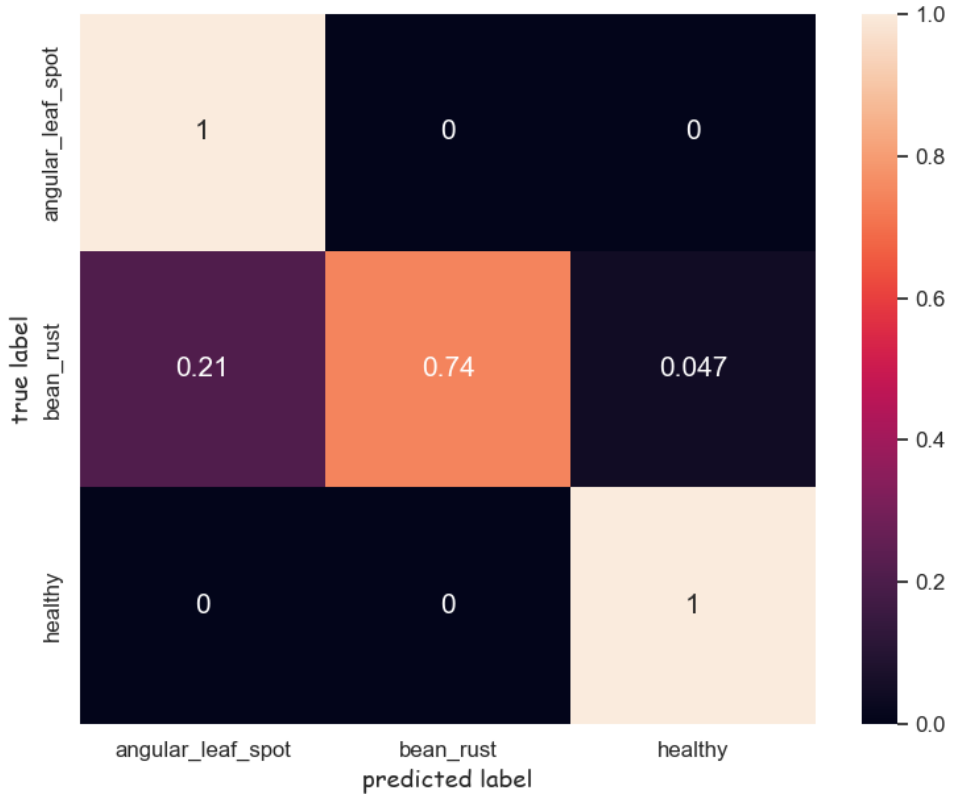
شکل 13. **5** تصویر تصادفی از داده‌های تست به همراه برچسب آن‌ها برای مدل **EfficienNetB6**



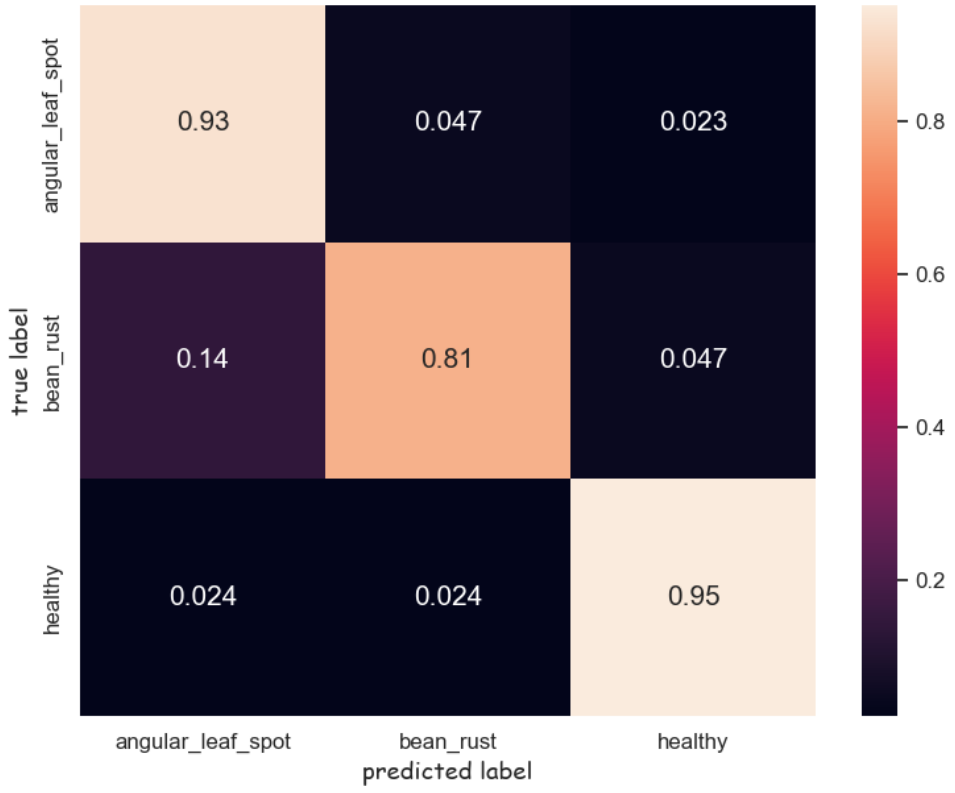
شکل 14. **5** تصویر تصادفی از داده‌های تست به همراه برچسب آن‌ها برای مدل **NasNetMobile**

برای این که متوجه شویم که مدل‌های ما در کدام دسته عملکرد بهتری داشته‌اند، کافی است confusion matrix هر مدل را بدست آوریم و دقت بدست آمده برای هر کلاس را بررسی کنیم.

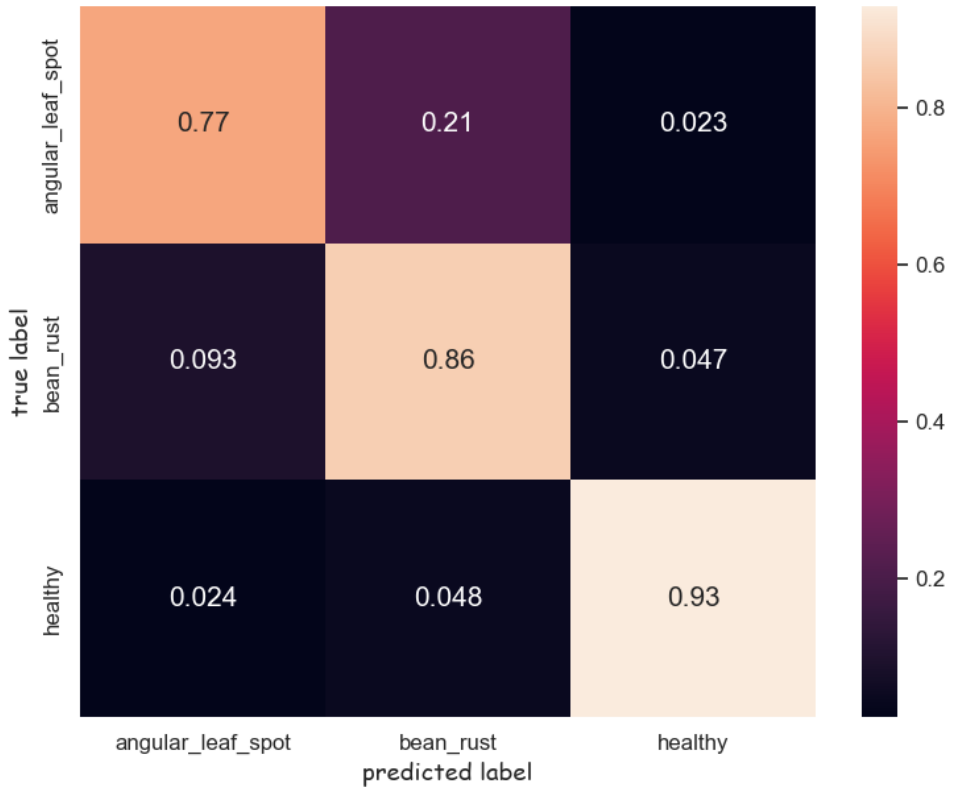
شکل 15، 16 و 17 این ماتریس‌ها را برای مدل‌ها با بهینه‌ساز RMSprop نشان می‌دهد.



شکل 15. **Confusion Matrix** متعلق به مدل **MobileNetV2** با بهینه‌ساز **RMSprop**



شکل 16. **Confusion Matrix** متعلق به مدل **EfficientNetB6** با بهینه‌ساز **RMSprop**



شکل 17. **Confusion Matrix** متعلق به مدل **NasNetMobile** با بهینه‌ساز **RMSprop**

با مقایسه این confusion matrix ها متوجه می‌شویم که مدل MobileNetV2 در تشخیص کلاس bean_rust مشکل دارد و درصد قابل توجهی از این کلاس را بخشی از کلاس angular_leaf_spot تشخیص می‌هد. به طور کلی می‌توان گفت که این مدل در تشخیص برگ های سالم فوق‌العاده عمل کرده (تمام آن ها را درست تشخیص داده) اما در تشخیص درست بیماری چندان مناسب نیست.

در رابطه با confusion matrix مدل EfficientNetB6 می‌توان گفت که این مدل عملکرد خوبی در تشخیص برگ‌های سالم دارد؛ همچنین، تشخیص قابل قبولی در مورد نوع بیماری دارد اما تشخیص بیماری bean_rust در آن ضعیفتر است و 14% از آن به اشتباه بیماری angular_leaf_spot شناسایی می‌شود.

در انتها نیز در مورد confusion matrix مدل NasNetMobile نیز می‌توان گفت که این مدل عملکرد خوبی در تشخیص برگ‌های سالم دارد؛ همچنین، تشخیص قابل قبولی در مورد نوع بیماری دارد اما تشخیص بیماری angular_leaf_spot در آن ضعیفتر است و 21% از آن به اشتباه بیماری bean_rust شناسایی می‌شود.

در انتها، برای مقایسه نتایج خود با نتایج مقاله، جدولی مشابه با جدول شماره 4 مقاله رسم می‌کنیم و نتایج بدست آمده را با نتایج بدست آمده از مقاله مقایسه میکنیم. جدول 1 نتایج بدست آمده توسط مدل پیاده‌سازی شده از مقاله و جدول 2 نتایج بدست آمده توسط مقاله را نشان می‌دهد.

جدول 1. لیست نتایج بدست آمده

Optimizer	*CNN Model*	*Tr-Acc*	*Val-Acc*	*Tr-loss*	*Val-loss*
Adam	MobileNetV2	98.11	93.23	0.0701	0.1868
	EfficientNetB6	95.25	92.48	0.1426	0.2424
	NasNetMobile	86.74	88.72	0.3199	0.4319
Nadam	MobileNetV2	97.69	93.98	0.0594	0.1999
	EfficientNetB6	94.75	93.23	0.1528	0.2435
	NasNetMobile	91.91	88.72	0.2608	0.4267
RMSprop	MobileNetV2	94.75	93.23	0.1126	0.2082
	EfficientNetB6	93.67	92.48	0.1700	0.2938
	NasNetMobile	87.94	87.22	0.3354	0.4400

جدول 2. لیست نتایج ادعا شده توسط مقاله

Optimizer	*CNN Model*	*Tr-Acc*	*Val-Acc*	*Tr-loss*	*Val-loss*
Adam	MobileNetV2	94.39	91.72	0.1489	0.2110
	EfficientNetB6	96.62	91.74	0.0936	0.2489
	NasNetMobile	84.14	86.47	0.3749	0.3224
Nadam	MobileNetV2	94.78	91.73	0.1450	0.2055
	EfficientNetB6	94.39	88.72	0.1864	0.2634
	NasNetMobile	85.59	86.45	0.3697	0.3349
RMSprop	MobileNetV2	94.78	91.72	0.1343	0.1888
	EfficientNetB6	94.87	88.72	0.1573	0.2492
	NasNetMobile	85.49	84.21	0.3579	0.3262