

به نام خدا



دانشگاه تهران



دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین ۵۹۵

آرمنین قاسمی	نام و نام خانوادگی	پرسش ۱
810100198	شماره دانشجویی	
امیرحسین شمودی	نام و نام خانوادگی	پرسش ۲
810100108	شماره دانشجویی	
بهار ۱۴۰۴	مهلت ارسال پاسخ	

فهرست

پروژه ۱. تشخیص بیماران مبتلا به COVID-19 با استفاده از تصاویر X-Ray	۵
۱-۱. تحلیل دیتاست	۵
۲-۱. پیش پردازش داده ها	۹
۳-۱. آماده سازی مدل	۱۴
۴-۱. آموزش و ارزیابی مدل	۱۷
۵-۱. یادگیری انتقالی	۲۲
پروژه ۲ - پیاده سازی یک سیستم طبقه بندی خودرو با استفاده از SVM و VGG16	۳۲
۱-۲. مقدمه	۳۲
۲-۲. پیش پردازش داده ها	۳۲
انتخاب ۱۰ کلاس:	۳۲
بالанс کردن دیتا ست:	۳۳
۳-۲. استخراج ویژگی ها	۳۵
۴-۲. آموزش و ارزیابی مدل	۳۵
۱- تمرین مدل های بخش قبل	۳۵
۲- تفاوت VGG Net و Alex Net	۴۰
۳- مدل CNN	۴۰
۴- مدل ترکیبی SVM و VGG	۴۲
۵- تحلیل نتایج	۴۵
۶- امتیازی	۴۷
۱- استفاده از کرنل RBF	۴۷
۲- استخراج ویژگی ها	۴۸

شکل‌ها

- 5..... شکل 1: تقسیم بندی داده‌ها
- 7..... شکل 2: هیستوگرام تعداد داده‌های هر کلاس، برای هر دو مجموعه Train و Test
- 9..... شکل 3: هیستوگرام تعداد داده‌های هر کلاس، برای هر دو مجموعه Train و Test پس از بالانس شدن
- 10..... شکل 4: تصویر اصلی
- 11..... شکل 5: تصویر تغییر یافته اول
- 11..... شکل 6: تصویر تغییر یافته دوم
- 12..... شکل 7: تصویر تغییر یافته سوم
- 12..... شکل 8: تصویر تغییر یافته چهارم
- 13..... شکل 9: تصویر تغییر یافته پنجم
- 13..... شکل 10: تصویر تغییر یافته ششم
- 14..... شکل 11: شکل مورد نظر داخل مقاله (شکل 4)
- 17..... شکل 12: نمودارهای مربوط به یادگیری مدل
- 21..... شکل 13: matrix confusion مربوط به مدل اول
- 23..... شکل 14: نمودارهای مربوط به یادگیری مدل VGG16
- 26..... شکل 15: confusion matrix مربوط به مدل VGG16
- 27..... شکل 16: نمودارهای مربوط به یادگیری مدل MobileNet
- 30..... شکل 17: confusion matrix مربوط به مدل MobileNet
- 32..... شکل 18: نمودار فراوانی آماری کلاس‌های دیتا است
- 33..... شکل 19: توزیع آماری کلاس‌های انتخاب شده قبل از بالانس کردن
- 34..... شکل 20: نحوه augment کردن داده‌ها
- 34..... شکل 21: توزیع آماری کلاس‌های انتخاب شده بعد از بالانس کردن
- 35..... شکل 22: گزارش کلاس‌ها و ابعاد دسته‌های آموزشی و تست
- 36..... شکل 23: تابع خطا مدل VGG بر حسب epoch برای داده‌های train و Validation
- 37..... شکل 24: نمودار دقت مدل VGG روی داده‌های Validation
- 37..... شکل 25: پارامترهای ارزیابی مدل طبقه‌بند VGG
- 39..... شکل 26: تابع خطا مدل AlexNet بر حسب epoch برای داده‌های train و Validation
- 39..... شکل 27: نمودار دقت مدل Alex Net روی داده‌های Validation

- 40..... شکل 28: پارامتر های ارزیابی مدل طبقه بند Alex Net
- 41..... شکل 29: تابع خطا مدل CNN بر حسب epoch برای داده های train و Validation
- 41..... شکل 30: نمودار دقیقیت مدل CNN روی داده های Validation
- 42..... شکل 31: پارامتر های ارزیابی مدل طبقه بند CNN
- 42..... شکل 32: پارامتر های ارزیابی مدل طبقه بند SVM
- 42..... شکل 33: پارامتر های ارزیابی مدل طبقه بند SVM به ازای هر کلاس
- 43..... شکل 34: ماتریس آشفتگی مدل VGG-16
- 43..... شکل 35: ماتریس آشفتگی مدل Alex Net
- 44..... شکل 36: ماتریس آشفتگی مدل CNN
- 44..... شکل 37: ماتریس آشفتگی مدل VGG+SVM
- 46..... شکل 38: مقایسه عملکرد 4 مدل
- 47..... شکل 39: پارامتر های ارزیابی مدل طبقه بند SVM با کرنل RBF
- 48..... شکل 40: ماتریس آشفتگی مدل VGG+SVM با کرنل RBF

جدول‌ها

- 5..... جدول 1: بررسی فرمت و مشخصات داده ها
- 14..... جدول 2: جدول مورد نظر داخل مقاله (جدول 3)
- 15..... جدول 3: Summay model
- 45..... جدول 4: مقایسه عملکرد مدل ها

پرسش 1. تشخیص بیماران مبتلا به COVID-19 با استفاده از تصاویر X

Ray

۱-۱. تحلیل دیتاست

با توجه به پوشه هایی که درون دیتاست وجود دارد متوجه میشویم که داده ها از قبل به دو بخش تست و تربین تقسیم شده اند و همچنین ۳ کلاس دارند:

COID-19 -1
NORMAL -2
(ذات الریه)PNEUMONIA -3

DATASETS

- ▼ chest-xray-covid19-pneumonia
 - ▼ Data
 - ▼ test
 - ▶ COVID19
 - ▶ NORMAL
 - ▶ PNEUMONIA
 - ▼ train
 - ▶ COVID19
 - ▶ NORMAL
 - ▶ PNEUMONIA

شکل 1: تقسیم بندی داده ها

برای پیدا کردن فرمت و دیگر مشخصات تصاویر میتوانیم چند تصویر از هر کلاس را به صورت رندوم باز کنیم و مشخصات آنها را بدست آوریم. این مشخصات در شکل 2 آورده شده است.

با توجه به این شکل تصاویر اندازه های مختلف دارند. همچنین نکته جالب توجه دیگر این است که برخی تصاویر رنگی هستند و از سه کanal رنگی تشکیل شده اند و برخی سیاه و سفید هستند و صرفا از یک کanal رنگی تشکیل شده اند. همه این موارد باید در پیاده سازی مدل مورد توجه قرار بگیرد.

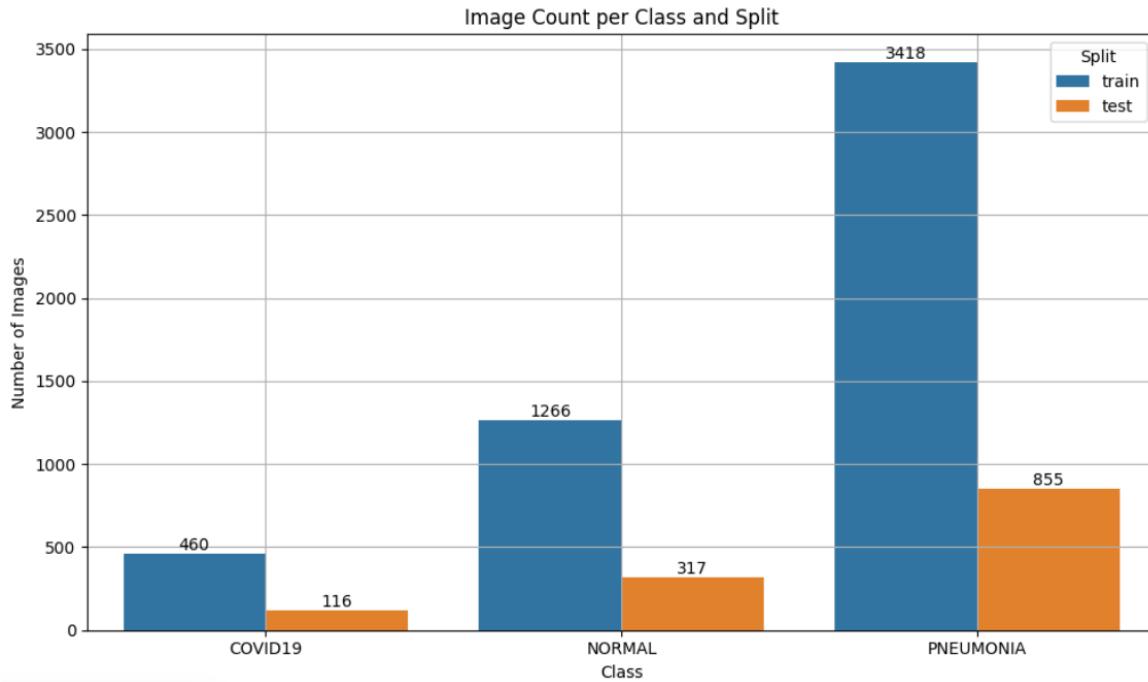
جدول 1: بررسی فرمت و مشخصات داده ها

Split	Class	Filename	Format	Size	Mode
0	train	COVID19	COVID19(189).jpg	JPEG (2000, 2000)	L
1	train	COVID19	COVID19(65).jpg	JPEG (2000, 2000)	L
2	train	COVID19	COVID19(239).jpg	JPEG (880, 891)	RGB
3	train	NORMAL	NORMAL(342).jpg	JPEG (1472, 1013)	RGB
4	train	NORMAL	NORMAL(545).jpg	JPEG (1738, 1442)	RGB
5	train	NORMAL	NORMAL(1103).jpg	JPEG (1488, 1337)	RGB
6	train	PNEUMONIA	PNEUMONIA(3189).jpg	JPEG (1184, 760)	RGB
7	train	PNEUMONIA	PNEUMONIA(2169).jpg	JPEG (1192, 736)	RGB
8	train	PNEUMONIA	PNEUMONIA(969).jpg	JPEG (1576, 1288)	RGB
9	test	COVID19	COVID19(563).jpg	JPEG (474, 397)	RGB
10	test	COVID19	COVID19(566).jpg	JPEG (1118, 1333)	RGB
11	test	COVID19	COVID19(531).jpg	JPEG (425, 448)	RGB
12	test	NORMAL	NORMAL(1312).jpg	JPEG (2338, 2430)	RGB
13	test	NORMAL	NORMAL(1335).jpg	JPEG (1673, 1291)	RGB
14	test	NORMAL	NORMAL(1578).jpg	JPEG (1288, 928)	RGB
15	test	PNEUMONIA	PNEUMONIA(3607).jpg	JPEG (1016, 656)	RGB
16	test	PNEUMONIA	PNEUMONIA(3961).jpg	JPEG (1616, 1288)	RGB
17	test	PNEUMONIA	PNEUMONIA(3958).jpg	JPEG (1008, 768)	RGB

در ادامه هیستوگرام داده های هر کلاس را برای دو مجموعه داده train و test رسم کرده (شکل 2) و توزیع داده ها در هر کلاس را بررسی می کنیم. به توجه به نمودار شکل 2 تعداد داده های مربوط به کلاس NORMAL و COVID19 از تعداد داده های مربوط به کلاس PNEUMONIA بسیار کمتر است. این عدم تعادل باعث می شود شبکه CNN کلاس COVID19 و NORMAL را درست یادنگیرد و نتواند به درستی ویژگی های مربوط به تصاویر این کلاس ها را استخراج کند. بدین ترتیب مدل روی کلاس

PNEUMONIA بایاس می شود. علاوه بر این احتمالاً مدل روی همین تعداد کم داده مربوط به این کلاس ها، overfit می شود و قدرت تعمیم پذیری خود را از دست می دهد.

ولی اگر تعداد داده های هر کلاس تقریباً برابر باشد مدل فرصت برابر برای یادگیری ویژگی های هر کلاس دارد و می تواند واقعاً بین کلاس ها تمایز ایجاد کند. همچنین accuracy نهایی بر عکس مدل هایی که با داده های نامتعادل آموزش می بینند، واقعی خواهد بود و مدل اگر با متريک های ديگر هم سنجideh شود نتیجه خوبی خواهد داشت.



شکل 2: هیستوگرام تعداد داده های هر کلاس، برای هر دو مجموعه Train و Test

برای حل کردن مشکل عدم تعادل داده های کلاس ها می توانیم از راه ها و ابزار های زیر استفاده کنیم:

Oversampling - 1 : در این روش داده های کلاس کمتر را تکرا می کنیم یا data Augmentation داده های بیشتری از روی آنها تولید می کنیم تا به تعادل برسیم. برای این کار می توان از کتابخانه ها و ابزار های زیر استفاده کرد:

Keras در ImageDataGenerator -

pytorch در torchvision.transforms -

برای داده های جدولی imbalanced-learn -

Undersampling - 2 : در این روش تعداد داده های کلاس هایی که داده زیادی دارند را کاهش می دهیم تا با تعداد داده های کلاس هایی کم داده برابر شوند. برای این کار می توان از ابزار های زیر استفاده کرد:

sklearn.utils.resample -

pandas.sample -

در loss function **Class Weights -3** مورد استفاده وزن بیشتری برای کلاس هایی که داده های

کمتری دارند درنظر میگیریم تا شبکه روی آنها حساس تر شود. ابزار ها:

keras در class_weight -

PyTorch در CrossEntropyLoss(weight=...) -

sklearn در compute_class_weight -

می توانیم برای کلاس هایی که داده های کمی دارند با انجام یکسری **Data Augmentation -4**

تغییرات مانند چرخش، آینه کردن، شیفت دادن و ... نمونه ای بیشتری تولیدی کنیم. در واقع به

نحوی می توان گفت Data Augmentation زیر مجموعه ای از oversampling است. ابزار ها و

کتابخانه های مورد استفاده:

Keras در ImageDataGenerator -

pytorch در torchvision.transforms -

albumentations -

در این روش ها به نمونه هایی که مدل سخت تر تشخیص میدهد وزن بیشتری **Focal Loss -5**

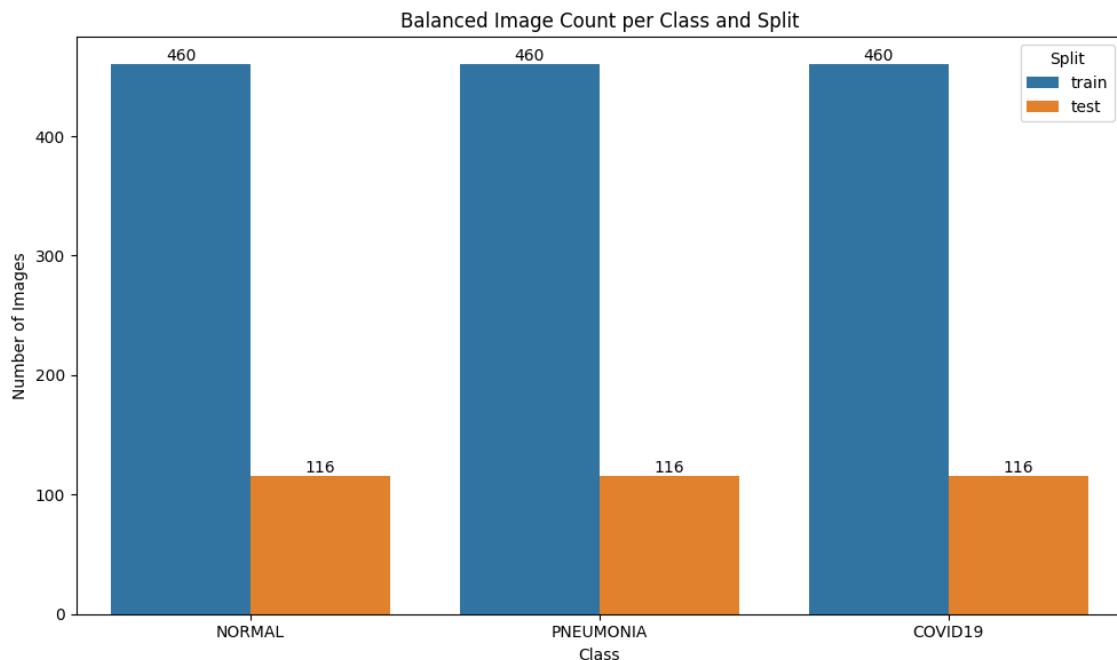
داده میشود. ابزار ها:

tensorflow-addons -

pytorch-focal-loss -

2- پیش پردازش داده ها

طبق چیزی که در صورت سوال گفته شده باید ابتدا undersampling انجام دهیم. با توجه به شکل 2 در مجموعه داده های تست باید از تمام کلاس ها 116 داده و در مجموعه داده های ترین باید از تمام کلاس ها 460 داده وجود داشته باشد. به صورت تصادفی این تعداد از داده های را از هر کلاس انتخاب می کنیم و مجددا هیستوگرام را رسم می کنیم.



شکل 3: هیستوگرام تعداد داده های هر کلاس، برای هر دو مجموعه **Test** و **Train** پس از بالانس شدن

پس از یکسان کردن تعداد داده های هر کلاس می خواهیم data augmentation را روی داده های خود پیاده سازی کنیم. data augmentation یعنی از روی داده ها و تصاویری که داریم، با استفاده از یکسری تغییرات کنترل شده (مثلاً اضافه کردن نویز، چرخاندن تصویر، کم و زیاد کردن نور، زوم کردن یا جا به جا کردن تصویر و....) داده ها و تصاویر جدید و متنوع تولید کنیم. در مدل هایی مانند CNN با augmentation مدل را با داده های بیشتری آموزش دهیم و همچنین مدل نهایی به تغییرات جزئی این چنینی مقاوم خواهد بود.

با استفاده از Albumentations داده های خود را به 6 برابر افزایش می دهیم. به بخشی از کد مورد استفاده برای این بخش توجه کنید:

```
transform = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.Rotate(limit=10, p=0.5),
    A.GaussianBlur(blur_limit=3, p=0.1),
    A.ShiftScaleRotate(shift_limit=0.02, scale_limit=0.1, rotate_limit=0, p=0.5),
])
```

در این بخش از کد یک توالی از تغییرات با احتمال های مختلف رو تصویر ورودی اعمال می شود:

- چرخاندن تصویر به صورت افقی از چپ به راست با احتمال 50% : در بعضی موارد در تصویر برداری های رادیولوژی قرینه سازی افقی منطقی است.
- چرخاندن تصاویر به صورت تصادفی تا 10 درجه با احتمال 50% : این مورد می تواند مثلا کج خوابیدن بیمار را توجیه کند.
- ایجاد مقدار کمی تاری روی تصویر با احتمال 10% : با توجه به اینکه تصاویر پزشکی ممکن است با تکنولوژی های مختلف ثبت شده باشند، این مورد می تواند مفید باشد ولی چون این مورد ممکن است اطلاعات تصویر را از بین ببرد با احتمال کم اجرا شده.
- جابه جایی تا 2% تصویر و scale کردن تصویر تا 10% با احتمال 10% : این کار به مدل کمک می کند نسبت به تغییرات جزئی موقعیت و اندازه مقاوم باشد.

یک نمونه از داده ها همراه با نمونه های تولید شده از روی آن در شکل های 4 تا 10 در صفحه بعدی آورده شده است.



شکل 4: تصویر اصلی



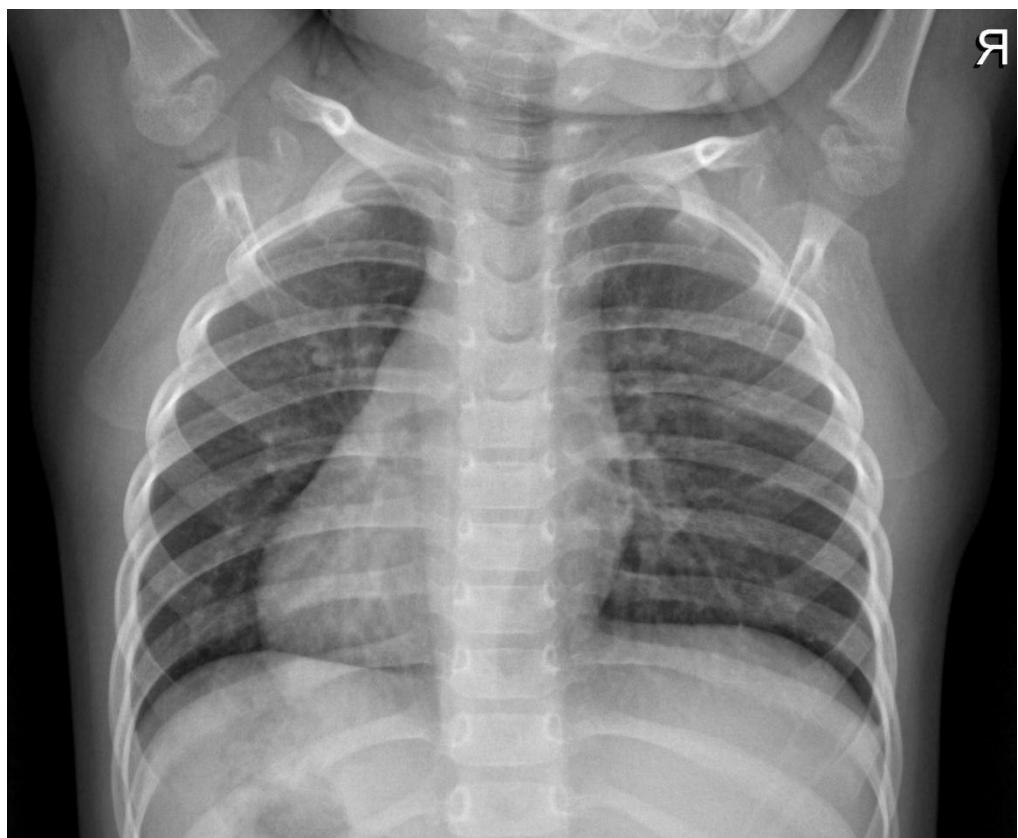
شكل 5: تصویر تغییر یافته اول



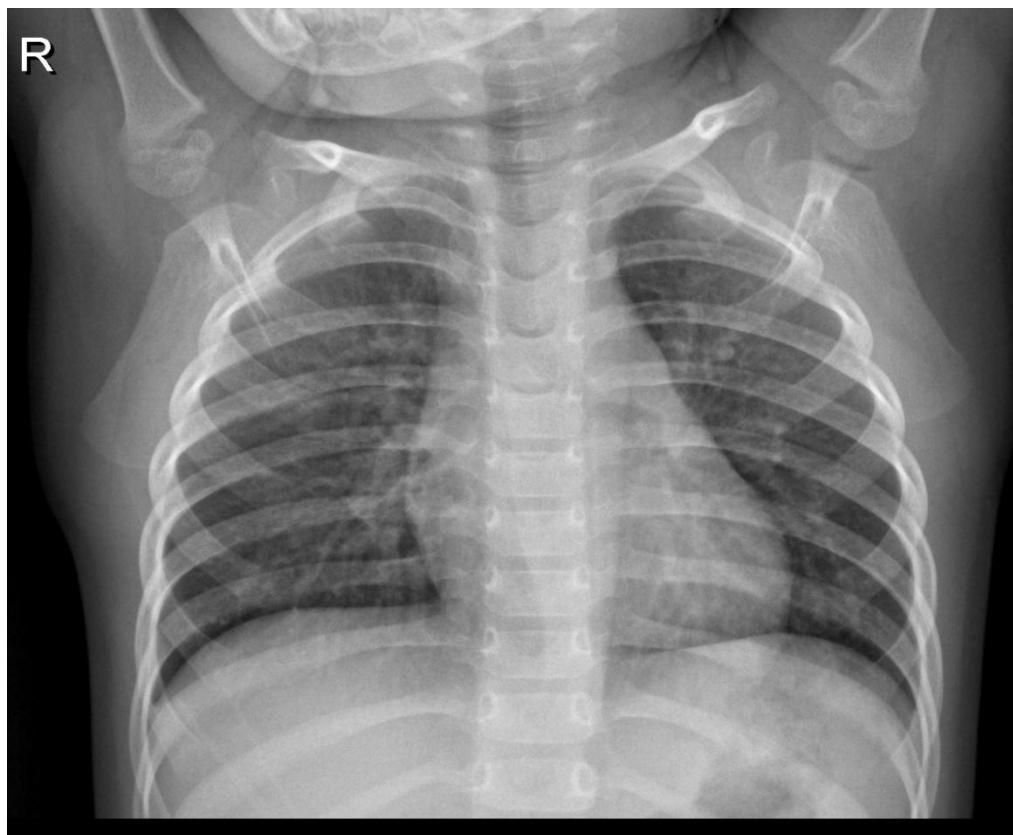
شكل 6: تصویر تغییر یافته دوم



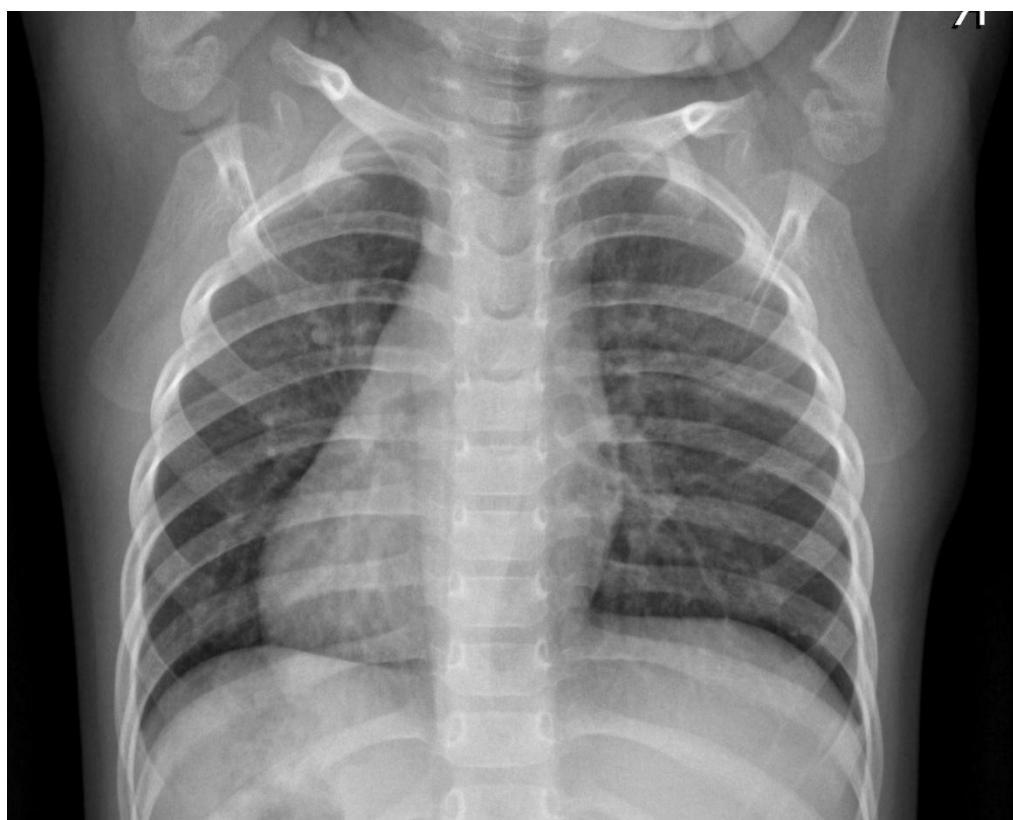
شكل 7: تصویر تغییر یافته سوم



شكل 8: تصویر تغییر یافته چهارم



شكل 9: تصویر تغییر یافته پنجم



شكل 10: تصویر تغییر یافته ششم

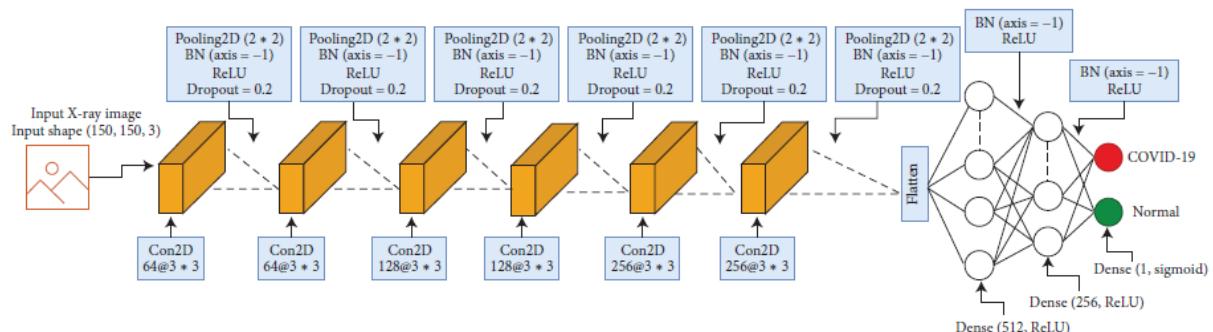
3- آماده سازی مدل

مطابق صورت مقاله مدل CNN را پیاده سازی و کامپایل میکنیم. همچنین با توجه به اینکه مدل CNN مورد نظر تصاویر را سه کاناله و در ابعاد 150×150 دریافت میکند، ما نیز تصاویر را به این اندازه resize میکنیم. همچنین خروجی شبکه را نیز با سه نرون و به صورت one hot نمایش میدهیم. شبکه مورد نظر در صفحه 12 بعد آورده شده است.

با توجه به اینکه در مقاله از learning rate متغیر استفاده شده است ما نیز از learning rate متغیر استفاده می کنیم. استفاده از learning rate کوچک همگرایی را کند می کند و همچنین ممکن است باعث شود مدل در یک local minimum گیر کند و پارامتر های بهتر را پیدا نکند. Learning rate بزرگ نیز ممکن است باعث شود مدل نوسان کند و هیچگاه به minimum نرسد. اما learning rate متغیر اینگونه عمل می کند که ابتدا با یک مقدار بزرگ شروع می کند و تا جایی که loss روی داده های validation رو به کاهش است با همان learning rate ادامه می دهد. ولی اگه در چند گام loss روی داده های validation کند است با همان learning rate شروع به کم شدن میکند تا به minimum برسد.

جدول 2: جدول مورد نظر داخل مقاله (جدول 3)

Parameter	Value
Input dimension	(150, 150, 3)
Filter to learn	64, 128, 256
Max pooling	2×2
Batch normalization	Axis = -1
Activation functions	ReLU, sigmoid
Dropout rate	20%
Kernel size	3×3
Epochs	50
Optimizer	Adam
Loss function	binary_crossentropy



شکل 11: شکل مورد نظر داخل مقاله (شکل 4)

با توجه به تصاویر بالا و summary مدل که در صفحه بعد آورده شده است می توان دید که مدل کاملاً منطبق بر مقاله است. نکته قابل توجه در مقاله این است که رفته رفته تعداد feature map ها افزایش ولی ابعاد تصویر با توجه به لایه های pooling کاهش می یابند. در تمامی لایه ها از dropout استفاده شده است و تابع غیر خطی تمام لایه ها ReLU است. همچنین در هر لایه از batch normalization استفاده شده است. Pooling های استفاده شده فیلتر های 2×2 هستند که یعنی از هر 4 پیکسل بزرگترین را انتخاب می کنند. تمامی kernel های استفاده شده 3×3 هستند ولی تعداد feature map ها در دو لایه اول 64، دو لایه بعدی 128 و در دو لایه آخر 256 است. تعداد ایپاک های مورد نظر مقاله 50 ایپاک است که ما با توجه به محدودیت های سخت افزاری و زمانی این مقدار را برابر با 30 قرار میدهیم ولی early stop نیز برای مدل قرار می دهیم که اگر مدل از جایی به بعد دیگر یاد نگرفت فرایند یادگیری متوقف شود. این مورد در کنار learning rate متغیر می تواند ترکیب جالبی باشد. در واقع در طی فرایند آموزش اگر به نوسان بیوفتیم ابتدا learning rate را کاهش می دهیم تا از حالت نوسانی خارج شویم ولی اگر پس از آن مجدداً نوسان کنیم یعنی واقعاً فرایند آموزش مدل پایان یافته است و باید یادگیری را متوقف کنیم. همچنین برای مدل checkpoint قرار می دهیم که بهترین مدلی که تا الان آموزش داده ایم را با توجه به دقت روی داده های validation ذخیره کنیم.

جدول 3: Summay model

Model:		"sequential"
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 150, 150, 64)	1,792
batch_normalization (BatchNormalization)	(None, 150, 150, 64)	256
activation (Activation)	(None, 150, 150, 64)	0
max_pooling2d (MaxPooling2D)	(None, 75, 75, 64)	0
dropout (Dropout)	(None, 75, 75, 64)	0
conv2d_1 (Conv2D)	(None, 75, 75, 64)	36,928
batch_normalization_1 (BatchNormalization)	(None, 75, 75, 64)	256
activation_1 (Activation)	(None, 75, 75, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 37, 37, 64)	0
dropout_1 (Dropout)	(None, 37, 37, 64)	0
conv2d_2 (Conv2D)	(None, 37, 37, 128)	73,856

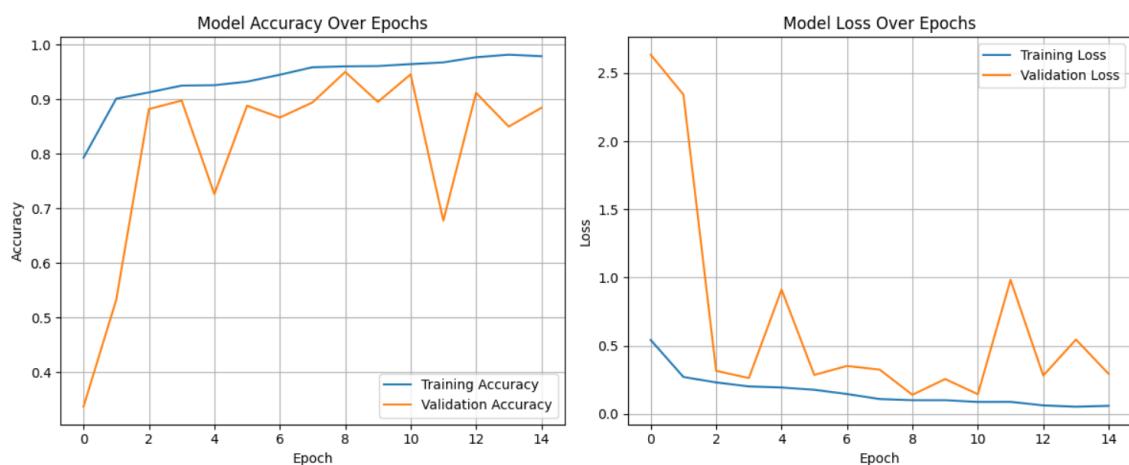
batch_normalization_2 (BatchNormalization)	(None, 37, 37, 128)	512
activation_2 (Activation)	(None, 37, 37, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 18, 18, 128)	0
dropout_2 (Dropout)	(None, 18, 18, 128)	0
conv2d_3 (Conv2D)	(None, 18, 18, 128)	147,584
batch_normalization_3 (BatchNormalization)	(None, 18, 18, 128)	512
activation_3 (Activation)	(None, 18, 18, 128)	0
max_pooling2d_3 (MaxPooling2D)	(None, 9, 9, 128)	0
dropout_3 (Dropout)	(None, 9, 9, 128)	0
conv2d_4 (Conv2D)	(None, 9, 9, 256)	295,168
batch_normalization_4 (BatchNormalization)	(None, 9, 9, 256)	1,024
activation_4 (Activation)	(None, 9, 9, 256)	0
max_pooling2d_4 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_4 (Dropout)	(None, 4, 4, 256)	0
conv2d_5 (Conv2D)	(None, 4, 4, 256)	590,080
batch_normalization_5 (BatchNormalization)	(None, 4, 4, 256)	1,024
activation_5 (Activation)	(None, 4, 4, 256)	0
max_pooling2d_5 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_5 (Dropout)	(None, 2, 2, 256)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 512)	524,800
dense_1 (Dense)	(None, 256)	131,328
dense_2 (Dense)	(None, 3)	771

Total params: 1,805,891 (6.89 MB)
 Trainable params: 1,804,099 (6.88 MB)
 Non-trainable params: 1,792 (7.00 KB)

4-1. آموزش و ارزیابی مدل

با توجه به دستور کار، مدل را تا 30 ایپاک آموزش میدهیم ولی با توجه به early stop احتمالاً یادگیری مدل زودتر تمام خواهد شد. از 35 درصد داده ها برای اعتبارسنجی استفاده میکنیم و learning rate را متغیر قرار میدهیم که در بخش قبلی از آن صحبت شد.

مدل را آموزش میدهیم. با توجه به early stop آموزش مدل روی ایپاک 15 تمام میشود. نمودار آموزش مدل به شکل زیر است و لاغ های مربوط به آموزش در صفحه بعدی آورده شده.



شکل 12: نمودار های مربوط به یادگیری مدل

با توجه به نمودارها بهترین دقت مدل روی داده های ولیدیشن در ایپاک نهم و برابر با 95.026٪ است که این مدل را سیو میکنیم و داده های تست را با آن پیش‌بینی میکنیم. (دقت کنید که روی نمودا ایپاک ها از صفر و در لاغ از 1 شروع می شوند!)

با توجه به لاغ های آموزش در ایپاک هفتم از 0.001 به 0.0005 ، در ایپاک دوازدهم از 0.00025 به 0.00005 و در ایپاک پانزدهم از 0.000125 به 0.000025 کاهش داشته است که نهایتاً در ایپاک 15 با توجه به early stop فرایند یادگیری تمام شده است. این کاهش learning rate ها به دلیل ثابت ماندن یا نوسان کردن loss مربوط به داده های validation است. اگر دقت کنیم خواهیم دید که در هر مرحله کاهش learning rate تابع loss مربوط به داده های validation حداقل سه ایپاک افزایش نداشته است و قبل از early stop نیز به مدت 6 ایپاک، loss روی داده های validation کاهش نیافته است.

نکته مهم این است که روی داده های train تابع loss همواره در حال کاهش و مقدار دقیق هموار در حال افزایش است. ولی روی داده های validation اینگونه نیست و این مقادیر کم و زیاد می شوند. این مورد می تواند نشان از overfit شدن مدل باشد. پس از تست مدل روی داده های تست کمی معماری آن را عوض می کنیم تا این مورد جلو گیری کنیم.

لگ های مربوط به آموزش مدل:

```

Epoch 1/30
169/169 ————— 0s 16s/step - accuracy: 0.6757 - loss: 0.8977
Epoch 1: val_accuracy improved from -inf to 0.33679, saving model to /content/drive/My
Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.

169/169 ————— 4216s 24s/step - accuracy: 0.6764 - loss: 0.8957
- val_accuracy: 0.3368 - val_loss: 2.6343 - learning_rate: 0.0010
Epoch 2/30
169/169 ————— 0s 705ms/step - accuracy: 0.8986 - loss: 0.2777
Epoch 2: val_accuracy improved from 0.33679 to 0.53264, saving model to /content/drive/My
Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.

169/169 ————— 184s 1s/step - accuracy: 0.8986 - loss: 0.2776 -
val_accuracy: 0.5326 - val_loss: 2.3414 - learning_rate: 0.0010
Epoch 3/30
169/169 ————— 0s 674ms/step - accuracy: 0.9137 - loss: 0.2329
Epoch 3: val_accuracy improved from 0.53264 to 0.88221, saving model to /content/drive/My
Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.

169/169 ————— 173s 1s/step - accuracy: 0.9137 - loss: 0.2328 -
val_accuracy: 0.8822 - val_loss: 0.3167 - learning_rate: 0.0010
Epoch 4/30
169/169 ————— 0s 667ms/step - accuracy: 0.9222 - loss: 0.2106
Epoch 4: val_accuracy improved from 0.88221 to 0.89775, saving model to /content/drive/My
Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.

169/169 ————— 173s 1s/step - accuracy: 0.9223 - loss: 0.2105 -
val_accuracy: 0.8978 - val_loss: 0.2636 - learning_rate: 0.0010
Epoch 5/30
169/169 ————— 0s 654ms/step - accuracy: 0.9220 - loss: 0.2064
Epoch 5: val_accuracy did not improve from 0.89775
169/169 ————— 168s 995ms/step - accuracy: 0.9220 - loss:
0.2063 - val_accuracy: 0.7264 - val_loss: 0.9124 - learning_rate: 0.0010
Epoch 6/30
169/169 ————— 0s 643ms/step - accuracy: 0.9316 - loss: 0.1848
Epoch 6: val_accuracy did not improve from 0.89775
169/169 ————— 167s 989ms/step - accuracy: 0.9316 - loss:
0.1848 - val_accuracy: 0.8884 - val_loss: 0.2867 - learning_rate: 0.0010
Epoch 7/30
169/169 ————— 0s 653ms/step - accuracy: 0.9485 - loss: 0.1406
Epoch 7: val_accuracy did not improve from 0.89775

Epoch 7: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

```

```

169/169 ----- 205s 1s/step - accuracy: 0.9484 - loss: 0.1406 -
val_accuracy: 0.8667 - val_loss: 0.3523 - learning_rate: 0.0010
Epoch 8/30
169/169 ----- 0s 645ms/step - accuracy: 0.9581 - loss: 0.1120
Epoch 8: val_accuracy did not improve from 0.89775
169/169 ----- 167s 990ms/step - accuracy: 0.9581 - loss:
0.1120 - val_accuracy: 0.8943 - val_loss: 0.3253 - learning_rate: 5.0000e-04
Epoch 9/30
169/169 ----- 0s 647ms/step - accuracy: 0.9536 - loss: 0.1145
Epoch 9: val_accuracy improved from 0.89775 to 0.95026, saving model to /content/drive/My
Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
169/169 ----- 204s 1s/step - accuracy: 0.9536 - loss: 0.1144 -
val_accuracy: 0.9503 - val_loss: 0.1409 - learning_rate: 5.0000e-04
Epoch 10/30
169/169 ----- 0s 647ms/step - accuracy: 0.9641 - loss: 0.0925
Epoch 10: val_accuracy did not improve from 0.95026
169/169 ----- 167s 990ms/step - accuracy: 0.9640 - loss:
0.0925 - val_accuracy: 0.8953 - val_loss: 0.2564 - learning_rate: 5.0000e-04
Epoch 11/30
169/169 ----- 0s 642ms/step - accuracy: 0.9660 - loss: 0.0900
Epoch 11: val_accuracy did not improve from 0.95026
169/169 ----- 166s 987ms/step - accuracy: 0.9660 - loss:
0.0900 - val_accuracy: 0.9458 - val_loss: 0.1450 - learning_rate: 5.0000e-04
Epoch 12/30
169/169 ----- 0s 640ms/step - accuracy: 0.9671 - loss: 0.0886
Epoch 12: val_accuracy did not improve from 0.95026

Epoch 12: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
169/169 ----- 165s 980ms/step - accuracy: 0.9671 - loss:
0.0886 - val_accuracy: 0.6777 - val_loss: 0.9835 - learning_rate: 5.0000e-04
Epoch 13/30
169/169 ----- 0s 653ms/step - accuracy: 0.9766 - loss: 0.0587
Epoch 13: val_accuracy did not improve from 0.95026
169/169 ----- 168s 995ms/step - accuracy: 0.9766 - loss:
0.0588 - val_accuracy: 0.9119 - val_loss: 0.2819 - learning_rate: 2.5000e-04
Epoch 14/30
169/169 ----- 0s 645ms/step - accuracy: 0.9834 - loss: 0.0524
Epoch 14: val_accuracy did not improve from 0.95026
169/169 ----- 168s 992ms/step - accuracy: 0.9834 - loss:
0.0524 - val_accuracy: 0.8501 - val_loss: 0.5459 - learning_rate: 2.5000e-04
Epoch 15/30
169/169 ----- 0s 659ms/step - accuracy: 0.9823 - loss: 0.0514
Epoch 15: val_accuracy did not improve from 0.95026

Epoch 15: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
169/169 ----- 169s 1s/step - accuracy: 0.9823 - loss: 0.0515 -
val_accuracy: 0.8846 - val_loss: 0.2936 - learning_rate: 2.5000e-04
Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 9.

```

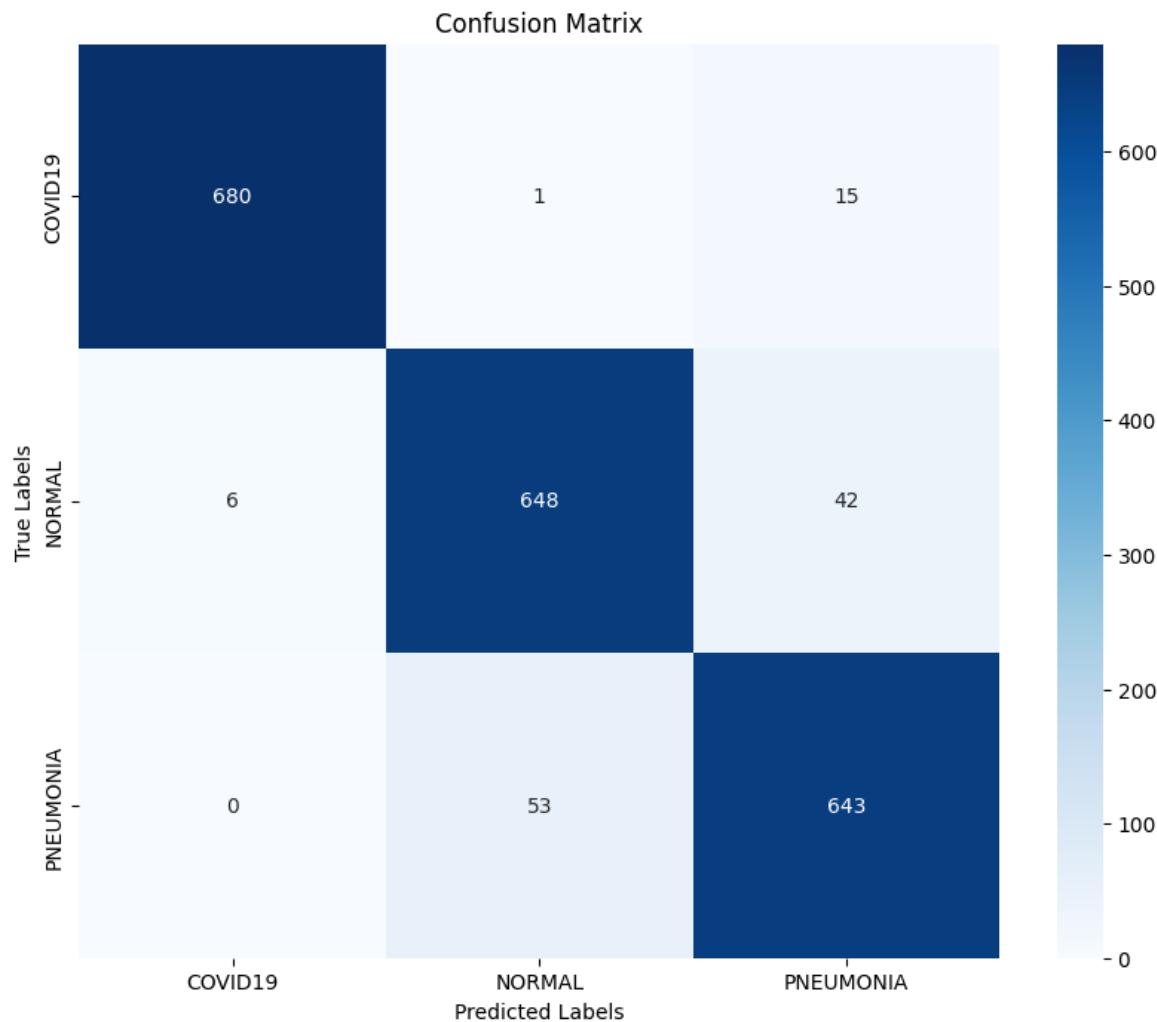
نتایج تست مدل با داده های تست در ادامه آورده شده:

Classification Report:

	precision	recall	f1-score	support
COVID19	0.99	0.98	0.98	696
NORMAL	0.92	0.93	0.93	696
PNEUMONIA	0.92	0.92	0.92	696
accuracy			0.94	2088
macro avg	0.94	0.94	0.94	2088

weighted avg 0.94 0.94 0.94 2088

با توجه به این موارد مدل دقت بسیار خوبی روی داده های تست داشته است. با توجه به مدل توانسته است 94 درصد کل داده های تست را به درستی پیشبینی کند. همچنین Precision برای تمام کلاس ها در یک حدود است ولی مدل بهتر توانسته داده های مربوط به COVID19 را پیشبینی کند. میانگین Precision برای تمام کلاس ها ۹۴٪ است. با توجه به داده های جدول بالا و Precision و Recall در یک حدود هستند. یعنی مدل بایاس نشده. برای مثال اگر بخواهیم برای کلاس COVID19 این مورد را بررسی کنیم یعنی وقتی مدل نمونه ای را مبتلا به کرونا پیشبینی کرده در ۹۹٪ موارد درست پیشبینی کرده (precision) و همچنین ۹۸٪ داده های مربوط به کلاس کرونا را نیز درست پیشبینی کرده (Recall). بنابراین با استفاده از precision و Recall می توانیم ببینیم مدل چند درصد از داده هایی که به عنوان کلاس A پیشبینی کرده واقعاً متعلق به کلاس A است و چند درصد از داده های واقعی کلاس A را درست پیشبینی کرده.



شکل 13 : matrix confusion مربوط به مدل اول

F1 score در واقع میانگین هارمونیک precision و recall است. در واقع این معیار می تواند تعادل بین این دو را برقرار کند. مخصوصا در داده های نا متوازن این معیار می تواند خیلی خوب مدل را توضیف کند. برای مثال فرض کنید در این دیتاست ، داده های مربوط به COVID19 10000 داده و داده های مربوط به باقی کلاس ها 1000 داده میبود. اگر مدل روی داده های COVID19 به دقت 99٪ میرسید می توانست معیار accuracy کل داده ها را به مقدار زیادی بالا ببرد. و در واقع سایر کلاس ها اهمیت خود را از دست میدادند.

بهتر کردن مدل:

برای بهتر کردن اوضاع از چیزی که هست می توانیم چند گام را انجام دهیم (به دلیل محدودیت زمان این گام ها را انجام ندادیم!):

- برای Augment کردن داده ها اول تعداد داده ها را کمتر نمی کنیم تا به تعداد کمترین کلاس برسد. بلکه با Augment کردن آنها کاری میکنیم تعداد داده ها کمی بیشتر از داده های بیشترین کلاس شود. این امر بخاطر آن است که از طرفی در کلاس های پر داده هم Augmentation داشته باشیم و هم داده های واقعی این کلاس ها را از دست ندهیم. نهایتاً مدل روی کلاس های پر داده کمتر به نویز و جابه جایی تصاویر مقاوم خواهد بود در عوض داده های کمتری را از دست خواهیم داد.

- نکته دیگر این است که سایز داده های متفاوت و اغلب بزرگ است به طوری که ما حتی داده 2000 در 2000 پیکسل هم داریم. با resize کردن این داده ها به 150 مقدار زیادی از اطلاعات داخل عکس از بین میروند. می توانیم در اینجا یک learnable resizer داشته باشیم. به طوری که این مازول داده ها را به ابعاد مختلف دریافت کند و با خروجی 150 در 150 به عنوان خروجی به ما بدهد. ولی با توجه به اینکه اگر سایز داده ها متغیر باشند کتابخانه های پایتون نمی توانند آنها را به شکل تنسور تبدیل کنند تا روی GPU این مدل ها را آموزش دهند فرایند آموزش ما بسیار کند خواهد شد (اما ایده بسیار جذاب است!! برای اطلاعات بیشتر می توانید به این [مقاله](#) مراجعه کنید).

- در ترین کردن مدل قبلی دیدیم که مقدار validation loss و به تبع آن دقت داده های validation بسیار متغیر هستند! درست است که ما بهترین مدل را ذخیره کردیم که مقدار دقت آن نیز مناسب بود ولی در حالت کلی مقدار validation loss هم باید یک روند مشخص داشته باشد. این مورد می تواند نشان از overfit شدن مدل باشد. بنابر این می توانیم چند مورد زیر را اعمال کنیم:

- به لایه های mlp نهایی regularization اضافه میکنیم.

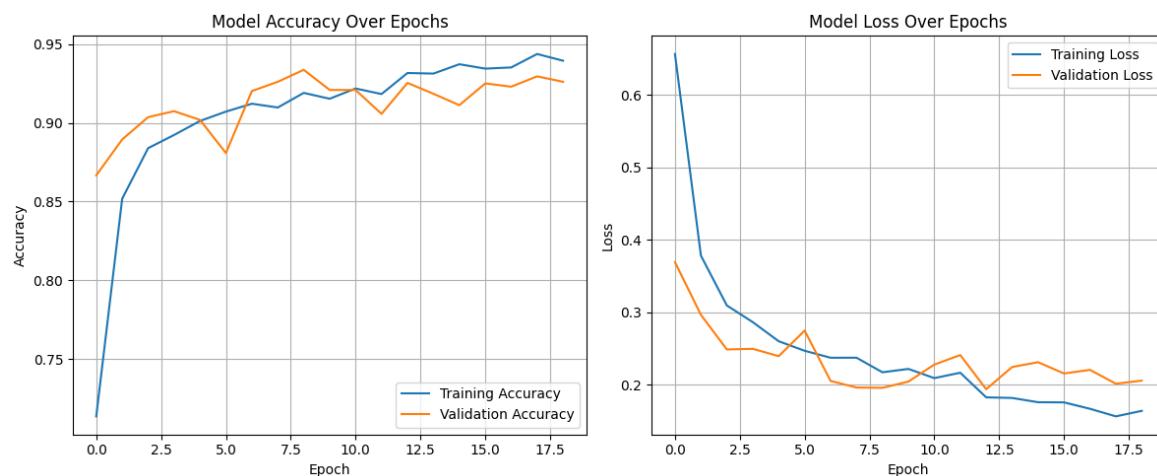
- dropout را در لایه های مختلف بیشتر میکنیم.

5- یادگیری انتقالی

یادگیری انتقالی یعنی در طی فرایند یادگیری از شبکه های از قبل آموزش داده شده استفاده کنیم. فرض کنید یک شبکه با تعداد بسیار زیادی لایه و پارامتر توسط بزرگترین متخصصان این حوزه با روش های حرfe ای روی سخت افزار های بسیار قوی و به مدت بسیار طولانی توسط داده های بسیار زیاد ترین شده باشند تا بتوانند ویژگی های مهم را از تصاویر استخراج کنند. در این حالت ما می توانیم از این شبکه ها در معماری خود استفاده کنیم. به شکلی که ویژگی های مورد نظر را توسط این شبکه ها استخراج کنیم و توسط یک شبکه mlp یک طبقه بند را روی این ویژگی ها آموزش دهیم. در این حالت ما شبکه از پیش

آموزش دیده را اصطلاحا فریز میکنیم و شبکه MLP خود را ترین میکنیم. با توجه به اینکه دیگر لازم نیست یک شبکه کانولوشنی را ترین کنیم احتمالا ترین شبکه ها با روش یادگیری انتقالی بسیار سریع تر خواهد بود بنابراین می توانیم مدل را در تعداد بالاتر ایپاک آموزش دهیم ولی early stop را قرار میدهیم تا اگر مدل به اندازه کافی آموزش دید فرایند آموزش متوقف شود.

ابتدا مدل VGG16 را در معماری قرار میدهیم و شبکه mlp را روی آن ترین میکنیم. شبکه mlp از سه لایه 128 و 64 و 3 نرونی تشکیل شده است. نمودار ها، لگ ها و گزارش های مربوطه در ادامه آورده شده اند.



شکل 14: نمودار های مربوط به یادگیری مدل **VGG16**

لگ های مربوط به آموزش مدل:

```

Epoch 1/100
169/169 ━━━━━━ 0s 3s/step - accuracy: 0.5919 - loss: 0.8543
Epoch 1: val_accuracy improved from -inf to 0.86667, saving model to /content/drive/My
Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)` . This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')` .
169/169 ━━━━━━ 753s 4s/step - accuracy: 0.5926 - loss: 0.8531 -
val_accuracy: 0.8667 - val_loss: 0.3694 - learning_rate: 0.0010
Epoch 2/100
169/169 ━━━━━━ 0s 745ms/step - accuracy: 0.8435 - loss: 0.3991
Epoch 2: val_accuracy improved from 0.86667 to 0.88946, saving model to /content/drive/My
Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)` . This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')` .
169/169 ━━━━━━ 195s 1s/step - accuracy: 0.8435 - loss: 0.3990 -
val_accuracy: 0.8895 - val_loss: 0.2966 - learning_rate: 0.0010
Epoch 3/100
169/169 ━━━━━━ 0s 706ms/step - accuracy: 0.8716 - loss: 0.3270
Epoch 3: val_accuracy improved from 0.88946 to 0.90363, saving model to /content/drive/My
Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)` . This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')` .

```

```

169/169 ----- 183s 1s/step - accuracy: 0.8716 - loss: 0.3269 -
val_accuracy: 0.9036 - val_loss: 0.2489 - learning_rate: 0.0010
Epoch 4/100
169/169 ----- 0s 689ms/step - accuracy: 0.8859 - loss: 0.2940
Epoch 4: val_accuracy improved from 0.90363 to 0.90743, saving model to /content/drive/My
Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
169/169 ----- 180s 1s/step - accuracy: 0.8860 - loss: 0.2939 -
val_accuracy: 0.9074 - val_loss: 0.2497 - learning_rate: 0.0010
Epoch 5/100
169/169 ----- 0s 680ms/step - accuracy: 0.8950 - loss: 0.2820
Epoch 5: val_accuracy did not improve from 0.90743
169/169 ----- 198s 1s/step - accuracy: 0.8951 - loss: 0.2818 -
val_accuracy: 0.9019 - val_loss: 0.2396 - learning_rate: 0.0010
Epoch 6/100
169/169 ----- 0s 665ms/step - accuracy: 0.9073 - loss: 0.2364
Epoch 6: val_accuracy did not improve from 0.90743
169/169 ----- 174s 1s/step - accuracy: 0.9073 - loss: 0.2365 -
val_accuracy: 0.8808 - val_loss: 0.2749 - learning_rate: 0.0010
Epoch 7/100
169/169 ----- 0s 666ms/step - accuracy: 0.9171 - loss: 0.2219
Epoch 7: val_accuracy improved from 0.90743 to 0.92021, saving model to /content/drive/My
Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
169/169 ----- 195s 1s/step - accuracy: 0.9170 - loss: 0.2220 -
val_accuracy: 0.9202 - val_loss: 0.2054 - learning_rate: 0.0010
Epoch 8/100
169/169 ----- 0s 679ms/step - accuracy: 0.9116 - loss: 0.2330
Epoch 8: val_accuracy improved from 0.92021 to 0.92608, saving model to /content/drive/My
Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
169/169 ----- 204s 1s/step - accuracy: 0.9116 - loss: 0.2330 -
val_accuracy: 0.9261 - val_loss: 0.1963 - learning_rate: 0.0010
Epoch 9/100
169/169 ----- 0s 675ms/step - accuracy: 0.9221 - loss: 0.2192
Epoch 9: val_accuracy improved from 0.92608 to 0.93368, saving model to /content/drive/My
Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
169/169 ----- 197s 1s/step - accuracy: 0.9221 - loss: 0.2192 -
val_accuracy: 0.9337 - val_loss: 0.1959 - learning_rate: 0.0010
Epoch 10/100
169/169 ----- 0s 659ms/step - accuracy: 0.9181 - loss: 0.2162
Epoch 10: val_accuracy did not improve from 0.93368
169/169 ----- 198s 1s/step - accuracy: 0.9181 - loss: 0.2162 -
val_accuracy: 0.9209 - val_loss: 0.2045 - learning_rate: 0.0010
Epoch 11/100
169/169 ----- 0s 650ms/step - accuracy: 0.9273 - loss: 0.1982
Epoch 11: val_accuracy did not improve from 0.93368
169/169 ----- 170s 1s/step - accuracy: 0.9273 - loss: 0.1983 -
val_accuracy: 0.9209 - val_loss: 0.2280 - learning_rate: 0.0010
Epoch 12/100
169/169 ----- 0s 683ms/step - accuracy: 0.9182 - loss: 0.2196
Epoch 12: val_accuracy did not improve from 0.93368

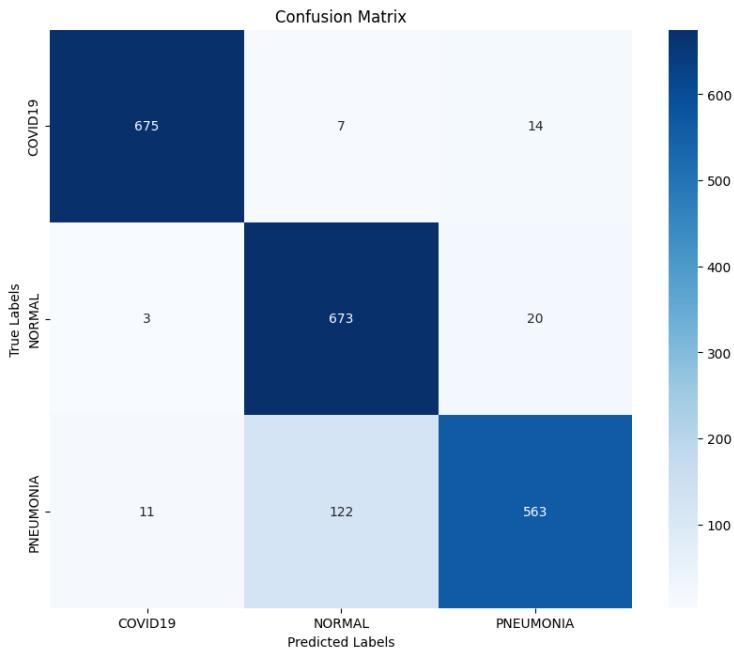
Epoch 12: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

```

```
169/169 ━━━━━━━━━━ 174s 1s/step - accuracy: 0.9182 - loss: 0.2196 -
val_accuracy: 0.9057 - val_loss: 0.2411 - learning_rate: 0.0010
Epoch 13/100
169/169 ━━━━━━━━━━ 0s 640ms/step - accuracy: 0.9228 - loss: 0.1998
Epoch 13: val_accuracy did not improve from 0.93368
169/169 ━━━━━━━━━━ 167s 990ms/step - accuracy: 0.9229 - loss:
0.1997 - val_accuracy: 0.9254 - val_loss: 0.1940 - learning_rate: 5.0000e-04
Epoch 14/100
169/169 ━━━━━━━━━━ 0s 650ms/step - accuracy: 0.9304 - loss: 0.1891
Epoch 14: val_accuracy did not improve from 0.93368
169/169 ━━━━━━━━━━ 169s 1s/step - accuracy: 0.9304 - loss: 0.1890 -
val_accuracy: 0.9185 - val_loss: 0.2246 - learning_rate: 5.0000e-04
Epoch 15/100
169/169 ━━━━━━━━━━ 0s 646ms/step - accuracy: 0.9347 - loss: 0.1810
Epoch 15: val_accuracy did not improve from 0.93368
169/169 ━━━━━━━━━━ 170s 1s/step - accuracy: 0.9348 - loss: 0.1810 -
val_accuracy: 0.9112 - val_loss: 0.2313 - learning_rate: 5.0000e-04
Epoch 16/100
169/169 ━━━━━━━━━━ 0s 686ms/step - accuracy: 0.9364 - loss: 0.1724
Epoch 16: val_accuracy did not improve from 0.93368

Epoch 16: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
169/169 ━━━━━━━━━━ 198s 1s/step - accuracy: 0.9364 - loss: 0.1724 -
val_accuracy: 0.9250 - val_loss: 0.2157 - learning_rate: 5.0000e-04
Epoch 17/100
169/169 ━━━━━━━━━━ 0s 662ms/step - accuracy: 0.9307 - loss: 0.1741
Epoch 17: val_accuracy did not improve from 0.93368
169/169 ━━━━━━━━━━ 171s 1s/step - accuracy: 0.9308 - loss: 0.1740 -
val_accuracy: 0.9230 - val_loss: 0.2207 - learning_rate: 2.5000e-04
Epoch 18/100
169/169 ━━━━━━━━━━ 0s 660ms/step - accuracy: 0.9425 - loss: 0.1580
Epoch 18: val_accuracy did not improve from 0.93368
169/169 ━━━━━━━━━━ 171s 1s/step - accuracy: 0.9425 - loss: 0.1580 -
val_accuracy: 0.9295 - val_loss: 0.2016 - learning_rate: 2.5000e-04
Epoch 19/100
169/169 ━━━━━━━━━━ 0s 643ms/step - accuracy: 0.9367 - loss: 0.1635
Epoch 19: val_accuracy did not improve from 0.93368

Epoch 19: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
169/169 ━━━━━━━━━━ 170s 1s/step - accuracy: 0.9368 - loss: 0.1635 -
val_accuracy: 0.9261 - val_loss: 0.2058 - learning_rate: 2.5000e-04
Epoch 19: early stopping
Restoring model weights from the end of the best epoch: 13
```



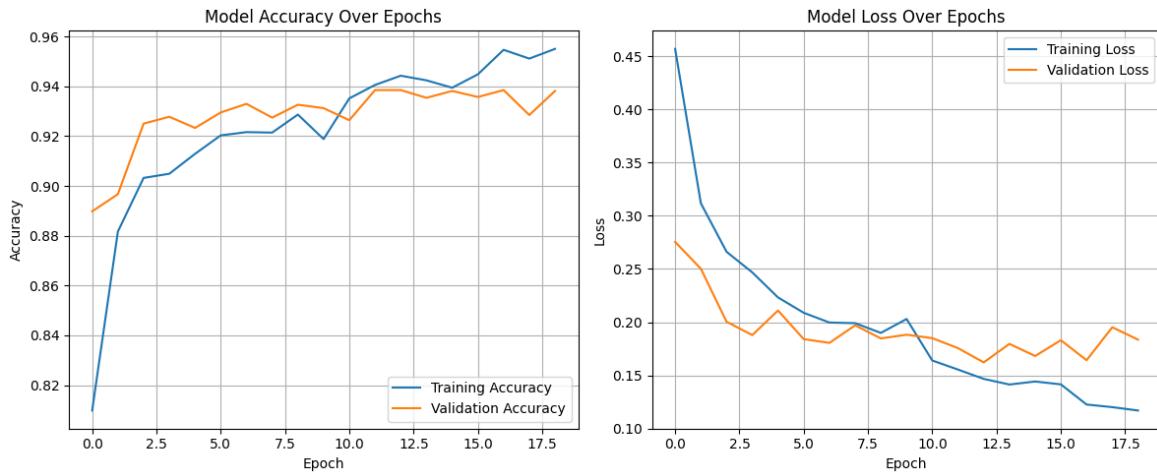
شکل 15: confusion matrix مربوط به مدل VGG16

Classification Report:

	precision	recall	f1-score	support
COVID19	0.98	0.97	0.97	696
NORMAL	0.84	0.97	0.90	696
PNEUMONIA	0.94	0.81	0.87	696
accuracy			0.92	2088
macro avg	0.92	0.92	0.91	2088
weighted avg	0.92	0.92	0.91	2088

با توجه به این موارد می توان گفت مدل به طور کلی مقدار کمی ضعیف تر عملکردی داشت. به عنوان نمونه recall کلاس PNEUMONIA مقدار 81 و PRECISION کلاس NORMAL برابر با 84 است. این یعنی مدل فقط 81 درصد از داده های واقعی کلاس PNEUMONIA را درست پیش‌بینی می‌کند و همچنین فقط 84 درصد از نمونه هایی که به عنوان NORMAL پیش‌بینی شده اند درست پیش‌بینی شده اند. ولی داده های مربوط به کلاس COVID19 همچنان با دقت بالایی پیش‌بینی شده اند.

مدل MobileNet را نیز مشابه به VGG به شبکه mlp وصل کرده و ترین میکنیم. نمودار ها، لگ ها و گزارش ها در ادامه آمده اند.



شکل 16: نمودار های مربوط به یادگیری مدل MobileNet

لگ های مربوط به آموزش مدل:

```

Epoch 1/100
169/169 ————— 0s 15s/step - accuracy: 0.7208 - loss:
0.6341
Epoch 1: val_accuracy improved from -inf to 0.88981, saving model to
/content/drive/My Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
169/169 ————— 4125s 24s/step - accuracy: 0.7213 -
loss: 0.6330 - val_accuracy: 0.8898 - val_loss: 0.2754 - learning_rate: 0.0010
Epoch 2/100
169/169 ————— 0s 713ms/step - accuracy: 0.8850 -
loss: 0.3109
Epoch 2: val_accuracy improved from 0.88981 to 0.89672, saving model to
/content/drive/My Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
169/169 ————— 229s 1s/step - accuracy: 0.8850 -
loss: 0.3109 - val_accuracy: 0.8967 - val_loss: 0.2503 - learning_rate: 0.0010
Epoch 3/100
169/169 ————— 0s 699ms/step - accuracy: 0.8959 -
loss: 0.2823
Epoch 3: val_accuracy improved from 0.89672 to 0.92504, saving model to
/content/drive/My Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
169/169 ————— 178s 1s/step - accuracy: 0.8960 -
loss: 0.2822 - val_accuracy: 0.9250 - val_loss: 0.2004 - learning_rate: 0.0010
Epoch 4/100
169/169 ————— 0s 662ms/step - accuracy: 0.9025 -
loss: 0.2456

```

```
Epoch 4: val_accuracy improved from 0.92504 to 0.92781, saving model to
/content/drive/My Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

169/169 ————— 196s 1s/step - accuracy: 0.9026 -
loss: 0.2456 - val_accuracy: 0.9278 - val_loss: 0.1879 - learning_rate: 0.0010
Epoch 5/100
169/169 ————— 0s 657ms/step - accuracy: 0.9176 -
loss: 0.2184
Epoch 5: val_accuracy did not improve from 0.92781
169/169 ————— 170s 1s/step - accuracy: 0.9176 -
loss: 0.2184 - val_accuracy: 0.9233 - val_loss: 0.2110 - learning_rate: 0.0010
Epoch 6/100
169/169 ————— 0s 652ms/step - accuracy: 0.9182 -
loss: 0.2159
Epoch 6: val_accuracy improved from 0.92781 to 0.92953, saving model to
/content/drive/My Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

169/169 ————— 172s 1s/step - accuracy: 0.9182 -
loss: 0.2158 - val_accuracy: 0.9295 - val_loss: 0.1841 - learning_rate: 0.0010
Epoch 7/100
169/169 ————— 0s 656ms/step - accuracy: 0.9257 -
loss: 0.1986
Epoch 7: val_accuracy improved from 0.92953 to 0.93299, saving model to
/content/drive/My Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

169/169 ————— 174s 1s/step - accuracy: 0.9257 -
loss: 0.1986 - val_accuracy: 0.9330 - val_loss: 0.1805 - learning_rate: 0.0010
Epoch 8/100
169/169 ————— 0s 663ms/step - accuracy: 0.9265 -
loss: 0.1953
Epoch 8: val_accuracy did not improve from 0.93299
169/169 ————— 194s 1s/step - accuracy: 0.9265 -
loss: 0.1953 - val_accuracy: 0.9275 - val_loss: 0.1970 - learning_rate: 0.0010
Epoch 9/100
169/169 ————— 0s 661ms/step - accuracy: 0.9246 -
loss: 0.1985
Epoch 9: val_accuracy did not improve from 0.93299
169/169 ————— 170s 1s/step - accuracy: 0.9246 -
loss: 0.1985 - val_accuracy: 0.9326 - val_loss: 0.1847 - learning_rate: 0.0010
Epoch 10/100
169/169 ————— 0s 650ms/step - accuracy: 0.9128 -
loss: 0.2138
Epoch 10: val_accuracy did not improve from 0.93299

Epoch 10: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
169/169 ————— 171s 1s/step - accuracy: 0.9129 -
loss: 0.2137 - val_accuracy: 0.9313 - val_loss: 0.1882 - learning_rate: 0.0010
Epoch 11/100
```

```
169/169 ----- 0s 654ms/step - accuracy: 0.9344 -
loss: 0.1703
Epoch 11: val_accuracy did not improve from 0.93299
169/169 ----- 168s 998ms/step - accuracy: 0.9344 -
loss: 0.1703 - val_accuracy: 0.9264 - val_loss: 0.1850 - learning_rate: 5.0000e-
04
Epoch 12/100
169/169 ----- 0s 646ms/step - accuracy: 0.9409 -
loss: 0.1594
Epoch 12: val_accuracy improved from 0.93299 to 0.93851, saving model to
/content/drive/My Drive/best_model_1.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
169/169 ----- 169s 1s/step - accuracy: 0.9409 -
loss: 0.1594 - val_accuracy: 0.9385 - val_loss: 0.1756 - learning_rate: 5.0000e-
04
Epoch 13/100
169/169 ----- 0s 641ms/step - accuracy: 0.9460 -
loss: 0.1424
Epoch 13: val_accuracy did not improve from 0.93851
169/169 ----- 167s 990ms/step - accuracy: 0.9460 -
loss: 0.1424 - val_accuracy: 0.9385 - val_loss: 0.1622 - learning_rate: 5.0000e-
04
Epoch 14/100
169/169 ----- 0s 639ms/step - accuracy: 0.9429 -
loss: 0.1443
Epoch 14: val_accuracy did not improve from 0.93851
169/169 ----- 166s 987ms/step - accuracy: 0.9429 -
loss: 0.1443 - val_accuracy: 0.9354 - val_loss: 0.1796 - learning_rate: 5.0000e-
04
Epoch 15/100
169/169 ----- 0s 649ms/step - accuracy: 0.9381 -
loss: 0.1466
Epoch 15: val_accuracy did not improve from 0.93851
169/169 ----- 167s 989ms/step - accuracy: 0.9381 -
loss: 0.1466 - val_accuracy: 0.9382 - val_loss: 0.1682 - learning_rate: 5.0000e-
04
Epoch 16/100
169/169 ----- 0s 633ms/step - accuracy: 0.9444 -
loss: 0.1407
Epoch 16: val_accuracy did not improve from 0.93851

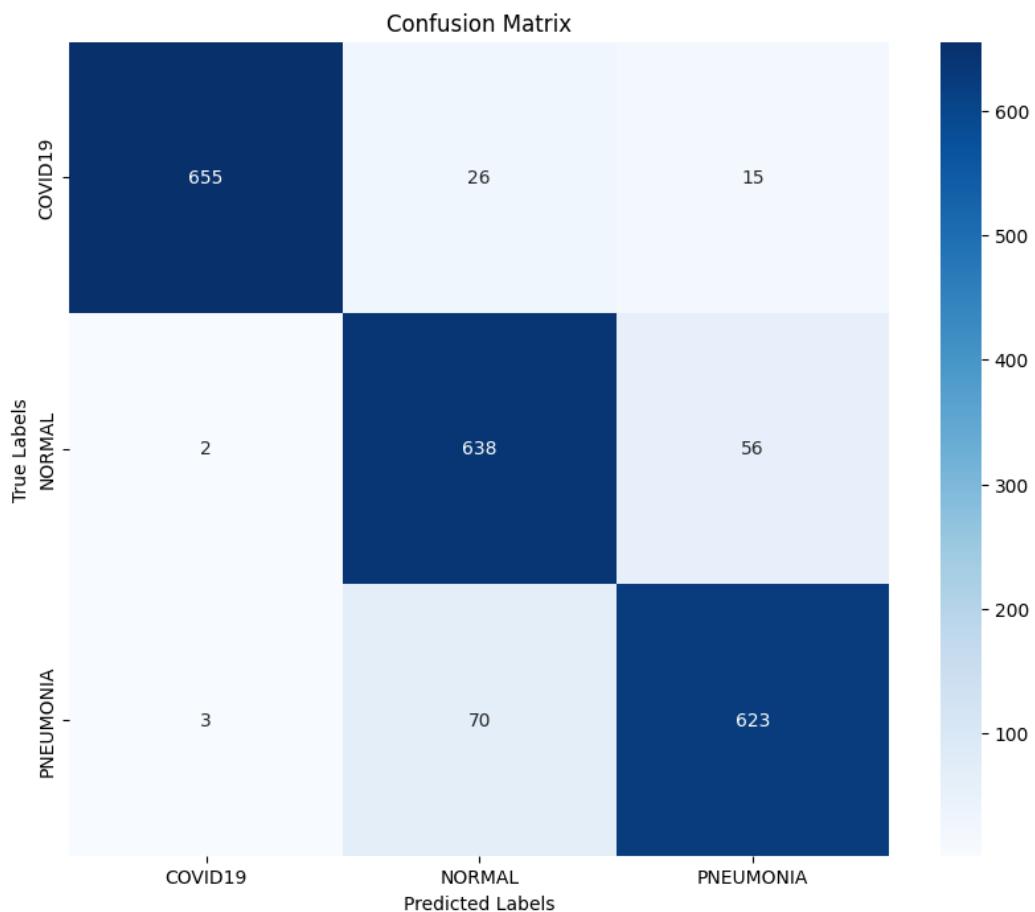
Epoch 16: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
169/169 ----- 165s 979ms/step - accuracy: 0.9444 -
loss: 0.1407 - val_accuracy: 0.9358 - val_loss: 0.1831 - learning_rate: 5.0000e-
04
Epoch 17/100
169/169 ----- 0s 624ms/step - accuracy: 0.9555 -
loss: 0.1219
Epoch 17: val_accuracy did not improve from 0.93851
169/169 ----- 162s 962ms/step - accuracy: 0.9555 -
loss: 0.1219 - val_accuracy: 0.9385 - val_loss: 0.1643 - learning_rate: 2.5000e-
04
Epoch 18/100
169/169 ----- 0s 637ms/step - accuracy: 0.9545 -
loss: 0.1190
Epoch 18: val_accuracy did not improve from 0.93851
```

```

169/169 ----- 166s 986ms/step - accuracy: 0.9545 -
loss: 0.1190 - val_accuracy: 0.9285 - val_loss: 0.1952 - learning_rate: 2.5000e-
04
Epoch 19/100
169/169 ----- 0s 643ms/step - accuracy: 0.9520 -
loss: 0.1176
Epoch 19: val_accuracy did not improve from 0.93851

Epoch 19: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
169/169 ----- 191s 1s/step - accuracy: 0.9520 -
loss: 0.1176 - val_accuracy: 0.9382 - val_loss: 0.1836 - learning_rate: 2.5000e-
04
Epoch 19: early stopping
Restoring model weights from the end of the best epoch: 13.

```



MobileNet مربوط به مدل **confusion matrix** :17 شکل

Classification Report:

	precision	recall	f1-score	support
COVID19	0.99	0.94	0.97	696
NORMAL	0.87	0.92	0.89	696
PNEUMONIA	0.90	0.90	0.90	696
accuracy			0.92	2088
macro avg	0.92	0.92	0.92	2088
weighted avg	0.92	0.92	0.92	2088

با توجه به نتایج بدست آمده می توان گفت MobileNet عملکرد بهتری نسبت به VGG داشته است. درست است که میانگین دقت هر دو مدل تقریباً برابر شده است ولی در MobileNet تعادل بین Precision و Recall بیشتر به چشم می خورد. در این مدل نیز داده های مربوط به کلاس COVID19 نسبت به بقیه کلاس ها بهتر تشخیص داده شده اند. با توجه به اینکه در تمام مدل ها داده های مربوط به این کلاس بهتر پیشبینی شده اند، احتمالاً داده های مربوط به کلاس COVID19 تفاوت معنی داری با داده های مربوط به بقیه کلاس ها دارند. همچنین با توجه به این confusion matrix و ماتریس مربوط به کلاس های قبلی، می توان گفت احتمالاً داده های کلاس های NORMAL و PNEUMONIA بسیار بهم نزدیک هستند زیرا در خیلی از موارد به جای هم پیشبینی می شوند.

درباره اینکه چرا با وجود شبکه های آماده پژوهشگران هنوز به دنبال طراحی مدل های جدید هستند چند دلیل وجود دارد. یکی از بدیهی ترین دلایل این است که یک مدل خاص که برای منظور و داده های خاص طراحی شده قطعاً نتیجه بهتری روی آن داده های خاص دارد. برای مثال همین تصاویر X-Ray یا MRI تفاوت زیادی با داده های ImageNet دارند و قاعده تحلیل و پردازش آنها نیز با داده های عادی تفاوت دارد. دلیل دیگر سبک تر و سریع تر شدن مدل است. در بسیاری از مواقع انتظار داریم مدل سریع و سبک باشد ولی در عین حال دقت آن زیاد پایین نباشد، مثلاً برای استفاده روی دستگاه های تلفن همراه نیاز است که مدل سبک باشد یا برای کاربرد های رانندگی خودکار، نیاز است که مدل سریع باشد. واضح است که مدل های آماده نمی توانند این خواسته را برآورده کنند. همچنین یکی دیگر از دلایل این است که مدل های کوچکتر برای ترین شدن به داده های کمتری نیاز دارند و به همین دلیل توسعه دادن آنها قطعاً ساده تر است.

پرسش ۲ - پیاده‌سازی یک سیستم طبقه‌بندی خودرو با استفاده از SVM و VGG16

۱-۱. مقدمه

در این سوال قصد داریم چند مدل ترکیبی را برای طبقه‌بندی خودروهای تویوتا طراحی و پیاده‌سازی کرده و عملکرد مدل‌ها را مقایسه نماییم. مدل‌های ترکیبی بر مبنای شبکه‌های VGG16 و Alex Net می‌باشند که از بخش Convolutional این شبکه‌ها برای استخراج ویژگی‌های تصاویرمان استفاده می‌کنیم.

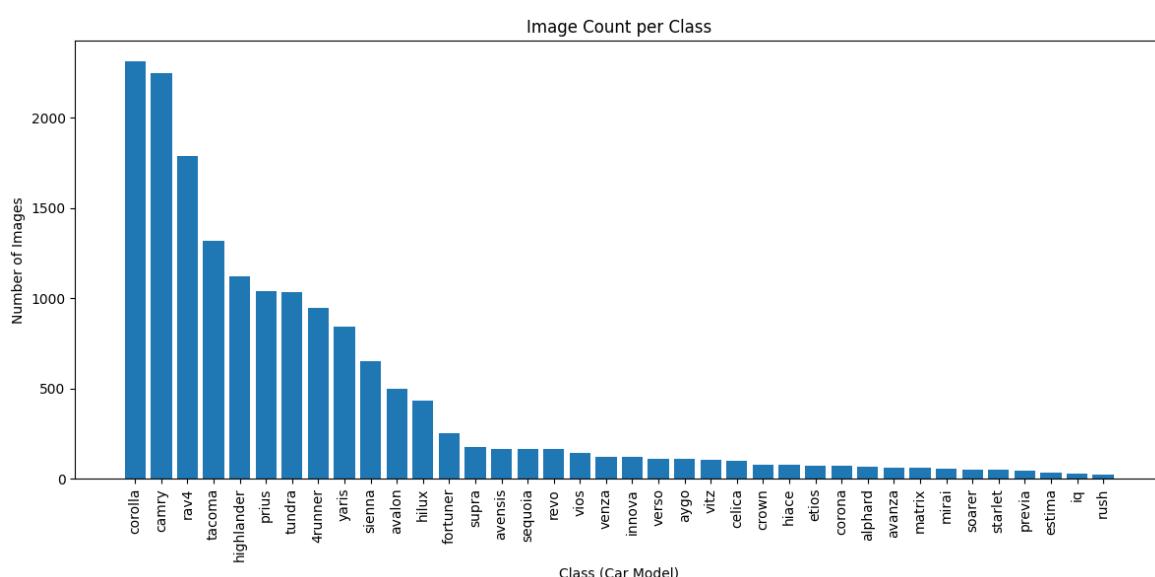
مدلی که در نهایت پیاده‌سازی می‌کنیم از شبکه از پیش تمرین داده شده VGG16 و برای طبقه‌بندی از مدل SVM استفاده می‌کند.

۲-۲. پیش‌پردازش داده‌ها

در ابتدا لازم است دیتا ست تصاویر خودرو هارا دانلود کرده و برای آماده‌سازی آن‌ها برای اعمال به شبکه مراحل زیر را انجام دهیم:

انتخاب ۱۰ کلاس:

ابتدا لازم است برای اینکه درکی از نحوه توزیع داده‌ها در دیتا ست داشته باشیم، نمودار فراوانی آماری داده‌ها که بر اساس فراوانی مرتب شده است رسم نماییم.



شکل ۱۸: نمودار فراوانی آماری کلاس‌های دیتا ست

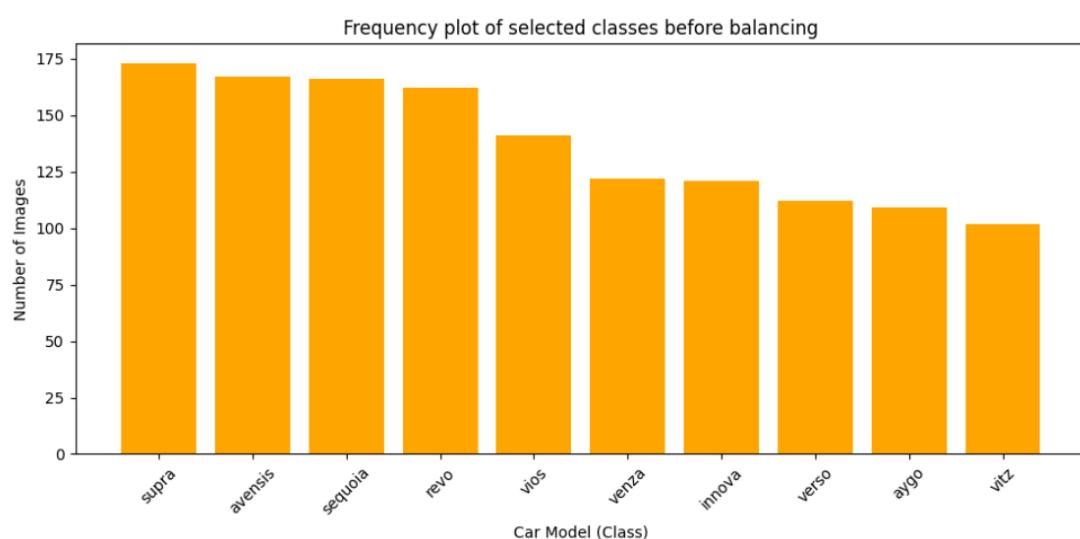
همانطور که میبینیم توزیع نمونه ها در بین کلاس های مختلف یکنواخت نیست و لازم است بعد از انتخاب کلاس های مورد نظر با تکنیک هایی آن هارا بالанс کنیم.

در اینجا ما کلاس های 13 تا 22 (شامل خودرو های Vios, Venza, Innova, Verso, Aygo, Vitz میشود) را انتخاب می نماییم.

حال لازم است برای آماده سازی این داده ها برای استفاده در شبکه، لیبل ها را به مقادیر عددی 0 تا 9 در بیاوریم. با انجام این کار امکان تعریف Loss function برای مسئله طبقه بندی نیز فراهم می شود.

بالанс کردن دیتا ست:

همانطور که در بخش قبل دیدیم توزیع فراوانی کلاس ها یکنواخت نیست. این امر باعث می شود تمرین دادن مدل به خوبی انجام نشود و باعث مشکلاتی مانند بایاس شدن مدل به یک کلاس خاص با تعداد نمونه بیشتر و یا کاهش دقیق مدل ببروی کلاس های کم نمونه می شود. روش های معمول یکنواخت سازی کلاس ها Undersampling و Oversampling می باشند. روش Undersampling با حذف تعدادی از اعضای کلاس ها همه را به سایز کوچکترین کلاس می رساند. مشخصا ایراد این روش این است که تعداد زیادی از داده ها را که میتوانست در تمرین مدل کمکمان کند از دست میدهیم. روش Oversampling با راهکار هایی مانند Data augmentation که با اعمال تغییراتی مانند چرخش و تغییر رنگ روی داده های موجود تصاویر جدید برای آن کلاس ایجاد می کند این مزیت را دارد که داده های همه کلاس هارا میتوان افزایش داد و همچنین مدل را نسبت به تغییرات ظاهری مانند رنگ و پرسپکتیو تصویر و ویژگی های مشابه از این دست که در تعیین کلاس موثر نیستند Robust نماییم.



شکل 19: توزیع آماری کلاس های انتخاب شده قبل از بالанс کردن

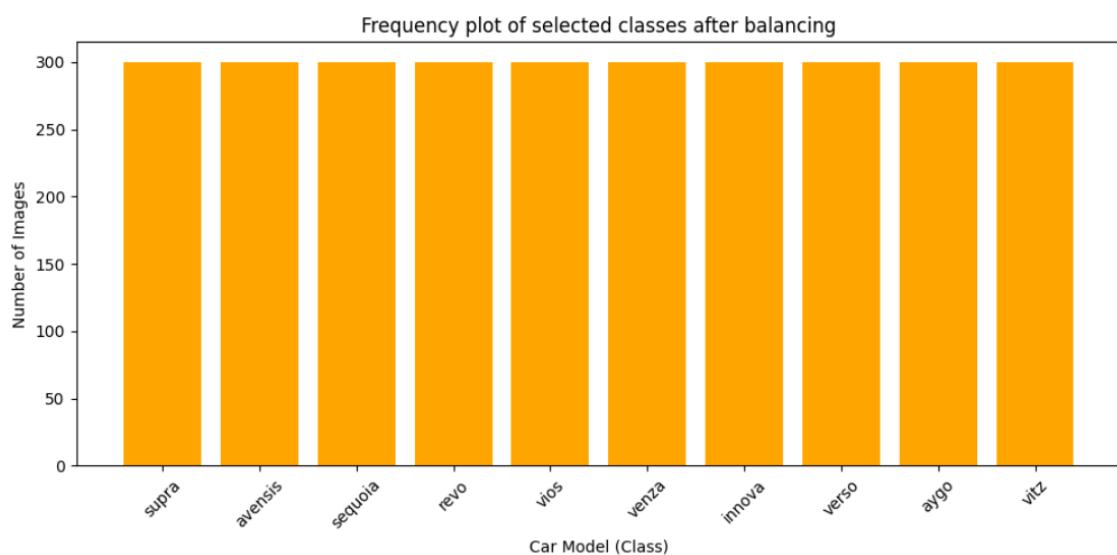
برای بالانس کردن نمونه های همه کلاس هارا به 300 می رسانیم.

```
augmentation_transforms = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(degrees=15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
    transforms.Resize((224, 224)),
])
```

شکل 20: نحوه augment کردن داده ها

شکل 20 نحوه انجام Augmentation را نشان میدهد. عملکرد های مختلف آن به شرح زیر هستند:

- تصاویر را به صورت تصادفی به صورت افقی می پرخاند.
- تصاویر را به صورت تصادفی در محدوده 0 تا 15 درجه می چرخاند.
- ویژگی هایی مانند روشنایی، کنترast و اشباع تصاویر را تغییر میدهد.
- سایز تصاویر را به همان اندازه مورد نظر سوال در می آورد.



شکل 21: توزیع آماری کلاس های انتخاب شده بعد از بالانس کردن

برای تغییر سایز ورودی ها به 224×224 و همچنین نرمال کردن پیکسل ها بین صفر و یک از یک استفاده مینماییم. Transform

```
Class to index mapping: {'avensis': 0, 'aygo': 1, 'innova': 2, 'revo': 3, 'sequoia': 4, 'supra': 5, 'venza': 6, 'verso': 7, 'vios': 8, 'vitz': 9}
Total samples: 3000
Training samples: 2400
Testing samples: 600
Image batch shape: torch.Size([32, 3, 224, 224])
Label batch shape: torch.Size([32])
```

شکل 22: گزارش کلاس ها و ابعاد دسته های آموزشی و تست

همانطور که میبینیم در مجموع 3000 داده داریم که با تقسیم 80 به 20 آنها 2400 داده آموزشی و 600 داده تست بدست میآید.

3-2. استخراج ویژگی ها

در این بخش میخواهیم تصاویرمان را به بخش Convolutional شبکه های AlexNet و VGG اعمال کرده، فیچرmap های خروجی آن هارا بدست آورده (به ازای هر عکس) و ویژگی ها را به صورت بردار های یک بعدی درآورده و ذخیره کنیم.

برای اینکار ابتدا باید مدل VGG را با وزن های دیفاللت اش لود کنیم. سپس با دستور Fully connected شبکه را حذف کنیم. حال با اعمال همه عکس ها به شبکه خروجی شبکه را ذخیره کرده، آن را به صورت یک بردار Reshape کرده و به صورت یک فایل pt. ذخیره میکنیم. در مراحل بعد با لود کردن این فایل ها میتوان بردار ویژگی ها را به شبکه مورد نظر اعمال نمود. برای شبکه AlexNet نیز همین مراحل را طی میکنیم.

همچنین نکته دیگری که وجود دارد این است که برای مشخص کردن کلاس هر بردار ویژگی ذخیره شده، نام آن را با فرمت img_{index}_label{data label} در میآوریم. با این کار در هنگام لود کردن ویژگی تصاویر، لیبل آنها نیز مشخص میشود.

4-2. آموزش و ارزیابی مدل

حال که بردار ویژگی ها را با شبکه های VGG و AlexNet استخراج کرده ایم، یک مدل طبقه بند برای هر کدام پیاده سازی میکنیم و به بررسی عملکرد آن میپردازیم.

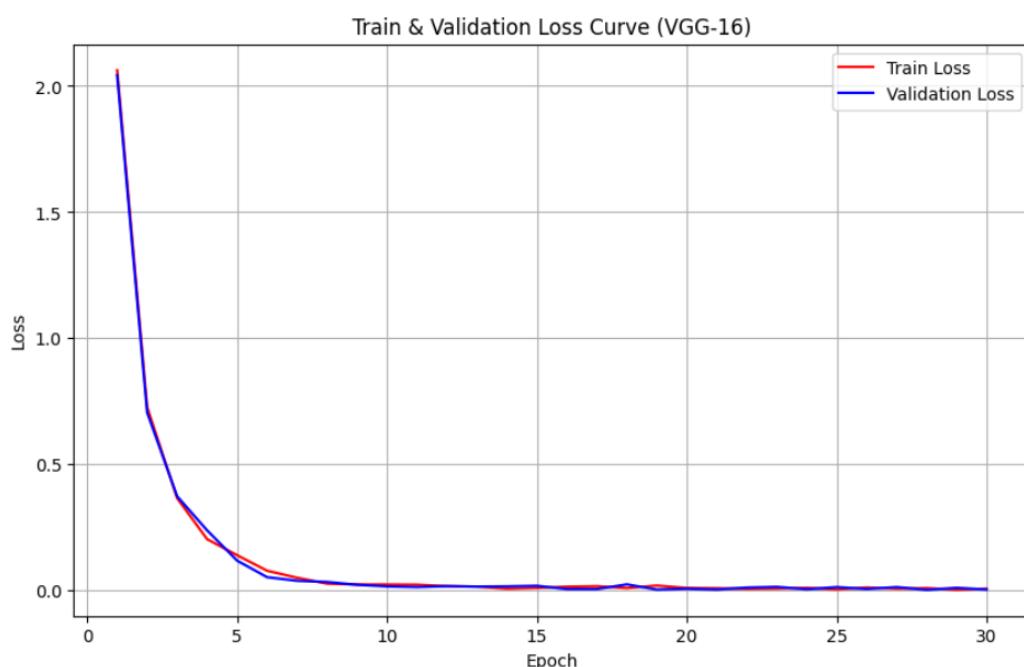
1- تمرین مدل های بخش قبل: مدل VGG16:

بردار خروجی ویژگی های بدست آمده از این شبکه ابعادی برابر $7*7*512$ که معادل 25088 هست را دارد. شبکه Fully connected 10 کلاس استفاده میشود را به صورت زیر تعریف میکنیم:

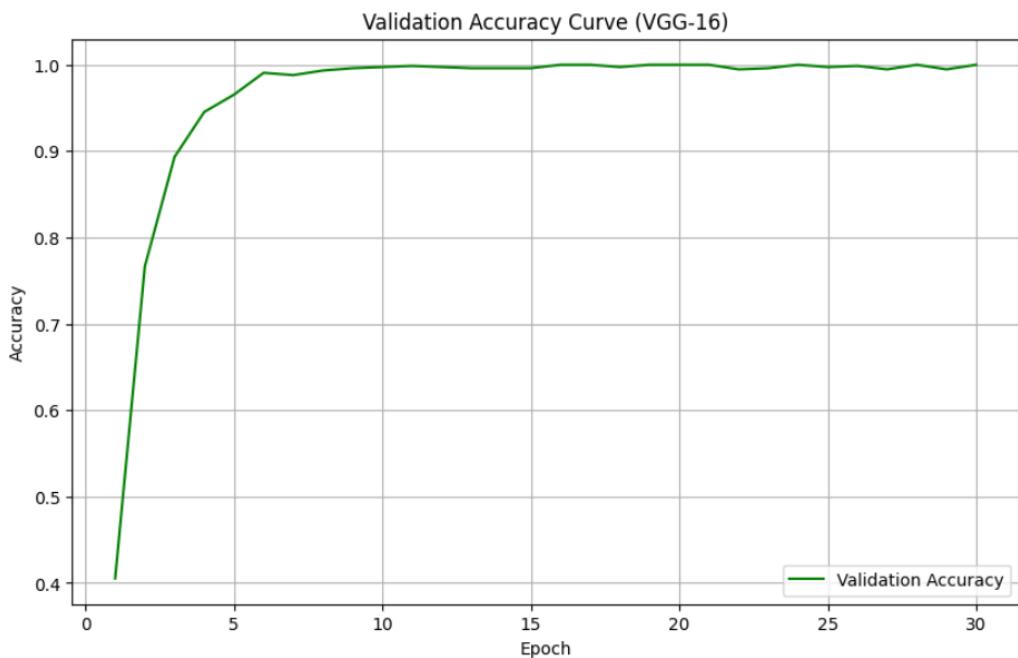
- لایه 1: 25088 به 512 ReLU
- لایه 2: 512 به 10 SoftMax
- نرخ یادگیری: 1e-3
- نرخ Regularization: 1e-4
- تعداد epoch: 30

نرخ یادگیری متغیر (هر 5 epoch نصف مقدار قبلی) اعمال شده است. همچنین در هر epoch 30 درصد از داده های batch را به صورت تصادفی به Validation اختصاص میدهیم تا عملکرد مدل حین تمرین برآورد شود.

نتایج تمرین مدل در تصاویر زیر آمده است.



شکل 23: تابع خطا مدل VGG بر حسب epoch و train Validation برای داده های



شکل 24: نمودار دقت مدل **VGG** روی داده های **Validation**

```
VGG-16 metrics:
Accuracy: 0.8516666666666667
Precision: 0.8528605521739642
Recall: 0.8516666666666667
F1 Score: 0.8515066364461905
```

شکل 25: پارامتر های ارزیابی مدل طبقه بند **VGG**

شکل 25 معیارهای ارزیابی مدل بر روی داده های تست را نشان میدهد. در ادامه نحوه عملکرد هر یک از این پارامترها توضیح میدهیم:

- Accuracy: این معیار نسبت تعداد نمونه هایی که به درستی در کلاس تشخیص داده شده اند (True Positive) و نمونه هایی که به درستی در کلاس منفی تشخیص داده شده اند را به کل نمونه ها مشخص میکند. به عبارت دیگر این معیار نشان دهنده درصد پیش‌بینی های صحیح مدل به کل پیش‌بینی ها میباشد.

- Precision: این معیار نسبت تعداد پیش‌بینی های مثبت از تمام پیش‌بینی های مثبت (که شامل پیش‌بینی هایی که به اشتباه به کلاس مثبت نسبت داده شده اند نیز می‌شود) را نشان میدهد. این میتواند معیار خوبی در موقعی باشد که پیش‌بینی اشتباه کلاس مثبت مطلوب ما نیست و میخواهیم ارزیابی از آن داشته باشیم.

- Recall: معیاری از توانایی مدل در تشخیص نمونه های یک کلاس (کلاس مثبت) است و نشان میدهد مدل چه درصدی از داده های آن کلاس را به درستی پیش‌بینی کرده و به آن کلاس

نسبت داده است و از نسبت پیش‌بینی های درست کلاس مثبت به کل پیش‌بینی های آن کلاس (شامل True Positive و False Negative) بدست می‌آید.

- F1 Score: ترکیبی از معیارهای precision و Recall می‌باشد و میتواند معیاری از تعادل میان دقت مدل و حساسیت آن به داده های یک کلاس خاص است و زمانی بالا میرود که هر دوی این معیار ها اعداد بالا باشند و در صورتی که در یکی عملکرد خوبی نداشته باشد، در F1 Score این امر مشخص میشود.

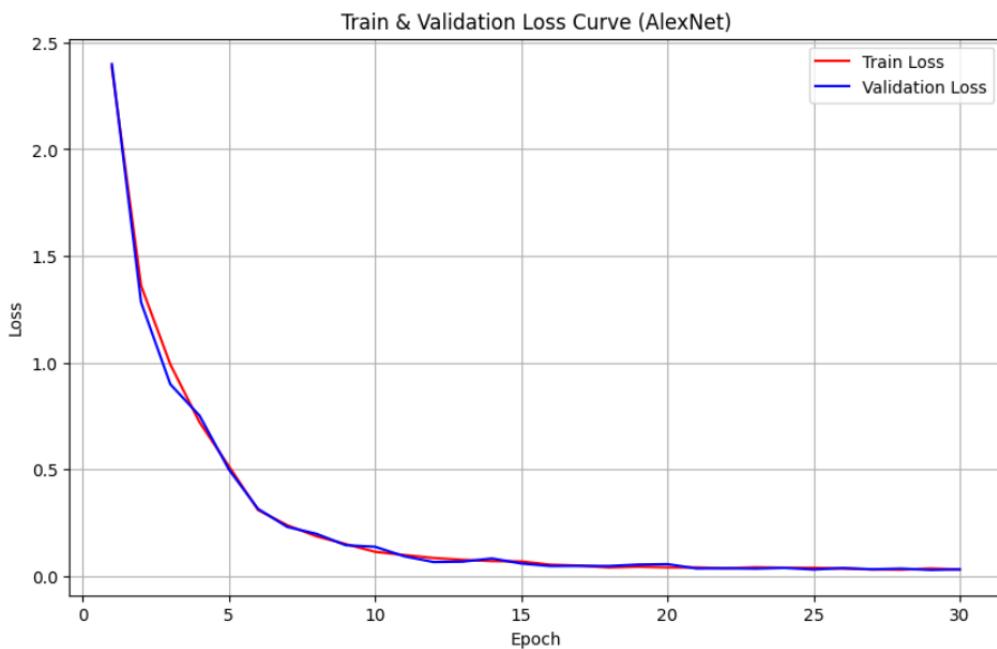
:AlexNet مدل

بردار خروجی ویژگی های بدست آمده از این شبکه ابعادی برابر $6 \times 6 \times 256$ که معادل 9216 هست را دارد. شبکه Fully connected 10 کلاس استفاده میشود را به صورت زیر تعریف میکنیم:

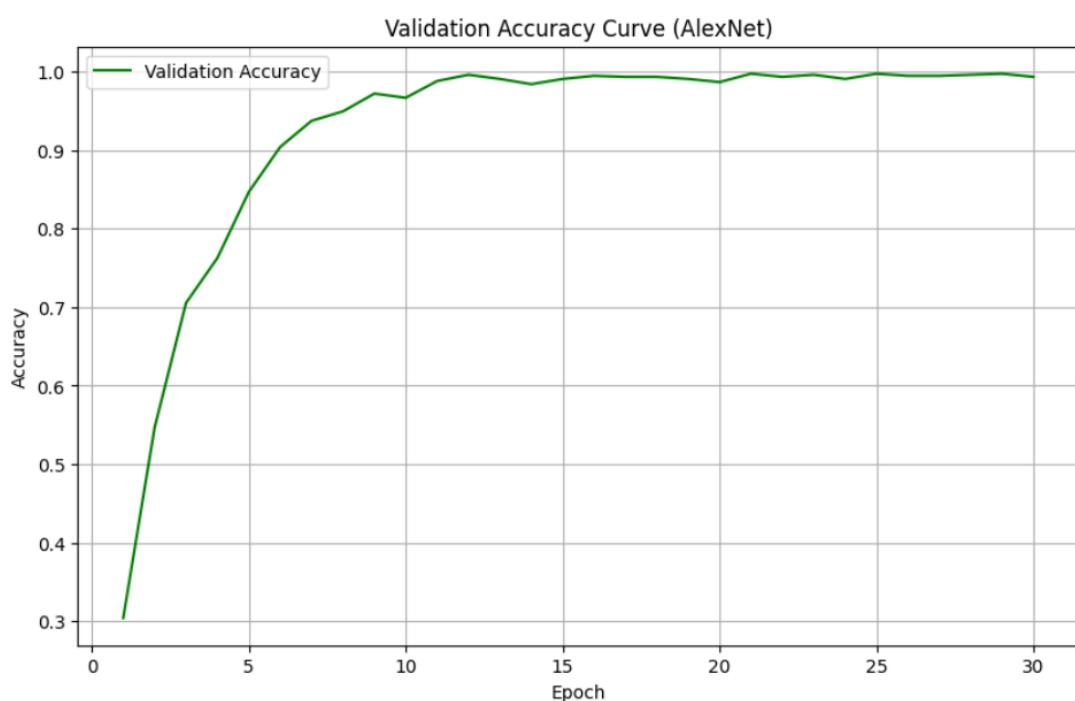
- لایه 1: 9216×512 به ReLU
- لایه 2: 512×10 به SoftMax
- نرخ یادگیری: $1e-3$
- نرخ Regularization: $1e-4$
- تعداد epoch: 30

نرخ یادگیری متغیر (هر 5 epoch نصف مقدار قبلی) اعمال شده است. همچنین در هر epoch درصد از داده های batch را به صورت تصادفی به Validation اختصاص میدهیم تا عملکرد مدل حین تمرین برآورد شود.

نتایج تمرین مدل در تصاویر زیر آمده است.



شکل 26: تابع خطای مدل **AlexNet** بر حسب epoch و train Validation برای داده های



شکل 27: نمودار دقت مدل **Alex Net** روی داده های Validation

```

Alex Net metrics:
Accuracy: 0.7866666666666666
Precision: 0.7900288346827894
Recall: 0.7866666666666666
F1 Score: 0.7872907629154676

```

شکل 28: پارامتر های ارزیابی مدل طبقه بند Alex Net

2- تفاوت VGG Net و Alex Net

شبکه Alex Net تعداد پارامترهای کمتری نسبت به VGG دارد (حدود نصف). اصلی ترین تفاوت آن ها در معماری شان است. شبکه Alex Net نسبتا عمق کمی دارد (8 لایه) و معماری لایه ها و سایز فیلتر ها در لایه های مختلف متفاوت می باشد و در کل معماری شبکه و نحوه اتصال لایه های مختلف پیچیده می باشد. در حالی که شبکه VGG شبکه عمیقی می باشد (با 19 لایه) و از معماری ساده و یکنواختی در همه لایه ها استفاده می کند (همه فیلتر ها 3 در 3 هستند)

3- مدل CNN

در این بخش یک مدل CNN را خودمان تعریف کرده و بر روی تصاویرمان با ابعاد 224 در 224 که در بخش های قبلی رسایز کرده بودیم تمرین میدهیم.

ساختار مدل به صورت زیر تعریف می شود:

بخش Convolution

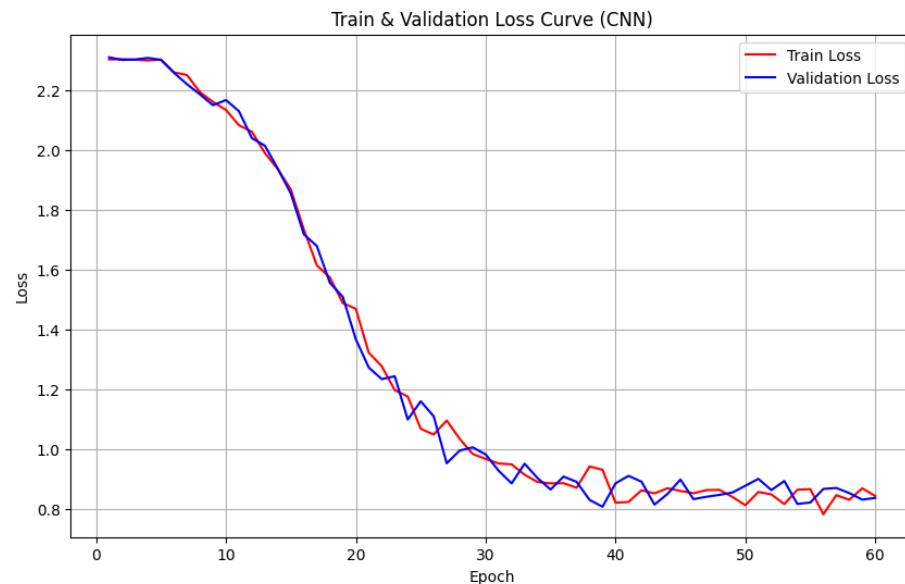
- لایه 1: 32 فیچرمپ، ReLU، Pooling=2، Kernel Size=9
- لایه 2: 64 فیچرمپ، ReLU، Pooling=2، Kernel Size=7
- لایه 3: 128 فیچرمپ، ReLU، Pooling=2، Kernel Size=5
- لایه 4: 256 فیچرمپ، ReLU، Pooling=2، Kernel Size=3
- لایه 5: 512 فیچرمپ، ReLU، Pooling=2، Kernel Size=3

بخش Fully connected

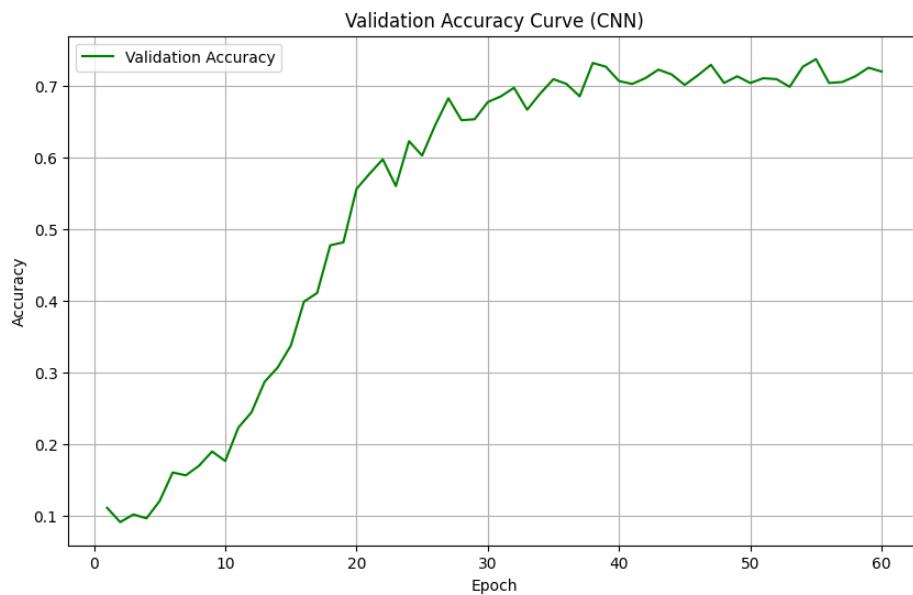
- لایه 1: 256 به 25088: ReLU
- لایه 2: 256 به 10: SoftMax
- نرخ یادگیری: 1e-3
- تعداد epoch: 60

نرخ یادگیری متغیر (هر 5 epoch نصف مقدار قبلی) اعمال شده است. همچنین در هر epoch، درصد از داده های batch را به صورت تصادفی به Validation اختصاص میدهیم تا عملکرد مدل حین تمرین برآورد شود.

نتایج به صورت زیر میباشد:



شکل 29: تابع خطا مدل CNN بر حسب epoch برای داده های train و Validation



شکل 30: نمودار دقیق مدل CNN روی داده های Validation

```

CNN metrics:
Accuracy: 0.4933333333333335
Precision: 0.49751213269353056
Recall: 0.4933333333333335
F1 Score: 0.4908417432678942

```

شکل 31: پارامتر های ارزیابی مدل طبقه بند CNN

4- مدل ترکیبی SVM و VGG

در این بخش ویژگی های استخراج شده در مراحل قبل توسط VGG را به یک مدل SVM اعمال میکنیم. کرنل SVM را از نوع خطی قرار میدهیم.

نتایج این مدل به صورت زیر میباشد:

```

VGG-16 + SVM metrics
Accuracy: 0.825
Precision: 0.8257123528643637
Recall: 0.825
F1 Score: 0.8244340710924798

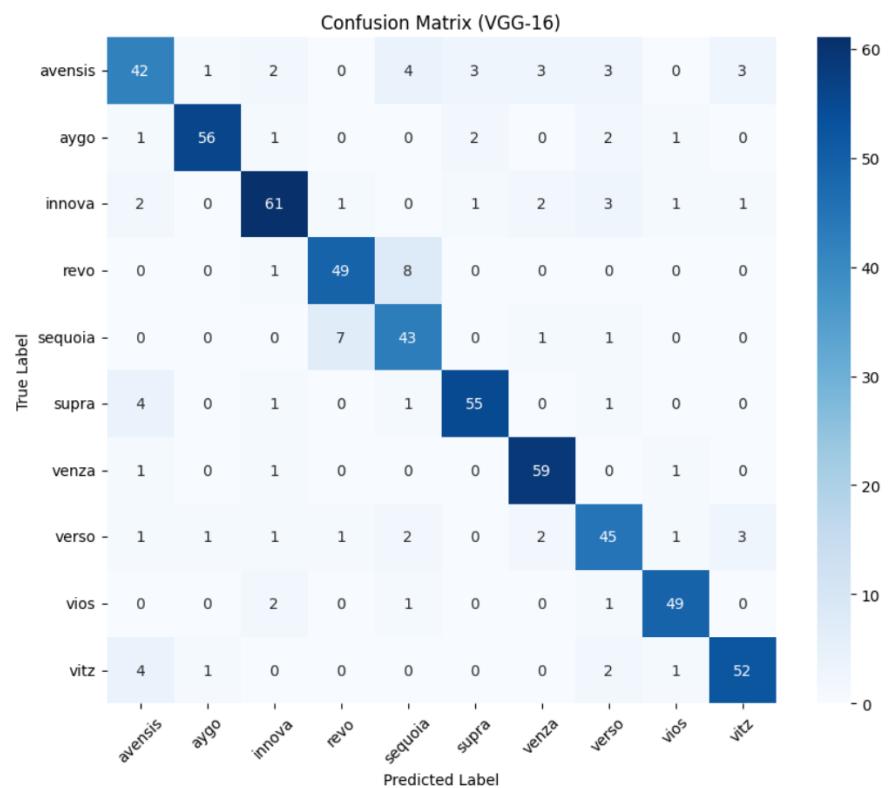
```

شکل 32: پارامتر های ارزیابی مدل طبقه بند

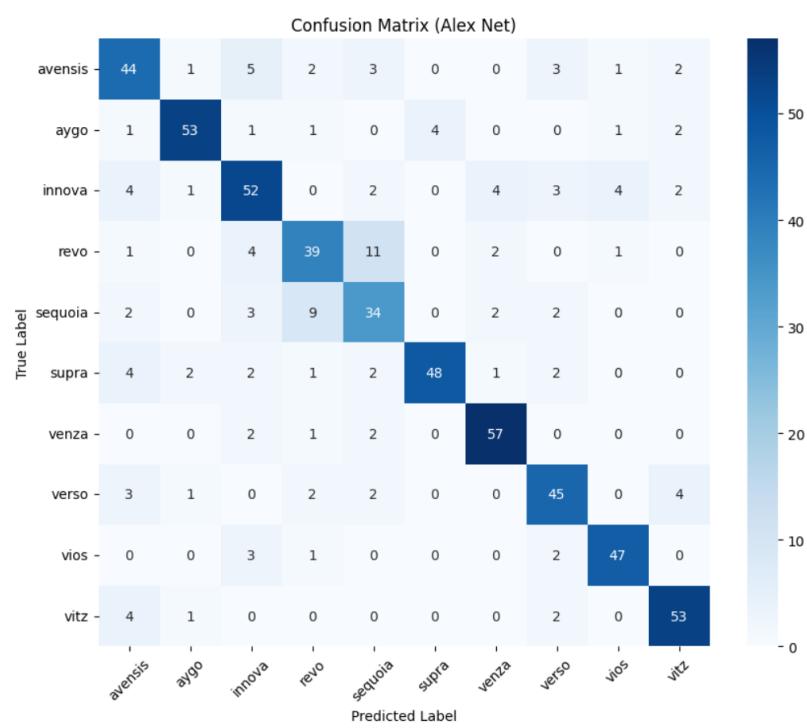
Classification Report:				
	precision	recall	f1-score	support
0	0.73	0.67	0.70	61
1	0.89	0.87	0.88	63
2	0.87	0.81	0.83	72
3	0.75	0.74	0.75	58
4	0.75	0.81	0.78	52
5	0.93	0.85	0.89	62
6	0.84	0.95	0.89	62
7	0.75	0.77	0.76	57
8	0.89	0.92	0.91	53
9	0.84	0.85	0.84	60

شکل 33: پارامتر های ارزیابی مدل طبقه بند SVM به ازای هر کلاس

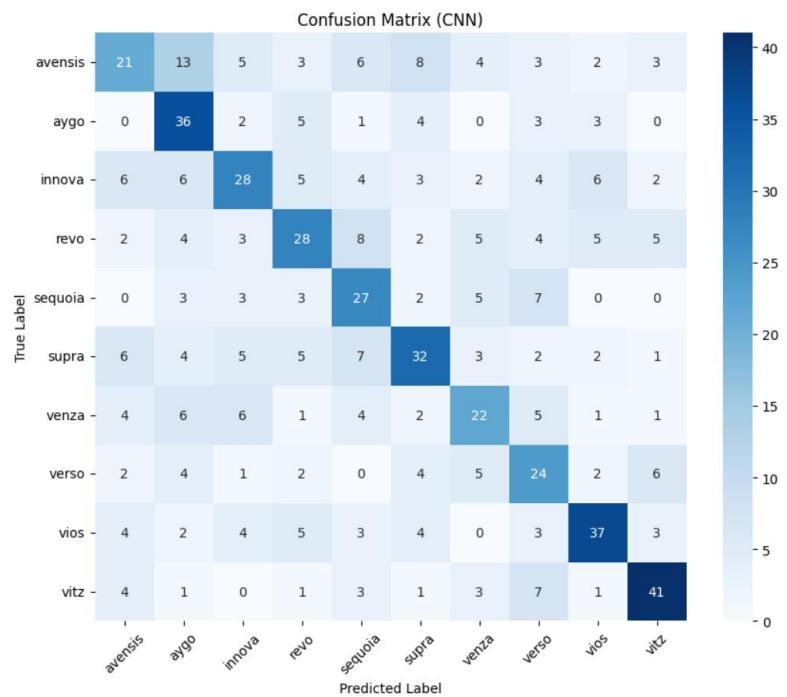
حال با رسم ماتریس های آشفتگی برای مدل های مختلف به تحلیل آنها میپردازیم. این ماتریس در دو محور کلاس های پیشビینی شده و کلاس های تارگت را نمایش داده و نشان میدهد در هر کلاس چه تعداد از داده ها درست پیشビینی شده و چه تعداد به اشتباه به کدام کلاس ها نسبت داده شده اند



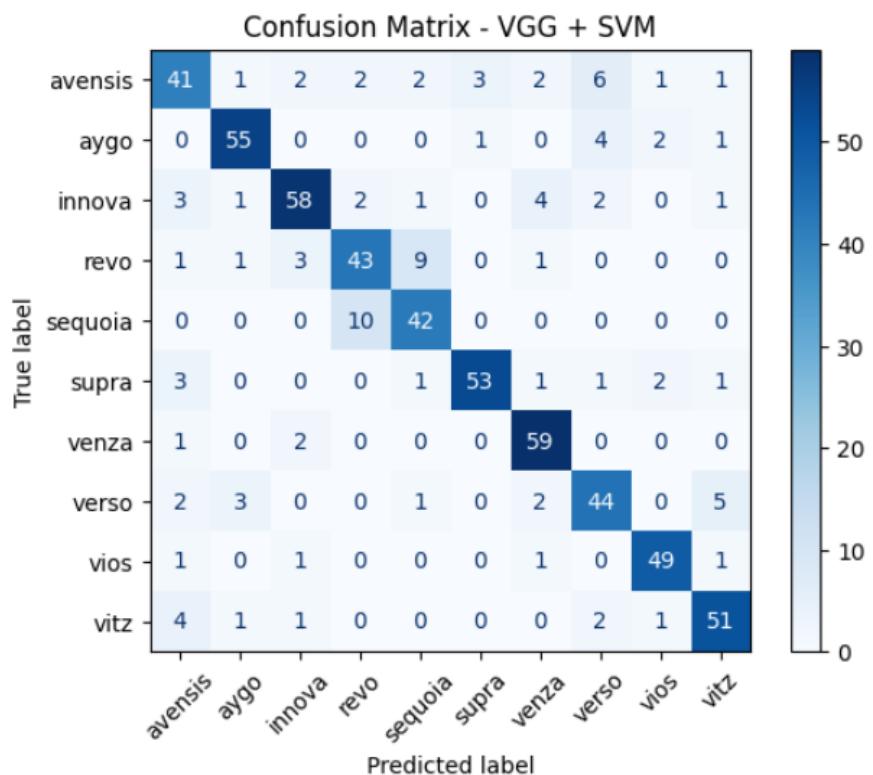
شکل 34: ماتریس آشتگی مدل VGG-16



شکل 35: ماتریس آشتگی مدل Alex Net



شکل 36: ماتریس آشفتگی مدل CNN



شکل 37: ماتریس آشفتگی مدل VGG+SVM

هر چه مقادیر این ماتریس بروی قطر آن جمع شوند نشان دهنده عملکرد بهتر مدل میباشد. همانطور که در تصاویر بالا مشاهده میشود عملکرد هر 4 مدل در تشخیص کلاس های Revo و Sequoia ضعیف تر از

کلاس های دیگر بوده و این دو کلاس را در تعداد قابل توجهی از داده ها به جای هم به اشتباه تشخیص داده است.

5-2: تحلیل نتایج

جدول 4: مقایسه عملکرد مدل ها

Model	Accuracy	Precision	Recall	F1 Score
VGG+SVM	82.5	82.57	82.5	82.44
VGG 16	85.16	85.28	85.16	85.15
Alex Net	78.66	79.00	78.66	78.72
CNN	49.33	49.75	49.33	49.08

ابتدا به تحلیل معیارهای جدول 4 می پردازیم. مدل VGG16 بهترین عملکرد را برروی کلاس های انتخاب شده نشان داد (با دقت حدود 85 درصد). مدل ترکیبی پیشنهادی نیز عملکرد خوبی داشت و دقتی برابر 82.5 درصد برروی داده های تست داشت. عملکرد مدل AlexNet نیز هم سطح این مدل ها بود (با دقت 78 درصدی). ولی مدل CNN بی که خودمان تعریف کردیم عملکرد خوبی از خود نشان نداد و دقتش نزدیک 50 درصد رسید. این امر نشان دهنده این است که با یک مدل CNN با عمق کم نمیتوان عملکردی در سطح شبکه های عمیقی مانند VGG16 که دارای لایه های کانولوشن زیادی هستند و یا شبکه Transfer Learning را میتوان در اینجا مشاهده کرد که با استفاده از این شبکه های قوی از پیش تمرین داده شده و بدست آوردن بردار ویژگی ها، با صرف هزینه و وقت کمی میتوانیم بر روی دیتا ست مد نظرمان به دقت خوبی برسیم. (تنها لایه های Fully Connected را تمرین میدهیم)

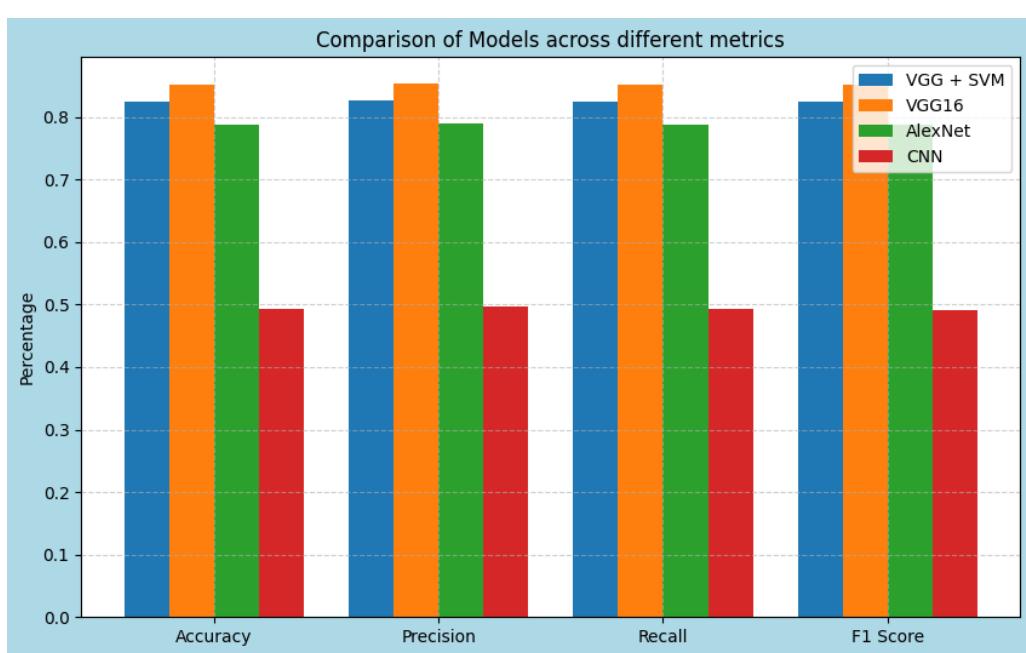
نکته دیگری که وجود دارد این است که هر 4 معیار در ارزیابی هر مدل مقادیری بسیار نزدیک به هم بودند که نشان دهنده بالانس بودن مدل است؛ به این معنا که برای مثال Overfit اتفاق نیفتاده و توزیع خطاهای متوازن است.

عملکرد هر مدل برروی کلاس های مختلف به کمک ماتریس آشفتگی ها که در بخش قبل رسم کردیم مشخص میشود:

- VGG16: این مدل بر روی کلاس های Aygo, Innova, Supra, Venzo عملکرد خیلی خوبی داشت و بیشتر داده های تست متعلق به این کلاس هارا به درستی لیبل گذاری میکند. ولی بر

روی کلاس های Revo و Sequoia نسبتاً ضعیف عمل کرده است و در بیشتر موارد خطأ، این دو کلاس را بجای هم اشتباه تشخیص داده است.

- عملکرد این مدل در تشخیص کلاس ها بسیار مشابه شبکه AlexNet میباشد.
- CNN: این شبکه به صورت کلی در تشخیص کلاس ها خوب عمل نمیکند. مدل در تشخیص کلاس های Aygo, Vitz, Vios به نسبت بقیه کلاس ها بهتر عمل میکند (به سطح عملکرد 3 مدل دیگر نمیرسد) ولی در کلاس هایی مثل Avensis, Venza, Verso عملکرد بسیار ضعیفی دارد.
- VGG + SVM: این مدل نیز عملکردش بر روی کلاس ها بسیار شبیه دو مدل ابتدایی می باشد



شکل 38: مقایسه عملکرد 4 مدل

برای ارتقای مدل‌ها میتوان از راهکارهای زیر استفاده نمود:

- بهبود داده‌ها: همانطور که دیدیم عملکرد مدل‌ها (به جز CNN) بر روی همه کلاس ها بجز دو کلاس Revo و Sequoia خوب بود. میتوانیم دیتا ست این دو کلاس را بررسی دقیق تری کنیم و با اعمال Data Augmentation و یا حتی ایجاد داده‌های جدید برای این دو کلاس، عملکرد مدل را بهبود ببخشیم.
- استفاده از Weighted Loss Function: با اینکار میتوان به داده های این دو کلاس که اشتباه تشخیص داده میشوند در تمرین مدل وزن بیشتری داد تا مدل به خطای آنها حساس تر بشود.

- میتوانیم یک Sub Network اضافه کنیم که تنها برای تفکیک داده هایی که توسط مدل متعلق به این دو کلاس تشخیص داده شده استفاده شود.
- برای بهبود عملکرد مدل CNN میباشد معماری شبکه را تغییر داده و با کارهایی مانند افزایش لایه های Convolution و استفاده از BatchNormalization دقیق شبکه را بهتر نمود.

6-2: امتیازی

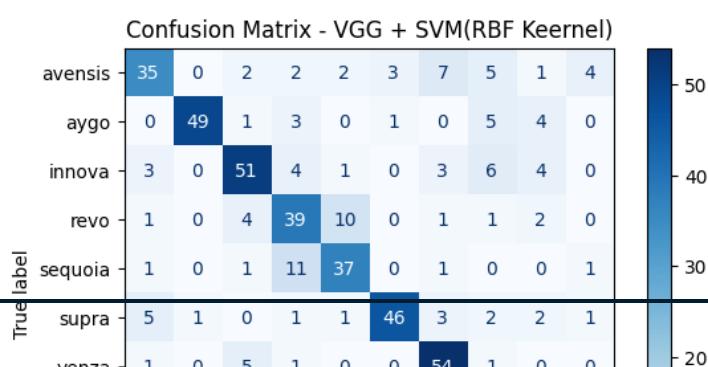
1- استفاده از کرنل RBF

حال میخواهیم از کرنل غیرخطی RBF برای مدل SVM بخش قبل استفاده کنیم. این کرنل برخلاف که یک صفحه صاف در فضای ویژگی ها پیدا میکند، میتواند مرزهای پیچیده تری برای ما ایجاد کند.

```
VGG-16 + SVM metrics(RBF Kernel)
Accuracy: 0.7383333333333333
Precision: 0.7468261945577734
Recall: 0.7383333333333333
F1 Score: 0.7390604939719709
```

شکل 39: پارامتر های ارزیابی مدل طبقه بند SVM با کرنل RBF

مشاهده میکنیم که نتیجه هم داشت. یکی از دلایل این امر میتواند این باشد که نقاط ما در فضای ویژگی ها به صورت خطی تا حد خوبی جدا پذیرند و با استفاده از کرنل غیر خطی تنها پیچیدگی مسئله را بالا بردیم که میتواند به آموزش مدل آسیب بزند.



شکل 40: ماتریس آشفتگی مدل **VGG+SVM** با کرنل **RBF**

2- استخراج ویژگی ها:

در مسائلی که حجم داده زیادی در اختیار نداریم میتوانیم از شبکه های از پیش آموزش داده شده برای استخراج موثر ویژگی تصاویر استفاده کنیم و در ادامه تنها با یک بخش MLP ساده مدل مورد نظرمان را پیاده سازی کنیم. این امر در سرعت آموزش مدل هم به شدت تاثیرگذار است و زمان آموزش را بهشدت کاهش میدهد.