به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین دوم

پرسش ۱	نام و نام خانوادگی	علیرضا نجفی مطیعی
	شماره دانشجویی	810100224
پرسش ۲	نام و نام خانوادگی	پوریا ذره پرور قوچانی نژاد
	شماره دانشجویی	۸۱۰۱۰۲۱۴۱
	مهلت ارسال پاسخ	**۱۴۰۴.02.01**

فهرست

شکل‌ها

جدول‌ها

پرسش ۱. تشخیص بیماران مبتلا به COVID-19 با استفاده از تصاویر X-RAY

2-1. تحلیل دیتاست

در این تمرین از notebook editor سایت Kaggle استفاده می‌کنیم و برای بررسی دیتاست و مراحل بعدی دیتاست گفته شده در تمرین را در بخش input اضافه می‌کنیم.

- کلاس‌هایی که در بخش ترین و تست این دیتاست وجود دارند به نام‌های زیر می‌باشند:

```
Root folders: ['test', 'train']
Train classes: ['PNEUMONIA', 'NORMAL', 'COVID19']
Test  classes: ['PNEUMONIA', 'NORMAL', 'COVID19']
```

شکل1. کلاس‌های موجود در دیتاست

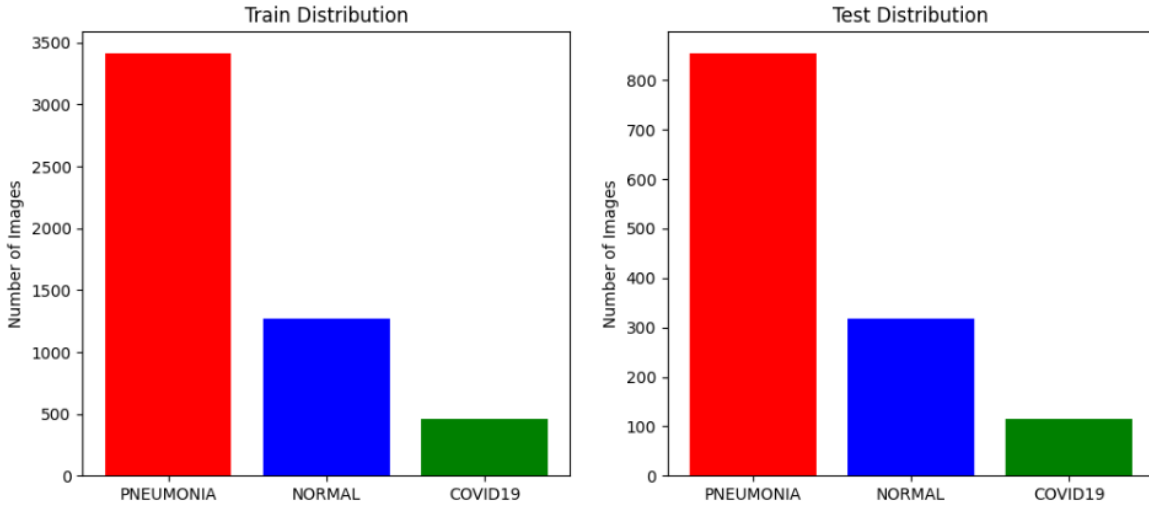
- فرمت داده‌های موجود در این دیتاست و همچنین توزیع داده‌های هر کلاس بصورت زیر است:

```
Train counts: {'PNEUMONIA': 3418, 'NORMAL': 1266, 'COVID19': 460}
Train formats: Counter({'.jpg': 5144})
Test  counts: {'PNEUMONIA': 855, 'NORMAL': 317, 'COVID19': 116}
Test  formats: Counter({'.jpg': 1288})
```

شکل2. فرمت و توزیع داده‌های هر کلاس

همانطور که مشاهده می‌شود همه داده‌ها در این دیتاست فرمت jpg. دارند. همچنین بیشتر داده‌ها چه در تست و چه در ترین مربوط به کلاس PNEUMONIA و کمترین آن‌ها مربوط به کلاس COVID19 می‌باشد.

- حال به رسم هیستوگرام تعداد داده‌ها برای هر دو مجموعه test و train می‌پردازیم:



شکل3. هیستوگرام داده‌های هرکلاس برای test و train

- بالانس بودن کلاس‌های دیتاست موجب می‌شود که در فرآیند یادگیری شبکه عصبی مدل نسبت به یک کلاس خاص که تعداد آن بطور محسوسی بیشتر از سایر کلاس‌هاست بایاس نشود و در

طی این فرآیند چون داده‌ها بالانسند به نسبت برابری با داده‌های هر کلاس روبرو شود و آن‌ها را یاد بگیرد.

- نامتعادل بودن دیتاست باعث خواهد شد تا مدل نسبت به کلاسی با تعداد داده‌های بیشتر بایاس شود بطوری که اگر بیشتر بودن این کلاس نسبت به سایر کلاس‌ها بطور محسوسی بیشتر باشد مدل بیشتر این کلاس را فرا خواهد گرفت و حتی اگر تمام داده‌های تست را بدون بررسی از داده‌های این کلاس تشخیص دهد باز هم به دقت بالایی خواهد رسید.

زمانی که دیتاست نامتعادل داریم از رایج‌ترین روش‌ها نمونه برداری از سایر کلاس‌ها به تعداد نمونه‌های کمترین کلاس و سپس درصورت نیاز استفاده از data augmentation و افزایش داده‌های نمونه برداری شده به تعداد برابر بگونه ای که درنهایت تمامی کلاس‌ها با نسبت تقریبا برابر از تعداد دیتای خوبی نیز برخوردار باشند.

برای این منظور برای نمونه برداری به تعداد کمترین کلاس از سایر کلاس‌ها می‌توانیم از کتابخانه random و تابع sample یعنی درواقع random.sample که باید در آن در آرگومان اول فایل‌های مرجع و در آرگومان بعدی تعداد اعضای نمونه را مشخص کنیم. برای data augmentation نیز از ImageDataGenerator موجود در tensorflow.keras.preprocessing.image استفاده می‌کنیم.

```
datagen=ImageDataGenerator(rotation_range=15, width_shift_range=0.10,
height_shift_range=0.10, zoom_range=0.10, brightness_range=[0.8,1.2],
horizontal_flip=True, fill_mode='nearest')
```

برای resample کردن داده‌ها از تابع‌های موجود در کتابخانه imblearn مانند SMOTEENN, ClusterCentroids, TomekLinks و ADASYN نیز می‌توان استفاده کرد اما برای دیتاست ما ابزارهای انتخاب شده خوب و پاسخگوی نیاز ما هستند.

3-1 . پیش‌پردازش داده‌ها

با استفاده ابزارهای معرفی شده در بخش قبل به تعداد داده‌های موجود در بخش test و train در کلاس COVID-19 که به ترتیب 116 و 460 هستند از داده‌های test و train سایر کلاس‌ها نمونه برداری می‌کنیم.

بطور خلاصه data augmentation یا داده افزایی به مجموعه راهکارهایی گفته می‌شود که از آن‌ها استفاده می‌کنیم تا تعداد داده‌های موجود خود را بدون اینکه دوباره به جمع‌آوری داده بپردازیم افزایش دهیم. از داده افزایی زمانی استفاده می‌کنیم که تعداد داده‌های موجود برای منظور خاص مورد نظر ما کافی نباشد. روش‌های متداول داده افزایی برای داده‌هایی که به فرم تصویر هستند شامل روشن یا تیره تر کردن تصویر، آینه کردن، شیفت دادن تصویر بصورت عمودی یا افقی، زوم کردن و چرخاندن تصویر در یک بازه مشخص می‌باشد که با استفاده از ImageDataGenerator می‌توانیم از تمامی این روش‌ها استفاده کنیم هرچند که چرخاندن تصویر یا آینه کردن آن و همچنین تا حدودی تغییر روشنایی تصویر احتمالا تاثیر بهتری روی این داده‌ها داشته باشند. با استفاده از این روش‌ها مدل با گستره خوبی از داده‌ها روبرو خواهد شد و می‌تواند بخوبی آموزش ببیند.

در این مرحله داده‌ها را حدودا 5 برابر کرده و در دایرکتوری خروجی Kaggle ذخیره می‌کنیم. برای منظور یک تابع می‌نویسیم تا با استفاده از ImageDataGenerator داده افزایی را انجام دهد و با استفاده از تابع‌های موجود در os آدرس را مشخص، داریکتوری را تشکیل و داده‌ها را در آن ذخیره می‌کنیم.

```python
def augment_and_save(src_dir,dst_dir,times=5,split_name="Train"):
    os.makedirs(dst_dir,exist_ok=True)
    for cls in tqdm(os.listdir(src_dir),
                    desc=f"Augmenting {split_name}",unit="cls"):
        src_cls=os.path.join(src_dir,cls)
        dst_cls=os.path.join(dst_dir,cls)
        os.makedirs(dst_cls,exist_ok=True)
        for fname in tqdm(os.listdir(src_cls),
                          desc=f" {cls}",unit="img",leave=False):
            img_path=os.path.join(src_cls,fname)
            img=img_to_array(load_img(img_path))
            x=img.reshape((1,)+img.shape)
            gen=datagen.flow(
                x,batch_size=1,
                save_to_dir=dst_cls,
                save_prefix='aug',
                save_format='jpeg'
            )
            for _ in range(times):
                next(gen)

augment_and_save(balanced_train,'/kaggle/working/augmented_train',times=5,split_name="Train")
augment_and_save(balanced_test,'/kaggle/working/augmented_test',times=5,split_name="Test")
```

در زیر بخش‌هایی از این فرآیند آمده است:

```
Sampling Train: 100%|          | 3/3 [00:22<00:00,  7.49s/cls]
Sampling Test: 100%|           | 3/3 [00:07<00:00,  2.44s/cls]
Augmenting Train:   0%|                | 0/3 [00:00<?, ?cls/s]
 COVID19:   0%|                | 0/460 [00:00<?, ?img/s]
 COVID19:   0%|                | 1/460 [00:03<26:09,  3.42s/img]
 COVID19:   0%|                | 2/460 [00:05<22:19,  2.92s/img]
 COVID19:   1%|                | 3/460 [00:07<18:23,  2.41s/img]
 COVID19:   1%|                | 4/460 [00:09<16:41,  2.20s/img]
 COVID19:   1%|                | 5/460 [00:10<14:12,  1.87s/img]
 COVID19:   1%||                | 6/460 [00:12<12:34,  1.66s/img]
```

شکل4. فرآیند نمونه برداری و آغاز فرآیند data augmentation

```
 NORMAL: 100%|          | 459/460 [29:39<00:03,  3.17s/img]
 NORMAL: 100%|          | 460/460 [29:42<00:00,  2.96s/img]
Augmenting Train: 100%|          | 3/3 [1:14:20<00:00, 1486.97s/cls]
Augmenting Test:   0%|                | 0/3 [00:00<?, ?cls/s]
 COVID19:   0%|                | 0/116 [00:00<?, ?img/s]
 COVID19:   1%|                | 1/116 [00:01<02:14,  1.17s/img]
 COVID19:   2%||                | 2/116 [00:01<01:45,  1.08img/s]
```

شکل5. پایان فرآیند برای داده‌های train و شروع برای داده‌های test

```
 NORMAL:  98%|          | 114/116 [07:45<00:08,  4.12s/img]
 NORMAL:  99%|          | 115/116 [07:47<00:03,  3.55s/img]
 NORMAL: 100%|          | 116/116 [07:51<00:00,  3.62s/img]
Augmenting Test: 100%|          | 3/3 [21:01<00:00, 420.53s/cls]
```

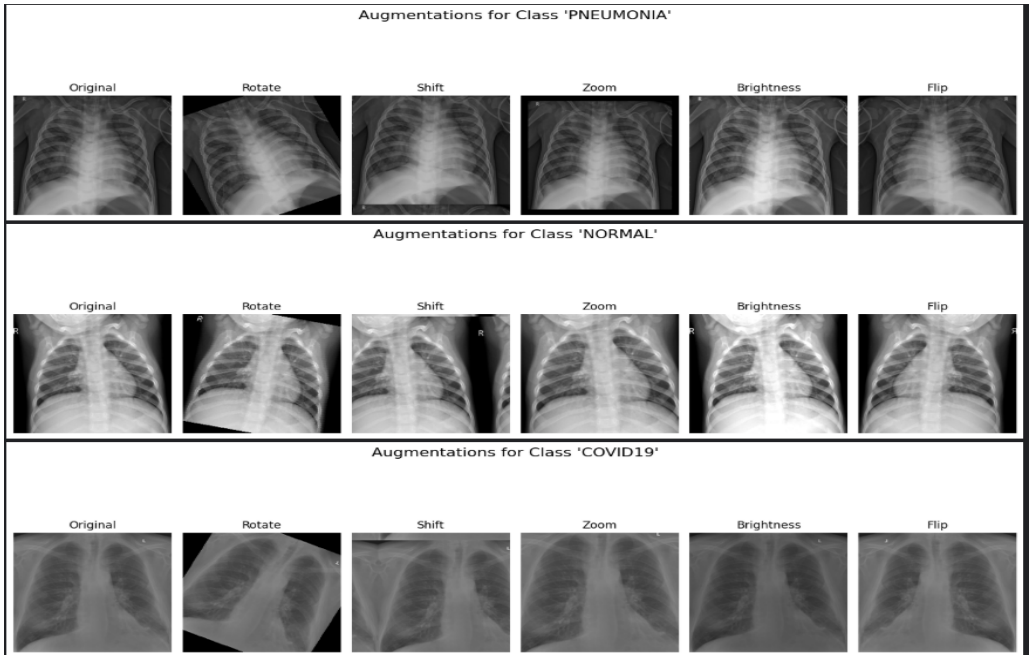
شکل6. پایان فرآیند برای داده‌های test

حال به منظور جلوگیری از انجام هرباره این فرآیند با متوقف شدن session فعال ادیتور از داده‌های ذخیره شده در دایرکتوری خروجی یک دیتاست جدید در سایت Kaggle تشکیل می‌دهیم. برای این کار از Kaggle API و نام کابری خود استفاده می‌کنیم و داده‌ها را در دیتاست تشکیل شده قرار می‌دهیم:

```
Starting upload for file augmented_train.zip
Warning: Looks like you're using an outdated API Version, please consider updating (server 1.7.4.2 / c
lient 1.6.17)
100%|                                        | 899M/899M [00:11<00:00, 81.1MB/s]
Upload successful: augmented_train.zip (899MB)
Starting upload for file augmented_test.zip
100%|                                        | 263M/263M [00:03<00:00, 79.1MB/s]
Upload successful: augmented_test.zip (263MB)
Starting upload for file README.md
100%|                                        | 116/116 [00:00<00:00, 191B/s]
Upload successful: README.md (116B)
Dataset version is being created. Please check progress at https://www.kaggle.com/alirezanajafimot/aug
mented-chest-xray-covid19-pneumonia
```

شکل7. تشکیل دیتاست در Kaggle

در زیر نمونه‌ای از انجام انواع روش‌های نامبرده برای داده‌افزایی روی یک داده از هر کلاس آورده شده است برای دیدن نمونه های بیشتر نیز دیتاست جدید ایجاد شده به نام augmented-chest-xray-covid19-pneumonia در سایت Kaggle موجود است.



شکل8. نمونه‌ای از دیتای داده افزایی شده

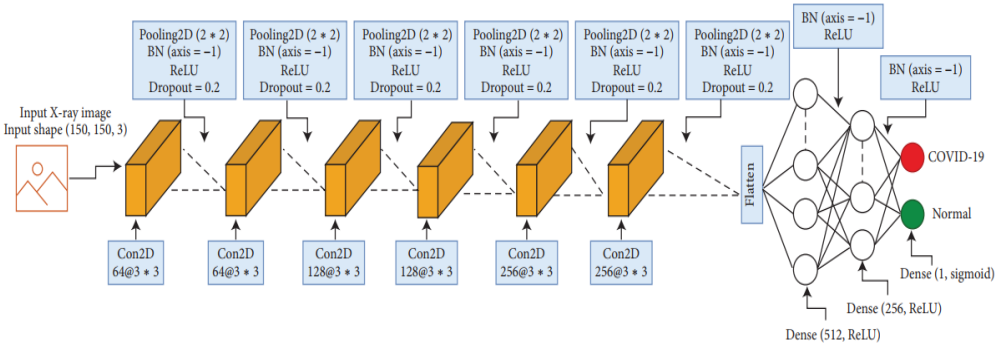
همچنین قبل از مرحله train و test نیاز است تا داده‌های خود را لیبل گذاری کنیم که از روش one-hot-encoding برای داده‌های کتگوریکال استفاده می‌کنیم. همچنین عکس‌ها را به فیچرهایی برحسب numpy array تبدیل می‌کنیم. برای این منظور از تابع‌های image_to_array و to-categorical موجود در tensorflow.keras استفاده می‌کنیم. همچنین چون ورودی مدل ارائه شده در مقاله بصورت 150*150*3 است باید ابتدا داده‌ها را resize کنیم که از تابع load_image برای این منظور استفاده می‌کنیم و سپس تصویر ها را برای ادامه فرآیند آماده‌سازی پاس می‌دهیم. در نهایت فرم کلی داده‌ها به شکل زیر است:

```
Training set shape: (6170, 150, 150, 3) (6170, 3)
Test set shape: (1686, 150, 150, 3) (1686, 3)
Class names: ['COVID19', 'NORMAL', 'PNEUMONIA']
```

شکل9. فرم کلی داده‌ها

4-1. آماده‌سازی مدل

ساختار کلی مدل طراحی شده در مقاله بصورت زیر است:



شکل10. فرم کلی ساختار مدل مقاله

برای طراحی این مدل لایه‌های ابتدایی را کاملا مشابه مقاله و لایه پایانی را چون بجای دو کلاس، سه کلاس داریم بگونه ای تغییر می‌دهیم که شامل سه نورون باشند و همچنین از فعالساز softmax بجای sigmoid استفاده می‌کنیم. همچنین در هنگام compile کردن مدل loss را بجای binary_crossentropy، categorical_crossentropy قرار می‌دهیم. کد استفاده شده و summary مدل ساخته شده در زیر آمده است که مطابق با گفته‌های مقاله است.

```python
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout,BatchNormalization,Input
input_shape=(150,150,3)
filter1_size=64
filter2_size=64
filter3_size=128
filter4_size=128
filter5_size=256
filter6_size=256
filter_shape=(3,3)
pooling_shape=(2,2)
dense_layer1=512
dense_layer2=256
out_layer=len(classes)
dropout_rate=0.2

model = Sequential()
model.add(Input(input_shape))
model.add(Conv2D(filter1_size,filter_shape,activation='relu',padding="same"))
model.add(BatchNormalization(axis=-1))
model.add(MaxPooling2D(pooling_shape))
model.add(Dropout(dropout_rate))
model.add(Conv2D(filter2_size,filter_shape,activation='relu',padding='same'))
model.add(BatchNormalization(axis=-1))
model.add(MaxPooling2D(pooling_shape))
model.add(Dropout(dropout_rate))

model.add(Conv2D(filter3_size,filter_shape,activation='relu',padding='same'))
model.add(BatchNormalization(axis=-1))
model.add(MaxPooling2D(pooling_shape))
model.add(Dropout(dropout_rate))

model.add(Conv2D(filter4_size,filter_shape,activation='relu',padding='same'))
model.add(BatchNormalization(axis=-1))
model.add(MaxPooling2D(pooling_shape))
model.add(Dropout(dropout_rate))

model.add(Conv2D(filter5_size,filter_shape,activation='relu',padding='same'))
model.add(BatchNormalization(axis=-1))
model.add(MaxPooling2D(pooling_shape))
model.add(Dropout(dropout_rate))

model.add(Conv2D(filter6_size,filter_shape,activation='relu',padding='same'))
model.add(BatchNormalization(axis=-1))
model.add(MaxPooling2D(pooling_shape))
model.add(Dropout(dropout_rate))
model.add(GlobalAveragePooling2D())
model.add(Flatten())

model.add(Dense(dense_layer1,activation='relu'))
model.add(Dense(dense_layer2,activation='relu'))
model.add(Dense(out_layer,activation='softmax'))
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 150, 150, 64)	1,792
batch_normalization_12 (BatchNormalization)	(None, 150, 150, 64)	256
max_pooling2d_12 (MaxPooling2D)	(None, 75, 75, 64)	0
dropout_11 (Dropout)	(None, 75, 75, 64)	0
conv2d_13 (Conv2D)	(None, 75, 75, 64)	36,928
batch_normalization_13 (BatchNormalization)	(None, 75, 75, 64)	256
max_pooling2d_13 (MaxPooling2D)	(None, 37, 37, 64)	0
dropout_12 (Dropout)	(None, 37, 37, 64)	0
conv2d_14 (Conv2D)	(None, 37, 37, 128)	73,856
batch_normalization_14 (BatchNormalization)	(None, 37, 37, 128)	512
max_pooling2d_14 (MaxPooling2D)	(None, 18, 18, 128)	0
dropout_13 (Dropout)	(None, 18, 18, 128)	0
conv2d_15 (Conv2D)	(None, 18, 18, 128)	147,584
batch_normalization_15 (BatchNormalization)	(None, 18, 18, 128)	512
max_pooling2d_15 (MaxPooling2D)	(None, 9, 9, 128)	0
dropout_14 (Dropout)	(None, 9, 9, 128)	0
conv2d_16 (Conv2D)	(None, 9, 9, 256)	295,168
batch_normalization_16 (BatchNormalization)	(None, 9, 9, 256)	1,024
max_pooling2d_16 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_15 (Dropout)	(None, 4, 4, 256)	0
conv2d_17 (Conv2D)	(None, 4, 4, 256)	590,080
batch_normalization_17 (BatchNormalization)	(None, 4, 4, 256)	1,024
max_pooling2d_17 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_16 (Dropout)	(None, 2, 2, 256)	0
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 256)	0
flatten_1 (Flatten)	(None, 256)	0
dense_1 (Dense)	(None, 512)	131,584
dense_2 (Dense)	(None, 256)	131,328
dense_3 (Dense)	(None, 3)	771

Total params: 1,412,675 (5.39 MB)
Trainable params: 1,410,883 (5.38 MB)
Non-trainable params: 1,792 (7.00 KB)

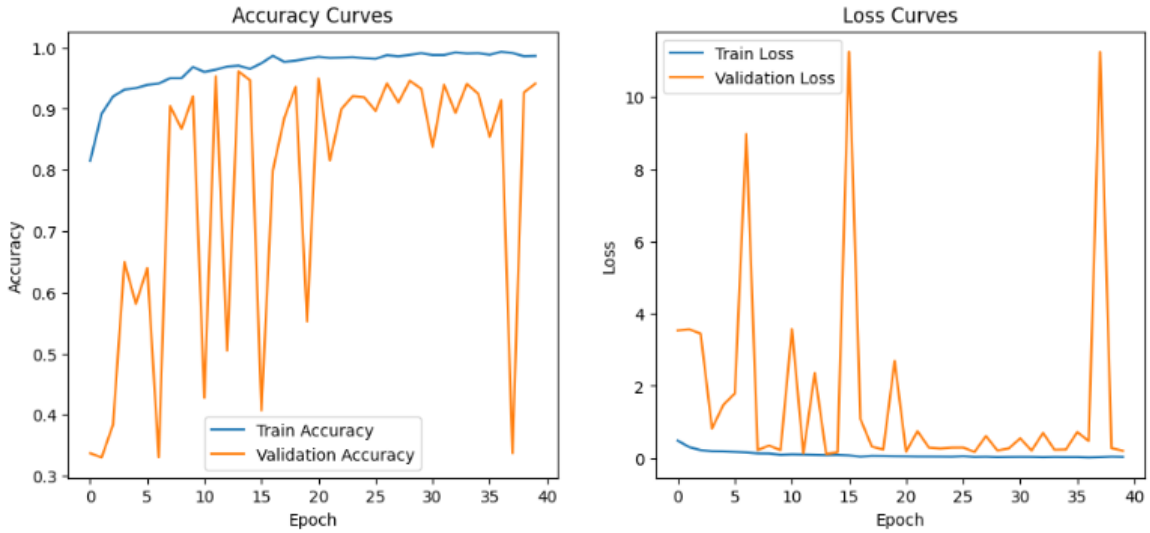
شکل11. خلاصه مدل طراحی شده بر اساس مدل مقاله

نرخ یادگیری متغیر: در این مقاله از نرخ یادگیری متغیر استفاده شده که روش مناسب‌تری از استفاده از یک نرخ یادگیری ثابت در سراسر طول یادگیری آن هم برای چنین دیتاستی می‌باشد. استفاده از نرخ یادگیری متغیر معمولا اینگونه است که optimizer با نرخ‌های یادگیری بالاتر شروع می‌کند و بتدریج نرخ یادگیری کاهش پیدا می‌کند. این به این معناست که نرخ یادگیری بزرگتر در منحنی تغییرات گرادیان در دامنه وسیعتری حرکت دارد و به همین سبب احتمال اینکه در نقاط اکسترمم نسبی گیر بیفتد کاهش می‌یابد. با جلوتر رفتن در روند آموزش و نزدیکتر شدن به اکسترمم واقعی نرخ یادگیری نیز کاهش می‌یابد و به سمت اکسترمم واقعی حرکت می‌کند.

با این حال مدل را با هر دو روش و به ازای چند نرخ یادگیری ثابت آموزش دادیم تا بررسی بهتری داشته باشیم.

در ابتدا مدل اصلی با نرخ یادگیری متغیر و در 40 epoch آموزش داده شد که نتایج بصورت زیر است:

```
Epoch 36/40
126/126 ━━━━━━━━━━━━━━━━━━━━ 7s 52ms/step - accuracy: 0.9885 - loss: 0.0261 - val_accuracy: 0.8542 - val_loss: 0.7229
Epoch 37/40
126/126 ━━━━━━━━━━━━━━━━━━━━ 7s 52ms/step - accuracy: 0.9937 - loss: 0.0162 - val_accuracy: 0.9139 - val_loss: 0.4731
Epoch 38/40
126/126 ━━━━━━━━━━━━━━━━━━━━ 7s 52ms/step - accuracy: 0.9972 - loss: 0.0095 - val_accuracy: 0.3370 - val_loss: 11.2483
Epoch 39/40
126/126 ━━━━━━━━━━━━━━━━━━━━ 7s 53ms/step - accuracy: 0.9821 - loss: 0.0455 - val_accuracy: 0.9269 - val_loss: 0.2772
Epoch 40/40
126/126 ━━━━━━━━━━━━━━━━━━━━ 7s 52ms/step - accuracy: 0.9861 - loss: 0.0343 - val_accuracy: 0.9412 - val_loss: 0.2047
```

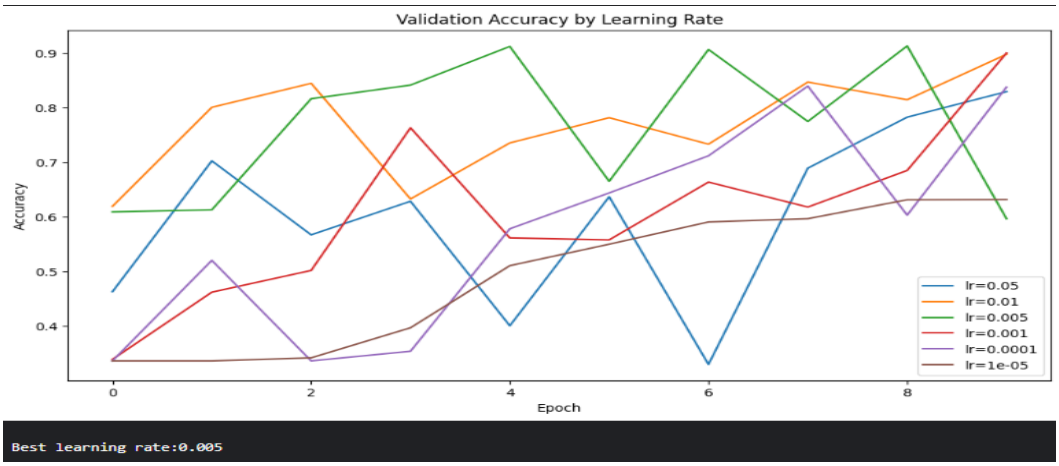


شکل12. نمودار accuracy و loss **برای مدل با نرخ یادگیری متغیر**

پس از آن چند نرخ یادگیری را در 10 epoch آموزش داده، بهترین آن‌ها را انتخاب و در نهایت بهترین نرخ یادگیری ثابت را در 40 epoch آموزش دادیم. همچنین نرخ‌های یادگیری استفاده شده بصورت زیر بودند.

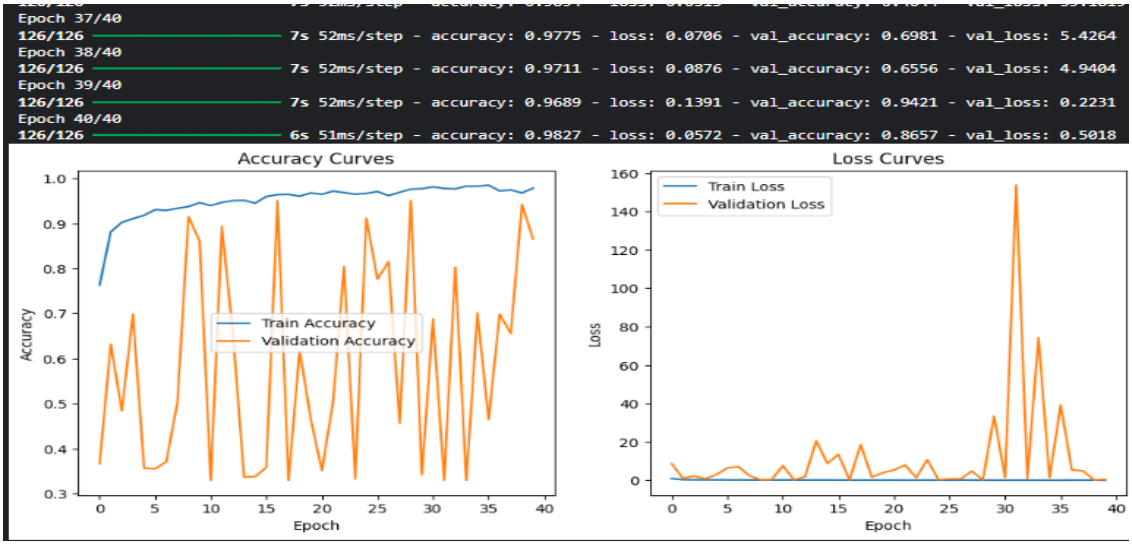
learning_rates=[5e-2,1e-2,5e-3,1e-3,1e-4,1e-5]

نتایج کلی بصورت زیر است:



شکل 13. نمودار validation_accuracy **بر حسب** epoch **بر حسب** lr **های نامبرده**

مشاهده می‌شود که در میان نرخ یادگیری‌های انتخاب شده بهترین آن‌ها 0.005 بوده که نتایج آن بصورت زیر است:



شکل14. نمودار accuracy و loss **برای** lr=0.005

مشاهده می‌شود که مدل با نرخ یادگیری متغیر کمتر overfit شده و مناسبتر است.

5-1. آموزش و ارزیابی مدل

در بخش قبلی مشاهده شد که استفاده از نرخ یادگیری متغیر برای این مدل مناسب‌تر است. همچنین نمودار خطا و دقت در طول روند شامل 40 epoch رسم شد و مشاهده شد دقت حاصله از داده‌های validation با اینکه در نهایت به عدد نسبتا خوبی رسیده است اما روند کلی آن نشان دهنده این است که مدل تاحدی overfit شده است و دقت کلی داده‌های test نیز چیزی شبیه به داده‌های validation خواهد بود. برای افزایش کارایی مدل می‌توانیم نرخ dropout را با جلو رفتن در لایه‌های مدل و با افزایش تعداد پارامترها افزایش دهیم و بجای استفاده از dropout و maxpooling در هر لایه در نصف لایه‌ها از آن‌ها استفاده کنیم تا اطلاعات بیشتری را در برخی مراحل نگه داریم. همچنین در لایه‌های Dense نهایی از regularizer استفاده می‌کنیم که اینجا از l2 استفاده کرده‌ایم تا میزان اورفیت را کاهش دهیم. همچنین سایز فیلترهای بعضی از مراحل را افزایش می‌دهیم و تعداد را در لایه‌های Dense نیز بیشتر می‌کنیم همچنین تابع فعالساز را بجای relu، swish قرار می‌دهیم که جریان گرادیانی بهتری دارد. در نهایت برای کنترل بهتر نرخ یادگیری از callback استفاده می‌کنیم. برای بررسی بهتر epoch را به 50 افزایش می‌دهیم و مجددا مرحله آموزش را تکرار می‌کنیم. با تدابیر اندیشیده شده احتمالا مدل هم در دقت آموزش و هم در دقت تست عملکرد بهتری از مدل مرحله قبل خواهد داشت.

در نهایت کد استفاده شده بصورت زیر است:

```python
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout,BatchNormalization,Input
input_shape=(150,150,3)
filter1_size=64
filter2_size=128
filter3_size=128
filter4_size=256
filter5_size=256
filter6_size=512
filter_shape=(3,3)
pooling_shape=(2,2)
dense_layer1=1024
dense_layer2=512
dense_layer3=128
out_layer=len(classes)
dropout_rate1=0.2
dropout_rate2=0.3
dropout_rate3=0.4

lr_scheduler=tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss',
                                                  factor=0.2,patience=3,min_lr=1e-9)


model = Sequential()
model.add(Input(input_shape))
model.add(Conv2D(filter1_size,filter_shape,activation='relu',padding="same"))
model.add(BatchNormalization(axis=-1))
model.add(MaxPooling2D(pooling_shape))

model.add(Conv2D(filter2_size,filter_shape,activation='relu',padding='same'))
model.add(BatchNormalization(axis=-1))

model.add(Conv2D(filter3_size,filter_shape,activation='relu',padding='same'))
model.add(BatchNormalization(axis=-1))
model.add(MaxPooling2D(pooling_shape))
model.add(Dropout(dropout_rate2))

model.add(Conv2D(filter4_size,filter_shape,activation='relu',padding='same'))
model.add(BatchNormalization(axis=-1))
model.add(MaxPooling2D(pooling_shape))
model.add(Dropout(dropout_rate2))

model.add(Conv2D(filter5_size,filter_shape,activation='relu',padding='same'))
model.add(BatchNormalization(axis=-1))
model.add(Dropout(dropout_rate3))

model.add(Conv2D(filter6_size,filter_shape,activation='relu',padding='same'))
model.add(BatchNormalization(axis=-1))
model.add(Flatten())

model.add(Dense(dense_layer1,activation='swish',kernel_regularizer='l2'))
model.add(Dropout(dropout_rate3))
model.add(Dense(dense_layer2,activation='swish',kernel_regularizer='l2'))
model.add(Dense(out_layer,activation='softmax'))
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
model.summary()
```

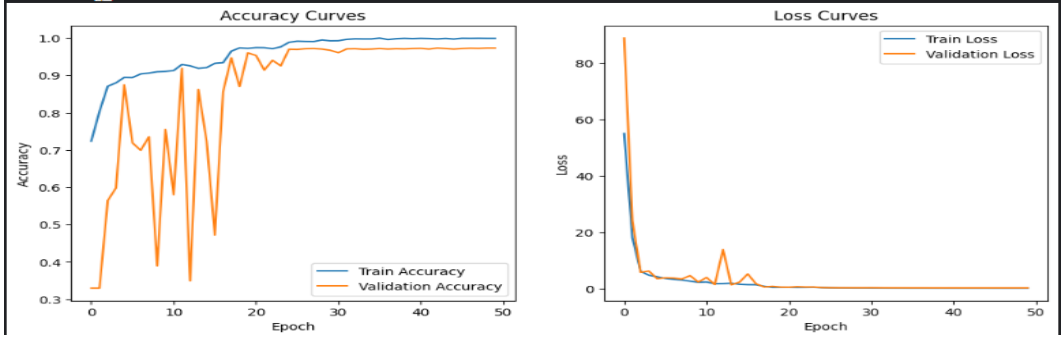
خلاصه مدل طراحی شده نیز در زیر آمده است:

Layer (type)	Output Shape	Param #
conv2d_95 (Conv2D)	(None, 150, 150, 64)	1,792
batch_normalization_90 (BatchNormalization)	(None, 150, 150, 64)	256
max_pooling2d_78 (MaxPooling2D)	(None, 75, 75, 64)	0
conv2d_96 (Conv2D)	(None, 75, 75, 128)	73,856
batch_normalization_91 (BatchNormalization)	(None, 75, 75, 128)	512
conv2d_97 (Conv2D)	(None, 75, 75, 128)	147,584
batch_normalization_92 (BatchNormalization)	(None, 75, 75, 128)	512
max_pooling2d_79 (MaxPooling2D)	(None, 37, 37, 128)	0
dropout_78 (Dropout)	(None, 37, 37, 128)	0
conv2d_98 (Conv2D)	(None, 37, 37, 256)	295,168
batch_normalization_93 (BatchNormalization)	(None, 37, 37, 256)	1,024
max_pooling2d_80 (MaxPooling2D)	(None, 18, 18, 256)	0
dropout_79 (Dropout)	(None, 18, 18, 256)	0
conv2d_99 (Conv2D)	(None, 18, 18, 256)	590,080
batch_normalization_94 (BatchNormalization)	(None, 18, 18, 256)	1,024
dropout_80 (Dropout)	(None, 18, 18, 256)	0
conv2d_100 (Conv2D)	(None, 18, 18, 512)	1,180,160
batch_normalization_95 (BatchNormalization)	(None, 18, 18, 512)	2,048
flatten_9 (Flatten)	(None, 165888)	0
dense_28 (Dense)	(None, 1024)	169,870,336
dropout_81 (Dropout)	(None, 1024)	0
dense_29 (Dense)	(None, 512)	524,800
dense_30 (Dense)	(None, 3)	1,539

شکل15. خلاصه مدل نهایی

نمودار خطا و دقت در طول روند آموزش را برای مدل نهایی نیز به تصویر می‌کشیم:

```
Epoch 46/50
126/126 ━━━━━━━━━━━━━━━━━━━━ 23s 183ms/step - accuracy: 0.9992 - loss: 0.1292 - val_accuracy: 0.9718 - val_loss: 0.2203
- learning_rate: 1.6000e-06
Epoch 47/50
126/126 ━━━━━━━━━━━━━━━━━━━━ 23s 182ms/step - accuracy: 0.9983 - loss: 0.1262 - val_accuracy: 0.9727 - val_loss: 0.2201
- learning_rate: 1.6000e-06
Epoch 48/50
126/126 ━━━━━━━━━━━━━━━━━━━━ 23s 183ms/step - accuracy: 0.9995 - loss: 0.1244 - val_accuracy: 0.9722 - val_loss: 0.2204
- learning_rate: 1.6000e-06
Epoch 49/50
126/126 ━━━━━━━━━━━━━━━━━━━━ 23s 183ms/step - accuracy: 0.9994 - loss: 0.1259 - val_accuracy: 0.9731 - val_loss: 0.2190
- learning_rate: 3.2000e-07
Epoch 50/50
126/126 ━━━━━━━━━━━━━━━━━━━━ 23s 182ms/step - accuracy: 0.9993 - loss: 0.1257 - val_accuracy: 0.9731 - val_loss: 0.2183
- learning_rate: 3.2000e-07
```



شکل16. نمودار accuracy و loss برای مدل نهایی

حال مدل را با معیارهای ارزیابی گفته شده می‌سنجیم:

```
Test Accuracy:0.9543
Precision:[0.97891037 0.93628319 0.94746377]
Recall:[0.99464286 0.93960924 0.92895204]
F1-Score:[0.98671391 0.93794326 0.93811659]

Classification Report:
              precision    recall  f1-score   support

     COVID19       0.98      0.99      0.99       560
      NORMAL       0.94      0.94      0.94       563
   PNEUMONIA       0.95      0.93      0.94       563

    accuracy                           0.95      1686
   macro avg       0.95      0.95      0.95      1686
weighted avg       0.95      0.95      0.95      1686
```

شکل17. معیارهای خواسته شده برای مدل نهایی

مشاهده می‌شود دقت کلی در داده‌های تست نیز تاحد خوبی به دقت داده‌های validation نزدیک است.

حال به توضیح اینکه هر پارامتر چه چیزی را توصیف می‌کند می‌پردازیم:

Accuracy: این معیار در واقع ارزیابی می‌کند که نسبت عناصری که درست تشخیص داده شده‌اند به کل داده‌های ارزیابی شده چقدر است. این معیار به این دلیل در داده‌های نامتوازن معیار مناسبی نیست که اگر کلاس بزرگتر ما برای مثال 95٪ دادگان را تشکیل داده باشد و ما مدلی داشته باشیم که در همه حالات کلاس بزرگتر را پیشبینی کند مدلی با 95٪ دقت خواهیم داشت درحالیکه این مدل تشخیص خاصی انجام نمی‌دهد. در مسائلی مانند تشخیص تقلب یا بیماری تشخیص کلاس کوچکتر از اهمیت بیشتری برخوردار است که در معیار Accuracy میزان موفقیت در تشخیص این نوع دادگان بخوبی ارزیابی نمی‌شود اما در اینجا چون کلاس‌ها با نسبت تقریبا برابری توزیع شده‌اند معیار Accuracy نیز معیار مناسبی خواهد بود.

فرمول Accuracy برای حالت باینری:

$$Accuracy = \frac{(TP + TN)}{TP + FP + FN + TN}$$

Precision: یا صحت در واقع بررسی می‌کند که چه مقدار از پاسخ‌های مثبت بدست آمده بدرستی مثبت بوده‌اند. زمانیکه ما داده‌هایی درکلاس‌های نامتوازن داریم یکی از معیارهایی که در ارزیابی به ما کمک می‌کند این معیار است چرا که داده‌های کلاس مثبت که معمولا وزن کمتری در داده‌های کلی یک جامعه را دارند بررسی می‌کند. در واقع به این معناست که اگر فردی متقلب یا بیمار تشخیص داده شد این تشخیص با چه احتمالی درست است. در اینجا چون سه کلاس داریم بهتر است این معیار برای هر کلاس بصورت جداگانه بررسی شود برای مثال Precision برای کلاس COVID-19 به این معناست که چه مقدار از داده‌هایی که عضو این کلاس تشخیص داده شده‌اند با چه احتمالی درست تشخیص داده شده‌است.

فرمول Precision برای حالت باینری:

$$Precision = \frac{TP}{TP + FP}$$

Recall: یا بازخوانی یک معیار دیگر ارزیابی است که تمرکز آن روی این است که ببیند چه مقدار از داده‌هایی که واقعا بیمار یا متقلب بوده‌اند تشخیص داده شده‌اند. بالا بودن آن به این معناست که افراد بیمار به نسبت خوبی شناسایی شده‌اند اما پایین بودن آن به این معنا خواهد بود که درصد کمی از افرادی که

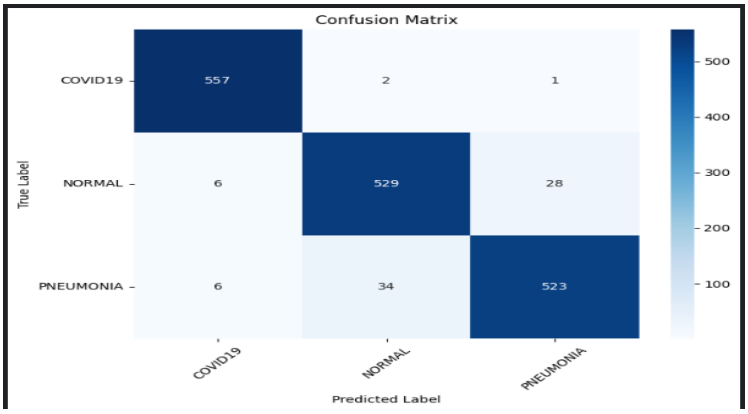
واقعا بیمار بوده‌اند را توانسته‌ایم تشخیص دهیم. این معیار را نیز بهتر است در اینجا برای هر سه کلاس جداگانه بررسی کنیم چرا که هم تشخیص بیمار کرونایی و هم کسی مه دارای ذات‌الریه است دارای اهمیت است. برای مثال در اینجا برای کلاس COVID-19 بیان کننده این است که چه مقدار از داده‌هایی که واقعا دارای کرونا بوده‌اند تشخیص داده شده‌اند و بالا بودن آن نیز به این معناست که افراد دارای کرونا بخوبی تشخیص داده شده‌اند.

F1-Score: نیز در واقع ترکیب یا میانگین هارمونیکی از معیارهای صحت و بازخوانی است که هر دوی این معیارها در مسائلی با توزیع نامتوازن کلاس‌ها معیارهای خوبی برای ارزیابی هستند. در حالت کلی این معیار ترکیبی از هر دو معیار یاد شده است که آن‌هم در اینجا برای هر کلاس بصورت جداگانه بررسی می‌شود.

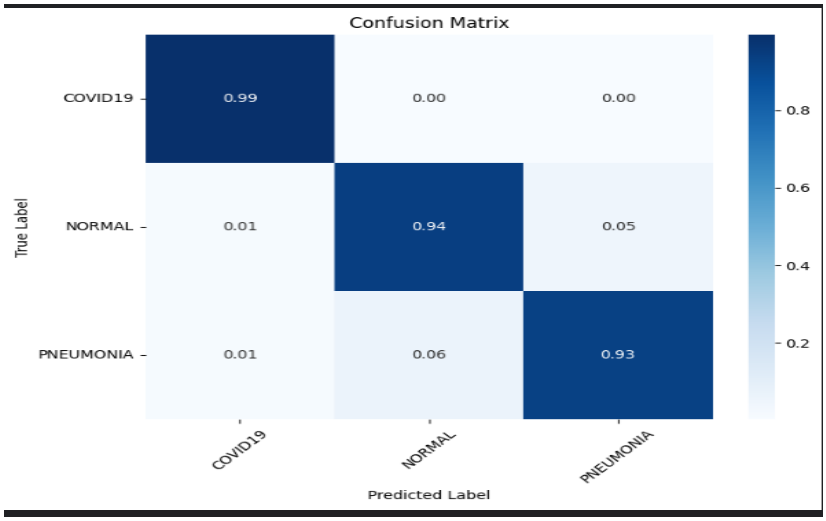
فرمول F1-Score:

$$F1_{score} = \frac{2 * Precision * Recall}{Precision + Recall}$$

حال به رسم ماتریس آشفتگی برای مدل نهایی می‌پردازیم:



شکل18. ماتریس آشفتگی برای مدل نهایی



شکل19. ماتریس آشفتگی نرمالایز شده برای مدل نهایی

مشاهده می‌شود که کلاس COVID-19 بهتر از دو کلاس دیگر تشخیص داده شده است که تا حدودی مطلوب ما نیز هست چرا که تشخیص بیمار و درمان آن براساس بیماری دارای اهمیت فراوانی است. دو

کلاس دیگر تقریبا به یک اندازه اما کلاس NORMAL اندکی سخت‌تر تشخیص داده شده است که بازهم می‌توان گفت مطلوب است چرا که درصورت بیمار تشخیص دادن یک فرد سالم درنهایت سالم بودن او محرض خواهد شد اما اگر یک بیمار دیر تشخیص داده شود ممکن است این امر آن فرد را در معرض خطر بیشتری قرار دهد.

در نهایت می‌توان گفت مدل دقت و قدرت تعمیم خوبی پیدا کرده است و می‌توان از آن استفاده کرد.

6-1 . یادگیری انتقالی

Transfer Leraning یا یادگیری انتقالی تکنیک استفاده از یک مدل پیشتر آموزش دیده(pretrained) که روی یک دیتاست بسیار بزرگ و به نوعی مرجع آموزش دیده شده‌است می‌باشد بخصوص زمانی که داده‌های در دسترس ما برای آموزش یک مدل جدید و قوی به اندازه کافی نباشد. به همین دلیل این روش تاحدودی زمان را کاهش و همچنین محاسبات را برای ما تاحدودی بهینه تر می‌کند(از لحاظ طراحی مجدد مدل وگرنه زمان و محاسبات مربوط به آموزش ممکن است افزایش پیدا کند.

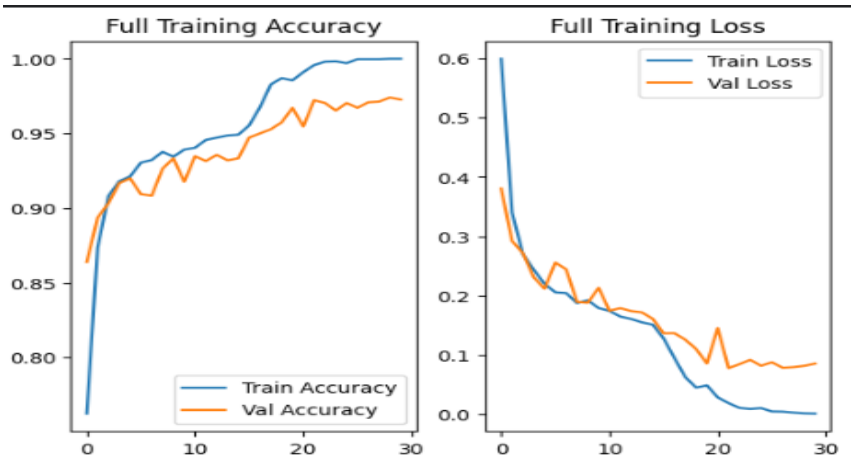
در یادگیری انتقالی بخش‌هایی از مدل که مربوط بخش های میانی کانولوشنی آن هستند حفظ و مجددا استفاده می‌شوند و لایه‌های ابتدایی یا سر مدل پیشین که برای استخراج ویژگی‌های تسک منبع ایجاد شده بودند را استفاده نخواهیم کرد و این بخش را متناسب با تسک خودمان دوباره تشکیل خواهیم داد همچنین لایه نهایی را نیز با توجه به تعداد کلاس‌های خودمان تغییر می‌دهیم.

در حالت کلی برای استفاده از مدل‌های از پیش آموزش داده شده پس از اینکه داده‌ها را مطابق با مدل تغییر دادیم (مثلا برای VGG16 بجای RGB باید از BGR استفاده کنیم و داده باید بین 0و1 باشد و برای MobileNetV2 نیز داده در باز -1و1 مورد قبول خواهد بود)، ابتدا یک سر جدید برای مدل تشکیل داده و یکبار با تعداد epoch پایین بدون اینکه وزن‌های آن‌ها را تغییر دهیم مدل را آموزش می‌دهیم و در مرحله بعد تعدادی از لایه‌های مدل را قابل آموزش مجدد کرده و دوباره فرآیند را اینبار با تعداد epoch بیشتر ادامه می‌دهیم تا به مدل نهایی برسیم که به این فرآیند fine-tune کردن می‌گویند.

کد استفاده شده برای VGG16:

```
vgg_base=VGG16(weights='imagenet',include_top=False,input_shape=(224,224,3))
vgg_base.trainable = False

inputs=Input(shape=(224,224,3))
x=vgg_base(inputs,training=False)
x=GlobalAveragePooling2D()(x)
x=Dense(512,activation='relu')(x)
x=Dropout(0.5)(x)
predictions=Dense(num_cls,activation='softmax')(x)

vgg_model=Model(inputs=inputs,outputs=predictions)

vgg_model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
trained_vgg=vgg_model.fit(x_train,y_train,epochs=15,validation_split=0.35,batch_size=32)

vgg_base.trainable=True
num_layers=len(vgg_base.layers)

for layer in vgg_base.layers[:15]:
    layer.trainable=False

vgg_model.compile(optimizer=Adam(learning_rate=1e-5),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
fine_trained_vgg=vgg_model.fit(x_train,y_train,epochs=30,initial_epoch=15,
                   validation_split=0.35,batch_size=32)
```

نتیجه حاصل از آموزش این مدل را رسم می‌کنیم:



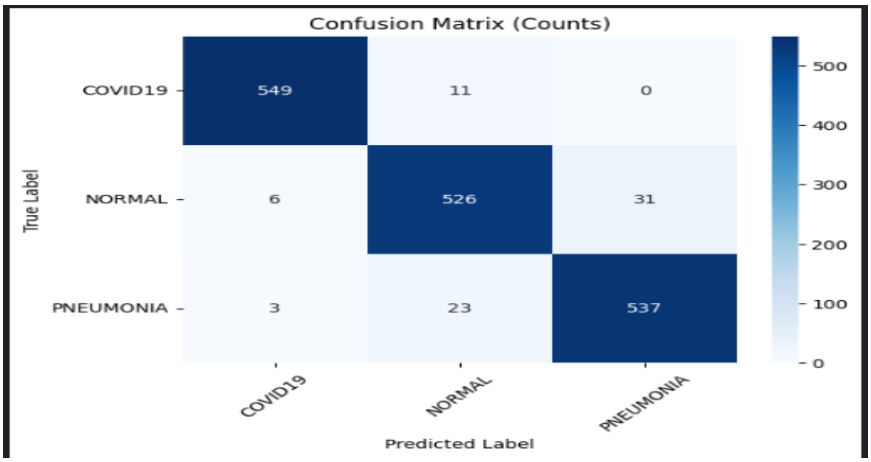
شکل20. نمودار accuracy و loss برای VGG16

حال معیارهای ارزیابی خواسته شده برای این مدل را بدست می‌آوریم:

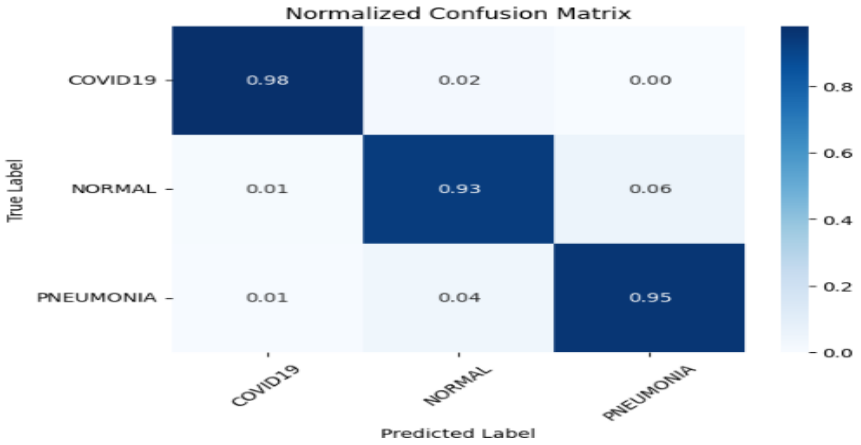
```
Test Accuracy: 0.9561091340450771
Precision: [0.98387097 0.93928571 0.94542254]
Recall: [0.98035714 0.93428064 0.95381883]
F1-Score: [0.98211091 0.93677649 0.94960212]

Classification Report:
              precision    recall  f1-score   support

     COVID19       0.98      0.98      0.98       560
      NORMAL       0.94      0.93      0.94       563
   PNEUMONIA       0.95      0.95      0.95       563

    accuracy                           0.96      1686
   macro avg       0.96      0.96      0.96      1686
weighted avg       0.96      0.96      0.96      1686
```

شکل21. معیارهای ارزیابی برای VGG16

ماتریس آشفتگی را نیز برای این مدل رسم می‌کنیم:



شکل22. ماتریس آشفتگی برای VGG16



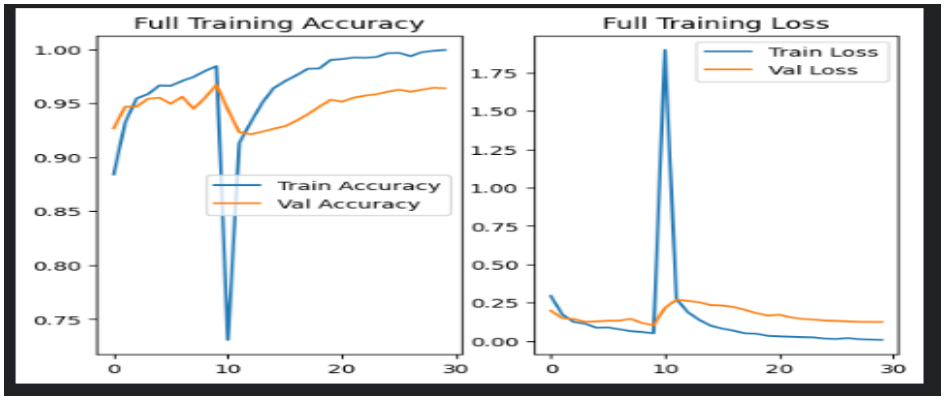
شکل23. ماتریس آشفتگی نرمالایز شده برای VGG16

مشاهده می‌شود که این مدل دقت درحدود 95.6 درصد را داراست. همچنین کلاس COVID-19 را بهتر از سایر کلاس‌ها تشخیص داده و کلاس را NORMAL را سختتر تشخیص می‌دهد.

کد مورد استفاده برای Mobile:

```
mobilenet_base=MobileNetV2(weights='imagenet',include_top=False,input_shape=(224,224,3))
mobilenet_base.trainable = False

x=mobilenet_base.output
x=GlobalAveragePooling2D()(x)
x=Dense(256,activation='relu')(x)
x=Dropout(0.3)(x)
predictions=Dense(num_cls,activation='softmax')(x)

mobilenet_model=Model(inputs=mobilenet_base.input,outputs=predictions)

mobilenet_model.compile(optimizer=Adam(learning_rate=1e-3),
                        loss='categorical_crossentropy',
                        metrics=['accuracy'])
trained_mobilenet=mobilenet_model.fit(train_generator,batch_size=batch_size,
    epochs=10,validation_data=val_generator)

mobilenet_base.trainable=True
num_layers=len(mobilenet_base.layers)
fine_tune_at=int(num_layers*0.7)

for layer in mobilenet_base.layers[:fine_tune_at]:
    layer.trainable=False

mobilenet_model.compile(optimizer=Adam(learning_rate=1e-5),
                        loss='categorical_crossentropy',
                        metrics=['accuracy'])

history_mobilenet_fine = mobilenet_model.fit(
    train_generator,epochs=30,initial_epoch=10,
    validation_data=val_generator,batch_size=batch_size)
```

نتیجه حاصله از آموزش این مدل را رسم می‌کنیم:



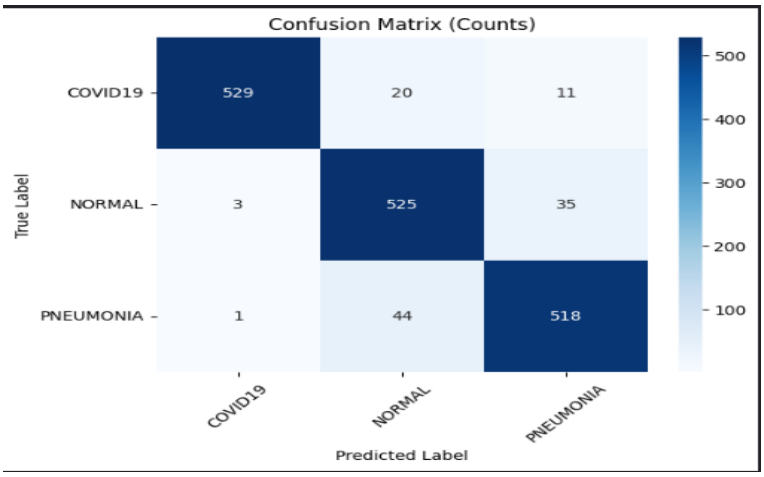
شکل24. نمودار Accuracy **و** loss **برای** MobileNetV2

حال معیارهای ارزیابی خواسته شده برای این مدل را بدست می‌آوریم:

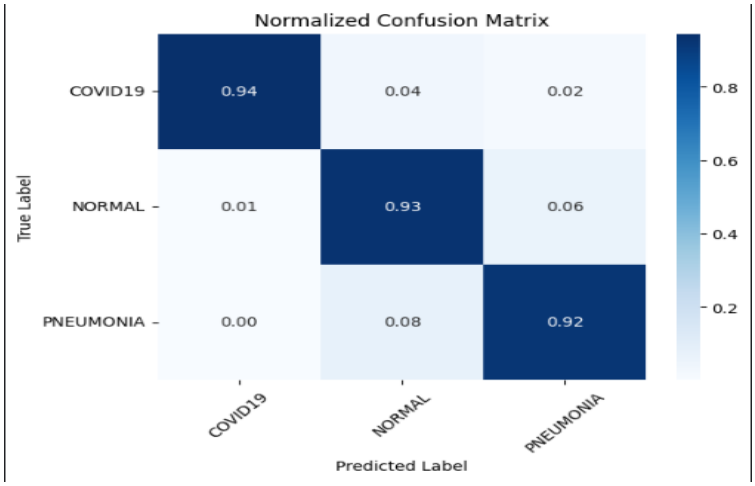
```
Test Accuracy: 0.9323843416370107
Precision: [0.99249531 0.89134126 0.91843972]
Recall: [0.94464286 0.93250444 0.92007105]
F1-Score: [0.96797804 0.91145833 0.91925466]

Classification Report:
              precision    recall  f1-score   support

     COVID19       0.99      0.94      0.97       560
      NORMAL       0.89      0.93      0.91       563
   PNEUMONIA       0.92      0.92      0.92       563

    accuracy                           0.93      1686
   macro avg       0.93      0.93      0.93      1686
weighted avg       0.93      0.93      0.93      1686
```

شکل25. معیارهای ارزیابی برای MobileNetV2

حال ماتریس آشفتگی را رسم می‌کنیم:



شکل26. ماتریس آشفتگی برای MobileNetV2



شکل27. ماتریس آشفتگی نرمالایز شده برای MobileNetV2

همانطور که مشاهده می‌شود COVID-19 را بهتر از بقیه و براساس معیار recall، PNEUMONIA را سختتر پیدا می‌کند.(البته در مجموع معیارها باز هم NORMAL عملکرد بدتری دارد)

حال یک جدول برای مقایسه کلی این سه مدل با یکدیگر تشکیل می‌دهیم:

نکته: داده ها بصورت [COVID-19 NORMAL PNEUMONIA] می‌باشند.

نام مدل	CNN	VGG16	MobileNetV2
Accuracy	0.9543	0.9561	0.9324
Precision	[0.97891037 0.93628319 0.94746377]	[0.98387097 0.93928571 0.94542254]	[0.99249531 0.89134126 0.91843972]
Recall	[0.99464286 0.93960924 0.92895204]	[0.98035714 0.93428064 0.95381883]	[0.94464286 0.93250444 0.92007105]
F1-Score	[0.98671391 0.93794326 0.93811659]	[0.98211091 0.93677649 0.94960212]	[0.96797804 0.91145833 0.91925466]
بهترین کلاس تشخیصی	COVID-19	COVID-19	COVID-19
بدترین کلاس تشخیصی	NORMAL	NORMAL	PNEUMONIA

جدول1. مقایسه سه مدل با یکدیگر

همانطور که مشاهده می‌شود اعداد برای هر سه مدل به یکدیگر نزدیک هستند و تفاوت فاحشی با یکدیگر ندارند هرچند که عملکرد MobileNetV2 اندکی از دو مدل دیگر ضعیفتر بوده است. همچنین مدل CNN از نظر F1-score و Recall در مجموع مقداری از VGG16 بهتر است اما در موجموع عملکرد نسبتا مشابهی دارند.

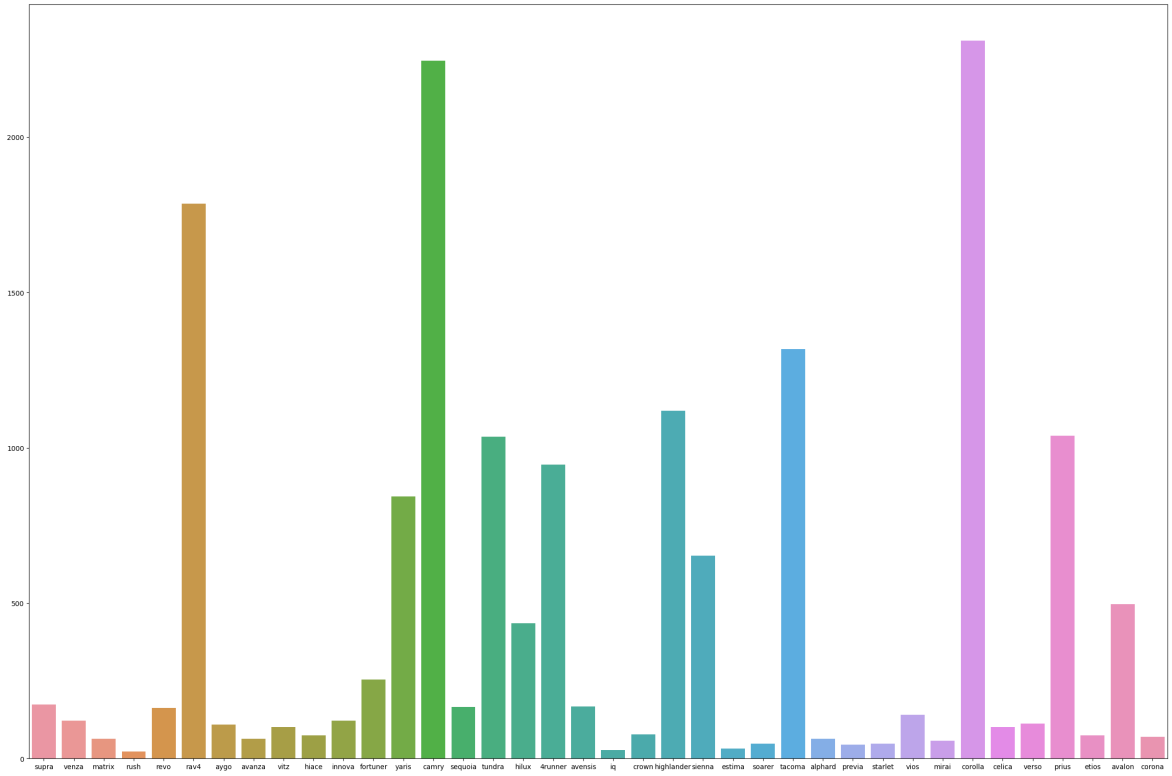
مهم‌ترین مزیت مدل ارائه شده در مقاله حداقل برای این دیتاست جدید این بود که چون به نسبت ساختار ساده‌تری از مدل‌های VGG16 و MobileNetV2 داشت زمان مورد نیاز برای آموزش آن بسیار کمتر از دو مدل دیگر بود و احتمالا بتوان از آن برای کاربردهای realtime استفاده کرد اما مزیت دو مدل transfer learning نیز در این است که اگر ما زمان یا سواد کافی برای طراحی یک مدل جدید برای این امر نداشتیم می‌توانیم از این مدل‌ها استفاده کنیم و مشاهده می‌شود که دقت خوبی نیز دارند.

با وجود چنین مدل‌های از پیش طراحی شده ای پژوهشگران باز هم به طراحی مدل می‌پردازند چرا که این نوع مدل‌ها معمولا روی دیتاست هایی ترین شده‌اند که شاید تصاویر و ویژگی‌هایشان با داده‌های هدف ما متفاوت باشد و بنابراین ممکن است ویژگی‌های که استخراج می‌شوند تاثیر مناسبی نداشته باشند. همچنین پارامترهای موجود در چنین مدل‌هایی معمولا تعداد بسیار زیادی دارند و اجرای مجدد و آموزش مجدد آن‌ها علاوه بر زمانبر بودن به سیستم قوی نیاز دارد بنابراین برای کاربردهای مثلا real-time که به پاسخگویی سریع نیاز داریم استفاده از مدل‌هایی با طراحی ساده‌تر و بهینه تر مناسبتر است که اصلی ترین دلیل در این حوزه نیز همین مطلب است. دلایل دیگری مانند برخی تسک‌ها نیاز به نگاه ریزبینانه ای به برخی ویژگی‌ها دارند که یک مدل یادگیری انتقالی ممکن است به آن‌ها توجه نکرده باشد.

پرسش۲. پیاده‌سازی یک سیستم طبقه‌بندی خودرو با استفاده از VGG16 و SVM

2-2. پیش پردازش داده ها

ابتدا داده های مورد نیز را از طریق لینک داده شده در نوت بوک kaggle به عنوان ورودی آپلود کرده سپس به بررسی فراوانی هر خودرو در دیتاست و تبدیل برچسب هر خودرو به مقادیر عددی می پردازیم :



شکل28 .نمودار فراوانی هر دسته در دیتاست

Folder: toyota_image_dataset_v2, Files: 4

Folder: toyota_cars, Files: 0

Folder: supra, Files: 173

Folder: venza, Files: 122

Folder: matrix, Files: 63

Folder: rush, Files: 23

Folder: revo, Files: 162

Folder: rav4, Files: 1786

Folder: aygo, Files: 109

Folder: avanza, Files: 63

Folder: vitz, Files: 102

Folder: hiace, Files: 75

Folder: innova, Files: 121

Folder: fortuner, Files: 254

Folder: yaris, Files: 844

Folder: camry, Files: 2246

Folder: sequoia, Files: 166

Folder: tundra, Files: 1035

Folder: hilux, Files: 435

Folder: 4runner, Files: 946

Folder: avensis, Files: 167

Folder: iq, Files: 27

Folder: crown, Files: 77

Folder: highlander, Files: 1119

Folder: sienna, Files: 652

Folder: estima, Files: 32

Folder: soarer, Files: 48

Folder: tacoma, Files: 1318

Folder: alphard, Files: 64

Folder: previa, Files: 44

Folder: starlet, Files: 48

Folder: vios, Files: 141

Folder: mirai, Files: 57

Folder: corolla, Files: 2311

Folder: celica, Files: 101

Folder: verso, Files: 112

Folder: prius, Files: 1039

Folder: etios, Files: 74

Folder: avalon, Files: 497

Folder: corona, Files: 70

Number of folders: 39

Total number of files: 16727

names_to_labels:

```
{'supra': 0,
 'venza': 1,
 'matrix': 2,
 'rush': 3,
 'revo': 4,
 'rav4': 5,
 'aygo': 6,
 'avanza': 7,
 'vitz': 8,
 'hiace': 9,
 'innova': 10,
 'fortuner': 11,
 'yaris': 12,
 'camry': 13,
 'sequoia': 14,
 'tundra': 15,
 'hilux': 16,
 '4runner': 17,
 'avensis': 18,
 'iq': 19,
 'crown': 20,
 'highlander': 21,
 'sienna': 22,
 'estima': 23,
 'soarer': 24,
 'tacoma': 25,
 'alphard': 26,
 'previa': 27,
 'starlet': 28,
 'vios': 29,
 'mirai': 30,
 'corolla': 31,
 'celica': 32,
 'verso': 33,
 'prius': 34,
 'etios': 35,
 'avalon': 36,
 'corona': 37}
```

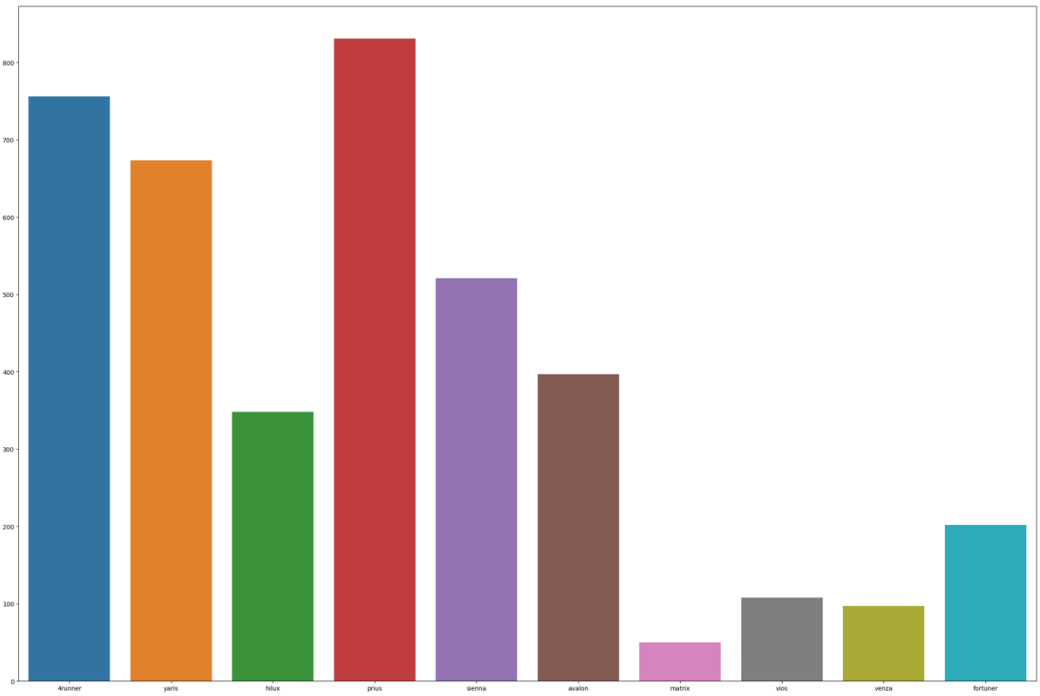
و سپس 10 کلاس زیر را انتخاب کرده به گونه ای که فراوانی های متفاوت را شامل شوند و سپس به تغییر ابعاد و نرمالسازی مقادیر پیکسل ها پرداخته و سپس به منظور جلوگیری از نشت اطلاعات در داده های تست ابتدا داده های هر کلاس را به نسبت ذکر شده به دو دسته داده های آموزش و تست تقسیم می کنیم و سپس به داده افزایی کلاس ها در داده های آموزش و به تعادل رساندن فراوانی کلاس ها در این داده ها می پردازیم :

```
# 10 Class Choose
folders = ['fortuner','hilux','venza','matrix','sienna','avalon','4runner','yaris','prius','vios']
```

```
train_directory = '/kaggle/working/train'
test_directory = '/kaggle/working/test'
output_directory = '/kaggle/working/train_resized'
3950/3990 images processed
```

همانطور که مشاهده می شود برخی از تصاویر قایل فراخوانی و پردازش نبودند و فراخوانی نشدند.

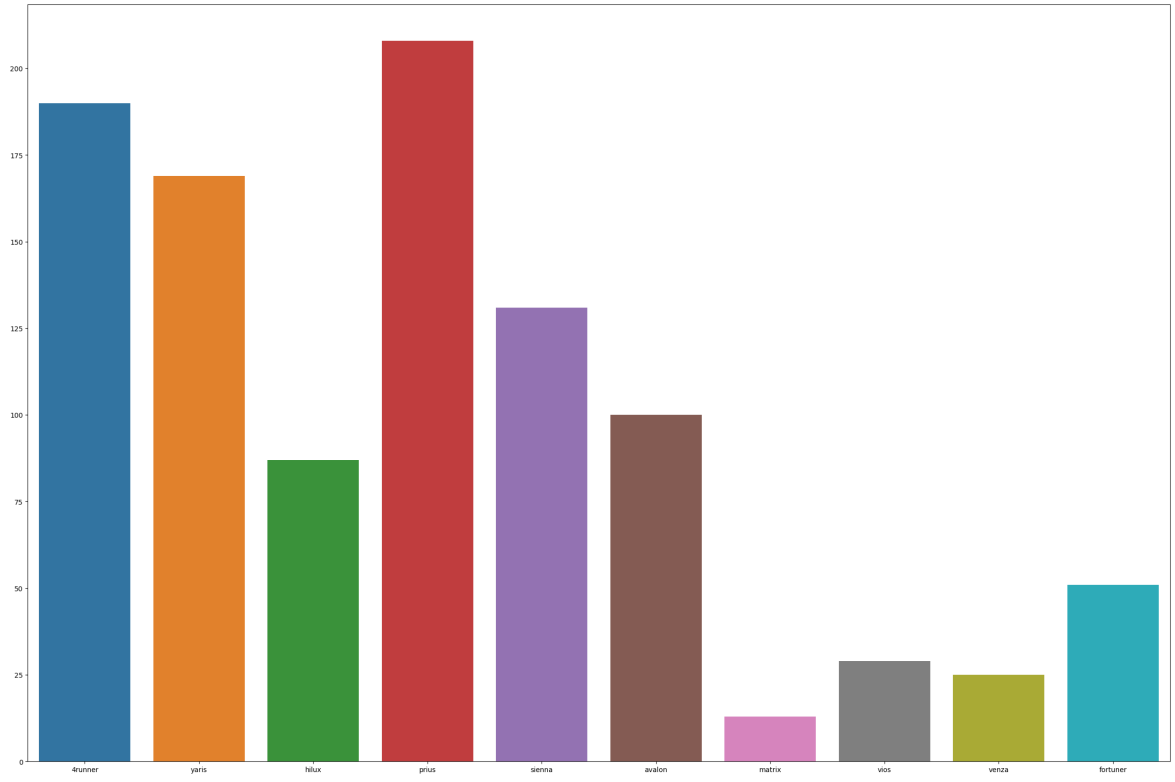
```
output_directory = '/kaggle/working/test_resized'
1000/1003 images processed

output_directory = '/kaggle/working/train_resized'
Folder: 4runner, File Count: 756
Folder: yaris, File Count: 673
Folder: hilux, File Count: 348
Folder: prius, File Count: 831
Folder: sienna, File Count: 521
Folder: avalon, File Count: 397
Folder: matrix, File Count: 50
Folder: vios, File Count: 108
Folder: venza, File Count: 97
Folder: fortuner, File Count: 202
```



شکل ۲۹. فراوانی داده های آموزش برای ۱۰ کلاس انتخابی

output_directory = '/kaggle/working/test_resized'

Folder: 4runner, File Count: 190
Folder: yaris, File Count: 169
Folder: hilux, File Count: 87
Folder: prius, File Count: 208
Folder: sienna, File Count: 131
Folder: avalon, File Count: 100
Folder: matrix, File Count: 13
Folder: vios, File Count: 29
Folder: venza, File Count: 25
Folder: fortuner, File Count: 51



شکل ۳۰. فراوانی داده های تست برای ۱۰ کلاس انتخابی

در ادامه روش های موجود برای به تعادل رساندن داده های آموزش مورد بررسی قرار می گیرد:

۱. افزایش داده‌ها (Data Augmentation)

با استفاده از تبدیل‌های تصویری مانند چرخش، تغییر مقیاس، برش، تغییر روشنایی، نویز و غیره، تصاویر جدیدی از داده‌های موجود تولید می‌شود تا تعداد نمونه‌های کلاس‌های کم‌داده افزایش یابد.

مزایا:

- ساده و کم‌هزینه در پیاده‌سازی.

- تنوع داده‌ها را افزایش می‌دهد و به جلوگیری از بیش‌برازش (Overfitting) کمک می‌کند.
- نیازی به جمع‌آوری داده‌های واقعی جدید ندارد.
- قابل استفاده در کنار سایر روش‌ها.

معایب:

- اگر تنوع تبدیل‌ها محدود باشد، ممکن است داده‌های تولیدشده تفاوت معناداری با داده‌های واقعی نداشته باشند.
- در مواردی که کلاس‌ها بسیار نامتوازن باشند، ممکن است به‌تنهایی کافی نباشد.

کاربرد: مناسب برای زمانی که داده‌های موجود حداقل تنوع اولیه را دارند و می‌توان با تغییرات ساده، داده‌های جدید تولید کرد.

۲. تولید داده‌های مصنوعی (Synthetic Data Generation)

استفاده از مدل‌های مولد مانند GANها (Generative Adversarial Networks) یا Variational Autoencoders برای تولید تصاویر مصنوعی مشابه داده‌های واقعی کلاس‌های کم‌داده.

مزایا:

- می‌تواند داده‌هایی با تنوع بالا تولید کند.
- برای کلاس‌های بسیار کم‌داده که افزایش داده به روش سنتی کافی نیست، مناسب است.
- امکان تولید داده‌های پیچیده‌تر نسبت به Data Augmentation.

معایب:

- نیاز به آموزش مدل‌های پیچیده و منابع محاسباتی بالا.
- کیفیت تصاویر تولیدشده ممکن است پایین باشد یا با داده‌های واقعی تفاوت زیادی داشته باشد.
- خطر اضافه کردن نویز یا داده‌های غیرواقعی به مجموعه داده.

کاربرد: مناسب برای زمانی که داده‌های واقعی بسیار محدود هستند و امکان جمع‌آوری داده جدید وجود ندارد.

۳. وزن‌دهی به کلاس‌ها (Class Weighting)

در هنگام آموزش مدل، به کلاس‌های کم‌داده وزن بیشتری در تابع هزینه (Loss Function) اختصاص داده می‌شود تا تأثیر آن‌ها در یادگیری مدل افزایش یابد.

مزایا:

- نیازی به تغییر داده‌ها یا تولید داده جدید ندارد.
- پیاده‌سازی ساده در اکثر فریم‌ورک‌های یادگیری عمیق (مانند PyTorch و TensorFlow).
- مناسب برای زمانی که نمی‌توان داده‌های جدید تولید کرد.

معایب:

- ممکن است باعث بیش‌برازش به کلاس‌های کم‌داده شود.

- در مواردی که تفاوت تعداد داده‌ها بین کلاس‌ها بسیار زیاد باشد، ممکن است به‌تنهایی کافی نباشد.

کاربرد: مناسب برای زمانی که داده‌ها قابل تغییر نیستند یا منابع محاسباتی محدود است.

۴. نمونه‌برداری (Resampling)

- Oversampling: افزایش تعداد نمونه‌های کلاس کم‌داده با کپی کردن یا تولید نمونه‌های جدید (مانند SMOTE برای داده‌های غیرتصویری).

- Undersampling: کاهش تعداد نمونه‌های کلاس پرداده برای بالانس کردن.

مزایا:

- Oversampling می‌تواند تعداد نمونه‌های کلاس کم‌داده را بدون نیاز به داده واقعی افزایش دهد.

- Undersampling می‌تواند زمان آموزش را کاهش دهد (به دلیل کاهش داده‌ها).

- پیاده‌سازی نسبتاً ساده.

معایب:

- Oversampling ممکن است باعث بیش‌برازش شود (به‌خصوص اگر فقط کپی انجام شود).

- Undersampling ممکن است اطلاعات مهم کلاس پرداده را حذف کند.

- روش‌هایی مانند SMOTE برای داده‌های تصویری پیچیده‌تر هستند و ممکن است نتایج غیرواقعی تولید کنند.

کاربرد: مناسب برای زمانی که تفاوت تعداد داده‌ها بین کلاس‌ها متوسط است و می‌توان تعادل را با تغییرات ساده برقرار کرد.

۵. یادگیری انتقالی (Transfer Learning)

استفاده از مدل‌های از پیش آموزش‌دیده (مانند ResNet، VGG یا EfficientNet) که روی مجموعه داده‌های بزرگ (مانند ImageNet) آموزش دیده‌اند و سپس تنظیم دقیق (Fine-tuning) آن‌ها روی داده‌های محدود.

مزایا:

- نیاز به داده‌های کمتری برای آموزش دارد.

- از دانش عمومی مدل‌های از پیش آموزش‌دیده استفاده می‌کند.

- عملکرد خوبی حتی با داده‌های محدود ارائه می‌دهد.

معایب:

- نیاز به تنظیم دقیق مدل برای داده‌های خاص.

- در صورتی که داده‌های هدف بسیار متفاوت از داده‌های آموزش اولیه مدل باشند، ممکن است عملکرد کاهش یابد.

- منابع محاسباتی برای تنظیم دقیق موردنیاز است.

کاربرد: مناسب برای زمانی که داده‌های بسیار محدود هستند و امکان استفاده از مدل‌های از پیش آموزش‌دیده وجود دارد.

۶. جمع‌آوری داده‌های جدید

جمع‌آوری تصاویر واقعی جدید برای کلاس‌های کم‌داده از منابع مختلف (مانند وب، دوربین‌ها یا پایگاه‌های داده عمومی).

مزایا:

- داده‌های واقعی و باکیفیت به مدل اضافه می‌شود.

- تنوع داده‌ها به‌طور طبیعی افزایش می‌یابد.

- معمولاً بهترین عملکرد را در بلندمدت ارائه می‌دهد.

معایب:

- زمان‌بر و پرهزینه است.

- ممکن است برای برخی کلاس‌ها داده‌های واقعی به‌سختی در دسترس باشند.

- نیاز به برچسب‌گذاری داده‌ها (که خود هزینه‌بر است).

کاربرد: مناسب برای پروژه‌هایی که منابع مالی و زمانی کافی دارند و کیفیت داده‌ها اولویت اصلی است.

انتخاب بهترین روش به عوامل مختلفی بستگی دارد، از جمله:

- تعداد داده‌های موجود: اگر داده‌ها بسیار محدود باشند، یادگیری انتقالی یا تولید داده مصنوعی مناسب‌تر است.

- منابع محاسباتی: روش‌هایی مانند Data Augmentation و Class Weighting منابع کمتری نیاز دارند.

- نوع مسئله: برای مسائل پیچیده با داده‌های خاص، جمع‌آوری داده جدید یا یادگیری انتقالی بهتر است.

- زمان : اگر زمان و منابع محدود باشد، Data Augmentation و Class Weighting سریع‌تر هستند.

توابع استفاده شده برای داده افزایی کلاس ها به صورت زیر است :

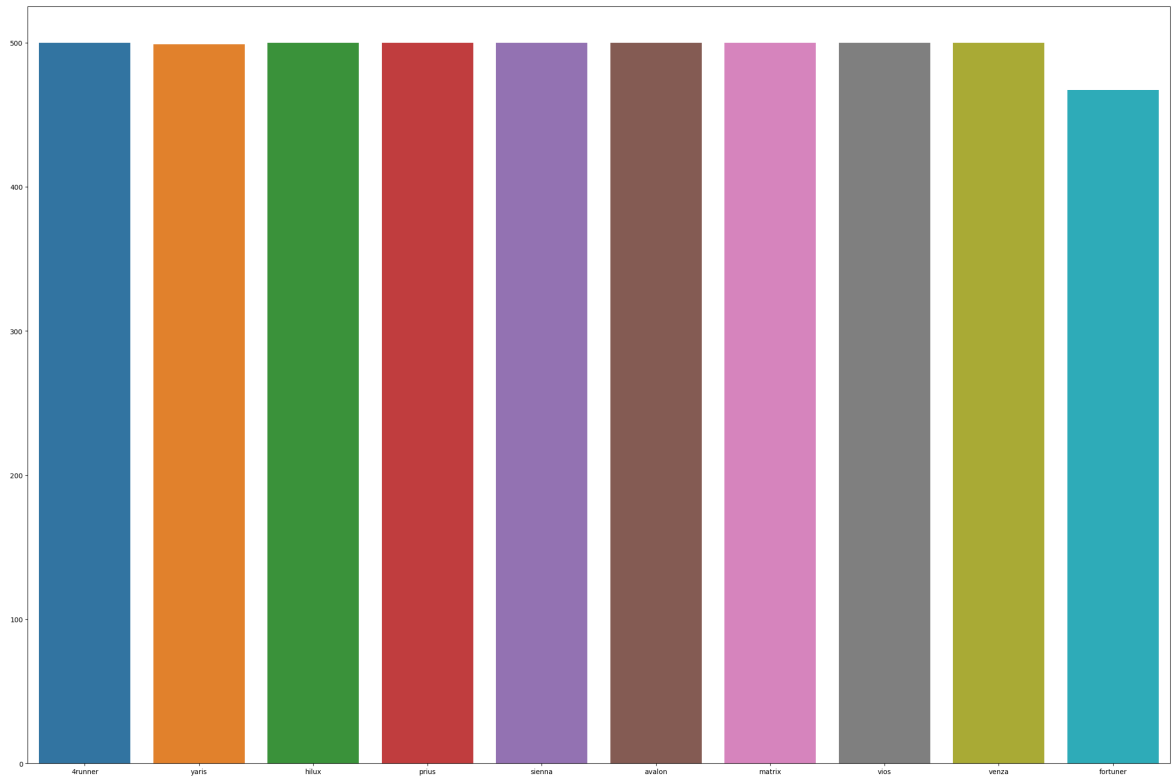
```
# Augmentation functions
def rotate_image(image, angle):
    (h, w) = image.shape[:2]
    center = (w // 2, h // 2)
```

```
    matrix = cv2.getRotationMatrix2D(center, angle, 1.0)
    rotated = cv2.warpAffine(image, matrix, (w, h))
    return rotated

def flip_image(image, flip_code):
    return cv2.flip(image, flip_code)

def scale_image(image, scale_factor):
    h, w = image.shape[:2]
    new_h, new_w = int(h * scale_factor), int(w * scale_factor)
    scaled = cv2.resize(image, (new_w, new_h), interpolation=cv2.INTER_LINEAR)
    return scaled

def translate_image(image, tx, ty):
    matrix = np.float32([[1, 0, tx], [0, 1, ty]])
    translated = cv2.warpAffine(image, matrix, (image.shape[1], image.shape[0]))
    return translated

def adjust_brightness_contrast(image, alpha, beta):
    adjusted = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)
    return adjusted
def add_gaussian_noise(image):
    row, col, ch = image.shape
    mean = 0
    var = random.uniform(0.1, 10.0)
    sigma = var ** 0.5
    gauss = np.random.normal(mean, sigma, (row, col, ch))
    noisy = image + gauss
    noisy = np.clip(noisy, 0, 255).astype(np.uint8)
    return noisy
```

```
def random_crop(image, crop_ratio=0.8):
    h, w = image.shape[:2]
    crop_h, crop_w = int(h * crop_ratio), int(w * crop_ratio)
    if crop_h > h or crop_w > w:
        return image
    x = random.randint(0, w - crop_w)
    y = random.randint(0, h - crop_h)
    cropped = image[y:y + crop_h, x:x + crop_w]
    return cv2.resize(cropped, (w, h))
```

و در نهایت به منظور به تعادل رساندن داده های آموزش برای هر کلاس تعداد داده آموزش هر کلاس را سعی می کنیم به تعداد ۵۰۰ داده برسانیم :

```
Folder: 4runner, File Count: 500
Folder: yaris, File Count: 499
Folder: hilux, File Count: 500
Folder: prius, File Count: 500
Folder: sienna, File Count: 500
Folder: avalon, File Count: 500
Folder: matrix, File Count: 500
Folder: vios, File Count: 500
Folder: venza, File Count: 500
Folder: fortuner, File Count: 467
```



شکل ۳۱. فراوانی داده های آموزش برای ۱۰ کلاس انتخابی بعد از تعادل

3-2. استخراج ویژگی ها

در این مرحله مدل های VGG ۱۶ و AlexNet را بدون لایه های اخر شامل fully connected که مرتبط با دسته بندی ویژگی های استخراج شده هستند مورد استفاده قرار می دهیم تا ویژگی مرتبط با هر تصویر را با استفاده از قدرت این شبکه ها استخراج کنیم و سپس بردار های ویژگی استخراج شده را به صورت یک بعدی ذخیره می کنیم تا در مراحل بعد با استفاده از دسته بند های مناسب بتوانیم از ویژگی های غنی برای دسته بندی این مسیله استفاده کنیم :

```
output_directory = '/kaggle/working/train_resized'
feature_output_directory = '/kaggle/working/feature_embedding_vgg'

os.makedirs(feature_output_directory, exist_ok=True)

# Load VGG16 model
VGGmodel = VGG16(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))


def extract_features(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)

    features = VGGmodel.predict(img_array)

    return features

def extract_and_save_features():
    all_folders = [folder for folder in os.listdir(output_directory)
                   if os.path.isdir(os.path.join(output_directory,
folder))]
    counter = 0

    for folder in all_folders:
        folder_path = os.path.join(output_directory, folder)
        feature_folder_path = os.path.join(feature_output_directory,
folder)
        os.makedirs(feature_folder_path, exist_ok=True)

        image_files = [f for f in os.listdir(folder_path)
                       if f.endswith(('.jpg', '.jpeg', '.png'))]

        print(f"Processing folder: {folder}, {len(image_files)}
images")
        for img_file in image_files:
            img_path = os.path.join(folder_path, img_file)
```

```
            try:
                features = extract_features(img_path)
                feature_file_name = os.path.splitext(img_file)[0] +
'_features.npy'
                feature_file_path = os.path.join(feature_folder_path,
feature_file_name)
                np.save(feature_file_path, features)

                print(f"Saved features for {img_file} at
{feature_file_path}")
                counter += 1
                print(counter)
            except Exception as e:
                print(f"Error processing {img_file}: {str(e)}")


extract_and_save_features()
```

و به طور مشابه با استفاده از مدل alexNet موجود در pytorch :

```
# Load pre-trained AlexNet
model = torch.hub.load('pytorch/vision:v0.10.0', 'alexnet',
pretrained=True)

for param in model.features.parameters():
    param.requires_grad = False
```

4-2. آموزش و ارزیابی مدل

۱ –

توضیح متریک‌های دسته‌بندی (Precision, Recall, F۱-Score, Accuracy)

متریک‌های ارزیابی در مسائل دسته‌بندی (Classification) برای سنجش عملکرد مدل‌های یادگیری ماشین استفاده می‌شوند. متریک‌های اصلی شامل Precision، Accuracy، Recall و F۱-Score هستند. در ادامه، هر یک از این متریک‌ها، نحوه محاسبه و عملکرد آن‌ها توضیح داده شده است.

برای درک متریک‌ها، ابتدا باید اصطلاحات زیر را بشناسیم (بر اساس ماتریس درهم‌ریختگی یا Confusion Matrix):

- True Positive (TP): مواردی که مدل به درستی به عنوان کلاس مثبت پیش‌بینی کرده است.
- True Negative (TN): مواردی که مدل به درستی به عنوان کلاس منفی پیش‌بینی کرده است.
- False Positive (FP): مواردی که مدل به اشتباه به عنوان کلاس مثبت پیش‌بینی کرده است (خطای نوع اول).
- False Negative (FN): مواردی که مدل به اشتباه به عنوان کلاس منفی پیش‌بینی کرده است (خطای نوع دوم).

۱. Accuracy (دقت کلی)

نسبت تعداد پیش‌بینی‌های درست (چه مثبت و چه منفی) به کل نمونه‌ها. این متریک نشان می‌دهد که مدل در کل چند درصد از پیش‌بینی‌ها را درست انجام داده است.

Accuracy = (TP + TN) / (TP + TN + FP + FN)

نحوه عملکرد:

- ساده‌ترین متریک برای ارزیابی مدل.

- در مسائل با کلاس‌های متعادل (تعداد نمونه‌های هر کلاس تقریباً برابر) بسیار مفید است.

- در مسائل نامتوازن (Imbalanced Data) ممکن است گمراه‌کننده باشد. مثلاً اگر ۹۵٪ داده‌ها متعلق به یک کلاس باشند، مدلی که همیشه همان کلاس را پیش‌بینی کند، Accuracy بالایی خواهد داشت، اما عملکرد واقعی آن ضعیف است.

مزایا:

- ساده و قابل‌فهم.

- برای مسائل متعادل مناسب است.

معایب:

- در مسائل نامتوازن گمراه‌کننده است.

- جزئیات عملکرد مدل روی کلاس‌های خاص را نشان نمی‌دهد.

کاربرد: مناسب برای زمانی که کلاس‌ها تقریباً متعادل هستند و هدف، ارزیابی کلی مدل است.

۲. Precision (دقت)

نسبت پیش‌بینی‌های درست مثبت به کل پیش‌بینی‌های مثبت. این متریک نشان می‌دهد که از بین نمونه‌هایی که مدل به عنوان مثبت پیش‌بینی کرده، چند درصد واقعاً مثبت بوده‌اند.

Precision = TP / (TP + FP)

نحوه عملکرد:

- تمرکز روی کاهش خطاهای مثبت کاذب (FP).

- در مواردی که هزینه خطای مثبت کاذب بالا باشد (مثلاً تشخیص اسپم ایمیل که نباید ایمیل‌های معتبر را اسپم تشخیص دهد)، اهمیت بیشتری دارد.

- Precision بالا به این معناست که مدل در شناسایی نمونه‌های مثبت دقیق است.

مزایا:

- برای مسائل نامتوازن که خطای مثبت کاذب مهم است، مفید است.

- دقت پیش‌بینی‌های مثبت را به‌خوبی نشان می‌دهد.

معایب:

- به‌تنهایی کافی نیست، زیرا ممکن است با کاهش Recall همراه باشد (مدل کمتر پیش‌بینی مثبت کند).
- اطلاعاتی درباره نمونه‌های منفی ارائه نمی‌دهد.

کاربرد: مناسب برای مسائلی که خطای مثبت کاذب (FP) هزینه‌بر است، مانند تشخیص بیماری‌های نادر.

۳. Recall

نسبت پیش‌بینی‌های درست مثبت به کل نمونه‌های واقعاً مثبت. این متریک نشان می‌دهد که مدل چند درصد از نمونه‌های مثبت واقعی را به درستی شناسایی کرده است.

Recall = TP / (TP + FN)

نحوه عملکرد:

- تمرکز روی کاهش خطاهای منفی کاذب (FN).
- در مواردی که از دست دادن نمونه‌های مثبت هزینه‌بر باشد (مثلاً تشخیص سرطان که نباید بیماران واقعی را از دست داد)، اهمیت بیشتری دارد.
- Recall بالا به این معناست که مدل اکثر نمونه‌های مثبت را شناسایی کرده است.

مزایا:

- برای مسائل نامتوازن که خطای منفی کاذب مهم است، مفید است.
- توانایی مدل در یافتن تمام نمونه‌های مثبت را نشان می‌دهد.

معایب:

- ممکن است با کاهش Precision همراه باشد (مدل بیش از حد نمونه‌ها را مثبت پیش‌بینی کند).
- اطلاعاتی درباره نمونه‌های منفی ارائه نمی‌دهد.

کاربرد: مناسب برای مسائلی که خطای منفی کاذب (FN) هزینه‌بر است، مانند تشخیص بیماری‌های خطرناک.

۴. F۱-Score

میانگین هارمونیک Precision و Recall. این متریک تعادلی بین Precision و Recall ایجاد می‌کند و زمانی مفید است که بخواهیم هر دو معیار را به طور همزمان در نظر بگیریم.

F۱-Score = ۲ * (Precision * Recall) / (Precision + Recall)

نحوه عملکرد:

- در مسائل نامتوازن که هم Precision و هم Recall مهم هستند، بهترین معیار است.

- اگر یکی از Precision یا Recall بسیار پایین باشد، F۱-Score نیز پایین خواهد بود، بنابراین معیار خوبی برای سنجش تعادل است.

- مقداری بین ۰ و ۱ دارد، که ۱ نشان‌دهنده عملکرد ایده‌آل است.

مزایا:

- تعادل بین Precision و Recall را به‌خوبی نشان می‌دهد.

- برای مسائل نامتوازن بسیار مناسب است.

- معیار جامعی برای مقایسه مدل‌ها ارائه می‌دهد.

معایب:

- ممکن است برای مسائل خاص که یکی از Precision یا Recall اهمیت بیشتری دارد، مناسب نباشد.

- پیچیدگی بیشتری نسبت به Accuracy دارد.

کاربرد: مناسب برای زمانی که داده‌ها نامتوازن هستند و نیاز به تعادل بین Precision و Recall وجود دارد، مانند مسائل پزشکی یا تشخیص تقلب.

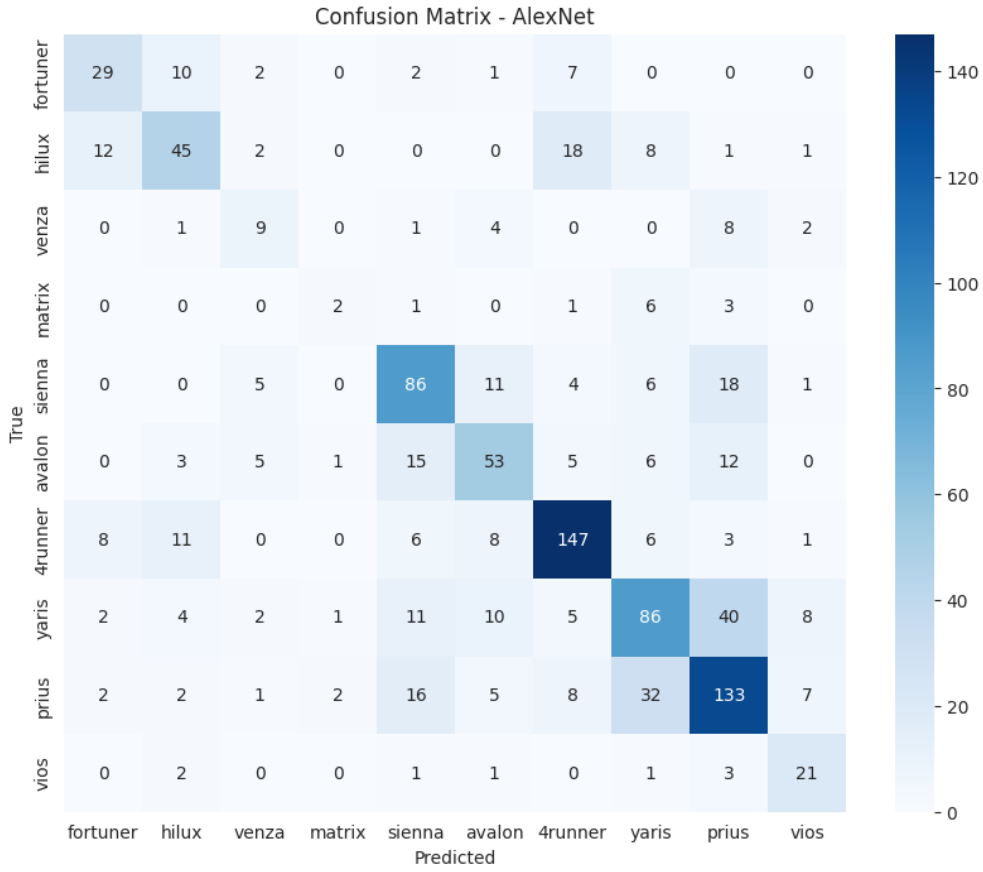
- در مسائل چندکلاسه، این متریک‌ها می‌توانند به‌صورت میانگین (Macro, Micro, Weighted) محاسبه شوند.

- برای مسائل نامتوازن، بهتر است از Precision، Recall و F۱-Score به جای Accuracy استفاده شود.

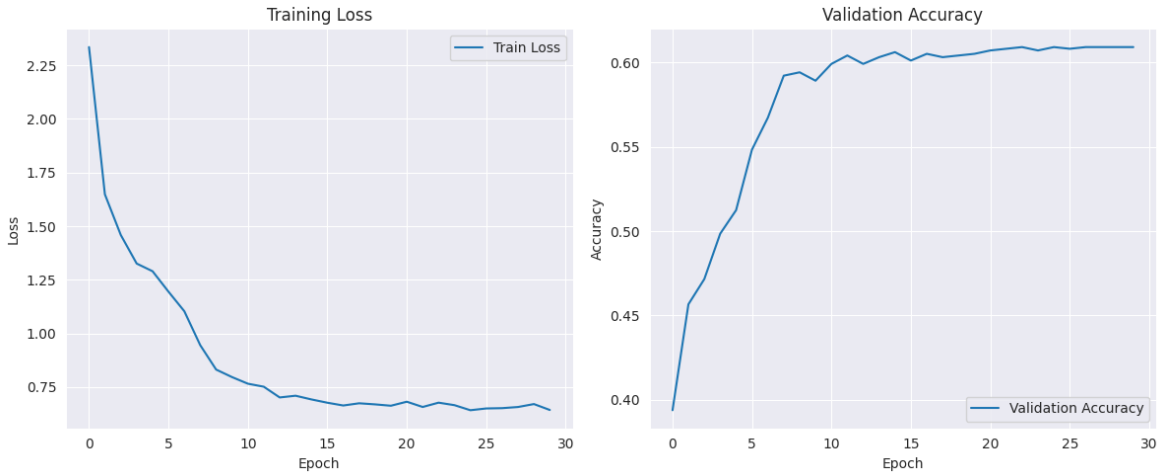
```
train_dir = '/kaggle/working/train_resized'
test_dir = '/kaggle/working/test_resized'
NUM_CLASSES = 10
BATCH_SIZE = 32
EPOCHS = 30

#(FC layers)
model.classifier = nn.Sequential(
    nn.Dropout(0.5),
    nn.Linear(9216, 4096),
    nn.ReLU(inplace=True),
    nn.Dropout(0.5),
    nn.Linear(4096, 4096),
    nn.ReLU(inplace=True),
    nn.Linear(4096, NUM_CLASSES)
)
```

```
Metrics for AlexNet:
Accuracy: 0.6092
Weighted Precision: 0.6068
```

```
Weighted Recall: 0.6092
Weighted F1-Score: 0.6063

Unweighted (Macro) Metrics:
Macro Precision: 0.5430
Macro Recall: 0.5432
Macro F1-Score: 0.5374

Per-Class Metrics:
fortuner - Precision: 0.5472, Recall: 0.5686, F1-Score: 0.5577
hilux - Precision: 0.5769, Recall: 0.5172, F1-Score: 0.5455
venza - Precision: 0.3462, Recall: 0.3600, F1-Score: 0.3529
matrix - Precision: 0.3333, Recall: 0.1538, F1-Score: 0.2105
sienna - Precision: 0.6187, Recall: 0.6565, F1-Score: 0.6370
avalon - Precision: 0.5699, Recall: 0.5300, F1-Score: 0.5492
4runner - Precision: 0.7538, Recall: 0.7737, F1-Score: 0.7636
yaris - Precision: 0.5695, Recall: 0.5089, F1-Score: 0.5375
prius - Precision: 0.6018, Recall: 0.6394, F1-Score: 0.6200
vios - Precision: 0.5122, Recall: 0.7241, F1-Score: 0.6000
```



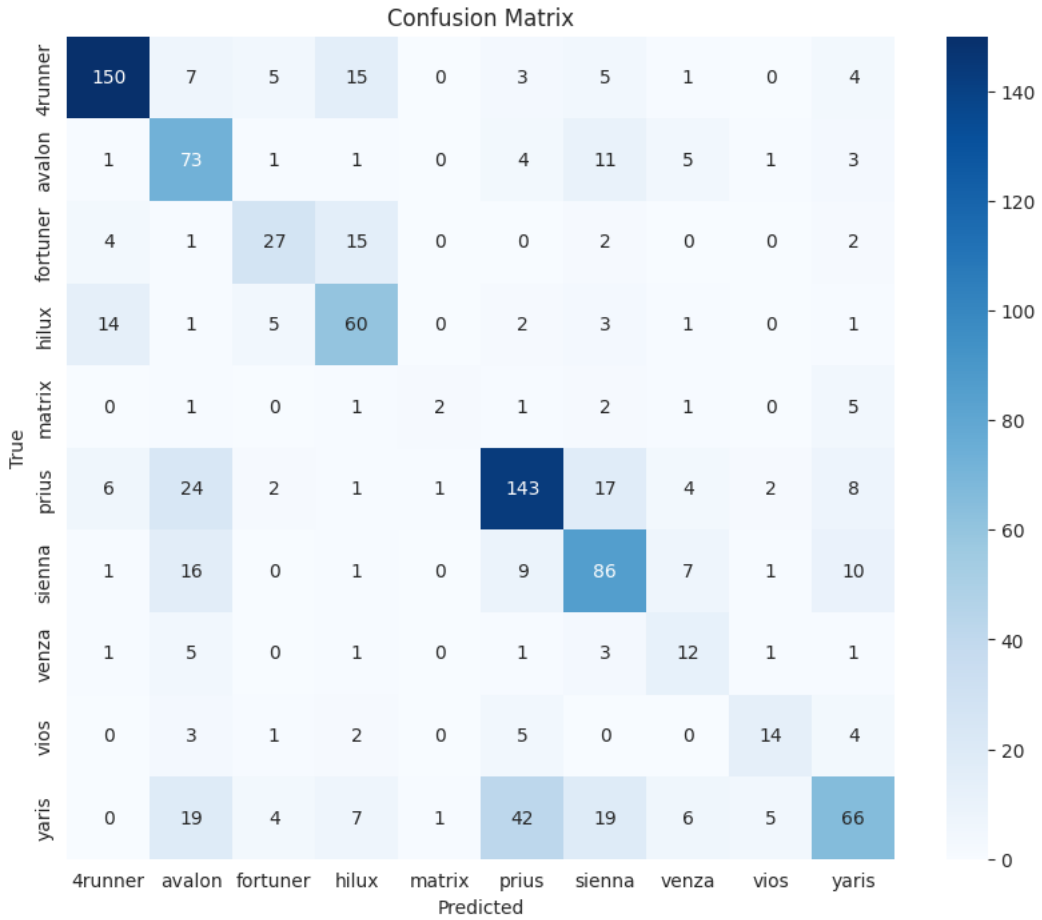
شکل ۳۲. ماتریس درهم ریختگی AlexNet



شکل ۳۳. نمودار خطای آموزش و اعتبارسنجی AlexNet

```
TRAIN_DIR = '/kaggle/working/train_resized'
TEST_DIR = '/kaggle/working/test_resized'
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
NUM_CLASSES = 10
EPOCHS = 45


x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(1024, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.3)(x)
predictions = Dense(NUM_CLASSES, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer=Adam(learning_rate=0.001, clipnorm=1.0),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=3, min_lr=1e-6)
```

```
Weighted F1-Score: 0.6280

Classification Report:
              precision    recall  f1-score   support

     4runner       0.85      0.79      0.82       190
      avalon       0.49      0.73      0.58       100
    fortuner       0.60      0.53      0.56        51
       hilux       0.58      0.69      0.63        87
      matrix       0.50      0.15      0.24        13
       prius       0.68      0.69      0.68       208
      sienna       0.58      0.66      0.62       131
       venza       0.32      0.48      0.39        25
        vios       0.58      0.48      0.53        29
       yaris       0.63      0.39      0.48       169

    accuracy                           0.63      1003
   macro avg       0.58      0.56      0.55      1003
weighted avg       0.65      0.63      0.63      1003
```



شکل ۳۴. ماتریس درهم ریختگی VGG

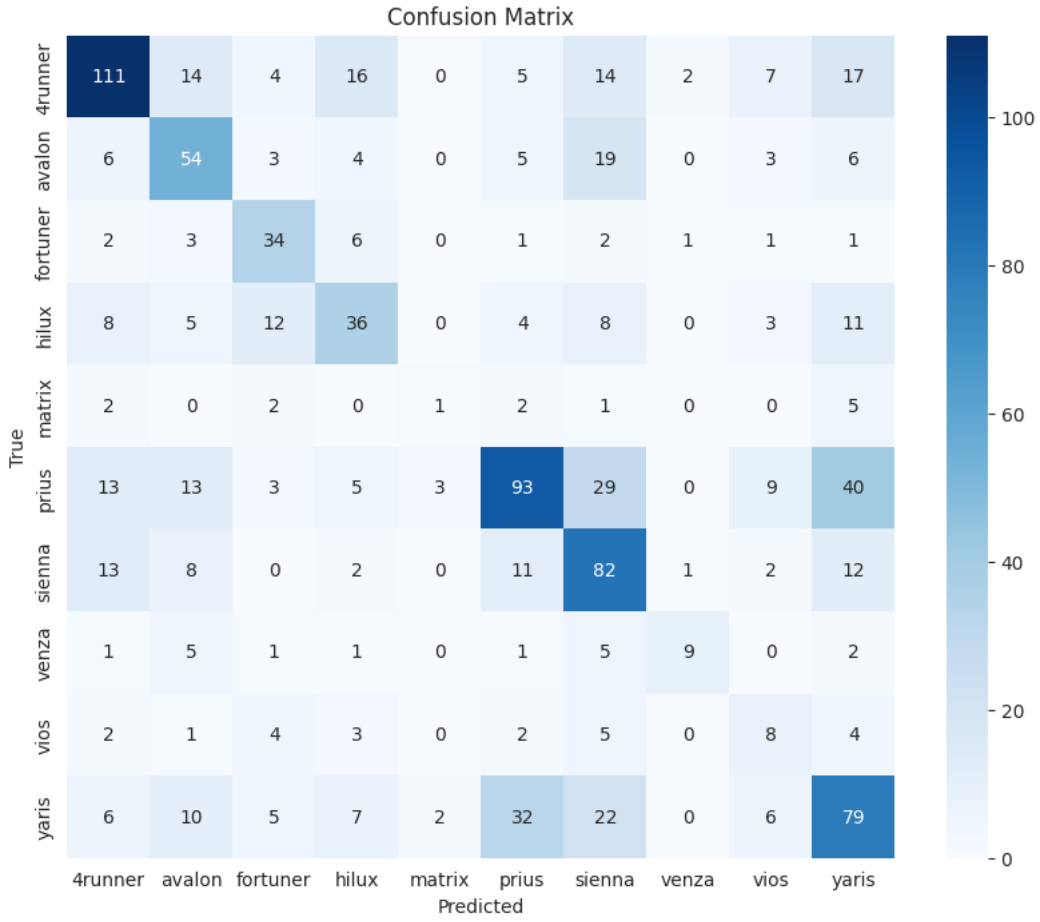
۲.

VGG Net با افزایش عمق شبکه و بدست آوردن ویژگی های پیچیده تر ، استفاده از فیلترهای کوچک‌تر و طراحی یکنواخت، دقت و تعمیم‌پذیری بهتری ارائه داد، اما به هزینه محاسباتی بیشتری نیاز دارد .

۳.

```
TRAIN_DIR = '/kaggle/working/train_resized'
TEST_DIR = '/kaggle/working/test_resized'
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
NUM_CLASSES = 10
EPOCHS = 60

model = Sequential([

    Conv2D(64, (3, 3), padding='same', input_shape=(224, 224, 3)),
    BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), padding='same'),
    BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), padding='same'),
    BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(2048),
    BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    Dropout(0.3),
    Dense(1024),
    BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    Dropout(0.2),
    Dense(512),
    BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    Dropout(0.2),
    Dense(NUM_CLASSES, activation='softmax')
])

model.compile(optimizer=Adam(learning_rate=0.001, clipnorm=1.0),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
Weighted F1-Score: 0.5060

Classification Report:
              precision    recall  f1-score   support

     4runner       0.68      0.58      0.63       190
      avalon       0.48      0.54      0.51       100
    fortuner       0.50      0.67      0.57        51
       hilux       0.45      0.41      0.43        87
      matrix       0.17      0.08      0.11        13
       prius       0.60      0.45      0.51       208
      sienna       0.44      0.63      0.52       131
       venza       0.69      0.36      0.47        25
        vios       0.21      0.28      0.24        29
       yaris       0.45      0.47      0.46       169

    accuracy                           0.51      1003
   macro avg       0.46      0.45      0.44      1003
weighted avg       0.52      0.51      0.51      1003
```



شکل ۳۵. ماتریس درهم ریختگی CNN

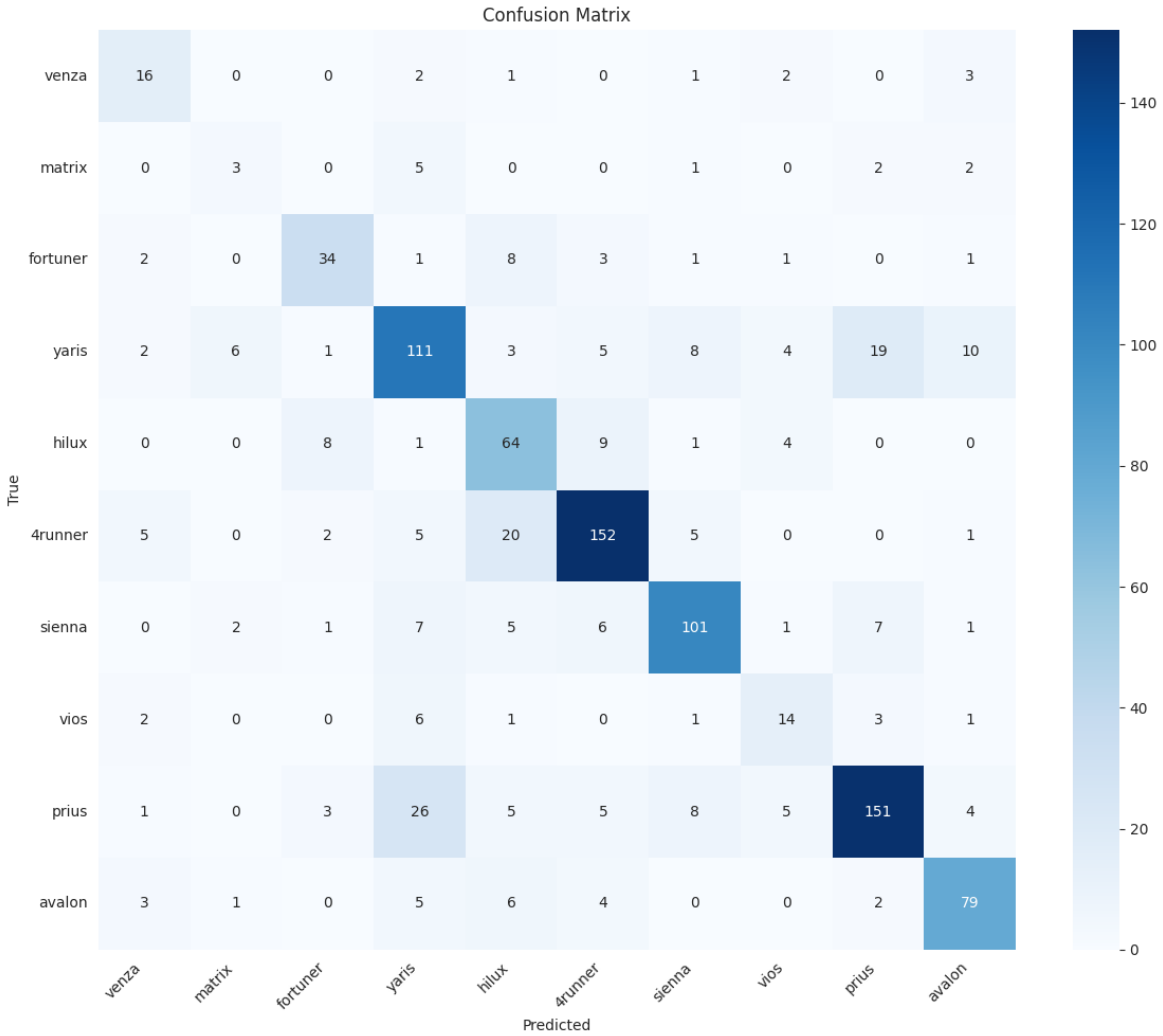
۴.

در این مرحله با فراخوانی بردار های ویژگی استخراج شده در مرحله قبل با استفاده از شبکه VGG ۱۶ و استفاده از SVM با کرنل خطی به عنوان دسته بند سعی می کنیم همانند مقاله دسته بندی مناسبی را روی این ۱۰ کلاس انتخاب شده ارایه دهیم :

```
classifier = SVC(kernel='linear', random_state=42, C=1.0)
classifier.fit(features, labels)

y_pred = classifier.predict(features_test)
accuracy = accuracy_score(labels_test, y_pred)
```

```
Test Accuracy: 0.7236

Classification Report:
              precision    recall  f1-score   support

       venza       0.52      0.64      0.57        25
      matrix       0.25      0.23      0.24        13
    fortuner       0.69      0.67      0.68        51
```

yaris	0.66	0.66	0.66	169
hilux	0.57	0.74	0.64	87
4runner	0.83	0.80	0.81	190
sienna	0.80	0.77	0.78	131
vios	0.45	0.50	0.47	28
prius	0.82	0.73	0.77	208
avalon	0.77	0.79	0.78	100
accuracy			0.72	1002
macro avg	0.64	0.65	0.64	1002
weighted avg	0.73	0.72	0.73	1002



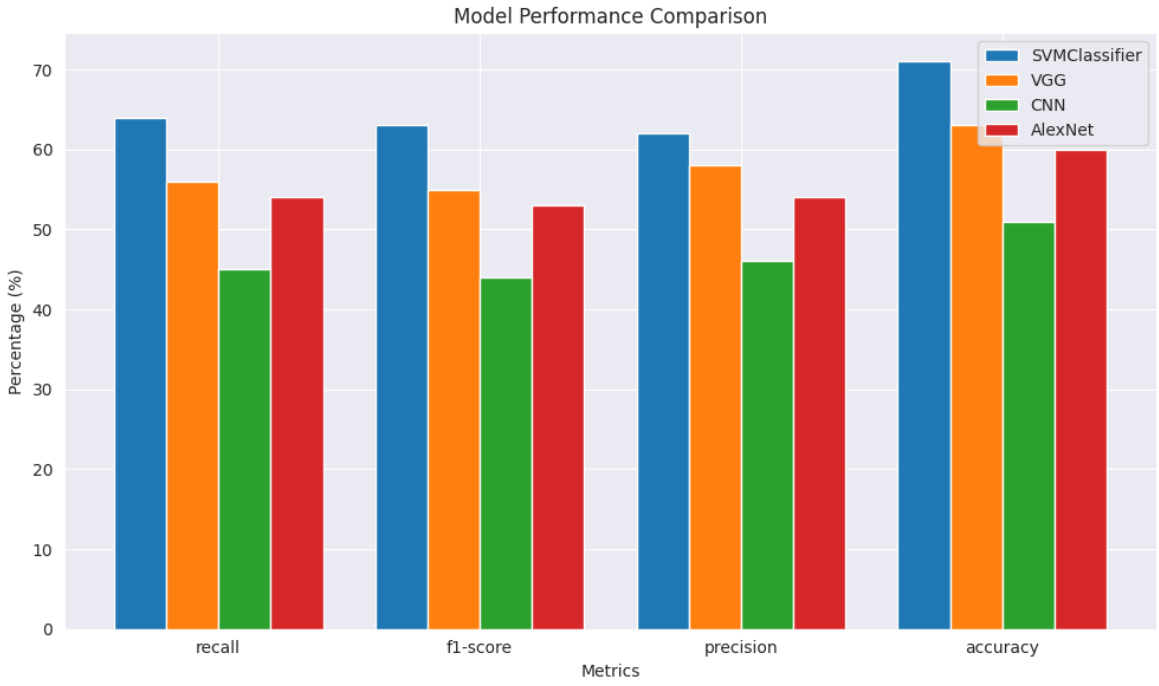
شکل ۳۶. ماتریس درهم ریختگی VGG + SVM

با توجه به تعریف انواع خطا در قسمت های قبل ماتریس اشفتگی به طور خلاصه نشان می دهد که مدل در مواجه با داده های تست از هر دسته چگونه عمل کرده است و تا چه حدی کلاس هر دسته را به خوبی تشخیص داده است که قطر اصلی آن بیانگر داده های است که به درستی به کلاس خود نسبت داده شده اند .

5-2. نتایج

جدول ۲. مقایسه مدل ها و معیار ها

Precision	F1-Score	Recall	Accuracy	
54%	53%	54%	60%	**AlexNet**
58%	55%	56%	63%	**VGG**
46%	44%	45%	51%	**CNN**
62%	63%	64%	71%	**VGG+SVM**



شکل ۳۷. نمودار مقایسه مدل ها به ازای معیار های دسته بندی

هر ۴ مدل در تشخیص کلاس ۴Runner و Prius بهتر از سایر کلاس ها عمل کرده اند و بد ترین عملکرد مدل ها در کلاس Matrix رخ داده است که می تواند به علت کمبود بیش از حد تصاویر اولیه در این دسته باشد و برای عملکرد بهتر مدل ها در همه دسته ها نیاز به داده های با کیفیت بیشتر داریم تا در مجموع عملکرد مدل ها با همین تعداد پارامتر ارتقا یابد .