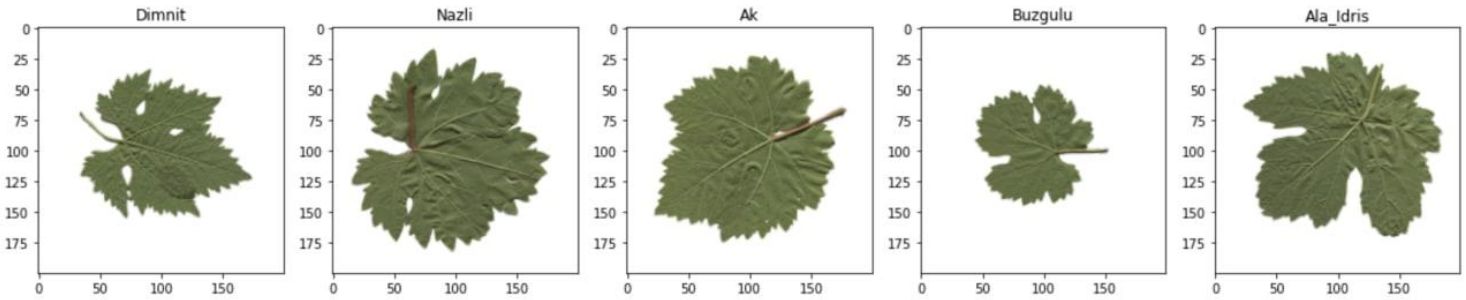
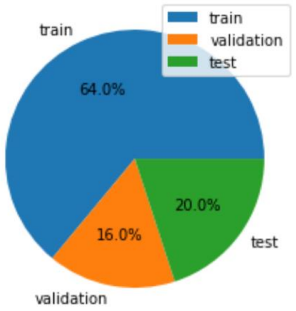
گزارشکار پروژه پایانترم درس داده کاوی
مهیار محمدی متین - 610398166
نیمسال دوم تحصیلی 1400-1401

هدف این پروژه، تشخیص و دسته‌بندی انواع مختلف برگ‌های درخت انگور است. در این پروژه، به ما 5 نوع از انواع مختلف برگ‌های درخت انگور داده شده است.

ابتدا دادگان از ورودی خوانده و دیتافریم مربوطه را تشکیل می‌دهیم. دیتافریم شامل دو ستون path و lable است. تعداد هر گروه از برگ‌ها نیز برابر 100 است. در زیر، نمونه‌ای از هر گروه آورده شده است.




طبق توضیح پروژه، دادگان را به دو دسته تست و آموزش با نسبت 80 به 20 تقسیم می‌کنیم. پس از آن، دادگان آموزش را با نسبت 20 به 80، به دو دسته validation و آموزش تقسیم می‌کنیم. در نهایت، دادگان اولیه به این نسبت تقسیم می‌شوند:

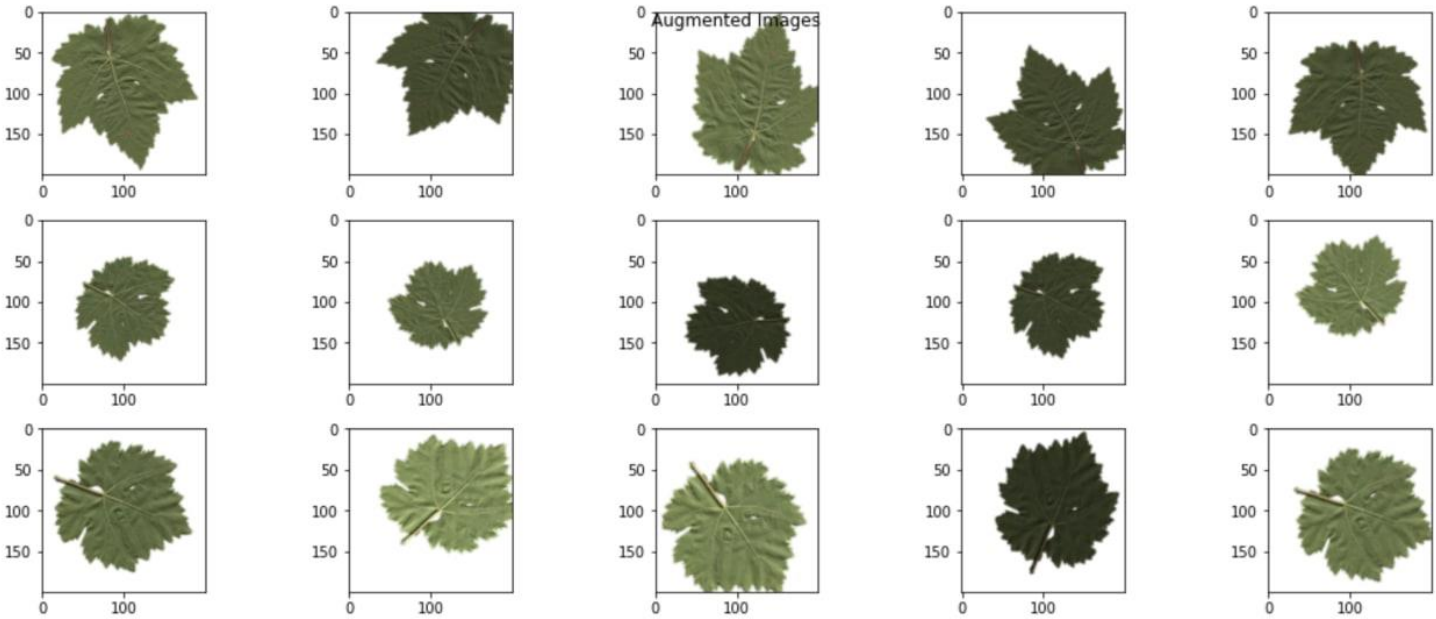



پس از آن تابع data generation را تشکیل می‌دهیم. به این منظور، دوتابع مختلف می‌نویسیم. یک تابع برای داده‌های آموزش و یک تابع برای داده‌های تست، اما توجه شود که دادگان تست نباید تغییر کنند. این تابع صرفا به این دلیل است که دادگان تست هم سایز با دادگان آموزش باشند و از این بابت مشکلی در ادامه بوجود نیاید. تابع ایجاد عکس های جدید به این صورت و با این پارامترها تعیین شد:

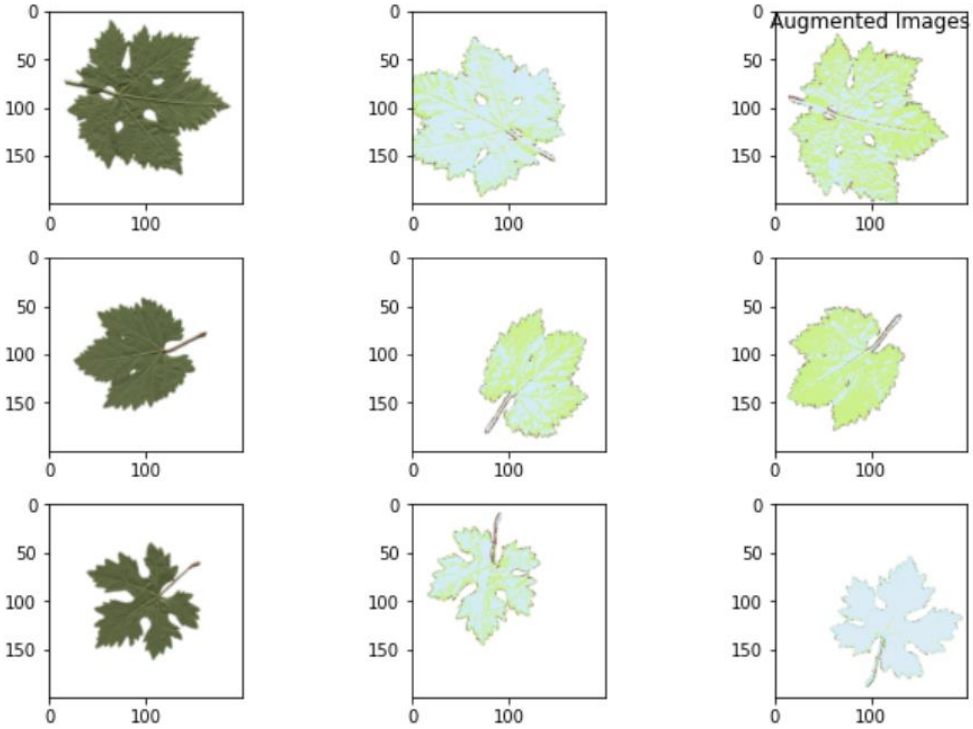
```
train_val_data_gen = ImageDataGenerator(
    rotation_range=45,
    zoom_range=0.1,
    brightness_range=[0.5,1.5],
    channel_shift_range=0.5,
    width_shift_range=0.15,
    height_shift_range=0.15,
    shear_range=0.15,
    horizontal_flip=True,
    vertical_flip=True,
    validation_split=0.2,
    fill_mode='constant',
    cval=0.0,
    #preprocessing_function= preprocess_input,
)
```

در این پروژه، دو تابع مختلف generation ایجاد کردم. یکی از توابع حافظ رنگ است و دیگری، رنگ را حفظ نمی‌کند. در عکس کد بالا خط کامنت شده مربوط به preproccess یکی از مدل های pre trained است که با اعمال آن تنها حاشیه عکس باقی مانده و رنگ آن تقریبا حذف می‌شود. طی چند مرتبه تست، تابعی که حافظ رنگ نبود، نتیجه بهتری داشت. به همین منظور، از تابع دوم استفاده شده است که در آن pre process داشتیم.

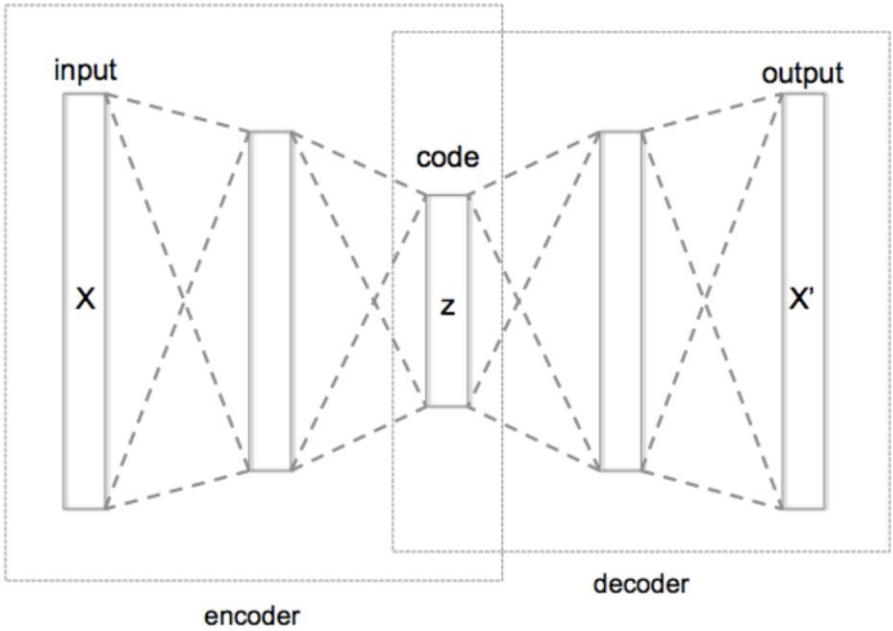
نتایج تابع اول:



نتایج تابع دوم:




امروزه از auto encoder ها در کاربردهای متنوعی استفاده میشود که از آن میتوان به denoising و dimension reduction و compression و... اشاره کرد. شمای کلی الگوریتم auto encoderبه صورت زیر است:




همانطور که مشخص است ابعاد تصویر در بخش z کم شده و عملا انگار انکد شده ورودی است. از طرف دقت کنید که ورودی و خروجی اتوانکودر باید یکسان باشد زیرا یک عکس انکود و سپس دیکود می‌شود به خودش.

در پروژه ما خیلی متوجه تعریف denoising نشدم زیرا اغلب عکس ها کیفیت خوبی داشتند و نویزی حداقل در حدی که مزاحممان باشد وجود نداشت. پس نیازی به این فیچر AEها نداریم.

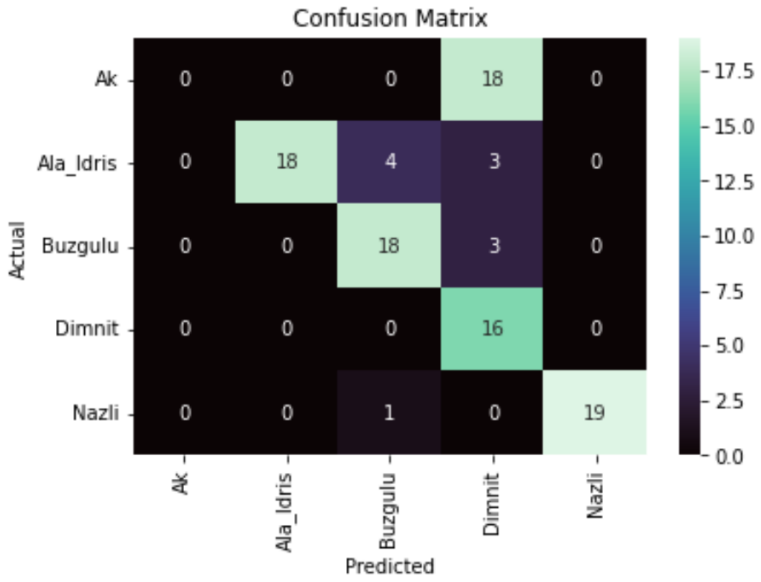
اما از طرف دیگر در کل هر عکسی دارای ابعاد بسیار بزرگی است و می‌توان از AEها برای کاهش ابعاد استفاده کرد.

همچنین برای این پروژه از سرویس google colab استفاده شده است. در این سرویس ما قابلیت استفاده از GPU با سرعت بالا را داریم که طبق مقایسه پایین می‌بینیم که سرعت هر epoch با GPU به شدت بیشتر از حالت عادی است:

```
10/10 - 258s - loss: 2.0562 - accuracy: 0.2438 - val_loss: 2.1388 - val_accuracy: 0.1875 - 258s/epoch - 26s/step
10/10 - 8s - loss: 3.4785 - accuracy: 0.2438 - val_loss: 3.6710 - val_accuracy: 0.2375 - 8s/epoch - 829ms/step
```

از مدل های مختلفی از لایبرری tf.keras.applications استفاده شد که 3 تا از برترین های آنها در بخش کد آمده است و با هم مقایسه شده است.

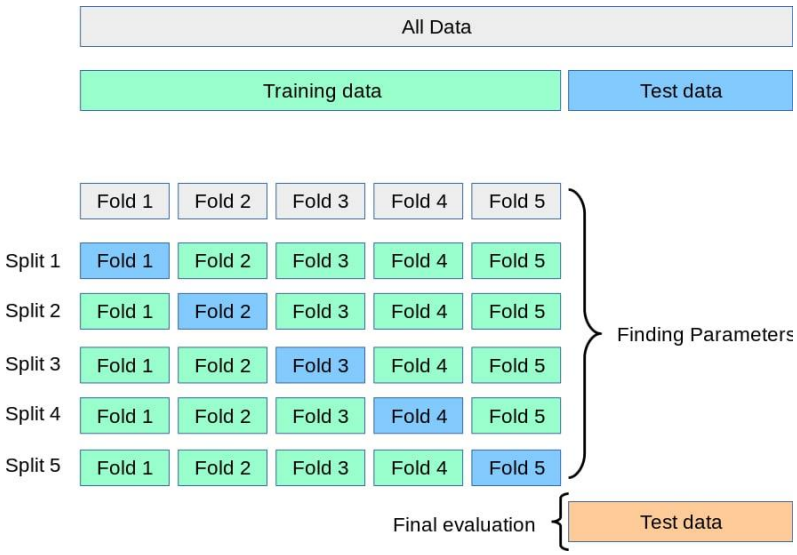
برای یکی از نتایج با دقت70 درصد confusion matrix را مشاهده کنیم که نتیجه جالبی در بر داشته است:




در این ماتریس همانطور که میبینیم بیشتر اشتباهات مدل ما سر اشتباه گرفتن دو نوع برگ dimnit و Ak است. در واقع هر برگ AK را مدل ما dimnit پیش بینی کرده است! این اشکال با تغییرات پارامتری و مدل برطرف شد.

حال مدل‌های مختلف را بر روی دادگان آموزش اجرا می‌کنیم و در نهایت دقت را بر روی دادگان تست حساب می‌کنیم. برای هر مدل، این فرآیند را 10 بار تکرار کرده و نهایتا بهترین مدل را برای ادامه کار انتخاب می‌کنیم.

در بهترین نتیجه ما مدل EfficientNetB3 انتخاب شد و برای بررسی بهتر نتیجه آن، بر روی آن الگوریتم $10fold$ پیاده شد. این الگوریتم در شکل زیر به تصویر کشیده شده است(الگوریتم $5fold$):




نتیجه اجرای $10fold$ بر روی مدل $ENB3$:

| دقت اجرا / مدل | ENB3 |
|---|---|
| دقت اجرای اول | 91 |
| دقت اجرای دوم | 89 |
| دقت اجرای سوم | 94 |
| دقت اجرای چهارم | 91 |
| دقت اجرای پنجم | 93 |
| دقت اجرای ششم | 97 |
| دقت اجرای هفتم | 89 |
| دقت اجرای هشتم | 87 |
| دقت اجرای نهم | 90 |
| دقت اجرای دهم | 91 |
| میانگین دقت مدل | 91.2 |