

**پروژه پایانی درس داده کاوی**

**دانشجو:**

**ابوالفضل کبیری**

**شماره دانشجویی :**

**۶۱۰۳۹۸۱۵۹**

**دانشگاه :**

**دانشگاه تهران (علوم کامپیووتر)**

**استاد:**

**دکتر هدیه ساجدی**

# فهرست

وارد کردن دیتاست به درایو و mount کردن درایو در کلب .....	۵
وارد کردن کتابخانه‌های مورد نیاز به محیط پایتون .....	۶
خواندن دیتاست اولیه در محیط پایتون.....	۶
ساخت دیتاست نهایی به همراه لیبل‌های دیتاست .....	۱۰
تغییر سایز تصاویر برای جلوگیری از crash شدن رم پس از Data Augmentation	۱۱
تغییر تعداد کانال‌های تصاویر موجود در دیتاست .....	۱۳
حالت اول : پیاده‌سازی شبکه‌ی CNN قبل از اعمال Data Augmentation	۱۴
حالت دوم : پیاده‌سازی شبکه‌ی CNN پس از اعمال Data Augmentation	۱۹
روش‌های مرسوم افزایش داده‌ها:.....	۲۰
حالت سوم : پیاده‌سازی شبکه‌ی pretrained-VGG19 پس از اعمال Data Augmentation	۳۰
حالت چهارم : پیاده‌سازی شبکه‌ی pretrained-VGG19 پس از اعمال Data Augmentation و استفاده از K Fold	۴۲
روش k Fold cross validation .....	۴۳
حالت پنجم : پیاده‌سازی شبکه‌های Res Net .....	۴۹
حالت ششم : پیاده‌سازی شبکه‌ی inception v3 .....	۵۴
ارزیابی نهایی نتایج : .....	۶۲

# فهرست شکل‌ها

شکل ۱ نمایش پنج نمونه از تصاویر دیتاست (از هر کلاس یک تصویر نمایش داده شده است). ۹.....
شکل ۲ شکل بالا تصویر ۴۹۹ ام در دیتاست قبل از ریسایز شدن و شکل پایین همان تصویر بعد از ریسایز شدن ۱۲.....
شکل ۳ نمایش تصویر ۴۹۹ ام در دیتاست قبل و بعد از حذف کانال چهارم ..... ۱۳.....
شکل ۴ از بالا به پایین به ترتیب نمایش loss و accuracy شبکه اول ..... ۱۶.....
شکل ۵ نمایش confusion matrix برای مدل ۱ ..... ۱۸.....
شکل ۶ نمایش precision, recall, F1 score برای مدل ۱ ..... ۱۸.....
شکل ۷ نمونه‌ای از حالات مختلف داده افزایی در تصویر ..... ۱۹.....
شکل ۸ نمونه‌ای از data augmentation با شیفت افقی ..... ۲۱.....
شکل ۹ نمونه‌ای از data augmentation با شیفت عمودی ..... ۲۱.....
شکل ۱۰ نمونه‌ای از data augmentation با چرخش ..... ۲۲.....
شکل ۱۱ نمونه‌ای از data augmentation با تغییر شدت روشنایی ..... ۲۳.....
شکل ۱۲ نمونه‌ای از data augmentation با تغییر اسکیل ..... ۲۳.....
شکل ۱۳ نمونه‌ای از تصاویر augment شده را برای یکی از تصاویر دیتاست ..... ۲۵.....
شکل ۱۴ از بالا به پایین به ترتیب نمایش loss و accuracy مدل CNN پس از Data Augmentation ..... ۲۷.....
شکل ۱۵ نمایش confusion matrix برای مدل ۲ ..... ۲۹.....
شکل ۱۶ نمایش precision, recall, F1 score برای مدل ۲ ..... ۲۹.....
شکل ۱۷ معماری شبکه VGG19 ..... ۳۱.....
شکل ۱۸ دو نمودار بالا برای چک کردن بالانس بودن داده‌های ورودی است. .... ۳۲.....
شکل ۱۹ آشکار ساز لبه ..... ۳۵.....
شکل ۲۰ آشکار ساز گوش ..... ۳۶.....
شکل ۲۱ ویژگی sift ..... ۳۶.....
شکل ۲۲ ویژگی HOG ..... ۳۷.....
شکل ۲۳ تصاویر augment شده ..... ۳۸.....
شکل ۲۴ نمایش loss و accuracy برای دادگان train و validation در مدل ۳ ..... ۴۰.....

۴۴	..... شکل ۲۵ نحوه عملکرد فرایند k fold cross validation
۴۶	..... شکل ۲۶ نتیجه آموزش شبکه VGG19 پس از ۵۰ ایپاک برای مدل ۴
۴۷	..... شکل ۲۷ نمایش confusion matrix برای مدل ۴
۴۸	..... شکل ۲۸ نمایش precision, recall, F1 score برای مدل ۴
۴۹	..... شکل ۲۹ resnet 101v2
۵۲	..... شکل ۳۰ نمودار خطای صحت آموزش و اعتبارسنجی برای مدل ۵
۵۲	..... شکل ۳۱ خطای حسب هر کلاس برای مدل ۵
۵۳	..... شکل ۳۲ confusion matrix برای مدل ۵ (رزنت)
۵۴	..... شکل ۳۳ نمودار بلوکی یک ماثول Inception دلخواه
۵۹	..... شکل ۳۴ از بالا به پایین به ترتیب نمایش loss و accuracy در دادگان برای مدل ۶
۶۱	..... شکل ۳۵ نمایش confusion matrix برای مدل ۶
۶۱	..... شکل ۳۶ نمایش precision, recall, F1 score برای مدل ۶

## مقدمه

دادگانی که در این پروژه در نظر گرفته شده است دادگان مربوط به شناسایی انواع برگ های درخت انگور است . علاوه بر میوه درخت انگور، برگ های درخت انگور هم قابل استفاده است و بر اساس نوع آن قیمت و کاربرد آن تعیین میشود.

جداسازی این برگ ها در مقیاس بزرگ به صورت دستی امکان پذیر نیست . بنابراین سعی میکنیم با دادگانی که داریم عمل دسته بندی برگ ها را به صورت خودکار انجام دهیم.

## وارد کردن دیتاست به درایو و mount کردن درایو در کلب

در ابتدا میبایست دادگان داده شده را وارد محیط پایتون کنیم . بدین منظور در ابتدا فایل zip مربوط به دادگان را در محیط google drive بارگذاری میکنیم . در ادامه نیز کافیست یک فایل ipynb . برای کدنویسی آنلاین پایتون ایجاد کنیم و شبکه های مدنظر را در این فایل پیاده سازی کنیم .



حال کافیست در ابتدا در فایل کد ساخته شده، درایو خود را mount کنیم تا بتوانیم به دیتاست آپلود شده در درایو دسترسی پیدا کنیم .

همچنین با توجه به اینکه هدف آموزش یک مدل کانولوشنال است، بهتر است به جای استفاده از gpu، az cpu، از برای اجرای کد خود استفاده کنیم، زیرا با استفاده از gpu ، زمان اجرای کدها به میزان بسیار بالایی کاهش می یابد و درواقع پایتون از رم آنلاین برای اجرای کدها استفاده می کند:

```
1- mount drive

[ ] 1 from google.colab import drive
2 drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)

Use GPU

[ ] 1 import tensorflow as tf
2 if len(tf.config.list_physical_devices('GPU')) > 0:
3     !nvidia-smi --query-gpu=gpu_name,driver_version,memory.total --format=csv
```

A screenshot of a Jupyter Notebook cell. The title of the cell is '1- mount drive'. The code in the cell is as follows:

```
from google.colab import drive
drive.mount('/content/drive')
```

A message indicates that the drive is already mounted at /content/drive. Below the cell, there is a section titled 'Use GPU' containing the following code:

```
import tensorflow as tf
if len(tf.config.list_physical_devices('GPU')) > 0:
    !nvidia-smi --query-gpu=gpu_name,driver_version,memory.total --format=csv
```

## وارد کردن کتابخانه‌های مورد نیاز به محیط پایتون

در مرحله بعد می‌دانیم در پایتون برای استفاده از هر دستوری میباشد کتابخانه‌ی مورد نیاز برای آن دستور را import کنیم. بدین منظور در این قسمت کتابخانه‌های مورد نیاز برای این پروژه را import می‌کنیم:

```
from zipfile import ZipFile
import numpy as np
from skimage.io import imread_collection
import sklearn
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Model
from tensorflow import keras
import random
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from keras.preprocessing.image import ImageDataGenerator #image data
from numpy import expand_dims
import cv2
from PIL import Image
from sklearn.utils import shuffle
from sklearn.metrics import classification_report
scaler = StandardScaler()
%matplotlib inline
random.seed(30)
```

## خواندن دیتاست اولیه در محیط پایتون

در مرحله بعد کافیست فایل زیپ مربوط به دیتاست را unzip کرده و تمامی دادگان موجود در این فایل که در ۵ کلاس مختلف زیر هستند، بخوانیم. کلاس‌های مختلف دادگان عبارتند از :

۱- کلاس AK

۲- کلاس Ala\_Idris

۳- کلاس Buzgulu

۴- کلاس Dimnit

۵- کلاس Nazli

توسط دستور زیر فایل زیپ `unzip` شده و پوشه‌های موجود در آن خوانده می‌شود.

```
1 # specifying the zip file name
2 file_name = "/content/drive/MyDrive/Grapevine_Leaves_Image_Dataset.zip"
3
4 # opening the zip file in READ mode
5 with ZipFile(file_name, 'r') as zip:
6     # printing all the contents of the zip file
7     zip.printdir()
8
9     # extracting all the files
10    print('Extracting all the files now...')
11    zip.extractall()
12    print('Done!')
```

بعد از اینکه فایل مربوط به دیتاست را `unzip` کردیم، حال کافیست تصاویر موجود در ۵ پوشه‌ی مربوط به این دیتاست را از مسیر اصلی که پوشه‌ی دیتاست در آن مسیر قرار دارد، به ترتیب بخوانیم و در متغیرهای `col1`، `col2`، `col3`، `col4` و `col5` بریزیم.

در قسمت قبل فایل زیپ مربوط به دیتاست، `unzip` شد و یک پوشه با نام `Grapevine_Leaves_Image_Dataset` در محیط کلب ساخته شد. این پوشه حاول ۵ فایل می‌باشد که هر فایل شامل ۱۰۰ تصویر است و در حقیقت هر فایل مربوط به یک کلاس است، در این قسمت کافیست هر فایل (که نشانده‌نده‌ی یک کلاس است) را به صورت مجزا خوانده و در `col1` تا `col5` بریزیم، بنابراین داریم:

```
1 path1 = '/content/Grapevine_Leaves_Image_Dataset/Ak/*.png'
2 path2 = '/content/Grapevine_Leaves_Image_Dataset/Ala_Idris/*.png'
3 path3 = '/content/Grapevine_Leaves_Image_Dataset/Buzgulu/*.png'
4 path4 = '/content/Grapevine_Leaves_Image_Dataset/Dimnit/*.png'
5 path5 = '/content/Grapevine_Leaves_Image_Dataset/Nazli/*.png'
6
7 #creating a collection with the available images
8 col1 = imread_collection(path1)
9 col2 = imread_collection(path2)
10 col3 = imread_collection(path3)
11 col4 = imread_collection(path4)
12 col5 = imread_collection(path5)
```

در ادامه میتوانیم ابعاد دادگان موجود در این ۵ کلاس را نیز بررسی کنیم، در این دیتاست در حقیقت هر کلاس حاوی ۱۰۰ عکس رنگی میباشد و هر عکس دارای ابعاد  $4 \times 511 \times 511$  میباشد.

```

1 print('class Ak = ',np.shape(col1))
2 print('class Ala_Idris = ',np.shape(col1))
3 print('class Buzgulu = ',np.shape(col1))
4 print('class Dimnit = ',np.shape(col1))
5 print('class Nazli = ',np.shape(col1))

class Ak = (100, 511, 511, 4)
class Ala_Idris = (100, 511, 511, 4)
class Buzgulu = (100, 511, 511, 4)
class Dimnit = (100, 511, 511, 4)
class Nazli = (100, 511, 511, 4)
```

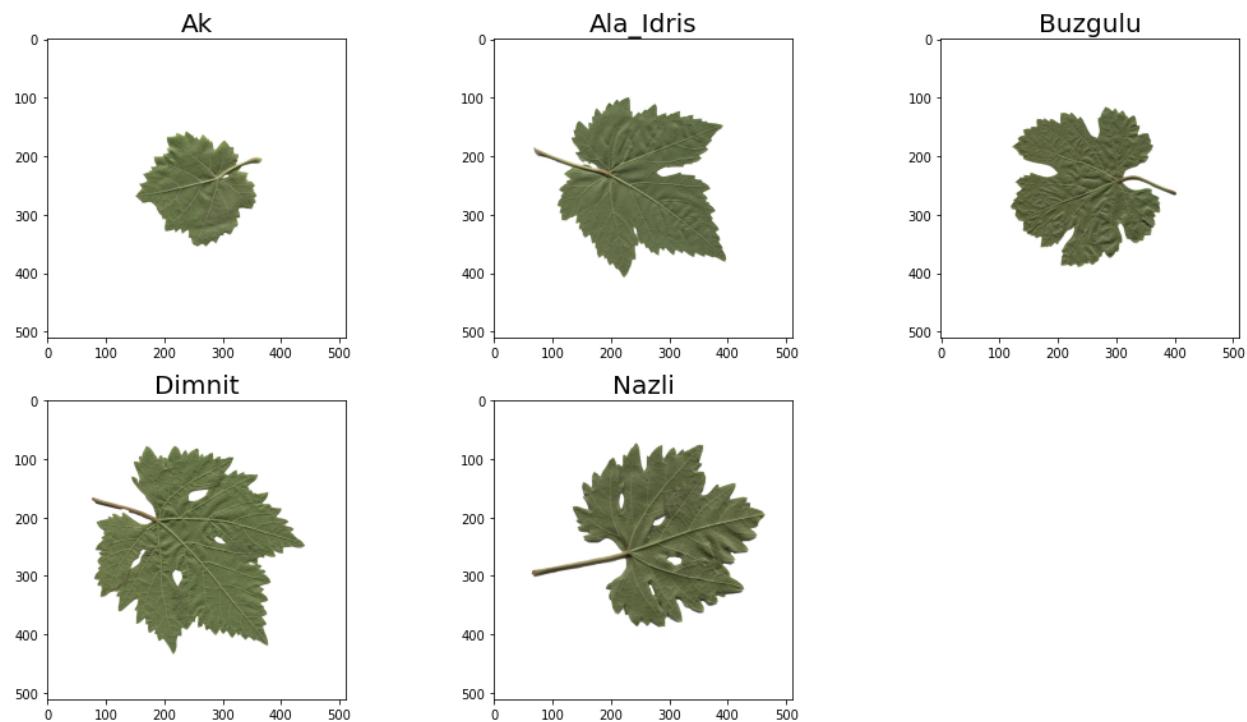
در ادامه تابع `show_image_list` را تعریف میکنیم که تابع فوق به عنوان ورودی پنج تصویر، لیبل این پنج تصویر، سایز تصاویر و برخی ورودی‌های مورد نیاز دیگر را خوانده و در خروجی از هر کلاس به عنوان نمونه یک تصویر را نمایش می‌دهد.

```

1 def img_is_color(img):
2     if len(img.shape) == 3:
3         # Check the color channels to see if they're all the same.
4         c1, c2, c3 = img[:, :, 0], img[:, :, 1], img[:, :, 2]
5         if (c1 == c2).all() and (c2 == c3).all():
6             return True
7     return False
8
9 def show_image_list(list_images, list_titles=None, list_cmaps=None, grid=True, num_cols=2, figsize=(20, 10), title_fontsize=30):
10    assert isinstance(list_images, list)
11    assert len(list_images) > 0
12    assert isinstance(list_images[0], np.ndarray)
13    if list_titles is not None:
14        assert isinstance(list_titles, list)
15        assert len(list_images) == len(list_titles), '%d imgs != %d titles' % (len(list_images), len(list_titles))
16    if list_cmaps is not None:
17        assert isinstance(list_cmaps, list)
18        assert len(list_images) == len(list_cmaps), '%d imgs != %d cmaps' % (len(list_images), len(list_cmaps))
19    num_images = len(list_images)
20    num_cols = min(num_images, num_cols)
21    num_rows = int(num_images / num_cols) + (1 if num_images % num_cols != 0 else 0)
22    # Create a grid of subplots.
23    fig, axes = plt.subplots(num_rows, num_cols, figsize=figsize)
24    # Create list of axes for easy iteration.
25    if isinstance(axes, np.ndarray):
26        list_axes = list(axes.flat)
27    else:
28        list_axes = [axes]
29    for i in range(num_images):
30        img = list_images[i]
31        title = list_titles[i] if list_titles is not None else 'Image %d' % (i)
32        cmap = list_cmaps[i] if list_cmaps is not None else (None if img_is_color(img) else 'gray')
33        list_axes[i].imshow(img, cmap=cmap)
34        list_axes[i].set_title(title, fontsize=title_fontsize)
35        list_axes[i].grid(grid)
36    for i in range(num_images, len(list_axes)):
37        list_axes[i].set_visible(False)
38    fig.tight_layout()
39    _ = plt.show()
```

برای نمایش تصاویر، کافیست از تابع فوق استفاده کنیم:

```
1 show_image_list(list_images=[col1[0], col2[1], col3[2], col4[3], col5[4]),
2                  list_titles=['Ak', 'Ala_Idris', 'Nazli', 'Ala_Idris', 'Nazli'],
3                  num_cols=3,
4                  figsize=(15, 8),
5                  grid=False,
6                  title_fontsize=20)
```



شکل ۱ نمایش پنج نمونه از تصاویر دیتاست (از هر کلاس یک تصویر نمایش داده شده است).

در شکل ۱ پنج نمونه از تصاویر موجود در این دیتاست را به همراه لیبل آنها نمایش دادیم. همانطور که در تصاویر نیز می‌بینیم، هدف طبقه‌بندی ۵ کلاسه برای این دادگان می‌باشد. بنابراین در ادامه می‌بایست دادگان این ۵ کلاس را با هم ترکیب کرده تا مجموعه داده نهایی خود را بسازیم، سپس این مجموعه داده را شافل کرده و از آن برای آموزش شبکه‌ی CNN یا سایر شبکه‌ها استفاده کنیم.

ساخت دیتاست نهایی به همراه لیبل‌های دیتاست

در مرحله بعد کافیست دادگان مربوط به ۵ کلاس را با هم ترکیب کرده و مجموعه دادهنهایی را بسازیم. همچنین لیبل مربوط به این دادگان را نیز می‌سازیم؛ به این ترتیب که کلاس Ak لیbel ۰، کلاس Ala\_Idris لیbel ۱، کلاس Buzgulu لیbel ۲، کلاس Dimnit لیbel ۳ و کلاس Nazli نیز لیbel ۴ می‌گیرد.

```
✓ 3s [10] 1 labels1 = [0] * len(col1)
      2 labels2 = [1] * len(col2)
      3 labels3 = [2] * len(col3)
      4 labels4 = [3] * len(col4)
      5 labels5 = [4] * len(col5)
      6 Y = labels1 + labels2 + labels3 + labels4 +labels5
      7 XX = []
      8 # X.append(list(col1))
      9 XX = list(col1) + list(col2) + list(col3) +list(col4) + list(col5)
     10 print (len(Y),len(XX))

500 500

✓ 0s [11] 1 XX = np.array(XX)
      2 Y = np.array(Y)
      3 print(np.shape(XX))
      4 print(np.shape(Y))

(500, 511, 511, 4)
(500,)
```

در ادامه لیل نهایی دادگان داده شده به ما را می‌بینیم از لیل ۰ تا ۴ می‌باشد (هر کلاس دارای ۱۰۰ داده است):

## تغییر سایز تصاویر برای جلوگیری از crash شدن رم پس از Data Augmentation

همانطور که در قسمت قبل نیز گفتیم، دیتاست نهایی دارای  $500 \times 511 \times 4$  تصویر است و سایز هر تصویر نیز برابر  $500 \times 511 \times 4$  می‌باشد. می‌دانیم سایز این دیتاست یعنی  $4 \times 511 \times 500$  بسیار بزرگ است و در صورتی که بخواهیم از Data Augmentation برای افزایش تعداد دادگان استفاده کنیم، حتماً با مشکل crash شدن رم مواجه خواهیم شد.

برای حل این مسئله کافیست سایز تصاویر را کمتر کنیم تا بتوانیم در مراحل بعدی بدون مشکل تصاویر را augment کنیم. بدین منظور من طول و عرض تصاویر را به میزان  $300 \times 153$  کاهش دادم تا ضمن حفظ اطلاعات هر تصویر، مشکل کمبود حافظه نیز حل شود. دیتاست نهایی دارای ابعاد  $4 \times 153 \times 153 \times 3$  خواهد شد.

```
[13] X = np.zeros((500,153,153,4))
      for i in range (500):
          data = XX[i]
          width, height = int(data.shape[1]*0.3), int(data.shape[0]*0.3)
          resized_data = cv2.resize(data,
                                     (width, height),
                                     interpolation = cv2.INTER_AREA,)
          X[i,:,:,:]=resized_data
```

پس از رسایز کردن تصاویر، در این مرحله می‌بینیم پیکسل‌های تصویر که حاوی intensity می‌باشد، به فرمت float64 شده است.

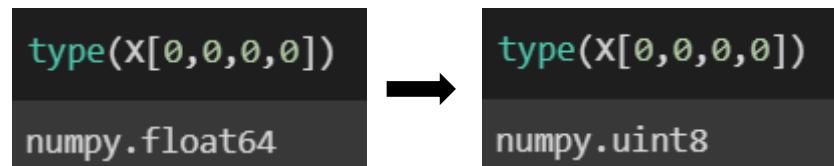
برای حل این مسئله می‌توان از تابع convert استفاده کرده که تصاویر را از هر فرمت گرفته و به فرمت دلخواه تبدیل می‌کند. به کمک این تابع، فرمت تصاویر را به شکل مدنظر یعنی uint8 درآوردم:

```
def convert(img, target_type_min, target_type_max, target_type):
    imin = img.min()
    imax = img.max()

    a = (target_type_max - target_type_min) / (imax - imin)
    b = target_type_max - a * imax
    new_img = (a * img + b).astype(target_type)
    return new_img

X = convert(X, 0, 255, np.uint8)
```

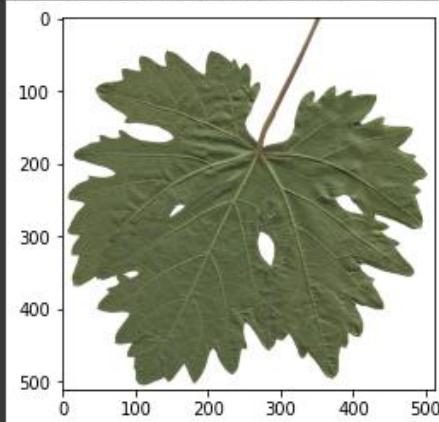
با اعمال تابع فوق داریم :



در ادامه یک نمونه از تصاویر موجود در دیتاست را قبل و بعد از ریسایز شدن می‌بینیم:

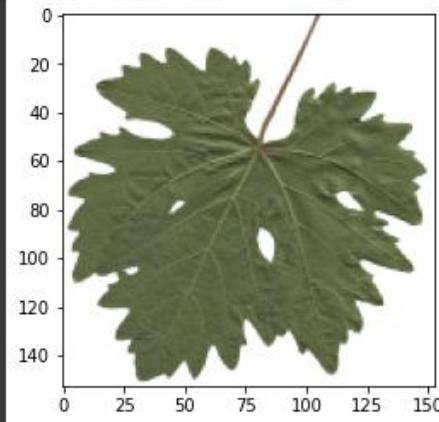
```
print(np.shape(XX[499,:,:,:]))
plt.imshow(XX[499,:,:,:])

(511, 511, 4)
<matplotlib.image.AxesImage at 0x7fa5e20a7350>
```



```
print(np.shape(X[499,:,:,:]))
plt.imshow(X[499,:,:,:])

(153, 153, 4)
<matplotlib.image.AxesImage at 0x7fa5e1f4bcd0>
```



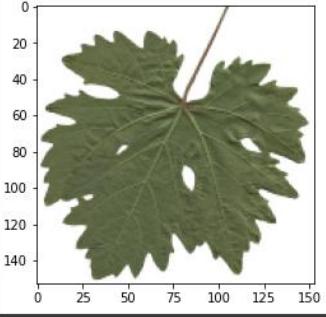
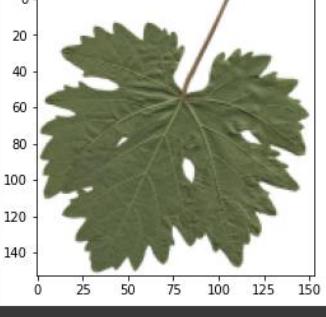
شکل ۲ شکل بالا تصویر ۴۹۹ ام در دیتاست قبل از ریسایز شدن و شکل پایین همان تصویر بعد از ریسایز شدن

## تغییر تعداد کانال‌های تصاویر موجود در دیتاست

همانطور که در قسمت قبل نیز گفتیم، تصاویر موجود در دیتاست دارای ۴ کانال هستند. در حقیقت تصویر CMYK دارای چهار کانال است: فیروزه‌ای، سرخابی، زرد و کلیدی (سیاه). CMYK استانداردی برای چاپ است که در آن از رنگ‌آمیزی کاهشی استفاده می‌شود. یک تصویر ۳۲ بیتی از چهار کانال ۸ بیتی، یکی برای فیروزه‌ای، یکی برای سرخابی، یکی برای زرد، و دیگری برای رنگ کلید (معمولًاً سیاه) ساخته شده است. با بررسی تصاویر دیدیم کانال چهارم در همه‌ی تصاویر دارای صفر است و در نتیجه حاوی اطلاعات معنی‌دار و مهمی نیست. به همین دلیل کانال چهارم از تمامی تصاویر را حذف کردیم، یعنی:

$$\begin{aligned} X &= X[:, :, :, 0:3] \\ Y &= Y \end{aligned}$$

نمایش یک نمونه از تصاویر موجود در دیتاست قبل و بعد از حذف کانال چهارم؛ (به وضوح می‌بینیم حذف کانال چهارم تاثیری در اطلاعات تصویر نداشته است).

```
print(np.shape(X[499,:,:,:]))
plt.imshow(X[499,:,:,:])
(153, 153, 4)
<matplotlib.image.AxesImage at 0x7f6a00dfca50>
  
[39] print(np.shape(X[499,:,:,:]))
plt.imshow(X[499,:,:,:])
(153, 153, 3)
<matplotlib.image.AxesImage at 0x7f6a00d3bf50>

```

شکل ۳ نمایش تصویر ۴۹۹ در دیتاست قبل و بعد از حذف کانال چهارم

## حالت اول : پیاده‌سازی شبکه‌ی CNN قبل از اعمال Data Augmentation

در مرحله بعد می‌بایست مدل‌های ذکر شده در پروژه را پیاده‌سازی کرده و نتایج را بررسی کنیم. در ابتدا به پیاده‌سازی یک مدل CNN بدون اعمال Data Augmentation بر روی تصاویر) می‌پردازیم. بدین منظور در train, test ابتدا می‌بایست دادگان خود را به ۳ مجموعه‌ی آموزش، آزمایش و اعتبارسنجی تقسیم کنیم. ( ، validation ) تقسیم کنیم.

بدین منظور ابتدا می‌بایست دادگان را شافل کنیم زیرا در دیتابست اولیه به ترتیب ۱۰۰ تصویر دارای لیبل ۰، ۱۰۰ تصویر دارای لیبل ۱، ۱۰۰ تصویر دارای لیبل ۲، ۱۰۰ تصویر دارای لیبل سه و ۱۰۰ تصویر دارای لیبل ۴ است، این امر موجب می‌شود تا شبکه به درستی آموزش نبیند و در مینیمم محلی‌ها (local minimum) گیر کند ولی با شافل کردن دیتابست، لیبل‌ها در دادگان هم خورده و پخش می‌شود، در نتیجه این مشکل حل می‌شود. برای شافل کردن دیتابست کافیست :

```
#randomly the data
new_x_train, new_y_train = shuffle(X, Y)
len(new_x_train),len(new_y_train)

(500, 500)
```

حال کافیست با دستور train\_test\_split ۲۰ درصد دادگان را برای test و ۸۰ درصد دادگان را برای train و validation جدا کنیم. در ادامه نیز از این ۸۰ درصد ، ۲۰ درصد را برای validation و مابقی را برای train (برمیداریم).

```
# new_Y = to_categorical(new_y_train)
x_trainn, x_test, y_trainn, y_test= train_test_split(new_x_train, new_y_train, test_size=0.2, random_state=42)
x_train, x_val, y_train, y_val= train_test_split(x_trainn,y_trainn, test_size=0.2, random_state=42)
print('x_train=',np.shape(x_train))
print('y_train=',np.shape(y_train))
print('x_val=',np.shape(x_val))
print('y_val=',np.shape(y_val))
print('x_test=',np.shape(x_test))
print('y_test=',np.shape(y_test))

x_train= (320, 153, 153, 3)
y_train= (320,)
x_val= (80, 153, 153, 3)
y_val= (80,)
x_test= (100, 153, 153, 3)
y_test= (100,)
```

در مرحله بعد میبایست ساختار شبکه کانولوشنال را طراحی کنیم، شبکه طراحی شده دارای ۲ لایه کانلوشنال، ۲ لایه drop out، ۱ لایه flattening، ۱ لایه max pooling و درنهایت ۳ لایه dense است. ساختار کلی لایه‌ها در شبکه‌ی طراحی شده به شکل زیر است:

```
# define the model
model1 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), input_shape=(153, 153, 3), activation='relu', kernel_initializer='he_uniform', padding='same'),
    tf.keras.layers.BatchNormalization(),

    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=(4, 4)),
    tf.keras.layers.Dropout(0.5),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(5, activation='softmax')
])

for layer in model1.layers:
    print(layer.output_shape)

(None, 153, 153, 32)
(None, 153, 153, 32)
(None, 153, 153, 32)
(None, 153, 153, 32)
(None, 38, 38, 32)
(None, 38, 38, 32)
(None, 46208)
(None, 512)
(None, 512)
(None, 128)
(None, 5)
```

برای کامپایل شبکه نیز از یک مدل learning SGD (Stochastic gradient descent) با نرخ یادگیری یا rate = 0.01 استفاده می‌کنیم. در نهایت کافیست مدل طراحی شده را بر روی دادگان fit کنیم.

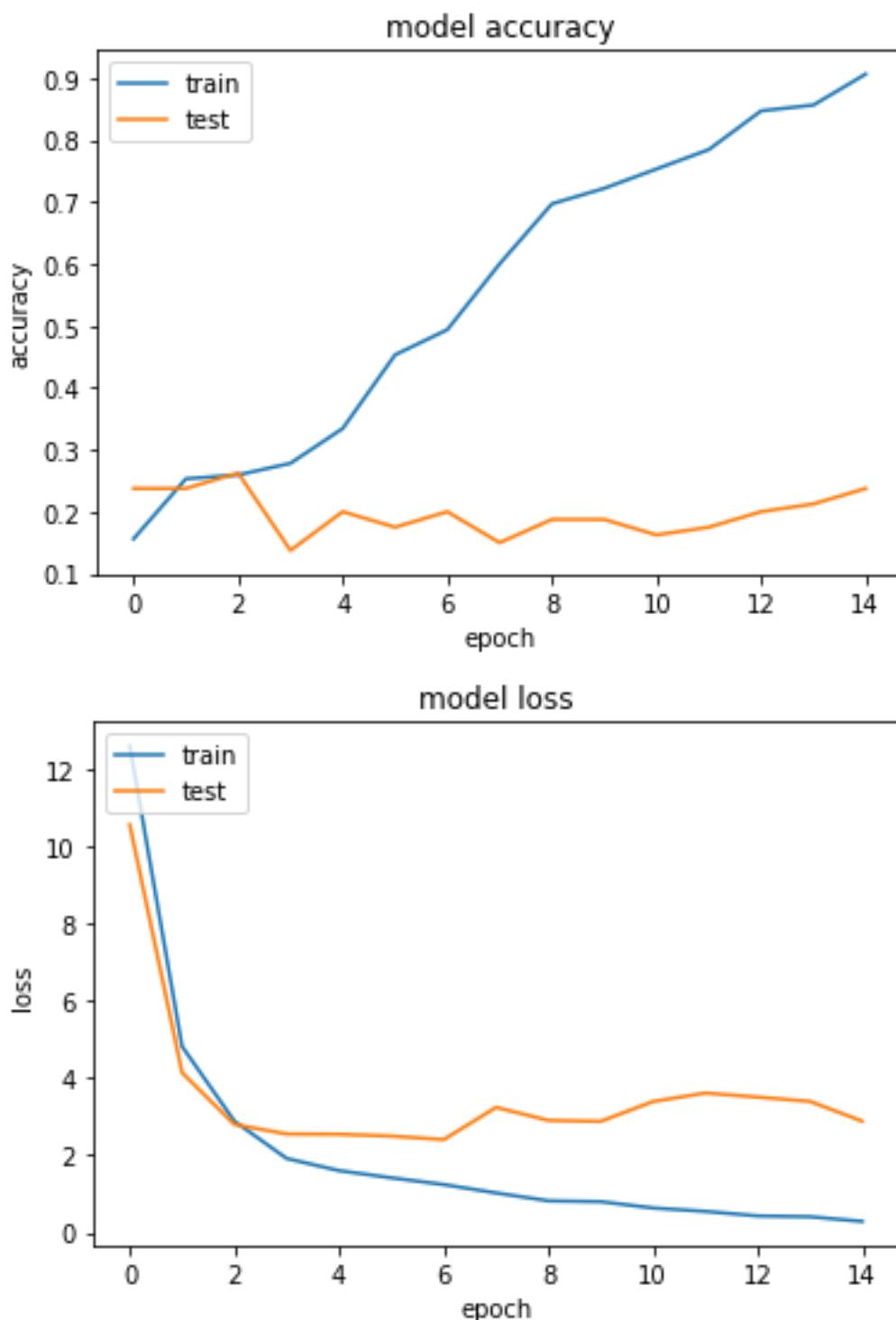
```
# compile and train the model
model1.compile(optimizer=keras.optimizers.SGD(lr = 0.001, decay=1e-6, momentum=0.9 ,nesterov=True),
               loss = tf.keras.losses.CategoricalCrossentropy(from_logits=True),
               metrics=['accuracy'])

history = model1.fit(x_train, to_categorical(y_train), batch_size=64, epochs=15, validation_data=(x_val, to_categorical(y_val)), validation_batch_size=64)

# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

در ادامه در ابتدا نمودار نهایی مربوط به loss و accuracy شبکه برابر میشود با :



شکل ۴ از بالا به پایین به ترتیب نمایش loss و accuracy شبکه اول

پس از آموزش شبکه میباشد شبکه را بر روی دادگان `test` ارزیابی کنیم. برای ارزیابی عملکرد شبکه روش‌های مختلفی وجود دارد از جمله : محاسبه `accuracy`, `loss`, `confusion matrix`, `f1 score`, `recall` و ... . بدین منظور در ابتدا با دستور `evaluate` میتوان میزان `loss` و `accuracy` را در دادگان `test` به دست آورد:

```
prediction = model1.predict(x_test)

# evaluate accuracy on test data
test_loss, test_acc = model1.evaluate(x_test, to_categorical(y_test), verbose=2)
print('\nTest accuracy:', test_acc,'Test loss:',test_loss)

4/4 - 0s - loss: 2.9555 - accuracy: 0.3200 - 89ms/epoch - 22ms/step

Test accuracy: 0.3199999928474426 Test loss: 2.955493211746216
```

همچنین به کمک دستور `predict` نیز میتوان لیبل دادگان تست را پیش‌بینی کرده و سپس لیبل پیش‌بینی شده برای دادگان را با لیبل واقعی آنها مقایسه کنیم.

در ابتدا برای بررسی بهتر عملکرد شبکه از ماتریس در هم ریختگی یا `confusion matrix` استفاده می‌کنیم. جدول یا ماتریس درهم ریختگی، نتایج حاصل از طبقه‌بندی را بر اساس اطلاعات واقعی موجود، نمایش می‌دهد. حال بر اساس این مقادیر می‌توان معیارهای مختلف ارزیابی دسته‌بند و اندازه‌گیری دقت را تعریف کرد. این ماتریس یک روش اندازه‌گیری عملکرد برای مساله طبقه‌بندی یادگیری ماشین است که در آن خروجی می‌تواند دو یا چند کلاس باشد.

در اینجا برای محاسبه `confusion matrix` کافی است در ابتدا لیبل دادگان تست پیش‌بینی شده را به دست آوریم. برای این کار تابع `predict` نوشته شده است که بر اساس عملکرد شبکه لیبل دادگان تست را پیش‌بینی می‌کند. سپس لیبل دادگان تست واقعی و لیبل دادگان تست به دست آمده را به `confusion matrix` داده و نتایج را بررسی کنیم.

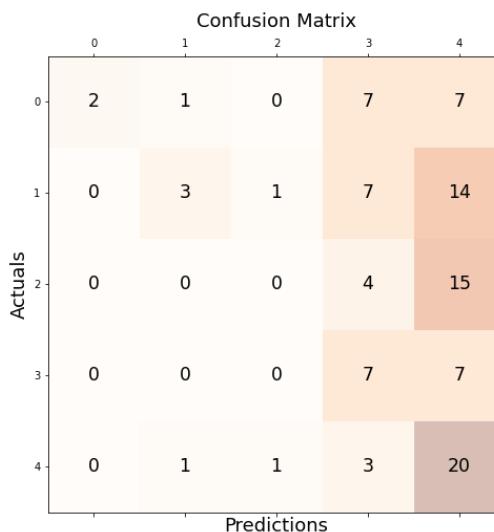
از طرفی برای بررسی عملکرد شبکه برای تشخیص لیبل کلاس‌های مختلف، ۳ معیار : `precision` ، `recall` و `f1 score` برای هر کلاس محاسبه می‌شود تا بتوان نتایج کلاس‌های مختلف را ارزیابی و کلاسی که بهترین نتیجه را می‌دهد انتخاب کنیم.

```
prediction = model1.predict(x_test)
```

در ادامه میتوان F1score، recall و confusion matrix را نیز برای این دادگان به دست آورد.

```
y_pred_bool = np.argmax(prediction, axis=1)
y_test_bool = y_test
#from sklearn.metrics.confusion_matrix import confusion_matrix
# showing the results
conf_matrix = sklearn.metrics.confusion_matrix(y_pred=y_pred_bool , y_true=y_test_bool)
fig, ax = plt.subplots(figsize=(8, 8))
ax.matshow(conf_matrix, cmap=plt.cm.Oranges, alpha=0.3)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```



شکل ۵ نمایش confusion matrix برای مدل ۱

print(classification_report(y_test_bool, y_pred_bool))				
	precision	recall	f1-score	support
0	0.00	0.00	0.00	23
1	0.00	0.00	0.00	22
2	0.00	0.00	0.00	17
3	0.00	0.00	0.00	22
4	0.16	1.00	0.28	16
accuracy			0.16	100
macro avg	0.03	0.20	0.06	100
weighted avg	0.03	0.16	0.04	100

شکل ۶ نمایش precision, recall, F1 score برای مدل ۱

## حالت دوم : پیاده‌سازی شبکه‌ی CNN پس از اعمال Data Augmentation

در ابتدا توضیح مختصری در رابطه با Data Augmentation می‌دهیم. بیشتر پایگاه داده‌های متداول و عمومی، دارای داده‌ها هزار ( یا بیشتر ) تصویر هستند. داشتن یک پایگاه داده بزرگ، برای حصول یک عملکرد خوب مهم است. زمانی که یک مدل یادگیری ماشین را آموزش می‌دهیم، پارامترهای آن را بگونه ای تنظیم می‌کنیم که بتواند یک ورودی خاص (مانند تصویر) را به خروجی (یک برچسب) نگاشت دهد. هدف از بهینه‌سازی ما، تعقیب آن نقطه مطلوبی است که تلفات مدل ما در آنجا کم باشد و این موضوع زمانی رخ می‌دهد که پارامترهای مدل به درستی تنظیم شده باشند.

از طرفی شبکه‌های عصبی جدید معمولاً دارای چندین میلیون پارامتر هستند. طبیعتاً اگر پارامترهای بسیار زیادی داشته باشیم، لازم است تعداد مناسبی از مثال‌ها را به مدل خود نشان دهیم تا عملکرد خوبی را بدست آوریم. همچنین، تعداد پارامترهایی که نیاز داریم، متناسب با پیچیدگی وظیفه‌ای است که مدل ما انجام می‌دهد.

نکته مهم اینکه لازم نیست به دنبال داده‌های جدیدی باشیم تا پایگاه داده خود را اضافه کنیم. زیرا شبکه‌های عصبی در ابتدا هوشمند نیستند. برای مثال، یک شبکه عصبی با آموزش ضعیف، تصور می‌کند سه توپ تنیس نشان داده در زیر، مجزا و بصورت تصاویر منحصر بفردی هستند. بنابراین، برای بدست آوردن داده‌های بیشتر، تنها لازم است تغییرات جزئی را در پایگاه داده فعلی خود انجام دهیم. از جمله تغییرات جزئی، قرینه‌سازی (Flip)، جابجایی و یا چرخش است. به‌حال، شبکه عصبی ما فکر می‌کند که این تصاویر مجزا هستند.



شکل ۷ نمونه‌ای از حالات مختلف داده افزایی در تصویر

یک شبکه عصبی کانولوشنی که می‌تواند اشیا را حتی اگر در جهات مختلف قرار گیرند، بصورت مقاوم کلاسه بندی کند، دارای ویژگی تغییرناپذیری ( Invariance ) خواهد بود. بطور خاص، یک CNN می‌تواند نسبت به جابجایی، نقطه دید ، اندازه و یا شدت روشنایی ( یا ترکیبی از این‌ها ) تغییرناپذیر باشد.

این موضوع، اساساً فرضیه داده‌افزایی است. در دنیای واقعی، ممکن است پایگاه داده‌ای داشته باشیم که تصاویر آن در شرایط محدودی تهیه شده باشد. اما برنامه هدف ما ممکن است در شرایط مختلفی مانند جهت، موقعیت، مقیاس، روشنایی و غیره وجود داشته باشد. این موقعیت‌ها ، توسط آموزش شبکه عصبی با داده‌های اصلاح شده مصنوعی اضافی در نظر گرفته می‌شوند.

## روش های مرسوم افزایش داده ها:

برای هر یک از این روش ها، ضریبی که با آن، اندازه پایگاه داده شما افزایش می یابد نیز تعیین می گردد (که تحت عنوان ضریب داده افزایی (Data Augmentation Factor) نامیده می شود).

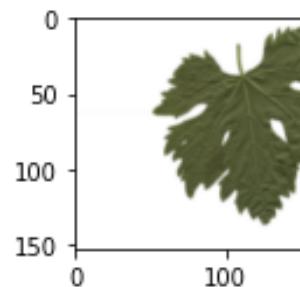
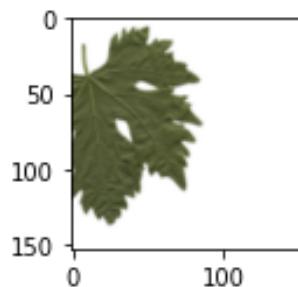
قرینه سازی (Flip)	می توان تصاویر را بصورت افقی و عمودی قرینه نمود. برخی از چارچوبها، تابعی برای قرینه سازی عمودی ندارند. اما می توان گفت قرینه عمودی، همان چرخش یک تصویر با زاویه $180^{\circ}$ درجه و سپس انجام یک قرینه افقی است.
چرخش (Rotation)	نکته ای که در مورد این عملیات باید بدان توجه نمود آن است که ابعاد تصویر ممکن است پس از چرخش، حفظ نگرددن. اگر تصویر شما یک مربع است، اندازه آن با یک چرخش نو دارجه ای تغییر نمی کند. اما اگر مستطیل باشد، یک چرخش $180^{\circ}$ درجه ای، اندازه تصویر را حفظ می نماید. چرخش تصویر با زوایای کوچکتر نیز، اندازه نهایی تصویر را تغییر می دهد.
مقیاس (Scale)	تصویر می تواند بصورت درونی (Inward) یا بیرونی (Outward)، مقیاس بندی شود. اگر بصورت بیرونی مقیاس شود، اندازه نهایی تصویر بزرگ تر از اندازه تصویر اولیه است. عمدۀ چارچوب های تصویر، بخشی از تصویر جدید را با اندازه ای برابر با تصویر اولیه برش می دهند.
برش (Crop)	برخلاف مقیاس بندی، شما تنها از بخشی از تصویر اولیه بصورت تصادفی نمونه برداری می کنید. سپس اندازه این بخش را به اندازه تصویر اولیه تغییر می دهید. این روش معمولاً تحت عنوان برش تصادفی خوانده می شود. در ادامه نمونه هایی از این برش نشان داده شده است.
جابجایی (Translation)	این عملیات، شامل حرکت تصویر در امتداد راستای X، Y یا هر دو راستا است. در مثال زیر، فرض می کنیم که تصویر دارای یک پس زمینه سیاه در بیرون از مرز خود می باشد، و بصورت مناسب جابجا شده است. این روش برای داده افزایی بسیار مفید است، چرا که اکثر اشیای موجود در تصاویر می توانند تقریباً در هرجایی از تصویر قرار گیرند. این موضوع، شبکه عصبی کانولوشنی شما را مجبور می کند که به همه جا نگاه کند.
تکنیک تغییر روشنایی به مقدار تصادفی	می توان برای آموزش شبکه با تصاویر بیشتر، از تکنیک های روشن تر یا تاریک کردن تصاویر اصلی برای افزایش اندازه دیتابست استفاده کرد. هدف این است که شبکه را توانا کنیم تا اشیا موجود در تصاویر را در سطوح مختلفی از روشنایی بتواند تشخیص دهد و قدرت تشخیص گسترده تری داشته باشد.

نکته مهم اینکه دقیق داشته باشید که **Image data augmentation** معمولاً باید فقط روی تصاویر آموزش اعمال کنید و نه داده های **test** یا **validation**، (مگر اینکه بنا به کاربرد شبکه، دلیل خوبی داشته باشید). دلیل اینکه ما در وهله اول از یک مجموعه **train** و **test** استفاده می کنیم این است که می خواهیم خطای سیستم خود را در واقعیت تخمین بزنیم. بنابراین داده های مجموعه **test** باید تا حد امکان به داده های واقعی نزدیک باشد. اگر این کار را در مجموعه **test** انجام دهید، ممکن است خطاهایی را معرفی کنید. به عنوان مثال، برای تشخیص ارقام، با چرخش، آن را افزایش دهید. سپس یک عدد ۶ ممکن است شبیه عدد ۹ باشد! اما خب بدیهی است که پیش پردازش هایی مانند تغییر اندازه و نرمال سازی به طور یکسان روی کل تصاویر اعمال می شود و از این

لحوظ با اعمال **Image data augmentation** متفاوت است. در ادامه چند نمونه از روش‌های فوق را بر روی یکی از تصاویر از دادگان خود اعمال می‌کنیم و نتیجه را نمایش می‌دهیم.

#### ۱- حالت اول : شیفت افقی

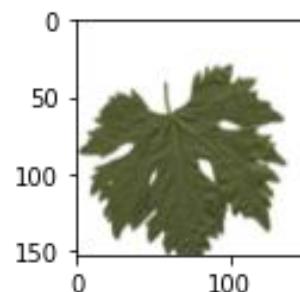
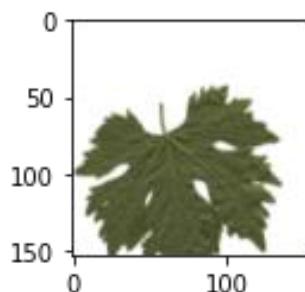
```
#Random horizontal shift
data = img_to_array(x_train[0])
samples = expand_dims(data,0)
datagen = ImageDataGenerator(width_shift_range=[-100,100])
it = datagen.flow(samples, batch_size=1)
for i in range(2):
    plt.subplot(220 + 1+ i)
    batch = it.next()
    image1 = batch[0].astype('uint8')
    plt.imshow(image1)
plt.show()
```



شکل 8 نمونه‌ای از data augmentation با شیفت افقی

#### ۲- حالت دوم : شیفت عمودی

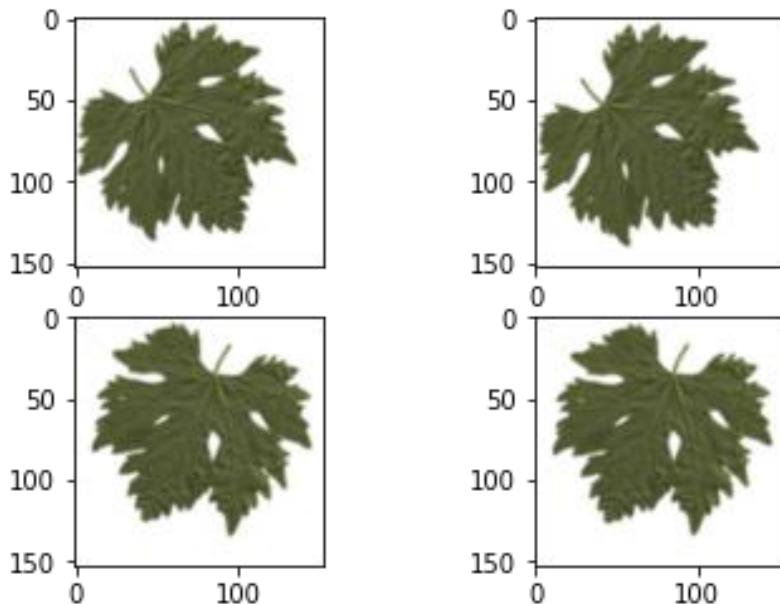
```
#Vertical shift data = img_to_array(image)
datagen = ImageDataGenerator(height_shift_range=0.5)
it = datagen.flow(samples, batch_size=1)
for i in range(2):
    plt.subplot(220 + 1 + i)
    batch = it.next()
    image2 = batch[0].astype('uint8')
    plt.imshow(image2)
plt.show()
```



شکل ۹ نمونه‌ای از data augmentation با شیفت عمودی

### ۳- حالت سوم : چرخش

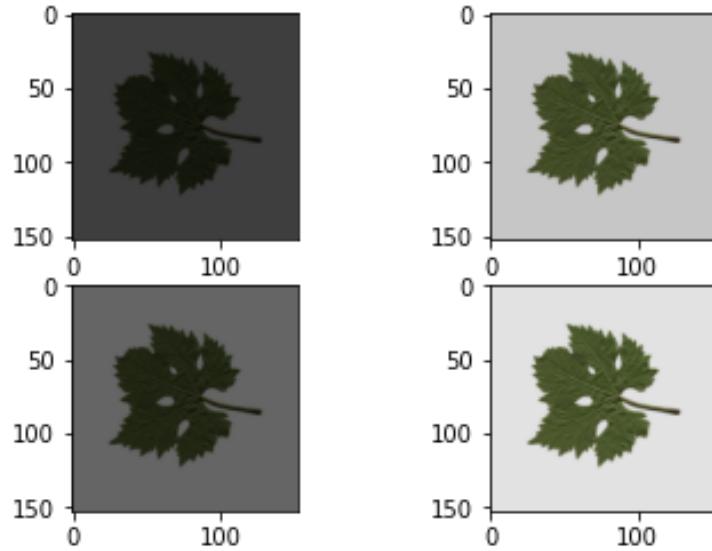
```
#Rotation
datagen = ImageDataGenerator(rotation_range=90) #Specify angle of rotation
it = datagen.flow(samples, batch_size=1)
for i in range(4):
    plt.subplot(220 + 1 + i)
    batch = it.next()
    image4 = batch[0].astype('uint8')
    plt.imshow(image4)
plt.show()
```



شکل ۱۰ نمونه‌ای از data augmentation با چرخش

### ۴- حالت چهارم : تغییر شدت نور

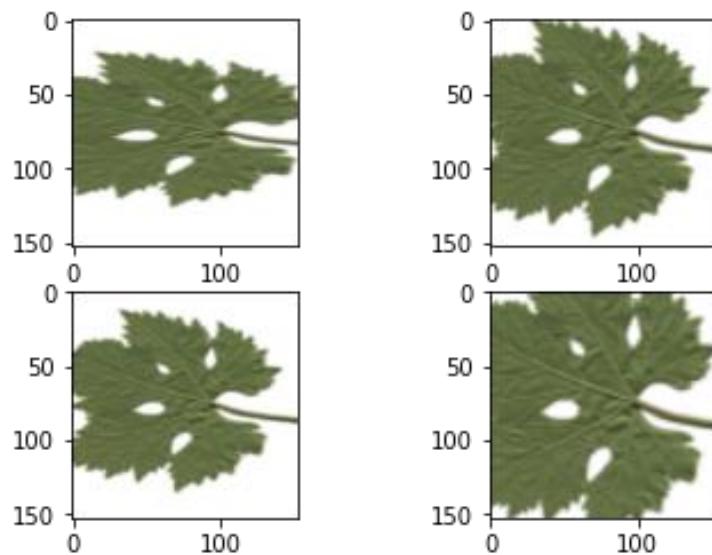
```
#brightness adjustment
datagen = ImageDataGenerator(brightness_range=[0.1,1.0])
it = datagen.flow(samples, batch_size=1)
for i in range(4):
    plt.subplot(220+ 1+ i)
    batch = it.next()
    image5 = batch[0].astype('uint8')
    plt.imshow(image5)
plt.show()
```



شکل ۱۱ نمونه‌ای از data augmentation با تغییر شدت روشنایی

##### ۵- حالت پنجم : تغییر اسکیل (zooming)

```
#Zooming
datagen = ImageDataGenerator(zoom_range=[0.2,1.0])
it = datagen.flow(samples, batch_size=1)
for i in range(4):
    plt.subplot(220 + 1 + i)
    batch = it.next()
    image6 = batch[0].astype('uint8')
    plt.imshow(image6)
plt.show()
```



شکل ۱۲ نمونه‌ای از data augmentation با تغییر اسکیل

در ادامه کافیست به کمک دستور `ImageDataGenerator` به داده‌افزایی یا `data augmentation` بپردازیم. بدین‌منظور می‌توان از روش‌های دلخواه و مناسب برای داده‌افزایی استفاده کرد که من در اینجا از `width_shift_range`, `height_shift_range`, `rotation_range`, `brightness_range`, `zoom_range`, `shear_range`, استفاده کردم.

در ادامه از هر تصویر، با داده‌افزایی، ۷ تصویر دیگر نیز ساختیم. نکته مهم اینکه عملیات `data augmentation` می‌بایست بر روی دادگان `train` و `validation` انجام شود پس در ابتدا دادگان را مشابه قبل شافل کرده، سپس به ۲ دسته‌ی `train`, `test` تقسیم کرده و `data augmentation` را صرفا بر روی دادگان آموزش و اعتبارسنجی (۸۰٪ درصد کل دادگان) اعمال کردیم.

```
new_x_train, new_y_train = shuffle(X, Y)
len(new_x_train),len(new_y_train)

(500, 500)

x_train, x_test, y_train, y_test= train_test_split(new_x_train, new_y_train, test_size=0.2, random_state=42)
# x_train, x_val, y_train, y_val= train_test_split(x_train, y_train, test_size=0.2, random_state=42)
# y_test_rr = np.argmax(y_test, axis = 1)
print('x_train=',np.shape(x_train))
print('y_train=',np.shape(y_train))
# print('x_val=',np.shape(x_val))
# print('y_val=',np.shape(y_val))
print('x_test=',np.shape(x_test))
print('y_test=',np.shape(y_test))

x_train= (400, 153, 153, 3)
y_train= (400,)
x_test= (100, 153, 153, 3)
y_test= (100,)
```

```
from keras.applications.vgg19 import preprocess_input
new_x_train = np.zeros((2800,153,153,3))
datagen = ImageDataGenerator(width_shift_range=0.15,zoom_range=0.10, height_shift_range=0.15,shear_range=0.15, rotation_range=20,brightness_range=[0.6,1.4], m=0
for j in range (400):
    data = img_to_array(x_train[j,:,:,:])
    for i in range(7):
        samples1 = expand_dims(data,0)
        it1 = datagen.flow(samples1, batch_size=1)
        batch1 = it1.next()
        image1 = batch1[0].astype('uint8')
        new_x_train[m,:,:,:,:]= batch1
        m+=1

new_y_train = np.zeros((2800,1))
m=0
for j in range (400):
    data = y_train[j]
    for i in range(7):
        new_y_train[m,:] = data
        m+=1
new_x_train = convert(new_x_train, 0, 255, np.uint8)
new_y_train = to_categorical(new_y_train)

print(np.shape(new_x_train))
print(np.shape(new_y_train))

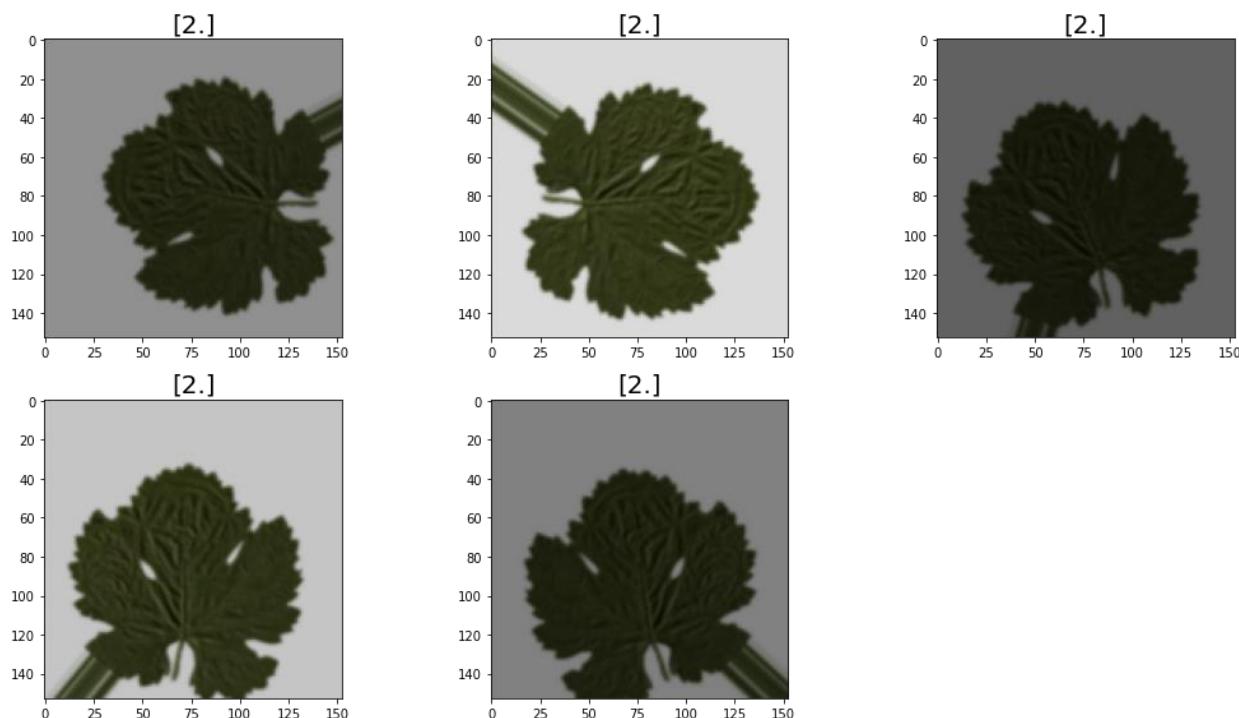
(2800, 153, 153, 3)
(2800, 5)
```

بنابراین در نهایت مجموعه داده اولیه در این قسمت از  $400 \times 2800$  تصویر به  $2800 \times 153 \times 153 \times 3$  تصویر افزایش پیدا کرد و ابعاد خروجی برای مجموعه داده و لیبل آنها به شکل زیر شد :

new\_x\_train :  $2800 \times 153 \times 153 \times 3$

new\_y\_train :  $2800 \times 1$

در ادامه میتوان نمونه‌ای از تصاویر augment شده را برای یکی از تصاویر دیتابست بینیم:



شکل ۱۳ نمونه‌ای از تصاویر augment شده را برای یکی از تصاویر دیتابست

حال کافیست این مجموعه داده‌ی augment شده را به دو قسمت validation و train تقسیم کرده و سپس به یک شبکه‌ی CNN بدهیم تا آموزش ببینند.

نکته‌ی مهم اینکه برای فیت کردن شبکه‌ی CNN بر روی دادگان augment شده، میتوانیم از روش K Fold cross validation استفاده کنیم.

در این قسمت در ابتدا مدل CNN را مشابه قبل تعریف می کنیم یعنی شبکه طراحی شده دارای ۲ لایه کانولوشنال، ۳ لایه drop out، ۱ لایه flattin، ۱ لایه max pooling و درنهایت ۳ لایه dense است. ساختار کلی لایه ها در شبکه طراحی شده به شکل زیر است:

```
# define the model
model1 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), input_shape=(153, 153, 3), activation='relu', kernel_initializer='he_uniform', padding='same'),
    tf.keras.layers.BatchNormalization(),

    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=(4, 4)),
    tf.keras.layers.Dropout(0.5),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(5, activation='softmax')
])

for layer in model1.layers:
    print(layer.output_shape)
```

سپس مشابه قبل این شبکه را کامپایل می کنیم:

```
# compile and train the model
model1.compile(optimizer=keras.optimizers.SGD(lr = 0.01, decay=1e-6, momentum=0.9 ,nesterov=True),
               loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
               metrics=['accuracy'])
```

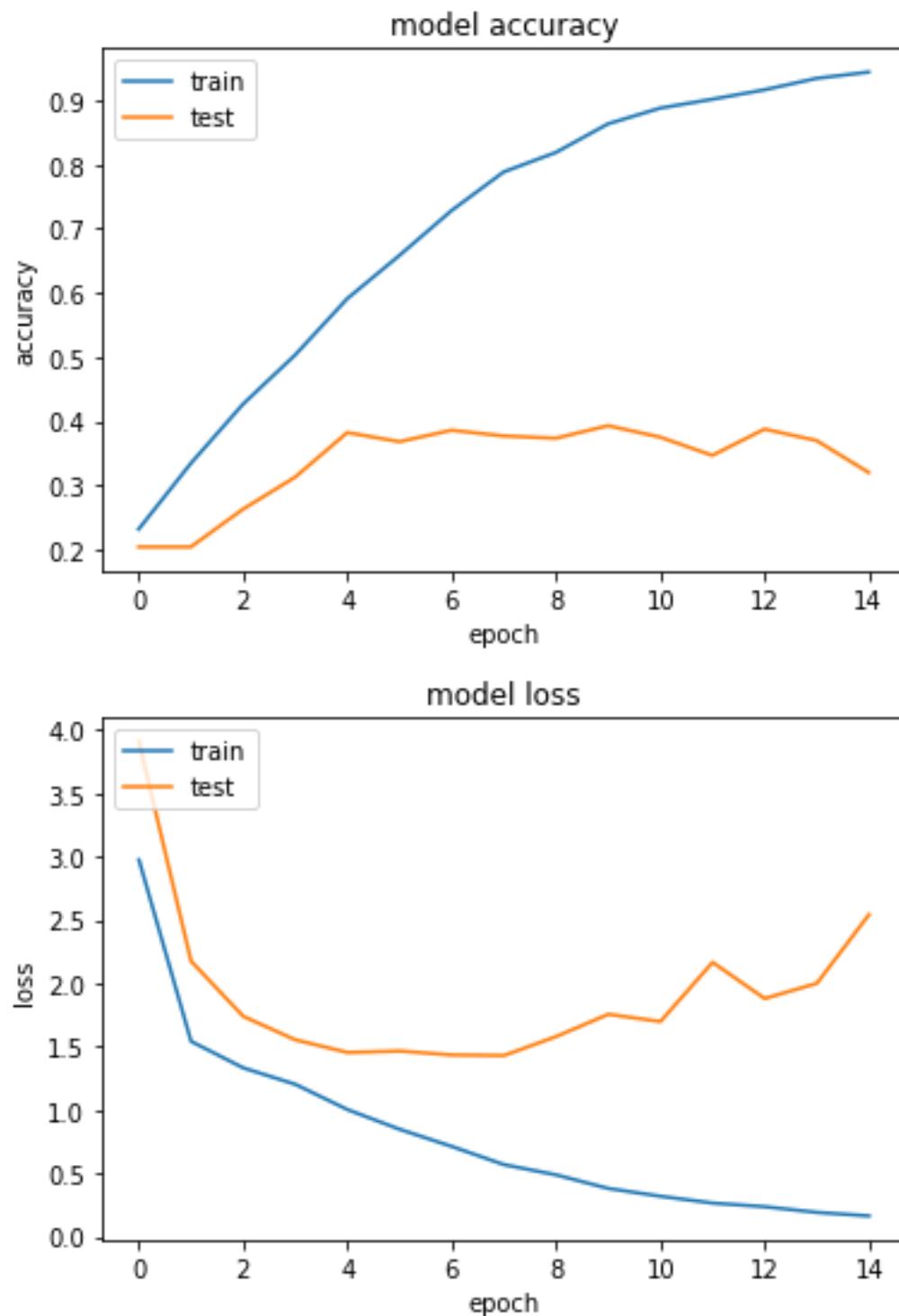
در ادامه کافیست مدل فوق را کامپایل کرده و فیت کنیم:

```
❶ # compile and train the model
model1.compile(optimizer=keras.optimizers.SGD(lr = 0.001, decay=1e-6, momentum=0.9 ,nesterov=True),
               loss = tf.keras.losses.CategoricalCrossentropy(from_logits=True),
               metrics=['accuracy'])

history = model1.fit(x_train, y_train, batch_size=64, epochs=15, validation_data=(x_val, y_val), validation_batch_size=32, verbose=1)

# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

نتایج نهایی به شکل زیر شد:



شکل ۱۴ از بالا به پایین به ترتیب نمایش CNN پس از loss و accuracy مدل

در ادامه مشابه حالت اول میتوان مدل فوق را نیز بر روی دادگان `test` ارزیابی کرد و معیارهای محاسبه شده در قسمت قبل را نیز برای دادگان تست با این مدل به دست آوریم:

```
# evaluate accuracy on test data
test_loss, test_acc = model1.evaluate(x_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc,'Test loss:',test_loss)

4/4 - 0s - loss: 1.8780 - accuracy: 0.3600 - 84ms/epoch - 21ms/step

Test accuracy: 0.36000001430511475 Test loss: 1.8780020475387573
```

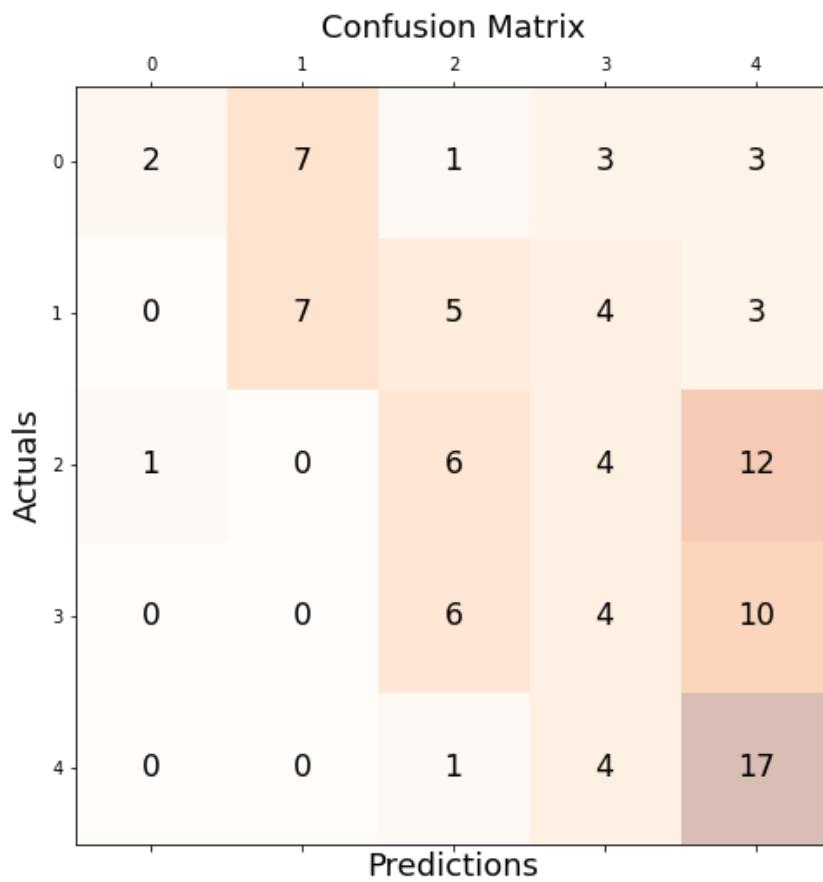
همانگونه که در شکل بالا میبینید، نتایج `accuracy` و `loss` نسبت به حالت قبل بهبود یافت ولی همچنان نتایج مطلوبی نیست پس میبایست از شبکه‌های `pretrained` استفاده کنیم.

در ادامه میتوان معیارهای دیگری نیز نظری `confusion matrix` را در خروجی بررسی کرد:

```
prediction = model1.predict(x_test)

y_pred_bool = np.argmax(prediction, axis=1)
y_test_bool = np.argmax(y_test, axis=1)
#from sklearn.metrics.confusion_matrix import confusion_matrix
# showing the results
conf_matrix = sklearn.metrics.confusion_matrix(y_pred=y_pred_bool , y_true=y_test_bool)
fig, ax = plt.subplots(figsize=(8, 8))
ax.matshow(conf_matrix, cmap=plt.cm.Oranges, alpha=0.3)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```



شکل ۱۵ نمایش confusion matrix برای مدل ۲

```
print(classification_report(y_test_bool, y_pred_bool))
```

	precision	recall	f1-score	support
0	0.67	0.12	0.21	16
1	0.50	0.37	0.42	19
2	0.32	0.26	0.29	23
3	0.21	0.20	0.21	20
4	0.38	0.77	0.51	22
accuracy			0.36	100
macro avg	0.41	0.35	0.33	100
weighted avg	0.40	0.36	0.33	100

شکل ۱۶ نمایش precision, recall, F1 score برای مدل ۲

## حالت سوم : پیاده‌سازی شبکه‌ی pretrained-VGG19 پس از اعمال Data Augmentation

در این قسمت هدف استفاده از مدل‌های pretrained برای افزایش دقت در طبقه‌بندی دادگان می‌باشد. در ابتدا توضیح مختصری در مورد شبکه‌ی VGG19 می‌دهیم.

AlexNet در سال ۲۰۱۲ منتشر شد و در شبکه‌های عصبی Convolutional سنتی پیشرفت کرد، بنابراین می‌توان VGG را به عنوان جانشین AlexNet درنظر گرفت. شبکه عصبی عمیق VGG یا به اصطلاح لاتین K. Simonyan VGG Deep Neural Network یک شبکه عصبی کانولوشنی است که توسط محققانی بنام A. Zisserman و Very Deep Convolutional Networks for Large-Scale Image Recognition پیشنهاد شد.

باتوجه به پیشرفت‌های اخیر شبکه‌های عصبی عمیق، شاید دیگر نتوان شبکه VGG را یک شبکه مدرن دانست، اما به دلیل ساختار خوب، سادگی، تعداد لایه‌های نه‌چندان زیاد هنوز در بسیاری از زمینه‌های بینایی کامپیوتر از این شبکه استفاده می‌شود. این شبکه بیشتر با خاطر شکل هرمی مانندش شناخته می‌شود که در آن لایه‌هایی که به تصویر نزدیکتر هستند، پهن‌تر و لایه‌های دورتر، عمیق‌تر هستند.

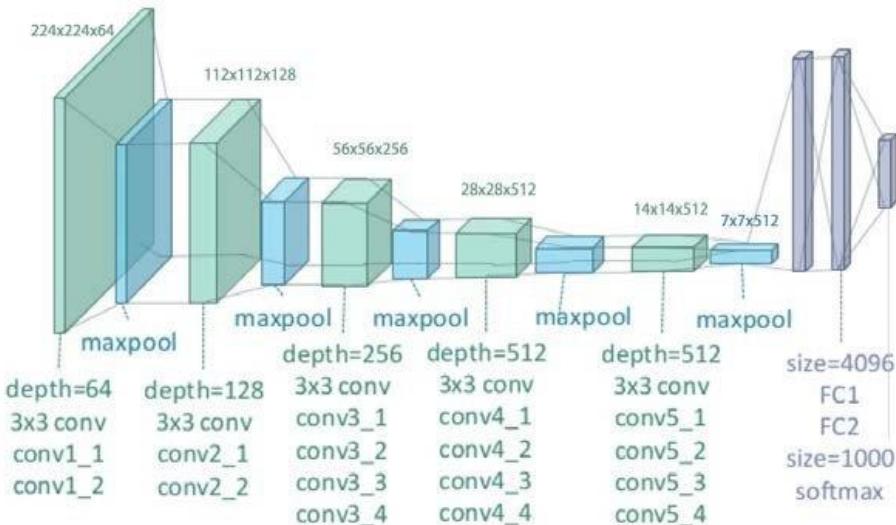
شبکه VGG در دو معماری مختلف با عنوان‌های ۱۶ و VGG ۱۹ ارائه شده است. ابتدا شبکه ۱۶ پیشنهاد شد و بعدها با تغییراتی جزئی در شبکه ۱۶، شبکه ۱۹ مطرح گردید. VGG19 به طور خلاصه از ۱۹ لایه شامل ۱۶ لایه کانولوشنی، ۳ لایه Fully Connected، ۵ لایه MaxPool و ۱ لایه تشکیل شده است.

به صورت جزئی‌تر شبکه ۱۹ VGG، شامل دو لایه کانولوشنی با  $64 \times 3 \times 3$  فیلتر و یک لایه ماکس‌پولینگ  $2 \times 2$  با پرش (Stride) به اندازه ۲ قرار گرفته است. این لایه ماکس‌پولینگ علاوه بر نمونه‌برداری، وظیفه کاهش بعد ویژگی‌ها به نصف را هم دارد.

در ادامه، دو لایه کانولوشنی دیگر با  $128 \times 3 \times 3$  و یک لایه ماکس‌پولینگ  $2 \times 2$  و پرش ۲ قرار گرفته‌اند. به طور مشابه، ۴ لایه کانولوشنی با  $256 \times 3 \times 3$  و یک لایه ماکس‌پولینگ  $2 \times 2$  با پرش ۲ قرار گرفته‌اند.

۴ لایه کانولوشنی با  $512 \times 3 \times 3$  و یک لایه ماکس‌پولینگ ادامه این شبکه هست که البته دو بار تکرار می‌شود. درنهایت، ویژگی‌ها تبدیل به یک بردار ویژگی می‌شوند تا در اختیار لایه‌های نورونی یا تمام اتصال یا Fully Connected قرار گیرند.

دو لایه FC به ابعاد ۴۰۹۶ و یک لایه FC به ابعاد ۱۰۰۰ پشت سر هم قرار گرفته‌اند. در حقیقت لایه FC آخر، یک لایه نورونی به ابعاد ۱۰۰۰ که متناظر با تعداد کلاس‌های کاربرد ما هست، در نظر گرفته شده است. باتوجه به اینکه پایگاه داده ImageNet شامل ۱۰۰۰ کلاس هست، در اینجا هم لایه خروجی شامل ۱۰۰۰ نورون است. در تمامی لایه‌های کانولوشنی و لایه‌های نورونی از تابع فعال‌ساز یا Activation Function بنام RELU استفاده شده است.



شکل ۱۷ معماری شبکه VGG19

در رابطه با نسخه‌های مختلف این شبکه، انواع دیگری از VGG مانند VGG16، VGG11 و ... وجود دارد. در حقیقت می‌توان در این شبکه تغییراتی به وجود آورد، به عنوان مثال، تعدادی از لایه‌های آن را کم کرد و یا تعدادی لایه به آن افزود، به عنوان مثال VGG16 شامل ۱۶ لایه کانولوشنی است (۳ لایه کمتر از VGG19). در مقایسه با VGG19، VGG16 کمی بهتر است اما حافظه بیشتری را نیاز دارد، به همین دلیل شبکه عصبی ۱۶ بسیار مورد توجه محققان و دانشجویان قرار گرفته و در زمینه‌های مختلفی از بینایی کامپیوتر از آن استفاده شده است.

از مزایای VGG19 می‌توان موارد زیر را نام برد:

۱. یک معماری خیلی خوب برای سنجش یک وظیفه مشخص است.
۲. شبکه‌های از قبل تعلیم دیده VGG به طور رایگان در اینترنت قرار دارند، به همین جهت در خیلی از اپلیکیشن‌ها استفاده می‌شوند.

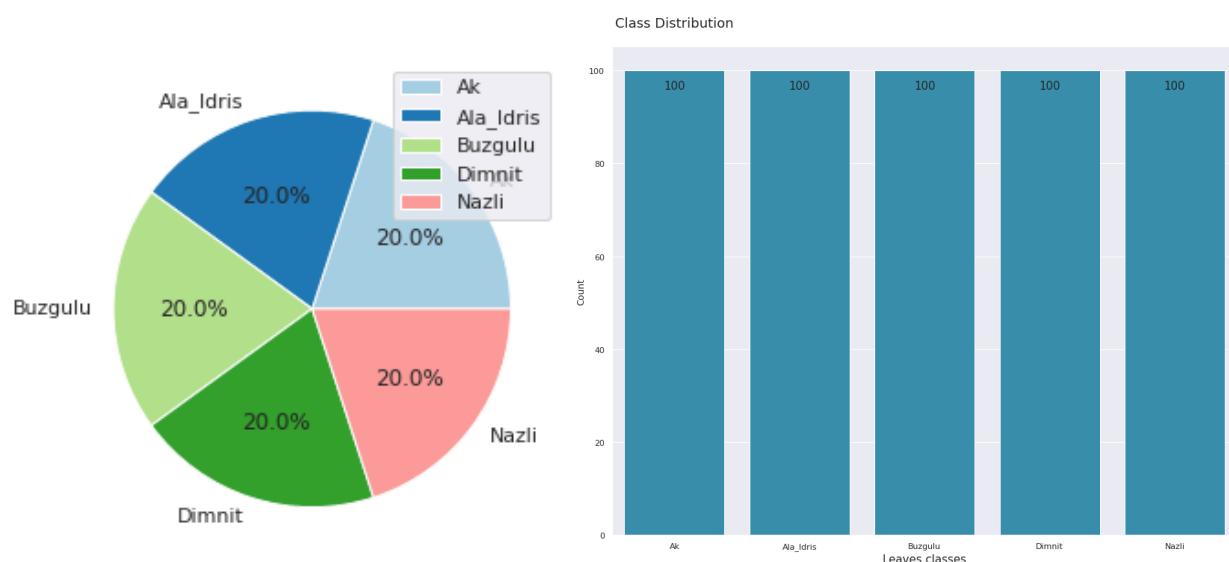
ولی مشکل اصلی این معماری این است که اگر بخواهید آن را از پایه تعلیم دهید، بسیار کند و زمان بر است. حتی در یک سیستم با GPU خوب هم راه اندازی آن بیشتر از یک هفته زمان می‌برد.

با توجه به اینکه معمولاً از transfer learning زمانی استفاده می‌شود که دیتاست کوچک باشد، بهتر است از نیز برای افزایش دادگان استفاده کنیم. درنتیجه من در این قسمت به کمک data augmentation به داده‌افزایی پرداختم.

در این روش ابتدا از داده‌ها با استفاده از کتابخانه opencv در پایتون که یک ابزار پردازش تصویر است ویژگی‌های از داده استخراج و به کمک این ویژگی‌ها به طبقه بندی می‌پردازیم. برای پردازش داده افزودن کل داده به محیط با هم باعث می‌شود رم crash شود برای همین با ساخت یک دیتا فریم ادرس تک به تک داده‌ها را به جای خود آن‌ها به همراه کلاس هر یک ذخیره می‌کنیم.

	images	classes	path
0	Ak (74).png	Ak	/content/Grapevine_Leaves_Image_Dataset/Ak/Ak ...
1	Ak (31).png	Ak	/content/Grapevine_Leaves_Image_Dataset/Ak/Ak ...
2	Ak (93).png	Ak	/content/Grapevine_Leaves_Image_Dataset/Ak/Ak ...
3	Ak (30).png	Ak	/content/Grapevine_Leaves_Image_Dataset/Ak/Ak ...
4	Ak (29).png	Ak	/content/Grapevine_Leaves_Image_Dataset/Ak/Ak ...

سپس چک می‌کنیم که داده‌ها مربوط به کلاس‌ها بالا نس هستند یا نه که می‌بینیم ۱۰۰ تصویر از هر کلاس داریم. با نمودار میله‌ای و دایره‌ای بالا نس بودن را نشان می‌دهیم. در تصاویر زیر این مسئله مشهود است. سپس چندین نمونه داده از هر کلاس را به نمایش می‌گذاریم.



شکل ۱۸ دو نمودار بالا برای چک کردن بالا نس بودن داده‌های ورودی است.

در این مرحله ابعاد داده را چک می کنیم که همه تصاویر یک ابعاد را داشته باشند و ابعاد را نیز در فایل مربوطه ذخیره می کنیم.

**طیف رنگی تصویر:** سپس چند چک ساده انجام می دهیم تا مطمئن شویم که داده‌ی ما در چه وضعیتی قرار دارد تا برای augmentation ایده در یافت کنیم ابتدا تابعی می‌نویسیم که به ما gray scale بودن و نبودن تصویر را نمایش دهد. برای این کار روی تصویر دور می‌زنیم و چک می‌کنیم که اگر سه بعد رنگ تصویر (rgb) با هم برابر بود یعنی تصویر gray scale است. برای این کار تعدادی نمونه دریافت و چک می‌کنیم و می‌بینیم با توجه به صفر شدن هیچ یک از تصاویر gray scale نیست.

```
[24] 1 sampleFrac = 0.5
2 #get our sampled images
3 isGreyList = []
4 for imageName in main_df['path'].sample(frac=sampleFrac):
5     val = Image.open(imageName).convert('RGB')
6     isGreyList.append(is_grey_scale(val))
7 print(np.sum(isGreyList) / len(isGreyList))
8 del isGreyList
```

**توزیع کanal‌های تصویر:** تصاویر طیف رنگ‌های مختلفی دارند که با دستور زیر تصاویر خود را به طیف rgb تبدیل می‌کنیم.

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

و سپس با جمع زن مقادیر هر کانال میزان آن رنگ در تصویر را بر می‌گردانیم. این تابع برای بیان میزان هر رنگ در تصاویر استفاده می‌شود.

```
1 def get_rgb_men(row):
2     img = cv2.imread(row['path'])
3     img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
4     return np.sum(img[:, :, 0]), np.sum(img[:, :, 1]), np.sum(img[:, :, 2])
5
6 tqdm.pandas()
7 main_df['R'], main_df['G'], main_df['B'] = zip(*main_df.progress_apply(lambda row: get_rgb_men(row), axis=1))
```

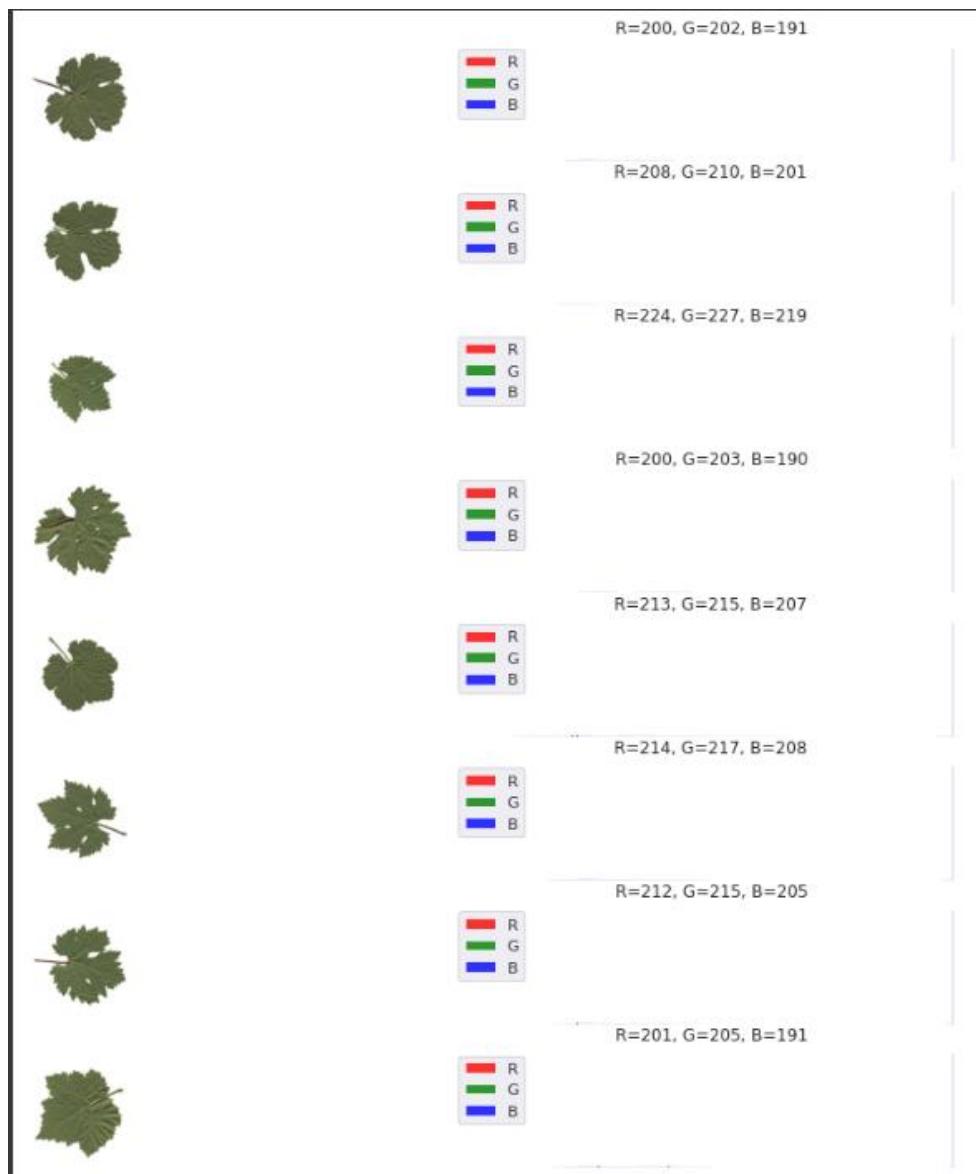
برای اینکه دقیقاً بدانیم هر رنگ در تصاویر چه توضیعی دارد بررسی می‌کنیم که در تصاویر رنگ غالب کدام است، یعنی رنگی که بیشتر است.

```
1 df = main_df[((main_df['B']) < main_df['R']) & ((main_df['G']) < main_df['R'])]
```

سپس آن تصاویری از داده‌ها به همراه میزان هر رنگ در تصویر را نمایش می‌دهیم. در اینجا همانطور که می‌بینید در هیچ یک از تصاویر قرمز و آبی بر سبز غالب نشده است و سبز همواره بیشترین مقدار را دارد.

Image intensity of selected color is weak

برای رنگ غالب در تصویر چند نمونه از تصاویر به همراه میزان هر رنگ در آن تصویر را نمایش می‌دهیم که به شکل زیر است.



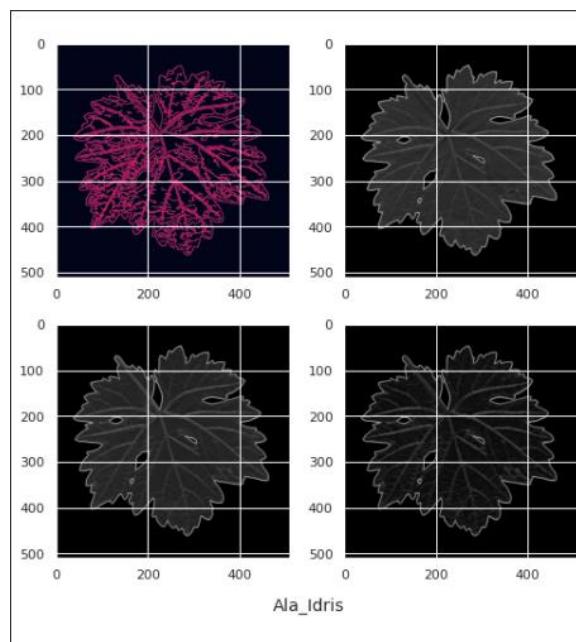
## افزودن داده با کمک ویژگی‌های استخراجی و آموزش مدل

شناسایی لبه تصویر: برای آشکارسازی لبه دو کار می‌توان کرد. می‌توان تصویر را ابتدا به **gray scale** تغییر داده و بعد با توابع **canny** برای شناسایی لبه استفاده کرد. یا می‌توان مستقیم خود تصویر **rgb** را به **sobel** داد برای شناسایی لبه که لبه افقی و عمودی و کلی را بدست می‌آورد. سوبیل یک روش مبتنی بر گرادیان بر

اساس مشتقات مرتبه اول است. اولین مشتقات تصویر را به طور جداگانه برای محورهای X و Y محاسبه می کند. اپراتور از دو هسته  $3 \times 3$  استفاده می کند که با تصویر اصلی در هم می آمیزد تا تقریب مشتقات را محاسبه کند - یکی برای تغییرات افقی و دیگری برای عمودی و در نهایت سه تصویر می دهد.

```
gray=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = sobel(image)
gray_edges=canny(gray)
dimension = edges.shape
```

در شکل زیر یک نمونه از تصاویر را می بینیم.

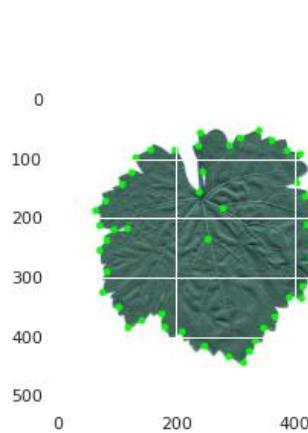


شکل ۱۹ آشکار ساز لبه

**آشکار ساز گو شه:** OpenCV یک تابع به نام `cv.goodFeaturesToTrack()` دارد. N قوی ترین گو شه تصویر را پیدا می کند. تصویر باید یک تصویر خاکستری باشد. سپس تعداد گوشه هایی را که می خواهید پیدا کنید مشخص می کنید. سپس سطح کیفیت را مشخص می کنید که مقداری بین ۰-۱ است که نشان دهنده حداقل کیفیت گوشه ای است که زیر آن همه رد می شوند. سپس حداقل فاصله اقلیدسی بین گوشه های شناسایی شده را ارائه می کنیم.

```
gray=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
corners_gray = cv2.goodFeaturesToTrack(gray, maxCorners=50, qualityLevel=0.02, minDistance=20)
```

یک نمونه از تصویر این بخش در زیر آورده شده است.

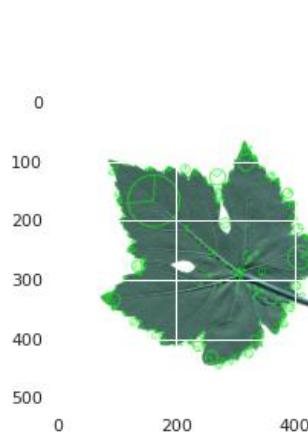


شکل ۲۰ آشکارساز گوشه

**ویژگی sift:** کلاس استخراج نقاط کلیدی و توصیفگرهای محاسباتی با استفاده از الگوریتم تبدیل ویژگی تغییر ناپذیر مقیاس (SIFT). برای انجام تشخیص ویژگی ها و تطبیق، از تابع sift استفاده می کنیم که با استفاده از آن، نمایش های برداری از محتواهای بصری تصویر برای انجام عملیات ریاضی روی آنها استخراج می شود. در این بخش ویژگی مربوطه را با کد زیر استخراج می کنیم.

```
gray=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
sift = cv2.SIFT_create()
kp, des = sift.detectAndCompute(gray, None)
kp_img = cv2.drawKeypoints(image, kp, None, color=(0, 255, 0), flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

یک نمونه از تصاویر در زیر آورده شده است.



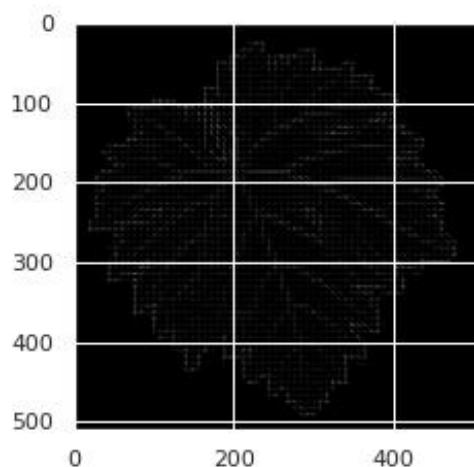
شکل ۲۱ ویژگی

**ویژگی HOG:** در توصیفگر ویژگی HOG، از توزیع (هیستوگرام) جهت گرادیان ها (گرادیان های جهت دار) به عنوان ویژگی استفاده می شود. گرادیان ها (مشتق های  $X$  و  $Y$ ) یک تصویر مفید هستند، زیرا بزرگی گرادیان ها در اطراف لبه ها و گوش ها زیاد است (مناطق با تغییرات شدید شدید) و می دانیم که لبه ها و گوش ها اطلاعات بسیار بیشتری در مورد شکل جسم نسبت به مناطق مسطح دارند. برای استخراج این ویژگی از کد زیر استفاده می کنیم.

```
fd, hog_image = hog(image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=True, multichannel=True)
```

یک نمونه ای از تصاویر در شکل زیر اورده شده است.

Ak



شکل ۲۲ ویژگی HOG

با کمک ویژگی های بالا و با augment کردن تصاویر مدل پیش تعلیم یافته vgg را تعلیم می دهیم.

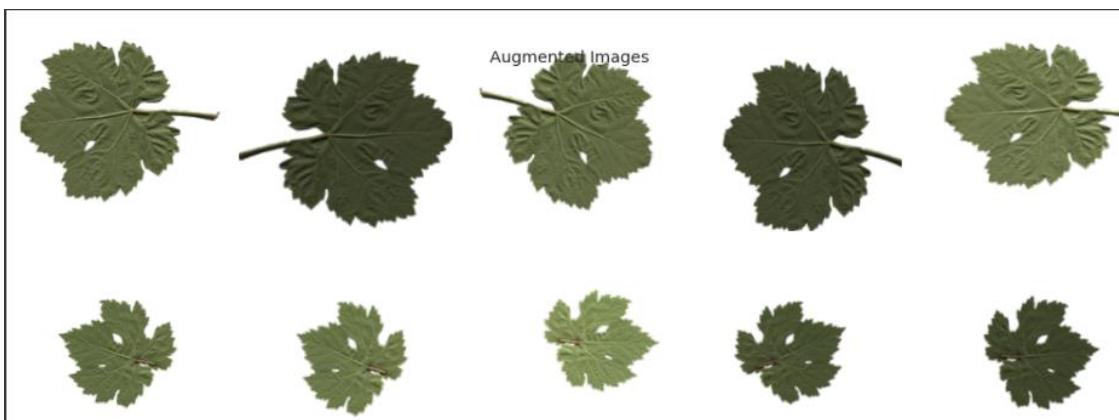
```

from keras.applications.vgg19 import preprocess_input
vgg_datagen = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.10,
    brightness_range=[0.6,1.4],
    channel_shift_range=0.7,
    width_shift_range=0.15,
    height_shift_range=0.15,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode='nearest',
    preprocessing_function=preprocess_input
)

train_generator_vgg = vgg_datagen.flow_from_dataframe(
    X_train, # This is the source directory for training images
    x_col='path',
    y_col='classes',
    target_size=(227, 227), # All images will be resized to 150x150
    batch_size=32,
    class_mode="categorical",
    shuffle=True,
)
val_generator_vgg = vgg_datagen.flow_from_dataframe(
    X_val, # This is the source directory for training images
    x_col='path',
    y_col='classes',
    target_size=(227, 227), # All images will be resized to 150x150
    batch_size=32,
    class_mode="categorical",
    shuffle=True,
)

```

برای افزایش داده از همان روش‌هایی که در بخش قبل بیان شده است استفاده می‌کنیم. چند نمونه از تصاویر augment شده در زیر آمده است.



شکل ۲۳ تصاویر augment شده

مدل به شرح زیر است ابتدا از vgg19 استفاده می کنیم به عنوان مدل pretrain که می گوییم آموزش نبیند و سپس یک لایه‌ی GlobalAveragePooling2D و در نهایت یه لایه DENSE که پنج نورون(تعداد کلاس‌های خروجی) دارد، قرار می‌دهیم.

```

1 vgg19 = VGG19(include_top = False, input_shape = (227,227,3), weights = 'imagenet')
2
3 # training of all the convolution is set to false
4 for layer in vgg19.layers:
5     layer.trainable = False
6
7 x = GlobalAveragePooling2D()(vgg19.output)
8 predictions = Dense(5, activation='softmax')(x)
9
10 model_vgg = Model(inputs = vgg19.input, outputs = predictions)
```

سپس مدل را کامپایل و fit می‌کنیم.

```

1 model_vgg.compile(loss='categorical_crossentropy', optimizer="adam", metrics=[ 'accuracy'])

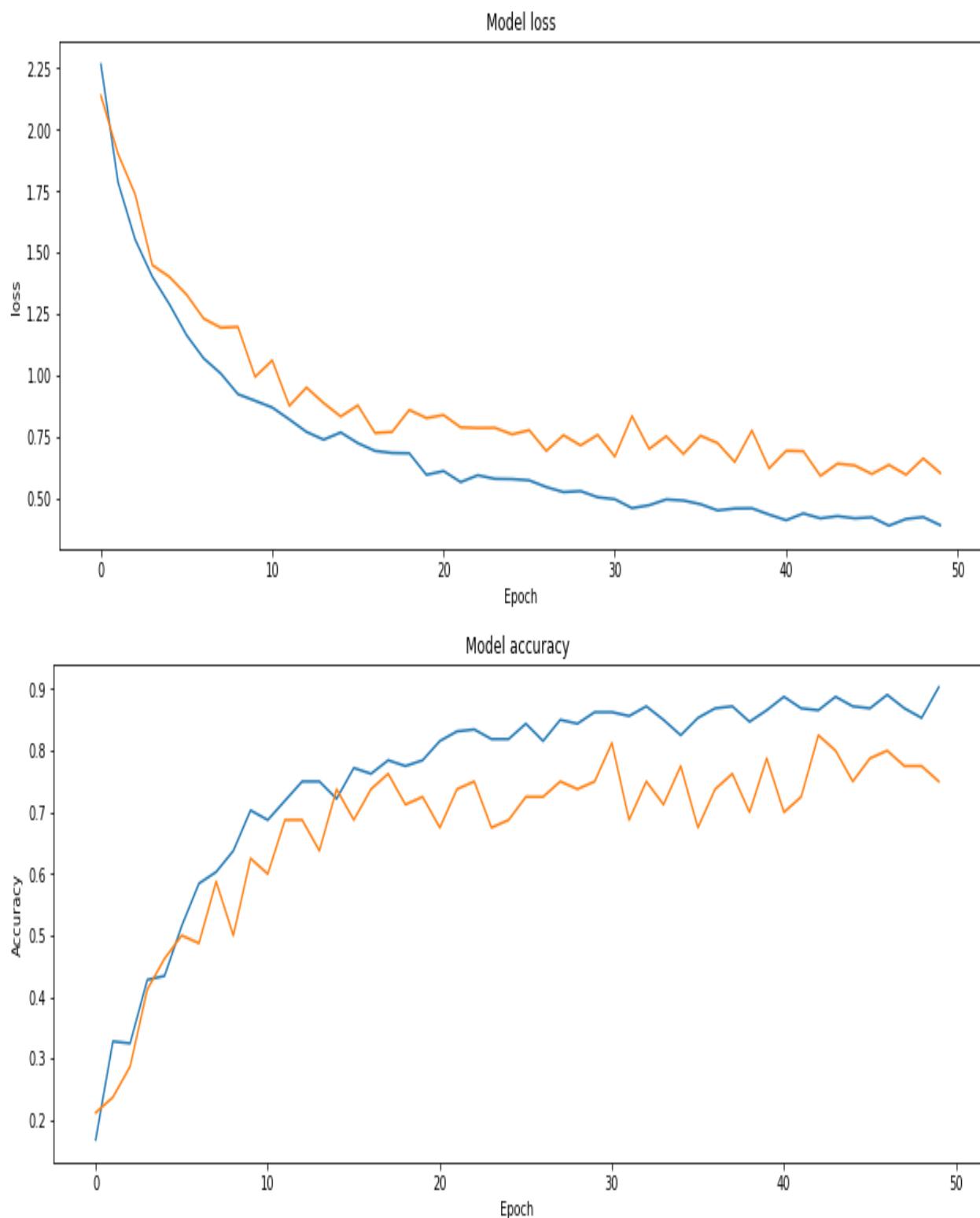
history_vgg = model_vgg.fit(
    train_generator_vgg,
    validation_data=val_generator_vgg,
    epochs=50,
    verbose=2)
```

نتایج این بخش برای ۱۳ ایپاک آخر به شرح زیر است:

```

10/10 - 267s - loss: 0.4777 - accuracy: 0.8531 - val_loss: 0.7563 - val_accuracy: 0.6750 - 267s/epoch - 27s/step
Epoch 37/50
10/10 - 268s - loss: 0.4525 - accuracy: 0.8687 - val_loss: 0.7260 - val_accuracy: 0.7375 - 268s/epoch - 27s/step
Epoch 38/50
10/10 - 270s - loss: 0.4602 - accuracy: 0.8719 - val_loss: 0.6490 - val_accuracy: 0.7625 - 270s/epoch - 27s/step
Epoch 39/50
10/10 - 271s - loss: 0.4611 - accuracy: 0.8469 - val_loss: 0.7775 - val_accuracy: 0.7000 - 271s/epoch - 27s/step
Epoch 40/50
10/10 - 270s - loss: 0.4356 - accuracy: 0.8656 - val_loss: 0.6232 - val_accuracy: 0.7875 - 270s/epoch - 27s/step
Epoch 41/50
10/10 - 268s - loss: 0.4125 - accuracy: 0.8875 - val_loss: 0.6947 - val_accuracy: 0.7000 - 268s/epoch - 27s/step
Epoch 42/50
10/10 - 291s - loss: 0.4400 - accuracy: 0.8687 - val_loss: 0.6933 - val_accuracy: 0.7250 - 291s/epoch - 29s/step
Epoch 43/50
10/10 - 272s - loss: 0.4202 - accuracy: 0.8656 - val_loss: 0.5928 - val_accuracy: 0.8250 - 272s/epoch - 27s/step
Epoch 44/50
10/10 - 270s - loss: 0.4293 - accuracy: 0.8875 - val_loss: 0.6427 - val_accuracy: 0.8000 - 270s/epoch - 27s/step
Epoch 45/50
10/10 - 271s - loss: 0.4202 - accuracy: 0.8719 - val_loss: 0.6353 - val_accuracy: 0.7500 - 271s/epoch - 27s/step
Epoch 46/50
10/10 - 272s - loss: 0.4242 - accuracy: 0.8687 - val_loss: 0.6006 - val_accuracy: 0.7875 - 272s/epoch - 27s/step
Epoch 47/50
10/10 - 277s - loss: 0.3905 - accuracy: 0.8906 - val_loss: 0.6375 - val_accuracy: 0.8000 - 277s/epoch - 28s/step
Epoch 48/50
10/10 - 273s - loss: 0.4177 - accuracy: 0.8687 - val_loss: 0.5974 - val_accuracy: 0.7750 - 273s/epoch - 27s/step
Epoch 49/50
10/10 - 272s - loss: 0.4253 - accuracy: 0.8531 - val_loss: 0.6639 - val_accuracy: 0.7750 - 272s/epoch - 27s/step
Epoch 50/50
10/10 - 272s - loss: 0.3922 - accuracy: 0.9031 - val_loss: 0.6045 - val_accuracy: 0.7500 - 272s/epoch - 27s/step
```

نتایج loss و accuracy برای دادگان train و validation برابر شد با :



شکل ۲۴ نمایش loss و accuracy برای دادگان train و validation در مدل ۳

حال کافیست از این مدل برای ارزیابی دادگان `test` استفاده کنیم:

```
from keras.applications.vgg19 import preprocess_input

test_generator_vgg = vgg_datagen.flow_from_dataframe(
    X_test, # This is the source directory for training images
    x_col='path',
    y_col='classes',
    target_size=(227, 227), # All images will be resized to 150x150
    batch_size=128,
    class_mode="categorical",
    shuffle=True,
)

Found 100 validated image filenames belonging to 5 classes.

# evaluate accuracy on test data
test_loss, test_acc = model_vgg.evaluate(test_generator_vgg, verbose=2)
print('\nTest accuracy:', test_acc,'Test loss:',test_loss)

1/1 - 69s - loss: 0.6922 - accuracy: 0.7700 - 69s/epoch - 69s/step

Test accuracy: 0.7699999809265137 Test loss: 0.6922000646591187
```

همانگونه که می‌بینیم، درنهایت در این مدل نتیجه‌ی طبقه‌بندی دادگان به میزان ۷۷ درصد رسید و نسبت به حالت قبل بهبود یافت.

نکته: شبکه‌های `resnet` و `alexnet` نیز برای این حالت بررسی شدند ولی با توجه به اینکه نتایج خوبی نداشتند در نهایت گزارش نشدند ولی کد مربوط به این دو شبکه نیز در فایل اصلی پروژه موجود است.

## حالت چهارم : پیاده‌سازی شبکه‌ی pretrained-VGG19 پس از اعمال Data Augmentation و استفاده از K Fold

در این قسمت نیز مشابه حالت سوم از شبکه‌ی VGG19 و Data Augmentation استفاده می‌کنم، با این تفاوت که اینبار برای آموزش شبکه از روش k fold استفاده می‌کنم. در ابتدا مشابه قبل، augmentation داریم:

```
new_x_train = np.zeros((2800,153,153,3))
# datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, rotation_range=90,brightness_range=[0.1,1.0], horizontal_flip=True)

m=0
for j in range (400):
    data = img_to_array(x_train[j,:,:,:])
    for i in range(7):
        samples1 = expand_dims(data,0)
        it1 = datagen.flow(samples1, batch_size=1)
        batch1 = it1.next()
        image1 = batch1[0].astype('uint8')
        new_x_train[m,:,:,:]= batch1
        m+=1

new_y_train = np.zeros((2800,5))
m=0
for j in range (400):
    data = y_train[j]
    for i in range(7):
        new_y_train[m,:] = data
        m+=1

new_x_train = convert(new_x_train, 0, 255, np.uint8)

print(np.shape(new_x_train))
print(np.shape(new_y_train))
(2800, 153, 153, 3)
```

در ادامه میبایست مجموعه داده را به شکل (۳۲،۳۲،۳) درآوریم:

```
trainX = np.zeros((2800,32,32,3))
for i in range (2800):
    data = new_x_train[i]
    width, height = int(data.shape[1]/4.78125), int(data.shape[0]/4.78125)
    resized_data = cv2.resize(data,
                            (width, height),
                            interpolation = cv2.INTER_AREA,)
    trainX[i,:,:,:]= resized_data

trainX = convert(trainX, 0, 255, np.uint8)
trainY = new_y_train

print('Train: X=%s, y=%s' % (trainX.shape, trainY.shape))
```

از طرفی پیاده سازی مدل pretrained-VGG دارای ۴ کاربرد مهم است که به شرح زیر است:

۱. به عنوان یک مدل استخراج ویژگی
۲. استفاده از مدل های از پیش آموزش داده شده (pretrained-VGG)
۳. تنظیم دقیق (Fine tuning) مدل های از پیش آموزش داده شده
۴. استفاده از وزن های مدل از پیش آموزش داده شده برای وزن دهی اولیه

در این قسمت از معماری VGG19 که در قسمت قبل گفتیم، استفاده کرده و آن را با مجموعه داده های خود از ImageNet آموزش می دهیم. بنابراین درنهایت در این قسمت از VGG-19 از پیش آموزش دیده با وزن های (با مشخص کردن weights='imagenet') استفاده کردیم، لایه FC را نیز با include\_top=False حذف کردم زیرا باید ویژگی ها را از تصویر دریافت کنیم، لایه های افزوده شده به شکل زیر است :

```
image_input = tf.keras.layers.Input(shape=(32,32, 3))
baseModel_VGG_19 = tf.keras.applications.VGG19(include_top=False,weights='imagenet',input_tensor=image_input)
baseModel_VGG_19.summary()

FC_layer_Flatten = tf.keras.layers.Flatten()(baseModel_VGG_19.output)
Dense=tf.keras.layers.Dense(units=1000,activation="relu")(FC_layer_Flatten)
Dense=tf.keras.layers.Dense(units=800,activation="relu")(Dense)
Dense=tf.keras.layers.Dropout(.3)(Dense)
Dense=tf.keras.layers.Dense(units=400,activation="relu")(Dense)
Dense=tf.keras.layers.Dense(units=200,activation="relu")(Dense)
Dense=tf.keras.layers.Dense(units=100,activation="relu")(Dense)
Classification=tf.keras.layers.Dense(units=5,activation="softmax")(Dense)
```

برای آموزش شبکه از روش k fold استفاده کردم، در ادامه این روش نیز توضیح داده می شود.

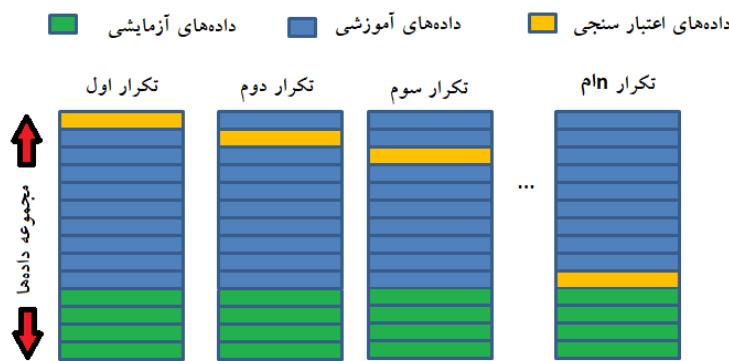
## روش k Fold cross validation

در روش اعتبارسنجی متقابل (CV)، طی یک فرآیند تکرار شونده، قسمت داده های آموزش (Training set) که به منظور مدل سازی به کار می رود، خود به دو بخش تفکیک می شود. در هر بار تکرار فرآیند CV، بخشی از داده ها برای آموزش و بخشی دیگر برای آزمایش مدل به کار می رود.

به این ترتیب این فرآیند یک روش بازنمونه گیری به منظور برآورد خطای مدل محسوب می شود. باید توجه داشت که داده های آزمایشی در فرایند CV ممکن است در تکرار بعدی به عنوان داده های آموزشی به کار روند، در نتیجه ماهیت آن ها با داده هایی که به عنوان داده های آزمایشی (Test set) معرفی شد، متفاوت است.

شاید تصویر زیر به درک ماهیت داده های تست در فرآیند CV کمک کند. مشخص است که داده های اعتبارسنجی بخشی از داده های آموزشی هستند و داده های آزمایشی نیز به عنوان بخشی مجزا از داده های آموزشی فرض شده اند. مراحل تکرار فرآیند CV نیز در تصویر به خوبی دیده می شود. نکته دیگری که در تصویر مشخص است،

مکمل بودن مجموعه داده‌های آموزشی و اعتبارسنجی است. با انتخاب بخشی از داده‌ها برای انجام فرایند CV، بقیه داده‌ها برای آموزش به کار گرفته می‌شوند.



شکل ۲۵ نحوه عملکرد فرایند k fold cross validation

در هر مرحله از فرایند CV، مدل بدست آمده توسط داده‌های آزمایشی برای پیش‌بینی داده‌های CV به کار گرفته و خطای یا دقت حاصل از برآش مدل روی داده‌های CV محاسبه می‌شود. معمولاً میانگین این خطاهای (دقتهای) به عنوان خطای (دقت) کلی مدل در نظر گرفته می‌شود. البته بهتر است انحراف معیار خطاهای (دقتهای) نیز گزارش شود. به این ترتیب با توجه به تعداد پارامترهای مختلف (پیچیدگی مدل)، می‌توان مدل‌های متفاوتی تولید و خطای برآورد آن‌ها را به کمک روش CV اندازه‌گیری کرد. در انتهای مدلی را به عنوان مدل مناسب انتخاب خواهیم کرد که دارای کمترین برآورد خطای باشد.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.losses import sparse_categorical_crossentropy
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
# y_pred = model.predict(X_test)

acc_per_fold = []
loss_per_fold = []

acc = []
val_acc = []
loss = []
val_loss = []
# Define the K-fold Cross Validator
num_folds = 10
kfold = KFold(n_splits=num_folds, shuffle=True)

# K-fold Cross Validation model evaluation
fold_no = 10

early = tf.keras.callbacks.EarlyStopping(monitor="val_loss", mode="min", patience=10)
check_point = tf.keras.callbacks.ModelCheckpoint("./k_fold_model.h5", monitor="val_loss", save_best_only=True)
rreduce = tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.01, patience=6)

callbacks_list = [early, check_point, rreduce]
```

```

for train, test in kfold.split(trainX,trainY):

    # # Define the model_final architecture
    IMG_HEIGHT = 128

    # Compile the model
    model_final.compile(optimizer='RMSprop',
                        loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                        metrics=['accuracy'])

    # Generate a print
    # print('-----')
    # print(f'Training for fold {fold_no} ...')

    # Fit data to model

    epochs = 50
    batch_size = 30
    history = model_final.fit(
        trainX[train],
        to_categorical(trainY[train]),
        steps_per_epoch= len(trainX) // batch_size,
        epochs=epochs
    )
    acc.append(history.history['accuracy'])
    loss.append(history.history['loss'])
    # Generate generalization metrics
    scores = model_final.evaluate(trainX[test],trainY[test] verbose=0)
    print(f'Score for fold {fold_no}: {model_final.metrics_names[0]} of {scores[0]}; {model_final.metrics_names[1]} of {scores[1]}')
    acc_per_fold.append(scores[1])
    loss_per_fold.append(scores[0])

    # Increase fold number
    fold_no = fold_no + 1

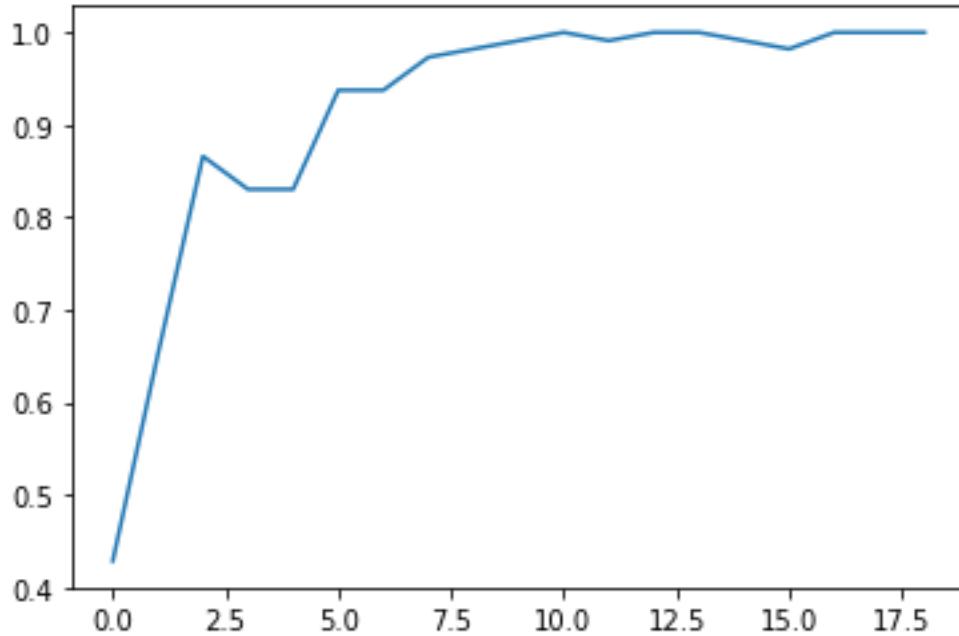
```

نتیجه آموزش شبکه پس از ۵۰ ایپاک به شکل زیر شد :

```

87/87 [=====] - 35s 402ms/step - loss: 0.2938 - accuracy: 0.8996
87/87 [=====] - 34s 396ms/step - loss: 0.2965 - accuracy: 0.8924
87/87 [=====] - 34s 395ms/step - loss: 0.2790 - accuracy: 0.8982
87/87 [=====] - 34s 395ms/step - loss: 0.2571 - accuracy: 0.9083
87/87 [=====] - 34s 396ms/step - loss: 0.2379 - accuracy: 0.9112
87/87 [=====] - 34s 396ms/step - loss: 0.2289 - accuracy: 0.9167
87/87 [=====] - 34s 396ms/step - loss: 0.2394 - accuracy: 0.9130
87/87 [=====] - 34s 396ms/step - loss: 0.2184 - accuracy: 0.9297
87/87 [=====] - 34s 396ms/step - loss: 0.1962 - accuracy: 0.9297
87/87 [=====] - 34s 394ms/step - loss: 0.1687 - accuracy: 0.9446
87/87 [=====] - 34s 395ms/step - loss: 0.1490 - accuracy: 0.9543
87/87 [=====] - 34s 393ms/step - loss: 0.1789 - accuracy: 0.9406
87/87 [=====] - 34s 393ms/step - loss: 0.1693 - accuracy: 0.9446
87/87 [=====] - 34s 393ms/step - loss: 0.1441 - accuracy: 0.9540
87/87 [=====] - 34s 394ms/step - loss: 0.1411 - accuracy: 0.9507
87/87 [=====] - 34s 394ms/step - loss: 0.1533 - accuracy: 0.9486
87/87 [=====] - 34s 395ms/step - loss: 0.1493 - accuracy: 0.9457
87/87 [=====] - 34s 394ms/step - loss: 0.1139 - accuracy: 0.9645
87/87 [=====] - 34s 393ms/step - loss: 0.1024 - accuracy: 0.9652
87/87 [=====] - 34s 394ms/step - loss: 0.1313 - accuracy: 0.9562
87/87 [=====] - 34s 395ms/step - loss: 0.1229 - accuracy: 0.9605
87/87 [=====] - 34s 394ms/step - loss: 0.1277 - accuracy: 0.9601
87/87 [=====] - 34s 394ms/step - loss: 0.1051 - accuracy: 0.9638
87/87 [=====] - 34s 394ms/step - loss: 0.0970 - accuracy: 0.9710
87/87 [=====] - 34s 394ms/step - loss: 0.0937 - accuracy: 0.9641
87/87 [=====] - 34s 394ms/step - loss: 0.0860 - accuracy: 0.9736
87/87 [=====] - 34s 394ms/step - loss: 0.0786 - accuracy: 0.9775
87/87 [=====] - 36s 417ms/step - loss: 0.0762 - accuracy: 0.9739
87/87 [=====] - 34s 394ms/step - loss: 0.1031 - accuracy: 0.9659
87/87 [=====] - 34s 394ms/step - loss: 0.0697 - accuracy: 0.9786
87/87 [=====] - 34s 394ms/step - loss: 0.1101 - accuracy: 0.9630
87/87 [=====] - 34s 394ms/step - loss: 0.0680 - accuracy: 0.9804
87/87 [=====] - 34s 394ms/step - loss: 0.0992 - accuracy: 0.9667
87/87 [=====] - 34s 394ms/step - loss: 0.0959 - accuracy: 0.9652
87/87 [=====] - 34s 394ms/step - loss: 0.0578 - accuracy: 0.9822
87/87 [=====] - 34s 395ms/step - loss: 0.0438 - accuracy: 0.9873
87/87 [=====] - 34s 395ms/step - loss: 0.0448 - accuracy: 0.9873

```



شکل ۲۶ نتیجه آموزش شبکه VGG19 پس از ۵۰ اپیک برای مدل ۴

در ادامه مشابه حالت‌های قبل می‌توان مدل فوق را نیز بر روی دادگان `test` ارزیابی کرد و معیارهای محاسبه شده در قسمت قبل را نیز برای دادگان تست با این مدل به دست آوریم:

```
# evaluate accuracy on test data
test_loss, test_acc = model_final.evaluate(x_test, to_categorical(y_test), verbose=2)
print('\nTest accuracy:', test_acc, 'Test loss:', test_loss)

4/4 - 3s - loss: 1.3857 - accuracy: 0.8100 - 3s/epoch - 803ms/step

Test accuracy: 0.8100000023841858 Test loss: 1.385744571685791
```

```
prediction = model_final.predict(x_test)
y_pred_bool = np.argmax(prediction, axis=1)
y_test_bool = y_test
```

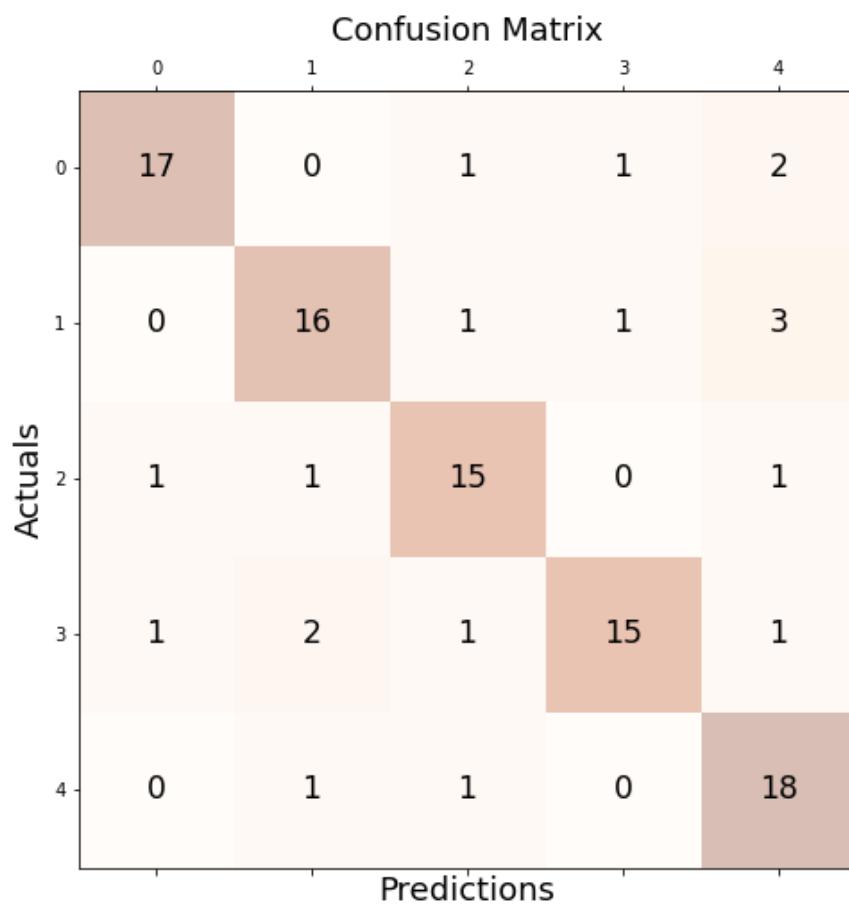
```

#from sklearn.metrics.confusion_matrix import confusion_matrix
# showing the results
y_pred = prediction
y_test_bool = np.argmax(testY, axis=1)
y_pred_bool = np.argmax(y_pred, axis=1)

conf_matrix = sklearn.metrics.confusion_matrix(y_pred=y_pred_bool , y_true=y_test_bool)
fig, ax = plt.subplots(figsize=(8, 8))
ax.matshow(conf_matrix, cmap=plt.cm.Oranges, alpha=0.3)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()

```



شکل ۲۷ نمایش confusion matrix برای مدل ۴

print(classification_report(y_test_bool, y_pred_bool))				
	precision	recall	f1-score	support
0	0.89	0.81	0.85	21
1	0.80	0.76	0.78	21
2	0.79	0.83	0.81	18
3	0.88	0.75	0.81	20
4	0.72	0.90	0.80	20
accuracy			0.81	100
macro avg	0.82	0.81	0.81	100
weighted avg	0.82	0.81	0.81	100

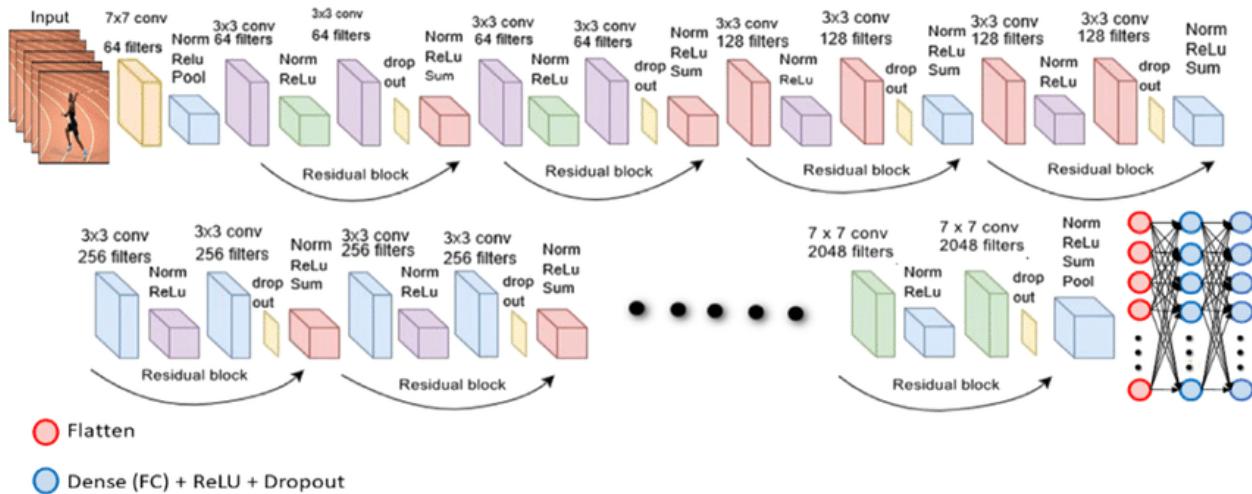
شکل ۲۸ نمایش precision, recall, F1 score برای مدل<sup>۴</sup>

پس از اینکه مدل VGG19 را با استفاده از این مجموعه داده‌ی ۵ کلاسه مجدد آموزش دادم نتیجه گرفته شده خوب شد و Accuracy نسبت به حالات قبل افزایش یافت و در حقیقت نتیجه به میزان زیادی بهبود یافت.

در حقیقت تاثیر شبکه‌های عمیقی نظری VGG19 در افزایش accuracy و کاهش loss منجر به افزایش اهمیت استفاده از این شبکه‌ها از طریق Transfer learning شد تا ضمن استفاده از مزیت‌های این شبکه‌های بسیار عمیق، سرعت آموزش شبکه برای "دیتاست‌های قابل تشخیص توسط مدل" به طور باور نکردنی زیاد شود. همچنین استفاده از k fold نیز منجر به افزایش صحت شد.

## حالت پنجم : پیاده‌سازی شبکه‌های ResNet

یک معماری شبکه عصبی است که برای طبقه بندی تصویر، رگرسیون و استخراج ویژگی استفاده می‌شود. از اتصالات پرش برای اضافه کردن ورودی گروهی از کانولوشن‌ها به خروجی خود استفاده می‌کند. ساختار رزنت مورد استفاده در شکل زیر آورده شده است.



شکل ۱۰۱v۲<sup>۲۹</sup>

از ویژگی‌های این معماری می‌توان به سرعت آموزش آن اشاره کرد.

در ابتدا با استفاده از augment کردن تعداد داده‌ها را از ۱۰۰ تصویر در هر کلاس به حدوداً ۳۰۰ تصویر در هر کلاس می‌رسانیم.

```
target=300
gen=ImageDataGenerator(horizontal_flip=True, rotation_range=20, width_shift_range=.2,
                      height_shift_range=.2, zoom_range=.2)
groups=df.groupby('labels')
for label in df['labels'].unique():
    group=groups.get_group(label)
    sample_count=len(group)
    if sample_count< target:
        aug_img_count=0
        delta=target-sample_count
        target_dir=os.path.join(aug_dir, label)
        aug_gen=gen.flow_from_dataframe( group, x_col='filepaths', y_col=None, target_size=(66,66), class_mode=None,
                                         batch_size=1, shuffle=False, save_to_dir=target_dir, save_prefix='aug-',
                                         save_format='jpg')
        while aug_img_count<delta:
            images=next(aug_gen)
            aug_img_count += len(images)
```

که داده‌ها از ۵۰۰ داده به حدود ۱۵۰۰ داده افزایش می‌یابند. سپس داده‌های مربوط به تست و آموزش و اعتبارسنجی را جدا می‌کنیم.

```

train_split=.8
valid_split=.1
dummy_split=valid_split/(1-train_split)
train_df, dummy_df=train_test_split(ndf, train_size=train_split, shuffle=True, random_state=123)
valid_df, test_df=train_test_split(dummy_df, train_size=dummy_split, shuffle=True, random_state=123)
print ('train_df length: ', len(train_df), ' test_df length: ', len(test_df), ' valid_df length: ', len(valid_df))

train_df length: 1220  test_df length: 153  valid_df length: 152

```

سپس با توجه به افزایش داده‌ها احتمال crash شدن رم از مدل استفاده می‌کنیم که داده‌ها را کم کم به شبکه می‌دهد و لزومی ندارد همه‌ی داده‌ها را با هم لود کنیم.

```

height=60
width=60
channels=3
batch_size=40
img_shape=(height, width, channels)
img_size=(height, width)
length=len(test_df)
test_batch_size=sorted([int(length/n) for n in range(1,length+1) if length % n ==0 and length/n<=80],reverse=True)[0]
test_steps=int(length/test_batch_size)
print ('test batch size: ',test_batch_size, ' test steps: ', test_steps)
def scalar(img):
    return img
trgen=ImageDataGenerator(preprocessing_function=scalar, horizontal_flip=True)
tvgan=ImageDataGenerator(preprocessing_function=scalar)
sdir='../input/grapesImages/Image/Images'
train_gen=trgen.flow_from_dataframe( train_df, x_col='filepaths', y_col='labels', target_size=img_size, class_mode='categorical',
                                      color_mode='rgb', shuffle=True, batch_size=batch_size)
test_gen=tvgan.flow_from_dataframe( test_df, x_col='filepaths', y_col='labels', target_size=img_size, class_mode='categorical',
                                      color_mode='rgb', shuffle=False, batch_size=test_batch_size)
valid_gen=tvgan.flow_from_dataframe( valid_df, x_col='filepaths', y_col='labels', target_size=img_size, class_mode='categorical',
                                      color_mode='rgb', shuffle=True, batch_size=batch_size)
classes=list(train_gen.class_indices.keys())
class_count=len(classes)
train_steps=int(len(train_gen.labels)/batch_size)

test batch size: 51  test steps: 3
Found 1220 validated image filenames belonging to 5 classes.
Found 153 validated image filenames belonging to 5 classes.
Found 152 validated image filenames belonging to 5 classes.

```

برای مدل ابتدا از resnet101v2 استفاده می‌کنیم و سپس از یک لایه Dense با ۲۵۶ نورون و در نهایت نیز باز یک لایه Dense که ۵ نورون (تعداد کلاس‌ها) دارد. در ادامه مدل را کامپایل می‌کنیم.

```

model_name='ResNet101V2'
basemodel = tf.keras.applications.ResNet101V2(weights = "imagenet",input_shape = img_shape,include_top=False,pooling='max')
x=basemodel.output
x=keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x)
x = Dense(256, kernel_regularizer = regularizers.l2(l = 0.02),activity_regularizer=regularizers.l1(0.005),
           bias_regularizer=regularizers.l1(0.006) ,activation='relu')(x)
x=Dropout(rate=.5, seed=123)(x)
output = Dense(class_count, activation="softmax")(x)
model = tf.keras.Model(inputs = basemodel.inputs, outputs = output)
model.compile(Adamax(learning_rate=.001), loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
plot_model(model, to_file='ResNet101V2_model_plot.png', show_shapes=True, show_layer_names=True)

```

سپس مدل را fit می‌کنیم. برای این مدل از callback نیز استفاده می‌کنیم تا بهترین وزن آموزش شده را ذخیره کند و ضریب یادگیری را نیز کاهش دهد در صورت عدم بهبود نتایج در چند epoch.

```

        callbacks=[LRA(model=model,patience=patience,stop_patience=stop_patience, threshold=threshold,
                         factor=factor, model_name=model_name, freeze=freeze, batches=batches,initial_epoch=0,epochs=epochs )]

    history=model.fit(x=train_gen, epochs=epochs, verbose=0, callbacks=callbacks, validation_data=valid_gen,
                       validation_steps=None, shuffle=False, initial_epoch=0)

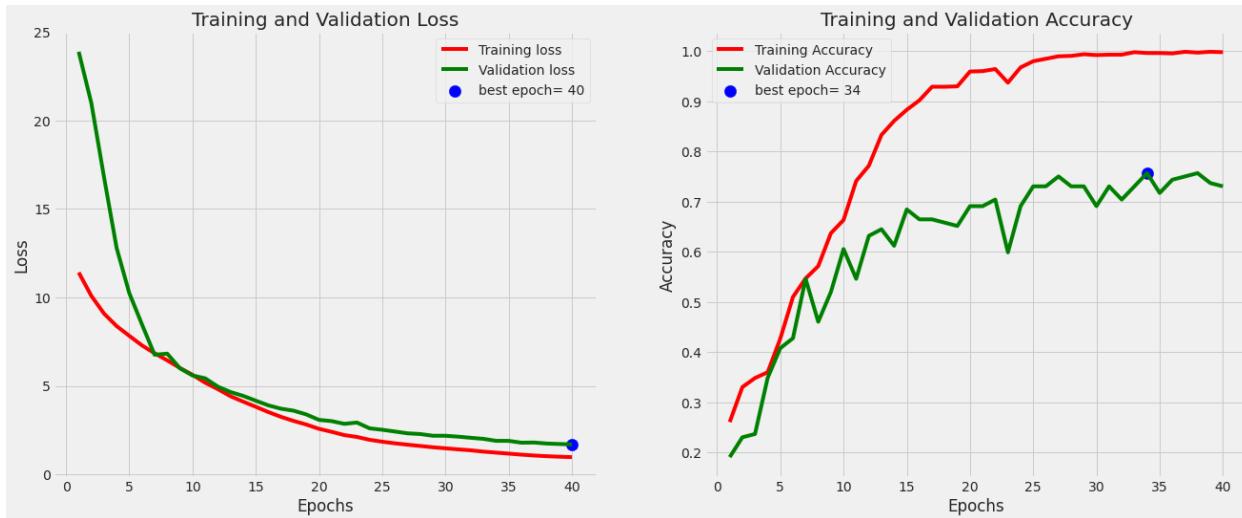
```

نتایج ۲۰ آخر تعلیم به شرح زیر است.

								val_loss	
20 /40	2.571	95.902	3.07541	69.079	0.00100	0.00100	val_loss	233.03	
21 /40	2.402	95.984	3.00635	69.079	0.00100	0.00100	val_loss	230.88	
22 /40	2.217	96.393	2.85304	70.395	0.00100	0.00100	val_loss	233.63	
23 /40	2.112	93.689	2.92488	59.868	0.00100	0.00050	val_loss	231.63	
24 /40	1.950	96.721	2.59896	69.079	0.00050	0.00050	val_loss	231.27	
25 /40	1.843	97.951	2.51785	73.026	0.00050	0.00050	val_loss	232.52	
26 /40	1.752	98.443	2.41896	73.026	0.00050	0.00050	val_loss	232.74	
27 /40	1.674	98.934	2.31646	75.000	0.00050	0.00050	val_loss	234.41	
28 /40	1.602	99.016	2.27640	73.026	0.00050	0.00050	val_loss	239.51	
29 /40	1.529	99.344	2.18064	73.026	0.00050	0.00050	val_loss	239.93	
30 /40	1.470	99.180	2.17791	69.079	0.00050	0.00050	val_loss	237.21	
31 /40	1.411	99.262	2.12886	73.026	0.00050	0.00050	val_loss	241.75	
32 /40	1.357	99.262	2.06422	70.395	0.00050	0.00050	val_loss	242.12	
33 /40	1.283	99.754	2.00376	73.026	0.00050	0.00050	val_loss	244.51	
34 /40	1.227	99.590	1.89348	75.658	0.00050	0.00050	val_loss	241.58	
35 /40	1.169	99.590	1.89069	71.711	0.00050	0.00050	val_loss	241.63	
36 /40	1.113	99.508	1.78716	74.342	0.00050	0.00050	val_loss	239.07	
37 /40	1.063	99.836	1.79690	75.000	0.00050	0.00025	val_loss	239.05	
38 /40	1.025	99.672	1.73938	75.658	0.00025	0.00025	val_loss	237.70	
39 /40	0.995	99.836	1.71336	73.684	0.00025	0.00025	val_loss	240.50	
40 /40	0.973	99.754	1.69161	73.026	0.00025	0.00025	val_loss	248.33	

برای رسم نمودار خطأ و صحت نیز از کد زیر استفاده می‌کنیم.

```
tr_plot(history,0)
save_dir=r'./'
subject= grapes'
acc=model.evaluate( test_gen, batch_size=test batch_size, verbose=1, steps=test_steps, return_dict=False)[1]*100
msg=f'accuracy on the test set is {acc:.2f} %'
print_in_color(msg, (0,255,0),(55,65,80))
save_id=str(model_name + '-' + subject + '-' + str(acc)[:str(acc).rfind('.')+3] + '.h5')
save_loc=os.path.join(save_dir, save_id)
model.save(save_loc)
```

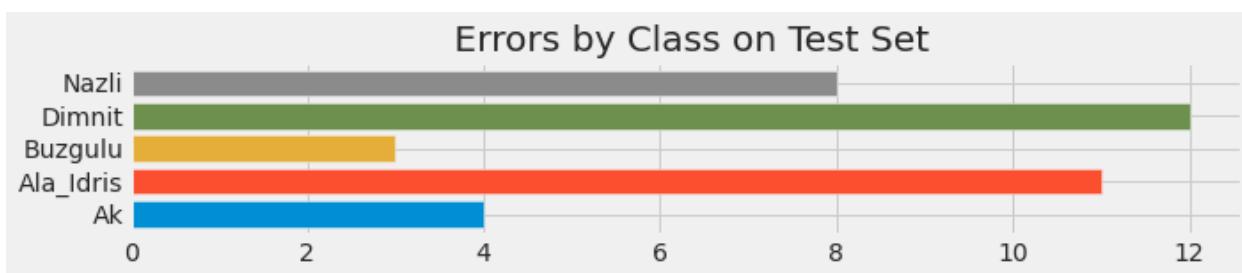


شکل ۳۰ نمودار خطأ و صحت آموزش و اعتبارسنجی

نتایج روی داده‌های `test` به صورت زیر شده است.

```
3/3 [=====] - 3s 1s/step - loss: 1.6404 - accuracy: 0.7516
accuracy on the test set is 75.16 %
```

```
print_code=0
preds=model.predict(test_gen)
print_info( test_gen, preds, print_code, save_dir, subject )
```



شکل 31 خطأ بر حسب هر کلاس

		Confusion Matrix				
		Ak	Ala_Idris	Buzgulu	Dimnit	Nazli
Actual	Ak	17	1	0	1	2
	Ala_Idris	4	16	2	4	1
	Buzgulu	0	1	25	0	2
	Dimnit	7	1	3	29	1
	Nazli	0	2	2	4	28
	Predicted	Ak	Ala_Idris	Buzgulu	Dimnit	Nazli

شکل ۳۲ confusion matrix برای رزنت

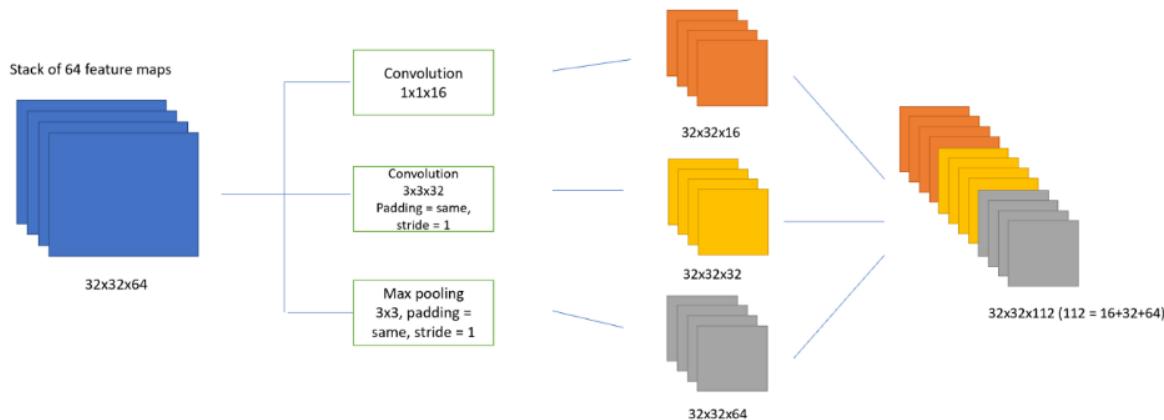
Classification Report:					
	precision	recall	f1-score	support	
Ak	0.61	0.81	0.69	21	
Ala_Idris	0.76	0.59	0.67	27	
Buzgulu	0.78	0.89	0.83	28	
Dimnit	0.76	0.71	0.73	41	
Nazli	0.82	0.78	0.80	36	
accuracy			0.75	153	
macro avg	0.75	0.76	0.75	153	
weighted avg	0.76	0.75	0.75	153	

## حالت ششم : پیاده‌سازی شبکه‌ی Inception v3

در این قسمت نیز هدف استفاده از شبکه‌ی از پیش آموزش دیده‌ی Inception v3 می‌باشد. در ابتدا توضیح مختصری در مورد شبکه Inception v3 می‌دهیم.

در شبکه‌های CNN فیلترهای کانولوشن برای استخراج ویژگی‌های تصویر استفاده می‌شوند. اندازه فیلترها یکی از عوامل تعیین کننده در ویژگی‌های استخراج شده از تصویر است. پیش از معرفی مازول‌های Inception طراحان شبکه باید با توجه به تصویر اندازه فیلترها را تعیین می‌کردند.

ایده مازول Inception استفاده از فیلترهای با ابعاد مختلف به طور همزمان است. به این صورت که چند فیلتر با ابعاد مختلف (فیلترهای کانولوشنی و pooling) روی ورودی اعمال می‌شوند. سپس خروجی آنها کنار یکدیگر قرار می‌گیرد (concat). در این حالت feature map‌ها مشتمل بر ویژگی‌های مختلف خواهند بود. هر مازول شامل چند عملیات کانولوشنی و pooling موازی است. شکل زیر به صورت بلوکی یک مازول دلخواه Inception را نشان می‌دهد.



شکل ۳۳ نمودار بلوکی یک مازول Inception دلخواه

در این شبکه‌ها محاسبات بسیار سنگین به صورت موازی انجام شده‌اند. طراحان مازول پس از این ضمن تلاش برای افزایش دقت، به اصلاحاتی برای کاهش هزینه محاسباتی نیز پرداختند.

Inception v3 یکی از آپدیت‌های شبکه‌ی Inception است. تغییرات زیر در ورژن سوم شبکه‌های Inception داده شد.

- استفاده از RMSProb
- فاکتوریزه کردن کرنل های  $7 \times 7$
- استفاده از BN auxiliary
- استفاده از تکنیک Label smoothing

**استفاده RMSProb:** یک الگوریتم بهینه سازی است که برای شبکه‌های عصبی طراحی شده این الگوریتم نرخ یادگیری را به صورت تطبیقی تغییر می‌دهد و موفقیت‌های خوبی در بسیاری از شبکه‌ها نشان داده در Inception نیز بهبود دقت نشان داد.

کرنل‌های  $7 \times 7$ : در این ورژن در مارژول‌ها به جای استفاده از کرنل‌های  $7 \times 7$  از کرنل‌های  $3 \times 3$  به صورت سری استفاده شد.

**شاخه BN auxiliary :** اگرچه در تحقیقات مربوط به  $V2$  نشان داده شد که شاخه auxiliary بر سرعت همگرایی تاثیر زیادی ندارد. اما دیده شد که استفاده از این شاخه زمانی که Batch normalized باشد، می‌تواند به عنوان Regularizer عمل کند.

**استفاده از Label smoothing:** اگر خروجی‌های نرمال نشده را  $Z$  بنامیم و خروجی مورد نظر را  $x$ ، آنگاه اینکه هر داده چقدر توسط شبکه احتمال داده شده که مربوط به دسته  $k$ ام باشد از رابطه  $p(k|x) = \exp(zk) / (1 + \sum \exp(zk))$ .

حال خروجی ایده آل این است که وقتی داده مربوط به دسته  $y$  است به آن احتمال  $1$  نسبت داده شده باشد و احتمال تعلق آن به دسته‌های دیگر  $\cdot$  یعنی  $q(k|x) = \delta[k-y]$  در حالت عادی شبکه را به گونه آموزش می‌دهیم که  $p$  مشابه  $q$  شود.

اما این می‌تواند منجر به بروز مشکلاتی شود. دقت کنید که زمانی  $p$  مشابه  $q$  خواهد شد که برای داده دسته  $y$  به سمت بی‌نهایت میل کند که با تعداد داده‌های آموزش محدود امکان پذیر نیست. به عبارت دیگر چون هیچ گاه  $p$  با  $q$  نمی‌تواند برابر شود، در حین آموزش همیشه مقداری خطا در شبکه محاسبه می‌شود و بر وزن‌ها تاثیر می‌گذارد این پدیده ممکن است باعث overfitting شود.

طراحان پیشنهاد دادند که به جای استفاده از تابع دلتا برای  $q$  از تابعی محافظه کارانه تر به صورت:

$$q(k|x) = (1 - \epsilon)\delta[k - y] + \frac{\epsilon}{N}$$

به این صورت احتمال تعلق به یک دسته کاملاً ۱ نبوده و با رسیدن به  $Z$  های محدود می‌توان خطا را صفر کرد.  
در  $V3$  از  $\epsilon=0.1$  استفاده شد.

Network	Top-1 Error	Top-5 Error	Cost Bn Ops
GoogLeNet [20]	29%	9.2%	1.5
BN-GoogLeNet	26.8%	-	<b>1.5</b>
BN-Inception [7]	25.2%	7.8	2.0
Inception-v2	23.4%	-	3.8
Inception-v2 RMSProp	23.1%	6.3	3.8
Inception-v2 Label Smoothing	22.8%	6.1	3.8
Inception-v2 Factorized $7 \times 7$	21.6%	5.8	4.8
Inception-v2 BN-auxiliary	<b>21.2%</b>	<b>5.6%</b>	4.8

در مقاله برای بررسی اثر بهبودهای داده شده در دقت جدول روبرو آورده شده است. شبکه‌ها با ILSVRC 2012 تست شده اند.

در ادامه در ابتدا دادگان در این شبکه split کردن دادگان به این صورت عمل کردم که از هر کلاس ۲۰ درصد در ابتدا برای test در نظر گرفته شد، در ادامه از ۸۰ درصد باقیماندهی کلاس نیز ۲۰ درصد برای validation در نظر گرفته شد. در ادامه مشابه قسمت قبل از generator استفاده کرده و augmentation نیز انجام دادم.

```
def train_val_generators(TRAINING_DIR, VALIDATION_DIR, TESTING_DIR):

    from tensorflow.keras.preprocessing.image import ImageDataGenerator

    train_datagen = ImageDataGenerator(rescale=1.0/255.,
                                        rotation_range=40,
                                        width_shift_range=0.2,
                                        height_shift_range=0.2,
                                        zoom_range=0.1,
                                        horizontal_flip=True,
                                        vertical_flip=True,
                                        fill_mode='nearest')

    # Pass in the appropriate arguments to the flow_from_directory method
    train_generator = train_datagen.flow_from_directory(directory=TRAINING_DIR,
                                                        batch_size=100,
                                                        class_mode='categorical',
                                                        target_size=(300, 300))
```

```

# Instantiate the ImageDataGenerator class (don't forget to set the rescale argument)
validation_datagen = ImageDataGenerator(rescale=1.0/255.)
# Pass in the appropriate arguments to the flow_from_directory method
validation_generator = validation_datagen.flow_from_directory(directory=VALIDATION_DIR,
                                                               batch_size=50,
                                                               class_mode='categorical',
                                                               target_size=(300, 300))

# Instantiate the ImageDataGenerator class (don't forget to set the rescale argument)
test_datagen = ImageDataGenerator(rescale=1.0/255.)
# Pass in the appropriate arguments to the flow_from_directory method
test_generator = test_datagen.flow_from_directory(directory=TESTING_DIR,
                                                               batch_size=50,
                                                               class_mode='categorical',
                                                               target_size=(300, 300))

### END CODE HERE
return train_generator, validation_generator, test_generator

```

حال کافیست برای آموزش شبکه inception module v3 بر روی دادگان خود، در خروجی چند لایه اضافه کنیم تا شبکه با ویژگی‌های گرفته شده از تصاویر ما آموزش ببیند. بنابراین در خروجی شبکه inception v3 نیز چند لایه به شکل زیر افزودم:

(لایه‌های آخر شبکه برداشته شده و برای fine tune شدن با دادگان ما، چند لایه افزوده شد.)

```

# The last 15 layers fine tune
for layer in InceptionV3_model.layers[:-15]:
    layer.trainable = False

x = InceptionV3_model.output
x = layers.GlobalAveragePooling2D()(x)
x = layers.Flatten()(x)
x = layers.Dense(units=512, activation='relu')(x)
x = layers.Dropout(0.3)(x)
x = layers.Dense(units=512, activation='relu')(x)
x = layers.Dropout(0.3)(x)
output = layers.Dense(units=5, activation='softmax')(x)
model = Model(InceptionV3_model.input, output)
model.summary()

```

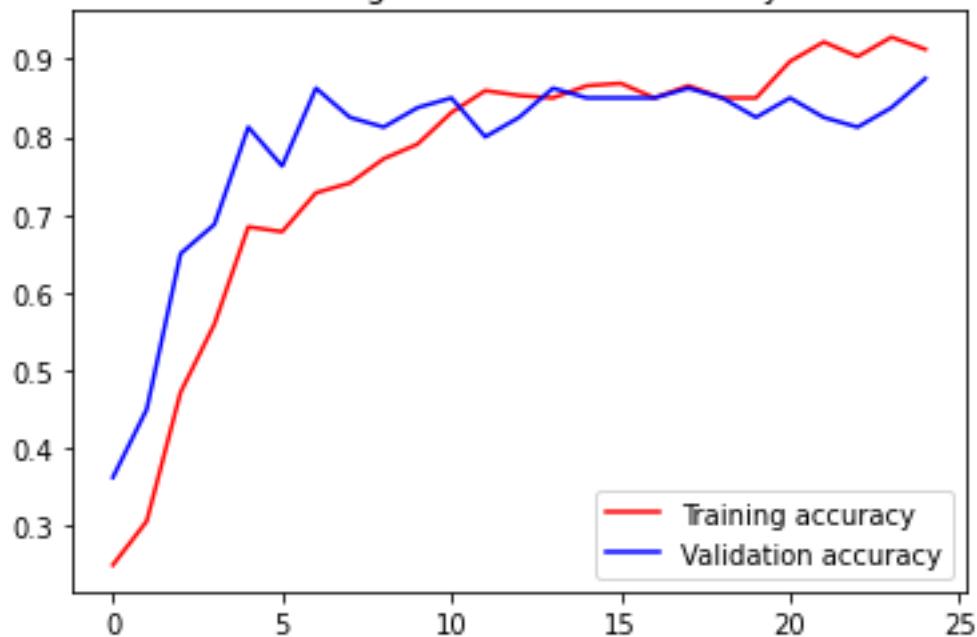
حال کافیست مدل خود را کامپایل کرده و درنهایت `fit` کنیم، نتایج نهایی به شکل زیر میشود:

```
Epoch 1/25
4/4 [=====] - 32s 3s/step - loss: 1.8275 - accuracy: 0.2500 - val_loss: 1.4692 - val_accuracy: 0.3625
Epoch 2/25
4/4 [=====] - 8s 2s/step - loss: 1.4661 - accuracy: 0.3063 - val_loss: 1.4079 - val_accuracy: 0.4500
Epoch 3/25
4/4 [=====] - 8s 2s/step - loss: 1.2486 - accuracy: 0.4719 - val_loss: 1.0484 - val_accuracy: 0.6500
Epoch 4/25
4/4 [=====] - 8s 2s/step - loss: 1.0441 - accuracy: 0.5594 - val_loss: 0.9819 - val_accuracy: 0.6875
Epoch 5/25
4/4 [=====] - 8s 3s/step - loss: 0.8603 - accuracy: 0.6844 - val_loss: 0.7409 - val_accuracy: 0.8125
Epoch 6/25
4/4 [=====] - 8s 2s/step - loss: 0.7901 - accuracy: 0.6781 - val_loss: 0.7268 - val_accuracy: 0.7625
Epoch 7/25
4/4 [=====] - 8s 2s/step - loss: 0.7138 - accuracy: 0.7281 - val_loss: 0.5899 - val_accuracy: 0.8625
Epoch 8/25
4/4 [=====] - 8s 2s/step - loss: 0.6477 - accuracy: 0.7406 - val_loss: 0.6058 - val_accuracy: 0.8250
Epoch 9/25
4/4 [=====] - 8s 3s/step - loss: 0.6055 - accuracy: 0.7719 - val_loss: 0.5846 - val_accuracy: 0.8125
Epoch 10/25
4/4 [=====] - 8s 3s/step - loss: 0.5036 - accuracy: 0.7906 - val_loss: 0.5093 - val_accuracy: 0.8375
Epoch 11/25
4/4 [=====] - 8s 2s/step - loss: 0.4646 - accuracy: 0.8313 - val_loss: 0.5011 - val_accuracy: 0.8500
Epoch 12/25
4/4 [=====] - 8s 2s/step - loss: 0.4015 - accuracy: 0.8594 - val_loss: 0.5252 - val_accuracy: 0.8000
Epoch 13/25
4/4 [=====] - 8s 2s/step - loss: 0.3879 - accuracy: 0.8531 - val_loss: 0.5129 - val_accuracy: 0.8250
Epoch 14/25
4/4 [=====] - 8s 2s/step - loss: 0.3725 - accuracy: 0.8500 - val_loss: 0.4610 - val_accuracy: 0.8625
Epoch 15/25
4/4 [=====] - 8s 3s/step - loss: 0.3812 - accuracy: 0.8656 - val_loss: 0.5047 - val_accuracy: 0.8500
Epoch 16/25
4/4 [=====] - 8s 2s/step - loss: 0.3579 - accuracy: 0.8687 - val_loss: 0.4545 - val_accuracy: 0.8500
Epoch 17/25
4/4 [=====] - 8s 2s/step - loss: 0.3912 - accuracy: 0.8500 - val_loss: 0.5344 - val_accuracy: 0.8500
Epoch 18/25
4/4 [=====] - 8s 2s/step - loss: 0.3827 - accuracy: 0.8656 - val_loss: 0.4525 - val_accuracy: 0.8625
Epoch 19/25
4/4 [=====] - 8s 2s/step - loss: 0.4320 - accuracy: 0.8500 - val_loss: 0.4729 - val_accuracy: 0.8500
Epoch 20/25
4/4 [=====] - 8s 3s/step - loss: 0.3439 - accuracy: 0.8500 - val_loss: 0.4604 - val_accuracy: 0.8250
Epoch 21/25
4/4 [=====] - 8s 2s/step - loss: 0.2815 - accuracy: 0.8969 - val_loss: 0.5697 - val_accuracy: 0.8500
Epoch 22/25
4/4 [=====] - 8s 2s/step - loss: 0.2673 - accuracy: 0.9219 - val_loss: 0.4743 - val_accuracy: 0.8250
Epoch 23/25
4/4 [=====] - 8s 2s/step - loss: 0.2319 - accuracy: 0.9031 - val_loss: 0.5829 - val_accuracy: 0.8125
Epoch 24/25
4/4 [=====] - 9s 2s/step - loss: 0.2363 - accuracy: 0.9281 - val_loss: 0.6009 - val_accuracy: 0.8375
Epoch 25/25
4/4 [=====] - 8s 3s/step - loss: 0.2744 - accuracy: 0.9125 - val_loss: 0.3647 - val_accuracy: 0.8750
```

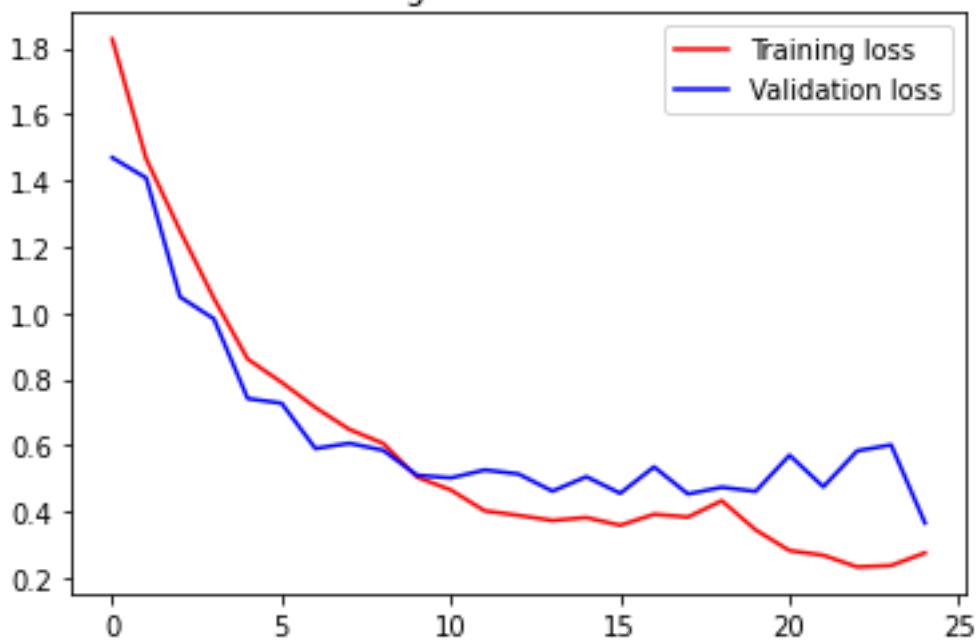
همانطور که در شکل بالا نیز به وضوح میبینیم، شبکه‌ی ما در یک روند بسیار مناسب و درست آموزش دیده است. بنابراین کافیست نتیجه‌ی آموزش شبکه بر روی دادگان `train` و `validation` را نیز در ادامه بررسی کنیم.

نمایش accuracy و loss حاصل برای دادگان train و validation

Training and validation accuracy



Training and validation loss



شکل ۳۴ از بالا به پایین به ترتیب نمایش accuracy و loss در دادگان برای مدل ۶

حال کافیست مشابه قبل به ارزیابی مدل خود بر روی دادگان test بپردازیم:

```
# evaluate accuracy on test data
test_loss, test_acc = model.evaluate(test_generator, verbose=2)
print('\nTest accuracy:', test_acc,'Test loss:',test_loss)

Test accuracy: 0.9009999952316284 Test loss: 1.8780020475387573

prediction = model.predict_generator(test_generator)
y_pred = prediction
y_pred_bool = np.argmax(y_pred, axis=1)
```

در ادامه مشابه قبل از مدل برای predict کردن دادگان test استفاده میکنیم:

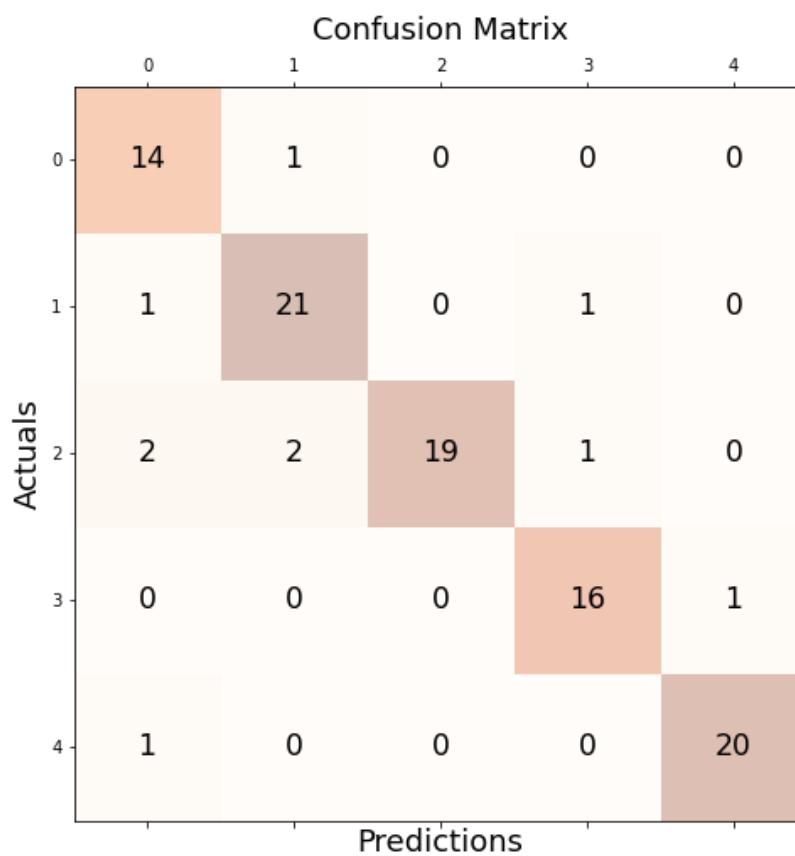
```
prediction = model.predict_generator(test_generator)
y_pred = prediction
y_pred_bool = np.argmax(y_pred, axis=1)
```

همانگونه که به وضوح میبینیم، نتیجه حاصل بر روی دادگان test نسبت به قبل بسیار بهبود یافت.

در ادامه میتوان confusion matrix را نیز برای این دادگان به دست آورد:

```
#from sklearn.metrics.confusion_matrix import confusion_matrix
import sklearn
conf_matrix = sklearn.metrics.confusion_matrix(y_pred=y_pred_bool , y_true=y_test_bool)
fig, ax = plt.subplots(figsize=(8, 8))
ax.matshow(conf_matrix, cmap=plt.cm.Oranges, alpha=0.3)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```



شکل ۳۵ نمایش confusion matrix برای مدل<sup>۶</sup>

	precision	recall	f1-score	support
0	0.78	0.93	0.85	15
1	0.88	0.91	0.89	23
2	1.00	0.79	0.88	24
3	0.89	0.94	0.91	17
4	0.95	0.95	0.95	21
accuracy			0.90	100
macro avg	0.90	0.91	0.90	100
weighted avg	0.91	0.90	0.90	100

شکل ۳۶ نمایش precision, recall, F1 score برای مدل<sup>۶</sup>

## ارزیابی نهایی نتایج :

نکته مهم اینکه نتایج ذکر شده به ازای seed های تصادفی مختلف حاصل شده و سپس میانگین این seedها به عنوان نتیجه نهایی بیان شده است.

Accuracy on train	Accuracy on test	مدل‌ها
90.62%	32.00%	CNN : ۱ مدل ۱
94.46%	36.00%	CNN + Data Augmentation : ۲ مدل ۲
90.31%	77.00%	CNN + data augmentation + : ۳ مدل ۳ pretrained VGG19
98.73%	81.04%	CNN + data augmentation + : ۴ مدل ۴ pretrained VGG19 + K Fold
99.836%	75.00%	CNN + data augmentation + : ۵ مدل ۵ pretrained Resnet
91.25%	90.09%	CNN + data augmentation + : ۶ مدل ۶ pretrained inception v3