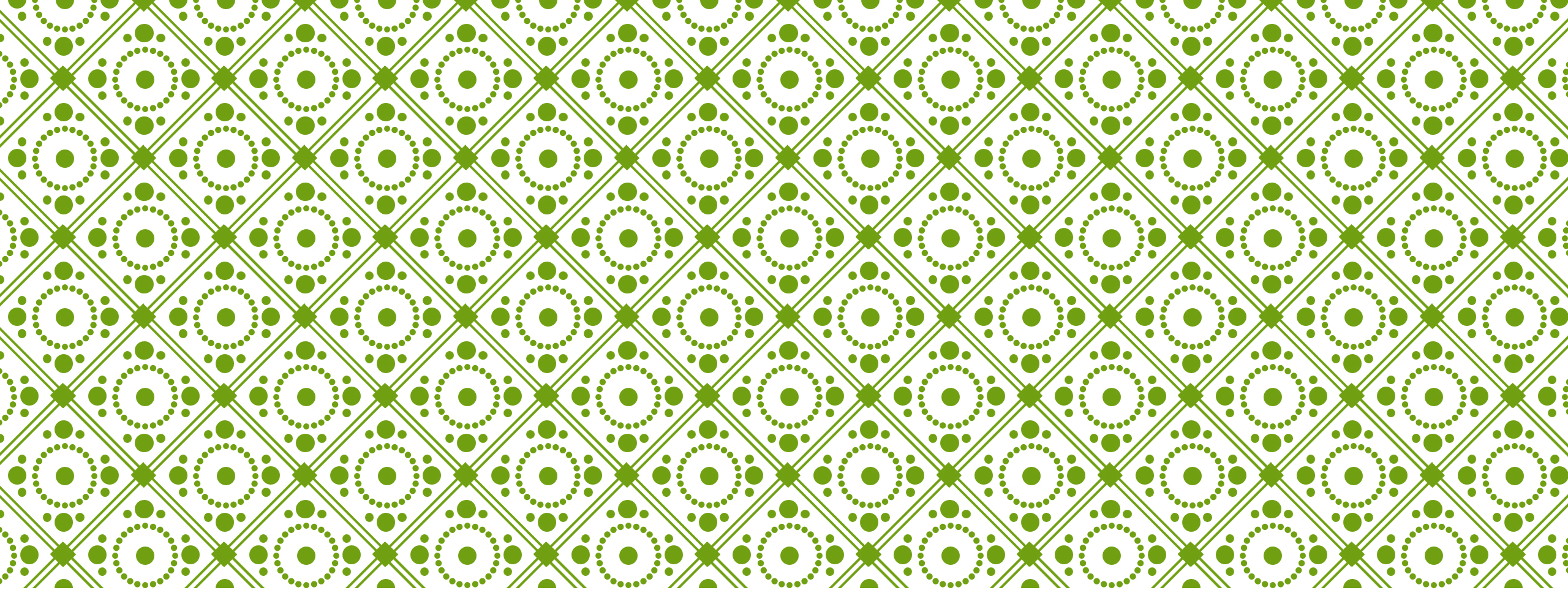


CASSAVA LEAF CLASSIFICATION
DEEP LEARNING

SHARIF UNIVERSITY OF TECHNOLOGY

Alireza Kazemi
95105035

مقدمه

در قسمت اول پروژه به سراغ کلاس بندی یکی از دیتاست های سایت ککگل می‌رویم که قرار است در این رقابت شرکت کنیم. چالش اصلی کسب بالاترین دقت بر روی داده تست است. در این پروژه با توجه به حجم زیاد دیتاست و عمیق بودن شبکه احتمالاً با مشکلات زیادی اعم از زمانی و حجمی رو به رو خواهیم شد. موضوع پروژه طبقه بندی تصاویری از برگ هاست که لیبل آن ها ۴ نوع بیماری و یا نوع سالم بودن است. من با استفاده از نوت بوک و ظرفیت پردازشی خود سایت ککگل، دیتاست را ادد کردم و به آماده سازی دیتا و مدل سازی بر روی آن پرداختم.

رویکرد

رویکرد اولیه من برای این پروژه، استفاده از ترنسفر لرنینگ است. در ترنسفر لرنینگ باید مدل مناسبی که از قبل یادگیری شده است را، وزن های آن را استخراج و سپس با یک یا چند لایه جدید قابل یادگیری مدل کلی را بسازیم. لازم به ذکر است که لایه های مدلی که ساخته ایم باید غیر قابل یادگیری باشند. مزیت این نوع رویکرد، کاستن از حجم محاسبات زیاد است اما از طرفی دقت مدل هم کاسته خواهد شد.

همچنین بر آن شدم تا مدل های از پیش یادگیری شده ماژول کراس را پیدا کنم، این ماژول چندین مدل را به صورت آماده و از پیش یادگیری شده دارد اما متاسفانه فقط وزن های یادگیری شده بر روی دیتاست **imagenet** را دارد. تعدادی از مدل هایی که در این ماژول آماده هستند را در صفحه بعدی آورده ام.

با توجه به سرچ هایی که کردم، ترنسفر لرنینگ هایی که مرتبط با دیتاست گیاهان کار شده است بیشتر از **inception** استفاده شده است. پس تلاش میکنم در این پروژه از این شبکه آماده استفاده کنم. همچنین معمولاً **Vgg19** هم ممکن است به درد بخورد.

Applications

Keras Applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.

Weights are downloaded automatically when instantiating a model. They are stored at `~/.keras/models/`.

The following image classification models (with weights trained on ImageNet) are available:

- Xception
- VGG16
- VGG19
- ResNet50
- InceptionV3
- InceptionResNetV2
- MobileNet
- MobileNetV2
- DenseNet
- NASNet

دیتاست

در این پروژه دیتاست ما حجم زیادی دارد و هر عکس جزئیات زیادی به همراه دارد. تصاویر ما سه کاناله ست و سایز آن تا ۵۰۰ در ۵۰۰ پیکسل نیز می باشد. اما من سعی کردم برای کاهش محاسبات که البته تریدآف کاهش دقت را به همراه دارد، سایز عکس را کمی دستکاری کنم و بتوانم با بازی کردن با آن نتایج زودتری به دست آوردم. اگر سایز تصویر را ۱۰۰ در ۱۰۰ انتخاب کنم جزئیات کاملاً از بین می‌رفت، بنابراین در کمترین حالت ۱۵۰ در ۱۵۰ پیکسل قرار دادم که البته باز هم دقت خوبی نداشت و بهترین حالت ۲۰۰ تا ۲۵۰ از نظر تایم و دقت برایم بود.

شماره کلاس ها به صورت زیر تعریف میشود:

```
"0": "Cassava Bacterial Blight (CBB)",
"1": "Cassava Brown Streak Disease (CBSD)",
"2": "Cassava Green Mottle (CGM)",
"3": "Cassava Mosaic Disease (CMD)",
"4": "Healthy"
```

```
Found 18188 validated image filenames belonging to 5 classes.
Found 4279 validated image filenames belonging to 5 classes.
```

دیتاست (ادامه)

توابع **train generator** و **valid generator** را برای استخراج از دیتاست استفاده کرده ام، به صورتی که با تابع **next** میتوان یک **batch** از سمپل ها را به صورت شافل انتخاب کرد. پس خروجی تابع **next** به صورت زیر میشود: **(Batch Size, Image Size, Image Size, Channels)**

در صفحه ی بعد تعدادی نمونه از کلاس های مختلف این دیتاست به تصویر کشیده ام.


Cassava Brown Streak Disease (CBSD)


Cassava Mosaic Disease (CMD)


Cassava Mosaic Disease (CMD)


Cassava Green Mottle (CGM)


Healthy


Cassava Mosaic Disease (CMD)


Cassava Mosaic Disease (CMD)


Cassava Brown Streak Disease (CBSD)


Healthy


Healthy


Cassava Bacterial Blight (CBB)


Cassava Mosaic Disease (CMD)


Cassava Bacterial Blight (CBB)


Cassava Mosaic Disease (CMD)


Cassava Mosaic Disease (CMD)


Cassava Green Mottle (CGM)


Cassava Mosaic Disease (CMD)


Healthy


Cassava Green Mottle (CGM)


Cassava Mosaic Disease (CMD)


Healthy


Cassava Mosaic Disease (CMD)


Cassava Mosaic Disease (CMD)


Cassava Green Mottle (CGM)


Cassava Mosaic Disease (CMD)

پیاده سازی

حال که دیتا آماده ورود به مدل می باشد، ابتدا باید مدل **inception** را ایمپورت کنیم. در اینجا از حالت وزن های آماده ی دیتاست **imagenet** استفاده میکنیم. سپس باید هر لایه این مدل را غیرقابل یادگیری کنیم. خروجی این مدل که در کد **base model** نامیده ام را پس از عبور از یک لایه **pooling** وارد یک یا چند لایه مخفی که قابل یادگیری هستند میکنم. همچنین بهینه ساز را از نوع **Adam** با نرخ یادگیری روتینی که همواره طی تمرین ها استفاده میکردم، میگذارم. و در انتها وارد لایه خروجی ۵ نورونه که تابع فعالیت ان سافتمکس می باشد، میکنم.

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.models import Model
from tensorflow.keras.applications import VGG19

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

base_model = InceptionV3(weights='imagenet', include_top=False)
#base_model = VGG19(weights='imagenet', include_top=False)
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(5, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
for layer in base_model.layers:
    layer.trainable = False
```

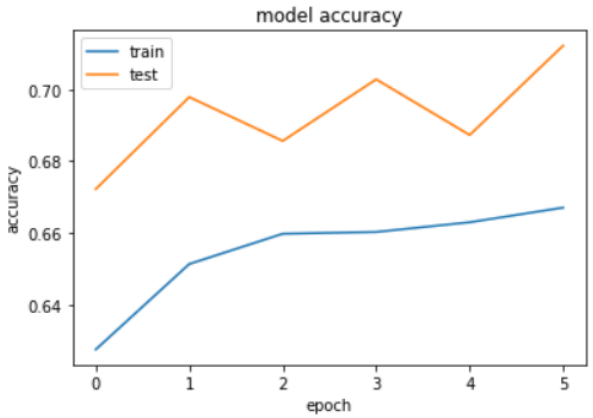
رگولاریزیشن

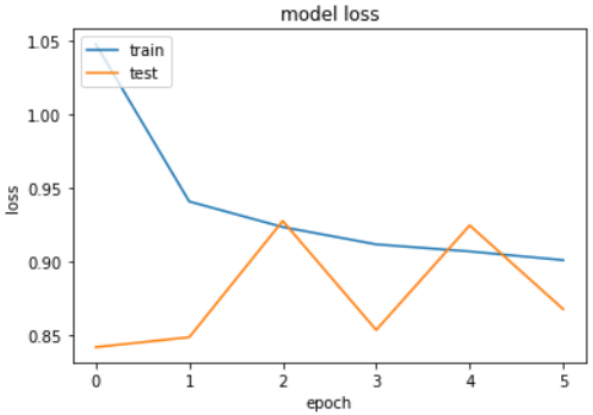
در این مدل از تکنیک **early stopping** استفاده کرده ام، به این معنی که در صورتی که دقت مدل بر روی داده ولیدیشن از یک ایپاک به بعد کمتر و کمتر شود مدل استاپ میشود و دیگر یادگیری انجام نمیدهد. یعنی ما بر روی نمودار ترید آف خطای واقعی و تجربی یک جایی در آن وسط می ایستیم. همچنین باز هم در لایه ماقبل آخر مدل از دراپ آوت استفاده کرده ام. دراپ آوت در لایه های قبلی منظور شده است. تعداد ایپاک معقول حدوداً ۱۰ تا ۲۰ در نظرم بود. با توجه به اینکه هنوز مشکل بزرگ به نظرم خود یادگیری بر روی داده های آموزش است، بیش از این به سراغ تکنیک های رگولاریزیشن نرفتم.

خروجی

```
Epoch 1/15
568/568 [==============================] - 805s 1s/step - loss: 1.2206 - acc: 0.5964 - val_loss: 0.8414 - val_acc: 0.6722

Epoch 00001: val_loss improved from inf to 0.84140, saving model to Model
Epoch 2/15
568/568 [==============================] - 743s 1s/step - loss: 0.9363 - acc: 0.6489 - val_loss: 0.8481 - val_acc: 0.6978

Epoch 00002: val_loss did not improve from 0.84140
Epoch 3/15
568/568 [==============================] - 739s 1s/step - loss: 0.9209 - acc: 0.6652 - val_loss: 0.9273 - val_acc: 0.6856

Epoch 00003: val_loss did not improve from 0.84140
Epoch 4/15
568/568 [==============================] - 736s 1s/step - loss: 0.9178 - acc: 0.6561 - val_loss: 0.8531 - val_acc: 0.7028

Epoch 00004: val_loss did not improve from 0.84140
Epoch 5/15
568/568 [==============================] - 741s 1s/step - loss: 0.9081 - acc: 0.6638 - val_loss: 0.9243 - val_acc: 0.6873

Epoch 00005: val_loss did not improve from 0.84140
Epoch 6/15
568/568 [==============================] - 741s 1s/step - loss: 0.9050 - acc: 0.6649 - val_loss: 0.8673 - val_acc: 0.7122

Epoch 00006: val_loss did not improve from 0.84140
Restoring model weights from the end of the best epoch.
Epoch 00006: early stopping
```

Submission and Description	Status	Public Score	Use for Final Score
notebook371e1e315c version95105035 (version 1/1) an hour ago by Alireza Kazemi From Notebook [notebook371e1e315c]	Succeeded	0.672	☐





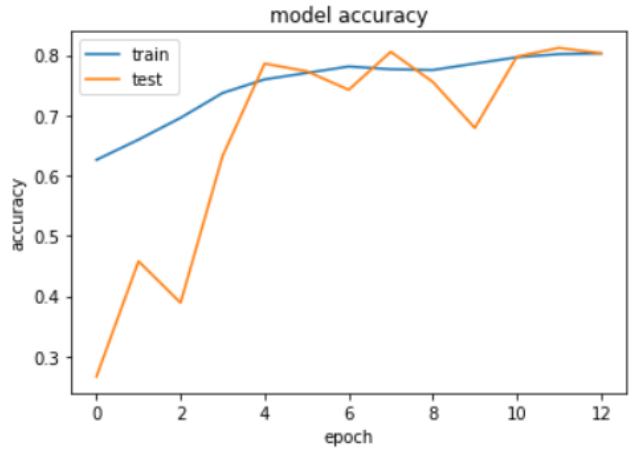
چالش

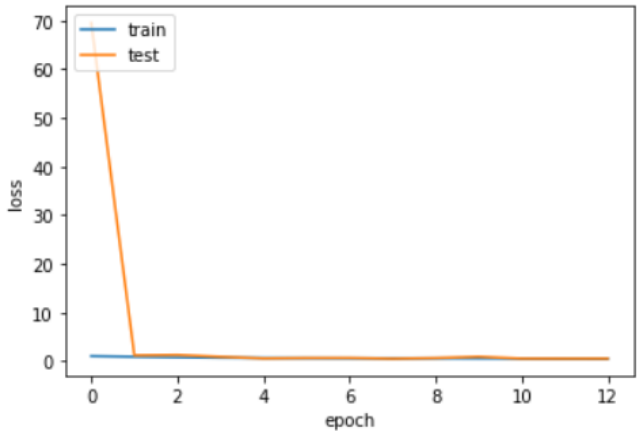
برای گرفتن دقت بر روی داده های تست، ابتدا مدل را مورد یادگیری قرار دادیم سپس آن را ذخیره کردیم و در یک نوتبوک جدید هم دیتای مسابقه و هم مدل را فراخوانی کردیم و بر روی داده های تست مدل را اجرایی کردم. در انتها فایل پیش بینی ها را ذخیره کردم و سابمیت کردم. (**Alireza Kazemi**)

لازم به ذکر است که ابتدا پیش از یادگیری کامل مدل، سایز عکس را ۳۵۰ قرار دادم. همچنین یک لایه پنهان دیگر علاوه بر مدل قبلی با ۲۰۴۸ نورون اضافه کردم. با توجه به خروجی قسمت قبل که به خروجی ۶۷/۲ درصد بر روی داده های تست رسیدم، که عددی نزدیک به دقت بر روی دیتای ولیدیشن بود. این مدل مبتنی بر این بود که اکثر لایه های آن، از وزن های فریز شده استفاده می کرد و فقط دو لایه یادگیری میکرد که با توجه به این به نظرم بسیار خوب عمل کرد، اما در حالت کلی شاید این خروجی خیلی مناسب نباشد، بنابراین تصمیم گرفتم که مدل جدید را بدون استفاده از وزن های فریز شده مورد یادگیری قرار دهم. برای این کار باید قابلیت یادگیری بر روی لایه های بیس مادل را **false** نکرد.

خروجی

```
plt.show()
568/568 [==============================] - 637s 1s/step - loss: 0.5969 - acc: 0.7867 - val_loss: 0.9493 - val_acc: 0.6790

Epoch 00010: val_loss did not improve from 0.54910
Epoch 11/15
568/568 [==============================] - 632s 1s/step - loss: 0.5735 - acc: 0.8009 - val_loss: 0.5721 - val_acc: 0.7972

Epoch 00011: val_loss did not improve from 0.54910
Epoch 12/15
568/568 [==============================] - 627s 1s/step - loss: 0.5799 - acc: 0.7972 - val_loss: 0.5609 - val_acc: 0.8118

Epoch 00012: val_loss did not improve from 0.54910
Epoch 13/15
568/568 [==============================] - 630s 1s/step - loss: 0.5595 - acc: 0.8019 - val_loss: 0.5639 - val_acc: 0.8029

Epoch 00013: val_loss did not improve from 0.54910
Restoring model weights from the end of the best epoch.
Epoch 00013: early stopping
```





توضیحات

همانطور که در صفحه قبلی دیدیم، توانستم به دقت حدود ۸۰ درصد بر روی داده ی ولیدیشن برسم که خب به نظر منطقی می آید. چون در اینجا دیگر وزن ها فریز نیستند. و هم چنین همانطور که دیدیم یادگیری پس از حدود ۸ ایپاک به سرانجام رسید که در مدل قبلی پس از ۱ ایپاک یادگیری به سرانجام رسید. در مورد خروجی دقت بر روی داده ی تست هنوز منتظر نتیجه هستم. اما به هر حال با نام **Alireza Kazemi** احتمالاً در لیست خواهم بود با درصد نزدیک به ۸۰.

راهکار

ایده مناسبی که برای این پروژه به ذهنم می‌رسد این است که از ترکیب دو روش قبلی استفاده کنم تا به نتیجه بهتری برسم. به این صورت که در ایپاک های اول از وزن های فریز شده ایمیجنت استفاده کنم تا دو لایه آخر **dense** به خوبی مورد یادگیری واقع شوند و سپس بعد از چند ایپاک، لایه های اولیه نیز قابل یادگیری شوند تا قابلیت استخراج ویژگی بهتر و اختصاص یافته تری نسبت به دیتاست برگ گیاهان داشته باشد. برای این کار لازم است که مدل را یک بار با وزن های فریز شده برای چند ایپاک ترین کنیم سپس دوباره این مدل را لود کنیم و همه لایه ها را قابل یادگیری کنیم و سپس یادگیری را برای ایپاک های باقی مانده انجام بدهیم. اما متاسفانه وقت کافی برای اجرای این مدل برایم نمانده است ...