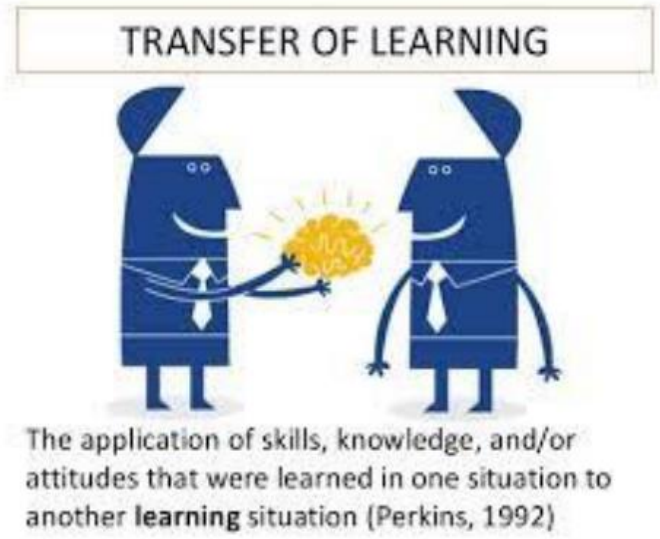
نقل التعلم :

ما هو نقل التعلم : كمثال تشبيه الطالب و المدرس حيث يتمتع المعلم بسنوات من الخبرة في موضوع معين يقوم بتدريسه , مع كل هذه المعلومات المتراكمة , فان المحاضرات التي يتلقاها الطلاب في نظرة عامة و موجزة عن الموضوع .

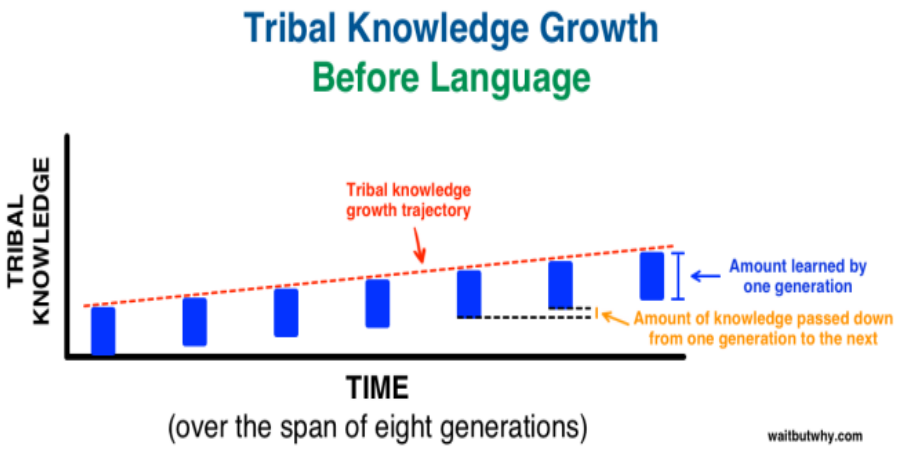
لذلك يمكن اعتباره بمثابة" نقل Transfer " للمعلومات من المتعلم الى المبتدئ .



مع الأخذ في الاعتبار هذا التشبيه , نقارن هذا بالشبكة العصبية ,يتم تدريب الشبكة العصبية على البيانات , تكتسب هذه الشبكة المعرفة من هذه البيانات , و التي يتم تجميعها على أنها "أوزان Weights " للشبكة , يمكن استخراج هذه الأوزان ثم نقلها الى أي شبكة عصبية أخرى . بدلا من تدريب الشبكة العصبية الأخرى من البداية نقوم "بنقل Transfer " الميزات المكتسبة.

حيث نفكر في أهمية نقل التعلم من خلال الارتباط بتطورنا , و ما هي أفضل طريقة من استخدام نقل التعلم ؟ لشرح ذلك بطريقة مبسطة هي طريقة اختراع اللغة في البشر .

قبل اختراع اللغة , كان على كل جيل من البشر إعادة ابتكار المعرفة لأنفسهم و هكذا كان نمو المعرفة يحدث من جيل الى أخر:



ثم اخترعت اللغة ! طريقة لنقل التعلم من جيل الى أخر و هذا ما حدث خلال نفس الاطار الزمني :

فان نقل التعلم عن طريق تمرير الأوزان يعادل اللغة المستخدمة لنشر المعرفة عبر الأجيال في التطور البشري .

خطوات أو نهج النموذج المدرب مسبقا :

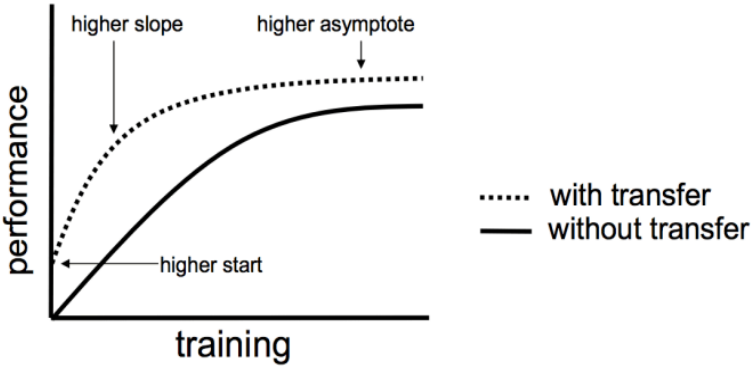
1- نحدد النموذج الأساسي select source Model : يتم اختيار نموذج مصدر مدرب مسبقا من النماذج المتاحة (في المشروع استخدمنا نموذج VGG16 المدرب على بيانات ImageNet عبارة عن 1000 فئة Class) , تطلق العديد من المؤسسات البحثية نماذج على مجموعات بيانات كبيرة و صعبة يمكن تضمينها في مجموعة النماذج المرشحة لاختيار من بينها .
2- إعادة استخدام النموذج Reuse Model : يمكن بعد ذلك استخدام النموذج الذي تم تدريبه مسبقا كنقطة انطلاق لنموذج في المهمة الثانية كموضع اهتمام , قد يشمل ذلك استخدام النموذج بالكامل أو أجزاء منه , اعتمادا على تقنية النمذجة المستخدمة (أي الغاء تفعيل طبقات معينة في الشبكة)
3- ضبط النموذج Tune Model : اختياريا , قد يحتاج النموذج الى تكييفه أو ضبطه على بيانات الادخال و الإخراج المتاحة للمهمة محل الاهتمام (مثلا ضبط Learning rate - batch size - epoch – إضافة Layer جديدة الى النموذج الى أخره .....)

متى نستخدم نقل التعلم ؟

نقل التعلم هو تحسين Optimization اي اختصار لتوفير الوقت أو الحصول على أداء أفضل .

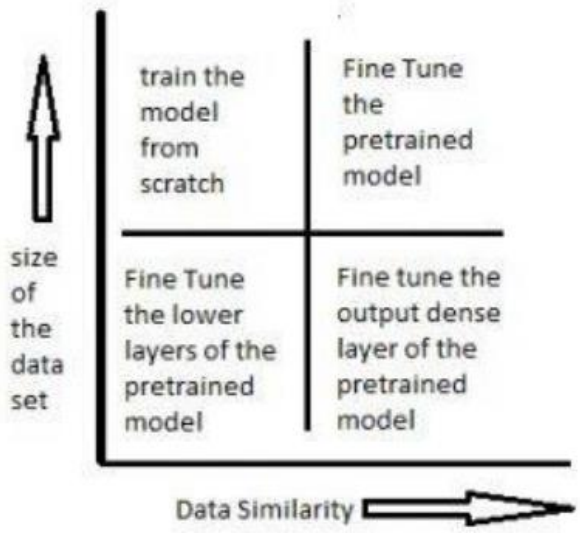
عند تدريب الشبكة العصبونية المبنية على نقل التعلم هنالك ثلاث فوائد محتملة جيب البحث عنها عند استخدام نقل التعلم :

1- بداية أعلى Higher start : أي الأداء الأولي (قبل ضبط النموذج) في نموذج الأصلي Source model أعلى مما ستكون عليه .
2- منحدر أعلى Higher slope : معدل تحسين الأداء أثناء تدريب نموذج المصدر أكثر حدة من النموذج الأصلي .
3- خط المقارب أعلى Higher asymptote : الأداء المتقارب للنموذج المدرب أفضل مما ستكون عليه في غير ذلك .



نهج النموذج المدرب مسبقا كيفية ضبط النموذج :

1- استخراج الميزات Feature extraction : يمكننا استخدام نموذج مدرب مسبقا كآلية لاستخراج الميزات , ما يمكننا القيام به هو أنه يمكننا إزالة طبقة الإخراج (الطبقة التي تعطي احتمالات التواجد في كل فئة من الفئات 1000 Softmax) ثم استخدام الشبكة بالكامل كمستخرج ميزة ثابتة لمجموعة البيانات الجديدة .
2- استخدام بنية النموذج المدرب مسبقا : ما يمكننا فعله هو استخدام بنية النموذج بينما نقوم بتهيئة جميع الأوزان بشكل عشوائي و تدريب النموذج وفقا لمجموعة البيانات الخاصة بنا مرة أخرى .
3- تدريب بعض الطبقات بينما نقوم بتجميد البعض الأخر : هنالك طريقة أخرى لاستخدام نموذج تم تدريبه مسبقا و هي التدريب جزئيا , ما يمكننا القيام به هو الاحتفاظ بأوزان الطبقات الأولية للنموذج مجمدة بينما نقوم بإعادة تدريب الطبقات العليا فقط , يمكننا تجربة و اختبار عدد الطبقات التي سيتم تجميدها و عدد الطبقات التي سيتم تدريبها .



السيناريو 1 :

حجم مجموعة البيانات صغير بينما تشابه البيانات مرتفع جدا – في هذه الحالة نظرا لأن تشابه البيانات مرتفع جدا , لا تحتاج الى إعادة تدريب النموذج , كل ما نحتاج الى القيام به هو تخصيص و تعديل طبقات الإخراج وفقا لبيان المشكلة , نحن نستخدم النموذج الجاهز كمستخرج للميزات , لنفترض أننا قررنا استخدام نماذج مدربة على Imagenet لتحديد ما اذا كانت مجموعة الصور الجديدة بها قطط أم كلاب , هنا الصور التي نحتاج الى تحديدها ستكون مشابهة ل imagenet لكننا نحتاج فقط الى فئتين كمخرجات في هذه الحالة كل ما نقوم به هو تعديل الطبقات الكثيفة و طبقة softmax النهائية لاخراج فئتين بدلا من 1000

السيناريو 2 :

حجم البيانات صغير و كذلك تشابه البيانات منخفض جدا – في هذه الحالة يمكننا تجميد الطبقات الأولية دعنا نقول K للنموذج الذي تم اختباره مسبقا و تدريب الطبقات المتبقية n-k مرة أخرى , سيتم بعد ذلك تخصيص الطبقات العليا لمجموعة البيانات الجديدة .

نظرا لأن مجموعة البيانات الجديدة ذات تشابه منخفض , فمن المهم إعادة تدريب الطبقات العليا و تخصيصها وفقا لمجموعة البيانات الجديدة , يتم تعويض الحجم الصغير لمجموعة البيانات من خلال حقيقة أن الطبقات الأولية يتم الاحتفاظ بها مسبقا (والتي تم تدريبها على مجموعة بيانات كبيرة مسبقا)و يتم تجميد أوزان تلك الطبقات .

السيناريو 3 :

حجم مجموعة البيانات كبير و لكن تشابه البيانات منخفض جدا – في هذه الحالة نظرا لأن لدينا مجموعة بيانات كبيرة , سيكون تدريب الشبكة العصبية لدينا فعالا , و مع ذلك , نظرا لأن البيانات التي لدينا مختلفة جدا مقارنة بالبيانات المستخدمة لتدريب نماذجنا التي تم اختبارها مسبقا , لن تكون التنبؤات التي تم اجراؤها باستخدام النماذج التي تم اختبارها مسبقا فعالة و بالتالي من الأفضل تدريب الشبكة العصبية من البداية وفقا لبياناتك .

السيناريو 4 :

حجم البيانات كبير و كذلك يوجد تشابه كبير في البيانات – هذا هو الوضع المثالي في هذه الحالة , يجب أن يكون النموذج الذي تم اختباره مسبقا هو الأكثر فعالية , أفضل طريقة لاستخدام النموذج هي الاحتفاظ ببنية النموذج و الأوزان الأولية للنموذج , ثم يمكننا إعادة تدريب هذا النموذج باستخدام الأوزان كما تم تهيئتها في النموذج المدرب مسبقا .

شبكة VGG16 :

Visual Geometry Group هي احدى أنواع الشبكات العصبونية Standard و اختصارا تسمى VGG و رقم 16 لأنها تحتوي على 16 طبقة (طبقة pooling لا تحتسب لأنها لا تحوي على بارامترات لتدريب)

1- تتكون الطبقتان التلافيفة الأولى والثانية من 64 مرشح و بحجم المرشح 3 × 3.

كصورة إدخال (صورة RGB بعمق 3) تم تمريرها إلى الطبقة التلافيفية Conv الأولى والثانية ، تتغير الأبعاد إلى 224 × 224 × 64.

ثم يتم تمرير الخرج الى طبقة التجميع max pooling لتقليل الأبعاد الى النصف أي 2

2- الطبقتان التلافيفيتان Conv الثالثة والرابعة من 124 مرشح و بحجم المرشح 3 × 3.

تتبع هاتان الطبقتان طبقة تجميع max pooling بخطوة أي تقليل الأبعاد بمقدار 2 وسيتم تقليل الناتج أي خريطة المميزات feature map إلى 56x56x128 .

3- الطبقات الخامسة والسادسة والسابعة هي طبقات تلافيفية بحجم نواة 3 × 3.

الثلاثة جميعها تستخدم 256 خريطة مميزات feature map.

هذه الطبقات متبوعة بطبقة تجمع max pooling بخطوة 2.

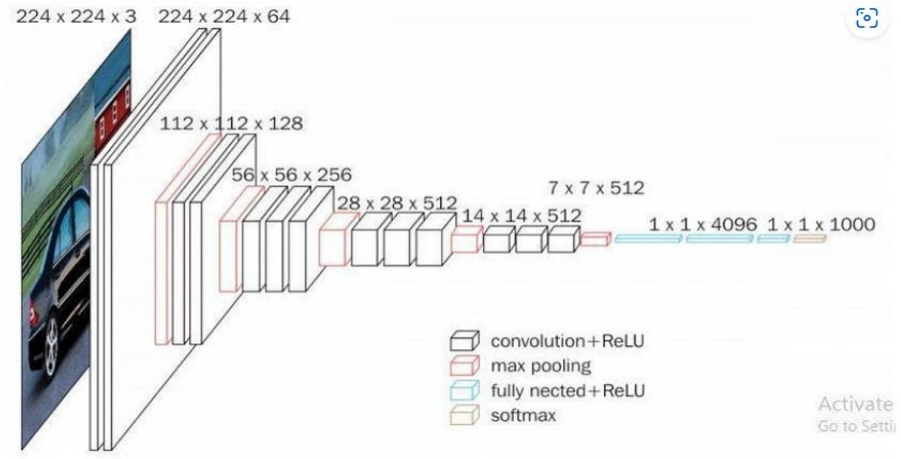
4- من الثامنة إلى الثالثة عشر مجموعتان من الطبقات التلافيفية Conv بحجم نواة 3 × 3.

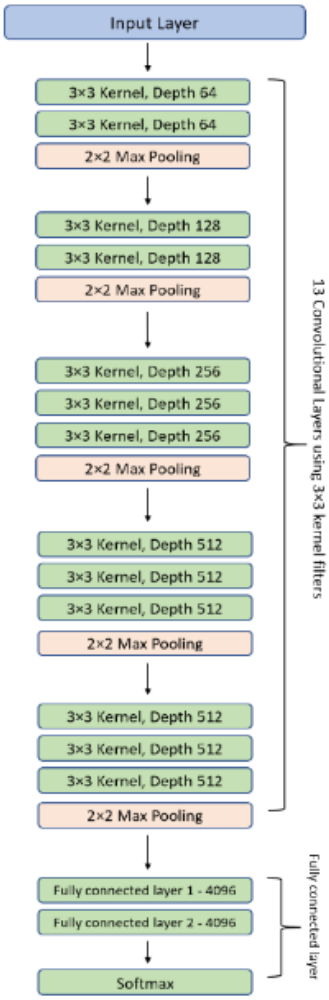
كل هذه المجموعات من الطبقات التلافيفية لها 512 مرشح نواة.

هذه الطبقات متبوعة بطبقة تجميع max pooling بخطوة 2.

5- أربعة عشر وخمسة عشر طبقة متصلة بالكامل FC بطبقات مخفية من 4096 وحدة

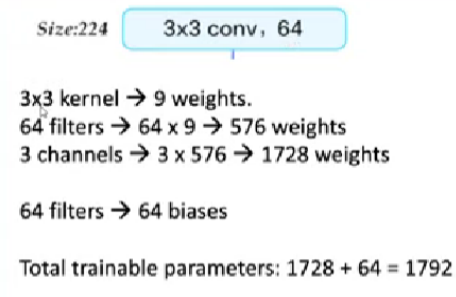
تليها طبقة إخراج softmax (الطبقة السادسة عشرة) من 1000 وحدة.



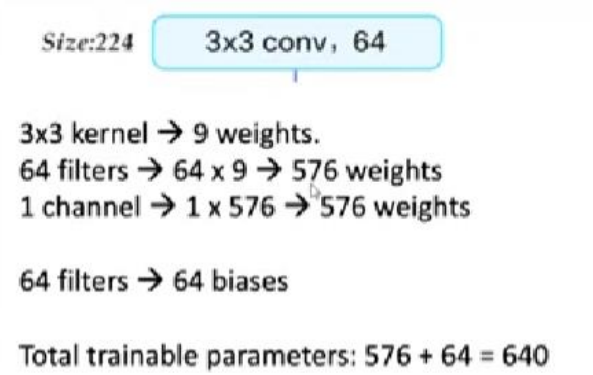


حساب بارامترات كل طبقة :

- في الطبقة الأولى لدينا Conv هو عبارة عن Kernel بأبعاد 3X3 و عدد Kernel 64 و عدد بارامترات الطبقة يحسب بهذه الطريقة لصورة ذو بعد RGB 3 و Padding same أي لا يتغير أبعاد خلال تطبيق kernel



أما اذا كانت الصورة عبارة عن Gray scale Img و حيث تتغير البارامترات القابلة لتدريب فقط لأول طبقة



- و لحساب بارامترات الطبقة الثانية نقوم بضرب أبعاد Kernal السابق 3 x3 بعدد Kernal السابق و هو 64 و نجمع معه 1+ ضرب عدد Kernal في الطبقة الثانية 36,928 = 64*(1+64*3*3)

layer is: ((shape of width of filter * shape of height filter * number of filters in the previous layer+1) * number of filters)

- طبقة Pool مهمتها تقليل أبعاد map لذلك لا يوجد لديها بارامترات

- لحساب FC طبقة الاتصال الكاملة نقوم بضرب عدد النيرون في FC بأبعاد الطبقة السابقة مثلا flatten +1*عدد النيرون في FC فنحصل على 102,764,544

layer is((current layer c*previous layer p)+1*c)

- لحساب خرج طبقة Softmax نضرب عدد الخرج من Softmax ب الطبقة السابقة FC + 1 ضرب بخرج الطبقة الحالية ل softmax لل VGG16 مدربة على ImageNET لديها 1000 classes

((current layer c*previous layer p)+1*c)

تدريب شبكة VGG16 :

1- استدعاء المكتبات الازمة مع Vgg16

```
#%% Libraries import
from keras.preprocessing.image import ImageDataGenerator #preprocessing for image
from tensorflow.keras.utils import img_to_array , load_img
from keras.models import Sequential #Creating a model
from keras.layers import Dense # for model
from keras.applications.vgg16 import VGG16 #for transfer learning
import matplotlib.pyplot as plt ##visualization
from glob import glob #f o r image import
```

2- تنزلي البيانات fruits 360 dataset من Kaggle من خلال رابط API

```
[ ]  #Set the enviroment variables
     import os
     os.environ['KAGGLE_USERNAME'] = "aubaialkhabbaz"
     os.environ['KAGGLE_KEY'] = "2715ab36f8842a62a77fc816d7f14851"
     !kaggle datasets download -d moltean/fruits

     Downloading fruits.zip to /content
     100% 1.28G/1.28G [00:40<00:00, 31.7MB/s]
     100% 1.28G/1.28G [00:41<00:00, 33.6MB/s]
```

3- فك ضغط الملف الذي تم تنزيله

```
[ ] ! unzip /content/fruits.zip
```

```
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/284_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/286_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/287_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/288_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/289_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/305_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/306_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/34_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/36_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/37_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/38_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/39_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/40_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/41_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/42_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/43_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/44_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/45_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/46_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/47_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/48_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/49_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/50_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/51_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/52_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/53_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/54_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/55_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/56_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/57_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/58_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/59_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/60_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/61_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/62_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/63_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/64_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/65_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/66_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/67_100.jpg
inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/68_100.jpg
```

4- نحذف بقية مجلدات الفئات داخل كل من Val و training و نبقي على فئة التفاح عبارة عن 12 فئة من خلال الكود التالي حيث نحدد المجلدات التي نريد حذفها

```
[ ] !rm -rf '/content/fruits-360-original-size/fruits-360-original-size/Validation/apple_6'
```

5- انشاء مسار لكل من بيانات التدريب و بيانات التحقق و ثم عرض احدى الصور من ضمن dataset حيث أبعاد الصورة هي 100x100x3 ملونة و نوع الأخر من البيانات ضمن dataset يختلف في أبعاد الصور مثلا 480X420

```
train_path = "/content/fruits-360-original-size/fruits-360-original-size/Training"
test_path = "/content/fruits-360-original-size/fruits-360-original-size/Validation"

img = load_img('/content/fruits-360-original-size/fruits-360-original-size/Validation/apple_braeburn_1/r0_1.jpg')
plt.imshow(img)
plt.axis("off")
plt.show()

x = img_to_array(img)
print(x.shape)
numberOfClass = len(glob('/content/fruits-360-original-size/fruits-360-original-size/Training'+'/*')) #We go into the folder and read the #CLass number. * by putting them we make them read them all. Thus, the n
```

6- بيانات التدريب و فهرستها

```
labels = {value: key for key, value in train_generator.class_indices.items()}

print("Label Mappings for classes present in the training and validation datasets\n")
for key, value in labels.items():
    print(f"{key} : {value}")
```

```
Label Mappings for classes present in the training and validation datasets

0 : apple_braeburn_1
1 : apple_crimson_snow_1
2 : apple_golden_1
3 : apple_golden_2
4 : apple_golden_3
5 : apple_granny_smith_1
6 : apple_pink_lady_1
7 : apple_red_1
8 : apple_red_2
9 : apple_red_3
10 : apple_red_delicios_1
11 : apple_red_yellow_1
```

7- تنزيل موديول Vgg16 المدرب مسبقا على بيانات ImageNet

```
vgg = VGG16()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467096/553467096 [==============================] - 16s 0us/step
```

8- عرض محتويات الموديول Vgg16

```
[ ] print(vgg.summary())
    print(type(vgg))

    vgg_layer_list = vgg.layers
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 fc1 (Dense)                 (None, 4096)              102764544

 fc2 (Dense)                 (None, 4096)              16781312

 predictions (Dense)         (None, 1000)              4097000

=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
```

9- حيث نقوم بحذف أخر طبقة من الموديول Vgg16

```
[ ] model = Sequential()
    for i in range(len(vgg_layer_list)-1):
        model.add(vgg_layer_list[i])
    print(model.summary())
```

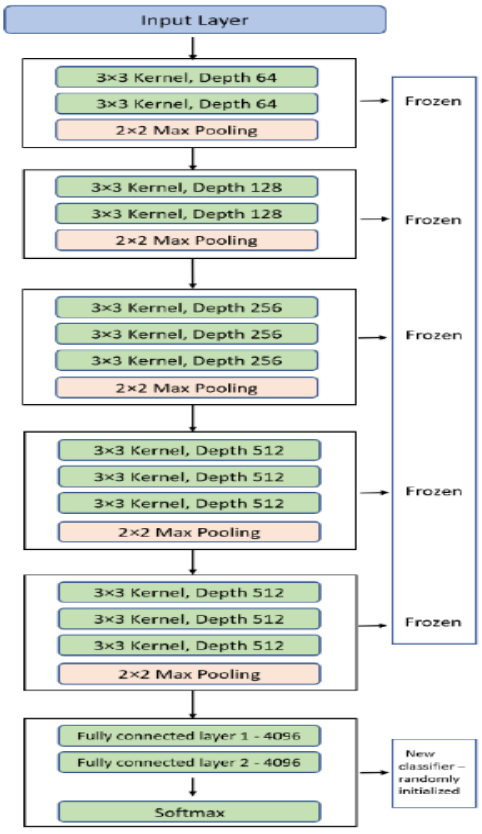
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 fc1 (Dense)                 (None, 4096)              102764544

 fc2 (Dense)                 (None, 4096)              16781312

=================================================================
Total params: 134,260,544
Trainable params: 134,260,544
Non-trainable params: 0
_________________________________________________________________
```

10- ضبط النموذج لنقل التعلم حيث نقوم بتجميد كل الطبقات و نقوم بإضافة طبقة الخرج و هي Softmax

و نقوم باختيار دالة الخسار Loss من نوع categorical crossentropy

و باختيار نوع المحسن optimizer من نوع Rmsprop

```
for layers in model.layers: # Here we turn off the ability to train the added layers. Because they are already highly educated. (Transfer Learning is here!)
    layers.trainable = False

model.add(Dense(numberOfClass, activation="softmax")) #Vgg16' We add the layer in accordance with our own data.

print(model.summary())

model.compile(loss = "categorical_crossentropy",
              optimizer = "rmsprop",
              metrics = ["accuracy"]) #compliting
```



```
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792

block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928

block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0

block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856

block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584

block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0

block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168

block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080

block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080

block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0

block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160

block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808

block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808

block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0

block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808

block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808

block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808

block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0

flatten (Flatten)            (None, 25088)             0

fc1 (Dense)                  (None, 4096)              102764544

fc2 (Dense)                  (None, 4096)              16781312

dense (Dense)                (None, 12)                49164

=================================================================
Total params: 134,309,708
Trainable params: 49,164
Non-trainable params: 134,260,544
_________________________________________________________________
None
```

11- نقوم بعمل data augmentation بالضافة الى تحديد مقدار الدفعة و تغير بعد الصور المدخلة لكل من مجلد training و val

```
train_datagen = ImageDataGenerator(rescale=1.0/255,
                                   zoom_range=0.2,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   fill_mode='nearest')
train_data = train_datagen.flow_from_directory('/content/fruits-360-original-size/fruits-360-original-size/Training',
                                               target_size=(224, 224),
                                               batch_size=32,
                                               class_mode='categorical',
                                               shuffle=True)
test_datagen = ImageDataGenerator(rescale=1.0/255,
                                  zoom_range=0.2,
                                  width_shift_range=0.2,
                                  height_shift_range=0.2,
                                  fill_mode='nearest')

test_data = test_datagen.flow_from_directory('/content/fruits-360-original-size/fruits-360-original-size/Validation',
                                             shuffle=False,
                                             batch_size=32,
                                             target_size = (224, 224),
                                             class_mode='categorical')
```

```
Found 3719 images belonging to 12 classes.
Found 1860 images belonging to 12 classes.
```

12- نبدأ بتدريب الشبكة العصبونية و نختارج 25 epoch

```
#train_data = ImageDataGenerator().flow_from_directory(train_path,target_size = (224,224))
#test_data = ImageDataGenerator().flow_from_directory(test_path,target_size = (224,224))

batch_size = 32

'''hist = model.fit_generator(train_data,
                              steps_per_epoch=1600//batch_size,
                              epochs= 25,
                              validation_data=test_data,
                              validation_steps= 800//batch_size,)'''
hist = model.fit_generator(train_data,
                              steps_per_epoch=116,
                              epochs= 25,
                              validation_data=test_data,
                              validation_steps=58 ,)
```

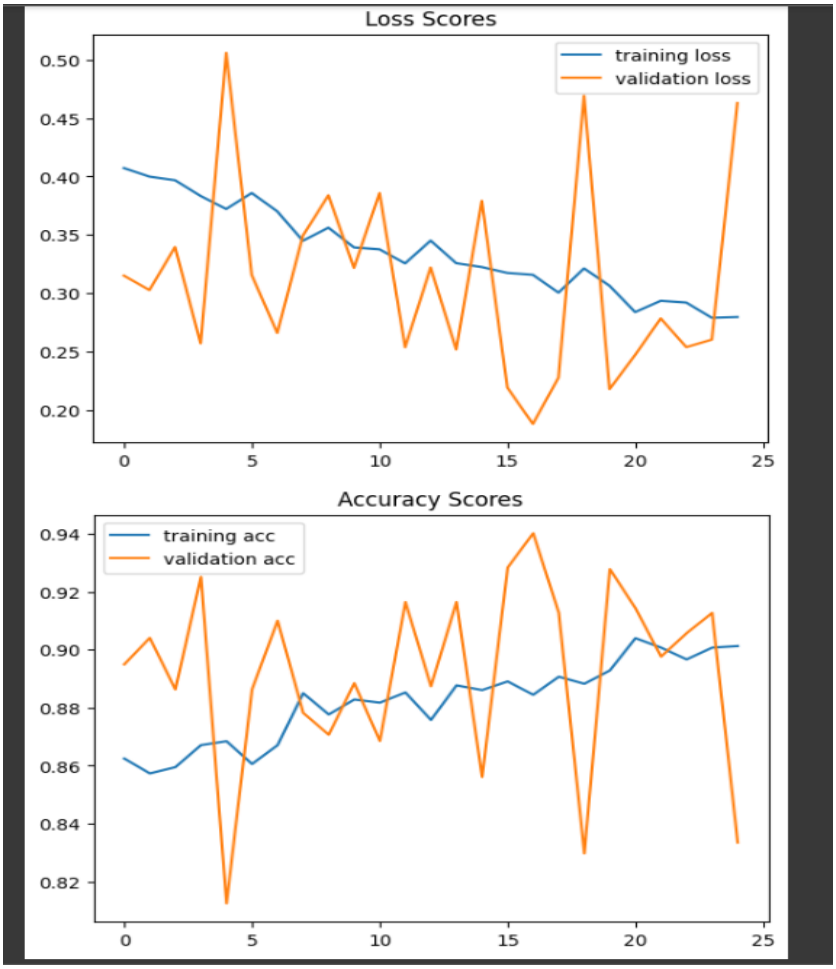
```
<ipython-input-48-f26e37529ae5>:11: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  hist = model.fit_generator(train_data,
Epoch 1/25
116/116 [==============================] - 97s 842ms/step - loss: 0.4072 - accuracy: 0.8625 - val_loss: 0.3151 - val_accuracy: 0.8949
Epoch 2/25
116/116 [==============================] - 96s 831ms/step - loss: 0.3999 - accuracy: 0.8573 - val_loss: 0.3027 - val_accuracy: 0.9041
Epoch 3/25
116/116 [==============================] - 97s 835ms/step - loss: 0.3967 - accuracy: 0.8595 - val_loss: 0.3395 - val_accuracy: 0.8863
Epoch 4/25
116/116 [==============================] - 98s 843ms/step - loss: 0.3833 - accuracy: 0.8671 - val_loss: 0.2571 - val_accuracy: 0.9251
Epoch 5/25
116/116 [==============================] - 97s 835ms/step - loss: 0.3721 - accuracy: 0.8685 - val_loss: 0.5058 - val_accuracy: 0.8125
Epoch 6/25
116/116 [==============================] - 84s 729ms/step - loss: 0.3858 - accuracy: 0.8606 - val_loss: 0.3157 - val_accuracy: 0.8863
Epoch 7/25
116/116 [==============================] - 92s 800ms/step - loss: 0.3701 - accuracy: 0.8671 - val_loss: 0.2660 - val_accuracy: 0.9100
Epoch 8/25
116/116 [==============================] - 79s 687ms/step - loss: 0.3449 - accuracy: 0.8850 - val_loss: 0.3492 - val_accuracy: 0.8782
Epoch 9/25
116/116 [==============================] - 93s 802ms/step - loss: 0.3562 - accuracy: 0.8777 - val_loss: 0.3838 - val_accuracy: 0.8707
Epoch 10/25
116/116 [==============================] - 94s 808ms/step - loss: 0.3392 - accuracy: 0.8828 - val_loss: 0.3217 - val_accuracy: 0.8885
Epoch 11/25
116/116 [==============================] - 98s 844ms/step - loss: 0.3376 - accuracy: 0.8817 - val_loss: 0.3858 - val_accuracy: 0.8685
Epoch 12/25
116/116 [==============================] - 83s 720ms/step - loss: 0.3256 - accuracy: 0.8853 - val_loss: 0.2537 - val_accuracy: 0.9165
Epoch 13/25
116/116 [==============================] - 94s 811ms/step - loss: 0.3451 - accuracy: 0.8758 - val_loss: 0.3221 - val_accuracy: 0.8874
Epoch 14/25
116/116 [==============================] - 80s 687ms/step - loss: 0.3257 - accuracy: 0.8877 - val_loss: 0.2519 - val_accuracy: 0.9165
Epoch 15/25
116/116 [==============================] - 78s 677ms/step - loss: 0.3224 - accuracy: 0.8861 - val_loss: 0.3791 - val_accuracy: 0.8561
Epoch 16/25
116/116 [==============================] - 95s 820ms/step - loss: 0.3174 - accuracy: 0.8891 - val_loss: 0.2193 - val_accuracy: 0.9283
Epoch 17/25
116/116 [==============================] - 97s 837ms/step - loss: 0.3158 - accuracy: 0.8845 - val_loss: 0.1881 - val_accuracy: 0.9402
Epoch 18/25
116/116 [==============================] - 93s 807ms/step - loss: 0.3005 - accuracy: 0.8907 - val_loss: 0.2278 - val_accuracy: 0.9127
Epoch 19/25
116/116 [==============================] - 94s 810ms/step - loss: 0.3213 - accuracy: 0.8883 - val_loss: 0.4691 - val_accuracy: 0.8297
Epoch 20/25
116/116 [==============================] - 95s 818ms/step - loss: 0.3064 - accuracy: 0.8929 - val_loss: 0.2179 - val_accuracy: 0.9278
```

13- حفظ أوزان النموذج المدرب

```
model.save('/content/fruit_360_last_model_Apples_12_classes_last_90_ACC.h5')
```

14- عرض النتائج لتدريب من دقة ACC و خسارة LOSS

```
plt.title('Loss Scores')
print(hist.history.keys())
plt.plot(hist.history["loss"],label = "training loss")
plt.plot(hist.history["val_loss"],label = "validation loss")
plt.legend()
plt.show()
plt.figure()
plt.title('Accuracy Scores')
plt.plot(hist.history["accuracy"],label = "training acc")
plt.plot(hist.history["val_accuracy"],label = "validation acc")
plt.legend()
plt.show()
```




```
Epoch 25/25
116/116 [==============================] - 108s 936ms/step - loss: 0.2797 - accuracy: 0.9013 - val_loss: 0.4628 - val_accuracy: 0.8335
```

15- حفظ ملف Excel لعناوين الكلاسات لاستفادة منها في طباعة اسم نوع التفاح لخرج الشبكة VGG16

```
[ ]
import pandas as pd



df = pd.DataFrame(data=labels, index=[0])

df = (df.T)

print (df)

df.to_excel('Apple_13_classes_new.xlsx')
```

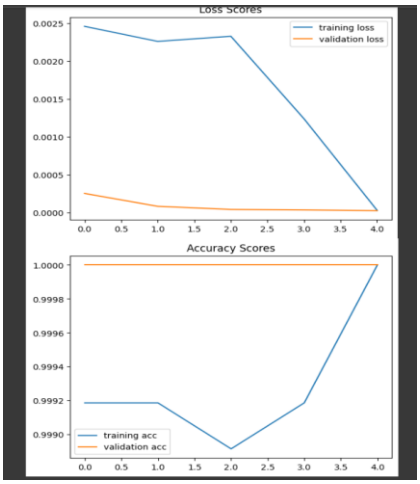
```
                    0
0       apple_braeburn_1
1   apple_crimson_snow_1
2         apple_golden_1
3         apple_golden_2
4         apple_golden_3
5   apple_granny_smith_1
6      apple_pink_lady_1
7            apple_red_1
8            apple_red_2
9            apple_red_3
10  apple_red_delicios_1
11    apple_red_yellow_1
```

16- اختبار VGG16 على صورة تفاحة (عملية الانتشار الأمامي)

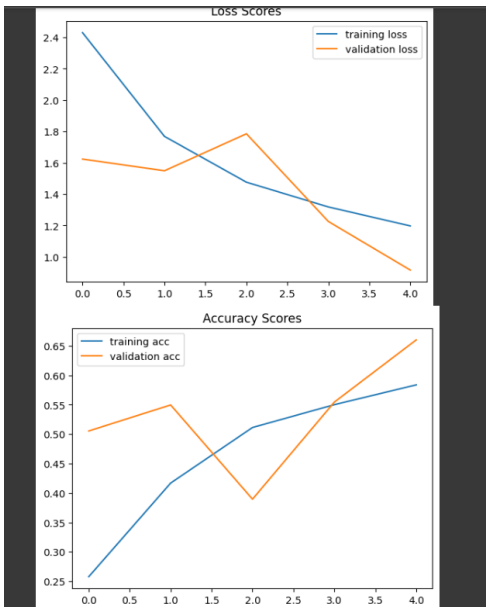
```python
import numpy as np
from tensorflow.keras.preprocessing import image
import tensorflow as tf

MODEL_PATH = "/content/fruit_360_last_model_Apples_12_classes_last_90_ACC.h5"
IMG_PATH = "/content/fruits-360-original-size/fruits-360-original-size/Test/apple_golden_3/r0_143.jpg"


model = tf.keras.models.load_model(MODEL_PATH)

img = image.load_img(IMG_PATH, target_size=(224,224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)

test_image = np.vstack([x])
#predictions = model.predict(test_image)
#print("Predicted classes : ", predictions)
#y_classes = classes.argmax(axis=0)
classes = np.argmax(model.predict(test_image), axis=-1)

print()
print('-------------------------------------')
print(classes)
print('Predicted Class: ' + CATEGORIES[classes[0]]  ,classes[0])
print('-------------------------------------')
```

```
1/1 [==============================] - 0s 185ms/step

-------------------------------------
[4]
Predicted Class: apple_golden_3 4
-------------------------------------
```

17- تجربة تدريب شبكة vgg16 بدون data augmentation ل 5 epoch



18- تجربة تدريب شبكة vgg16 مع data augmentation ل 5 epoch

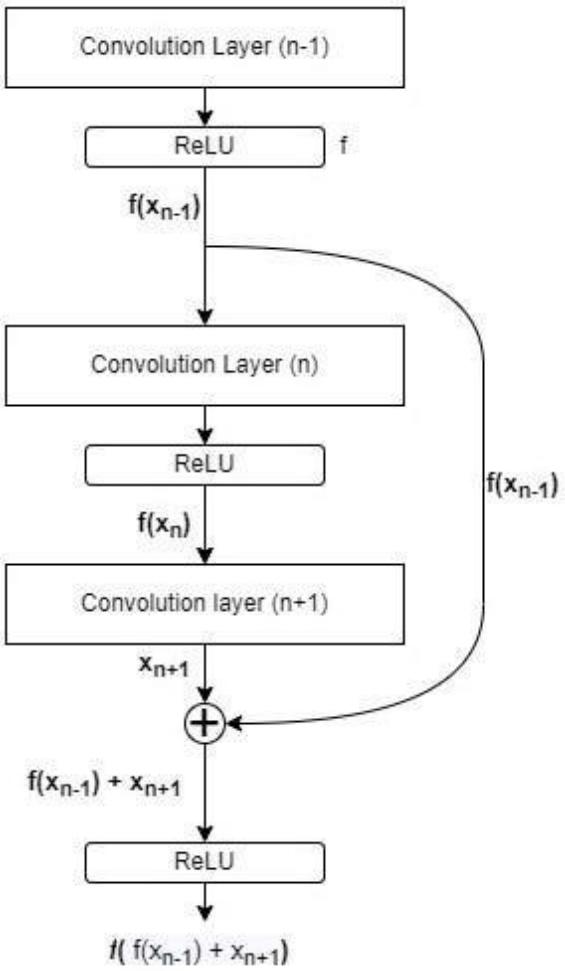


: Resnet 18

في الاعتقاد أنه كلما كانت الشبكة العصبية أعمق ، كان الأداء أفضل ، ولكن عندما جرب الباحثون شبكات عصبية أكثر عمقًا ، وجد أن إضافة المزيد من الطبقات إلى شبكة عميقة لا يؤدي دائمًا إلى تحسين أدائها بل يقللها ، وهو ما كان بسبب ظاهرة تلاشي التدرجات vanishing gradients في الشبكات العصبية العميقة جدًا ,حيث يعاد انتشار قيمة المشتق خلفاً إلى الطبقات الأولى بالضرب المتكرر وبالتالي تصبح قيمة المشتق لا متناهية في الصغر، وكنتيجة لذلك كلما كانت الشبكة أكثر عمقاً يصل الأداء لحد الإشباع أو يسوء بشكل سريع. و نتيجة لذلك ، تم اقتراح أن إضافة المزيد من الطبقات إلى الشبكة العصبية العميقة يجب إما زيادة أدائها أو السماح لها بالبقاء كما هي ، ولكن يجب ألا تقلل من الأداء. من أجل تحقيق ذلك ، توصلوا إلى مفهوم تخطي الاتصالات / الاتصالات المتبقية Skip connections/Residual connections، والتي من خلالها يمكننا تجنب فقدان تدفق المعلومات.

يأخذ اتصال التخطي / المتبقي التنشيطات من (n-1) طبقة التفاف ويضيفها إلى ناتج الالتفاف لطبقة (n + 1) ثم يطبق ReLU على هذا المجموع ، وبالتالي تخطي الطبقة n.

يوضح الرسم كيفية عمل اتصال التخطي(f (x) للإشارة إلى Relu المطبق على x حيث x هو الناتج بعد تطبيق عملية الالتفاف).

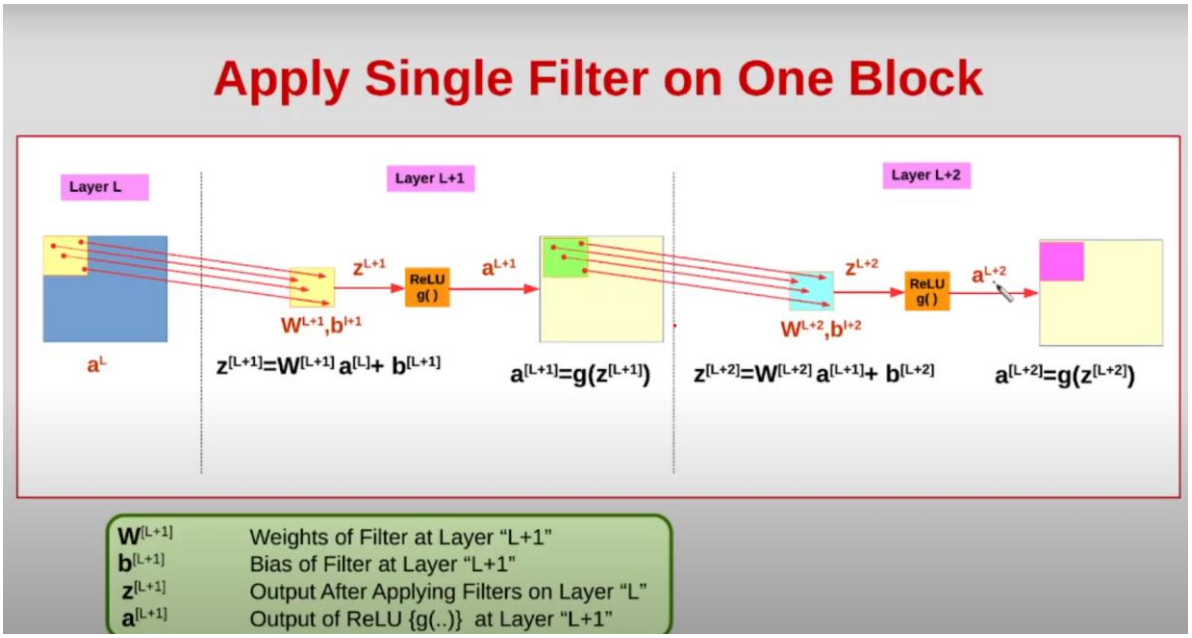


يساعد هذا ببساطة ، إذا كانت الطبقة n لا تتعلم أي شيء حتى في حالة لا نريد أي فقد للمعلومات ، لأن في طبقة $(n + 1)^{th}$ نستخدم إخراج الطبقة layer (n-1) عندما نكمل الى الأمام من خلال الطبقات ثم نطبق تابع التنشيط على هذا المجموع.

وبالتالي ، فإننا نقوم بتفعيل خاصية في الشبكة لتجاوز تابع تفعيل أو تنشيط واحد بينهما إذا لم تقدم أي معلومات مفيدة أو لا تقدم أي معلومات على الإطلاق ، أي 0 ، وستستخدم الشبكة المعلومات السابقة ، وبالتالي الحفاظ على أداء ثابت.

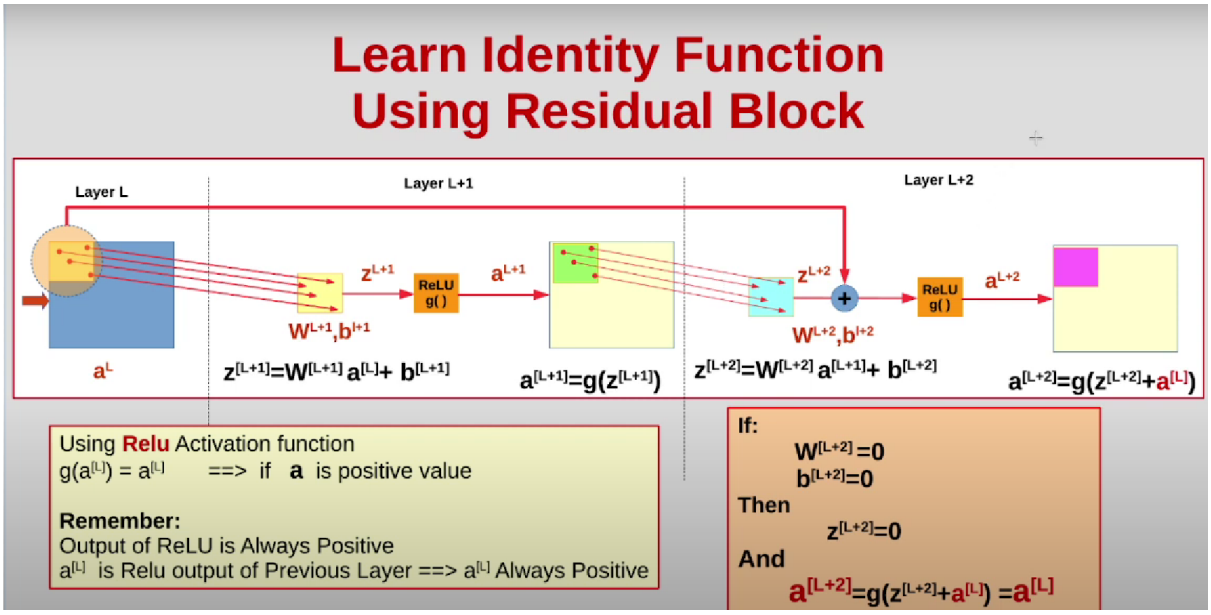
إذا كانت كلتا الطبقتين توفران معلومات مهمة ، فإن الحصول على معلومات سابقة سيعزز الأداء على أي حال.

الجزء النظري من res block



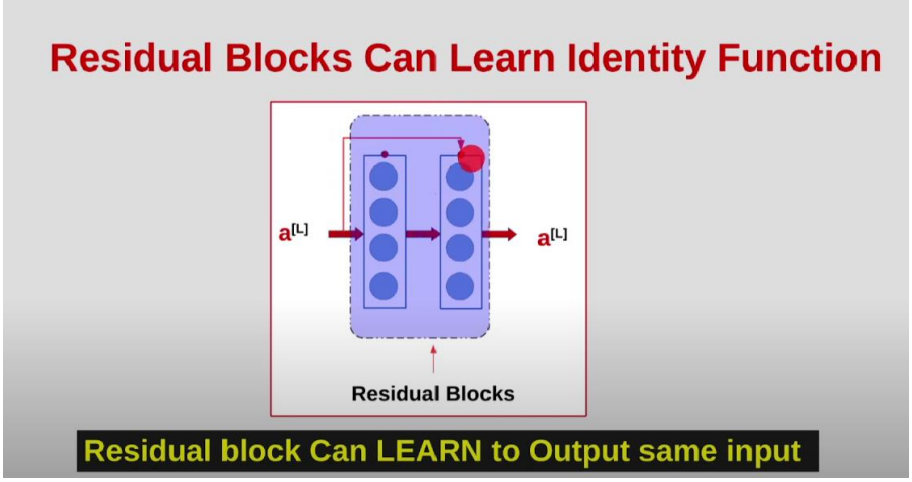
في الطبقات الأساسية من CNN يتم ضرب خرج الطبقة aL مع طبقة التالية L+1 حيث يتم ضرب كل عنصر خرج من aL بمقابله من وزن ضمن filter WL+1 + الانحياز bL+1 طبعا يكون مجموع سيغما و من ثم ندخل ZL+1 على تابع التفعيل غير خطي أي aL+1 = g(ZL+1)

عندما نزيد عدد الطبقات ضمن CNN يمكن أن تكون بدون فائدة ولا تزيد الدقة بل العكس تنقصها لأن optimizer عند تعدل الأوزان يكون الفارق بسيط جد بحيث لا يرى أي تغير أي يكون خرج الطبقة 0 أو قريب من 0 و يأثر على باقي الطبقات .

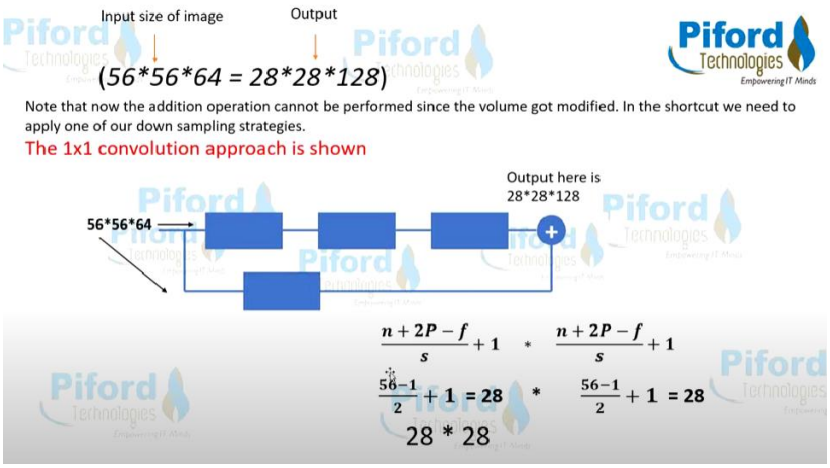


في res block نقوم بأخذ اتصال التخطي Skip connections/Residual connections ونقوم بجمعك مع الخرج ZL+2 بحيث اذا كانت طبقة conv لا تعطي أي معلومات في هذه الحالة يكون الدخل نفس الخرج لطبقة و هذه يسمى Learning identity function

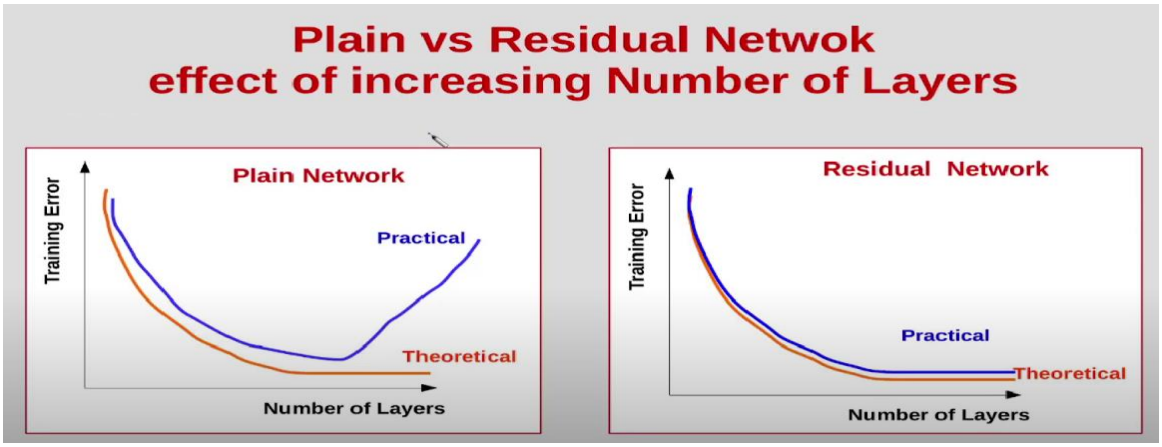
و هذا ما يسمى Residual Block



في حالة أن الطبقة تعطي خرج ليس 0 لجمع دخل الطبقة الأولى مع خرج conv من خلال عملية اتصال التخطي Skip connections/Residual connections نستخدم طريقة 1x1 conv لكي تصبح أبعاد feature map الدخل نفس الخرج لعملية الجمع و هذا يحسب من عملية التدريب



مقارنة في الطبقات CNN العادية و Residual Block عند حساب الخطأ :



بنية resnet 18 :

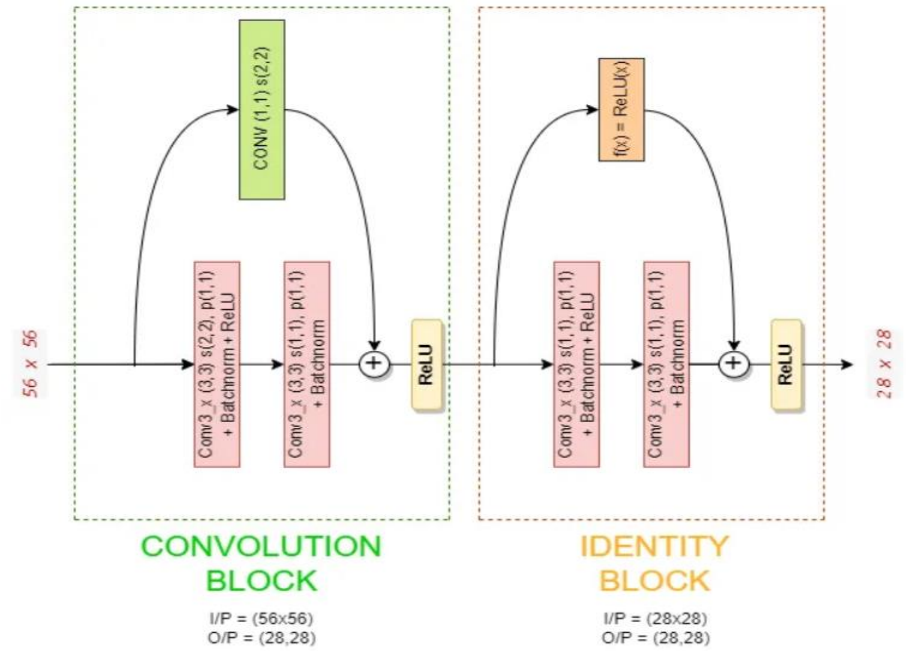
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet18 - Structural Details

#	Input Image			output			Layer	Stride	Pad	Kernel		in	out	Param
1	227	227	3	112	112	64	conv1	2	1	7	7	3	64	9472
	112	112	64	56	56	64	maxpool	2	0.5	3	3	64	64	0
2	56	56	64	56	56	64	conv2-1	1	1	3	3	64	64	36928
3	56	56	64	56	56	64	conv2-2	1	1	3	3	64	64	36928
4	56	56	64	56	56	64	conv2-3	1	1	3	3	64	64	36928
5	56	56	64	56	56	64	conv2-4	1	1	3	3	64	64	36928
6	56	56	64	28	28	128	conv3-1	2	0.5	3	3	64	128	73856
7	28	28	128	28	28	128	conv3-2	1	1	3	3	128	128	147584
8	28	28	128	28	28	128	conv3-3	1	1	3	3	128	128	147584
9	28	28	128	28	28	128	conv3-4	1	1	3	3	128	128	147584
10	28	28	128	14	14	256	conv4-1	2	0.5	3	3	128	256	295168
11	14	14	256	14	14	256	conv4-2	1	1	3	3	256	256	590080
12	14	14	256	14	14	256	conv4-3	1	1	3	3	256	256	590080
13	14	14	256	14	14	256	conv4-4	1	1	3	3	256	256	590080
14	14	14	256	7	7	512	conv5-1	2	0.5	3	3	256	512	1180160
15	7	7	512	7	7	512	conv5-2	1	1	3	3	512	512	2359808
16	7	7	512	7	7	512	conv5-3	1	1	3	3	512	512	2359808
17	7	7	512	7	7	512	conv5-4	1	1	3	3	512	512	2359808
	7	7	512	1	1	512	avg pool	7	0	7	7	512	512	0
18	1	1	512	1	1	1000	fc					512	1000	513000
Total														11,511,784

نختار كتلة Conv3_x ونحاول فهم ما يحدث بداخلها حيث هذه الكتلة تحتوي الى كتل الالتفاف والهوية Convolution and Identity blocks.

Conv3_x block تدفق البيانات باستخدام Convolution و Identity Blocks



تخبرنا الصورة أعلاه بالتفاصيل حول كيفية انتشار صورة 56 × 56 خلال كتلة Conv3_x ، والآن سننظر في كيفية تحول الصورة في كل خطوة داخل هذه الكتل.

- كتلة التلافيفية Convolution Block :

الإدخال إلى كتلة Conv3_x عبارة عن صورة لشكل (56 × 56) مع 64 قناة channels أي العمق ، تقوم طبقة الالتفاف الأولى بتحويل الصورة إلى صورة (28 × 28) مع 128 قناة باستخدام نواة 3 × 3 وخطوة 2 × 2 مع حشوة 1 × 1 ، وتطبق عليها ReLU. طبقة الالتفاف الثانية تأخذ هذه الصورة كمدخلات وتخرج الصورة بنفس الشكل (28 × 28 × 128). الآن من أجل تطبيق اتصال متبقي / تخطي ، يتعين علينا إضافة ناتج كتلة Conv2_x وهي (56 × 56 × 64) مع إخراج الالتفاف الثاني وهو (28 × 28 × 128) ، للقيام بذلك نحتاج إلى تحويل إخراج Conv2_x إلى (28x28x128) ، يتم ذلك عن طريق تطبيق التفاف آخر مع مرشح1x1 وخطوة 2 ، مع قنوات الإدخال 64 وقنوات الإخراج 128.

ببساطة ، فإن كتلة Convolution مسؤولة عن تحويل الإخراج من كتلة واحدة باستخدام عملية الالتواء بحيث يمكن استخدامها بشكل فعال للإضافة مع إخراج كتلة التفاف أخرى.

بعد الانتهاء من الإضافة ، يتم تطبيق التنشيط (ReLU) على مخرجاته وإرساله إلى كتلة الهوية identity Block.

- كتلة الهوية identity Block :

المدخلات والمخرجات من كتلة الهوية هي نفسها ، وبالتالي لتطبيق اتصال Residual / Skip ، لا نحتاج إلى تطبيق أي تحويل على إخراج Convolution Block. لذلك لتطبيق اتصال Skip / Residual ، كل ما نحتاج إليه هو إضافة ناتج كتلة Convolution إلى إخراج طبقة الالتفاف الرابعة في Conv3_x block ، ثم تطبيق ReLU عليها.

نحن نفهم الآن أنه كلما كانت هناك حاجة لضبط الإخراج لإتاحة إمكانية تطبيق اتصال متبقي ، سنحتاج إلى كتلة التفاف ، وكلما كانت المدخلات والمخرجات هي نفسها ، نحتاج إلى كتلة الهوية.

تدريب resnet 18 :

1- نقوم بتعريف المكتبات الازمة

```
import os, time
import numpy as np
import random
random.seed(42)
import pandas as pd
import seaborn as sns #Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score, classification_report

import torch
torch.manual_seed(42)
from torch import nn
from torch.optim import SGD, Adam
from torch.utils.data import DataLoader, RandomSampler
from torch.utils.data.dataset import Dataset
from torchvision.models import resnet
from torchvision import transforms, datasets, models
from torch.optim.lr_scheduler import ReduceLROnPlateau #Reduce learning rate when a metric has stopped improving

print('PyTorch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))
```

```
PyTorch 2.0.1+cu118 _CudaDeviceProperties(name='Tesla T4', major=7, minor=5, total_memory=15101MB, multi_processor_count=40)
```

2- تعريف دوال كل من Load_transform_images لتهيئة الصور مثل Resize(224,224) دخل شبكة resnet 18 و فتل الصورة مثلا **RandomHorizontalFlip** , تحويل الصورة هو عملية لتغيير القيم الأصلية لوحدات البكسل في الصورة إلى مجموعة من القيم الجديدة.
أحد أنواع التحول الذي نقوم به على الصور هو تحويل الصورة إلى PyTorch tensor. عندما يتم تحويل صورةPyTorch tensor تصبح قيم البكسل بين 0.0 و 1.0. في PyTorch ، يمكن إجراء هذا التحويل باستخدام torchvision.transforms.ToTensor (). يقوم بتحويل صورة PIL بنطاق بكسل من [0 ، 255] إلى PyTorch FloatTensor بالشكل (C، H، W) بنطاق [0.0 ، 1.0].

يعد تقييس الصور transforms.Normalize() ممارسة جيدة جدًا عندما نعمل مع الشبكات العصبية العميقة. تقييس الصور يعني تحويل الصور إلى قيم بحيث يصبح المتوسط والانحراف المعياري للصورة 0.0 و 1.0 على التوالي.

[Transforming and augmenting images — Torchvision 0.15 documentation (pytorch.org)]

```
def load_transform_images(images_path, presplit, train_split, test_split, val_split, batch_size, threads, mean, std):
    train_transform = transforms.Compose([
                                        #transforms.RandomRotation(degrees=15),
                                        #transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
                                        #transforms.RandomResizedCrop((224,224)),
                                        transforms.Resize((224,224)),
                                        transforms.RandomHorizontalFlip(),
                                        transforms.ToTensor(),
                                        transforms.Normalize(torch.Tensor(mean),
                                                             torch.Tensor(std))])

    test_transform = transforms.Compose([
                                         transforms.Resize((224,224)),
                                         #transforms.CenterCrop((224,224)),
                                         transforms.ToTensor(),
                                         transforms.Normalize(torch.Tensor(mean),
                                                              torch.Tensor(std))])

    val_transform = transforms.Compose([
                                        transforms.Resize((224,224)),
                                        #transforms.CenterCrop((224,224)),
                                        transforms.ToTensor(),
                                        transforms.Normalize(torch.Tensor(mean),
                                                             torch.Tensor(std))])

    if presplit:
        try:
            training_set = datasets.ImageFolder(root=images_path+'/Training', transform=train_transform)
            validation_set = datasets.ImageFolder(root=images_path+'/Test', transform=val_transform)
        except FileNotFoundError:
            raise Exception('Not presplit into Training and Validation sets')
        try:
            testing_set = datasets.ImageFolder(root=images_path+'/val', transform=test_transform)
        except:
            testing_set = validation_set
        dataset = training_set
```

```
if presplit:
    try:
        training_set = datasets.ImageFolder(root=images_path+'/Training', transform=train_transform)
        validation_set = datasets.ImageFolder(root=images_path+'/Test', transform=val_transform)
    except FileNotFoundError:
        raise Exception('Not presplit into Training and Validation sets')
    try:
        testing_set = datasets.ImageFolder(root=images_path+'/val', transform=test_transform)
    except:
        testing_set = validation_set
    dataset = training_set
else:
    dataset = datasets.ImageFolder(root=images_path, transform=train_transform)
    train_size = int(train_split * len(dataset))
    test_size = int(test_split * len(dataset))
    val_size = len(dataset) - train_size - test_size
    training_set, testing_set, validation_set = torch.utils.data.random_split(dataset, [train_size, test_size, val_size])

training_set_loader = DataLoader(training_set, batch_size=batch_size, num_workers=threads, shuffle=True)
validation_set_loader = DataLoader(validation_set, batch_size=batch_size, num_workers=threads, shuffle=True)
testing_set_loader = DataLoader(testing_set, batch_size=batch_size, num_workers=threads, shuffle=False)

return training_set_loader, testing_set_loader, validation_set_loader, dataset, training_set, testing_set, validation_set
```

3- نقوم بعمل دالة load_network () عند استدعاء شبكة resnet 18

```
def load_network(net_model, net_name, dropout_ratio, class_names):
    for name, param in net_model.named_parameters():
        param.requires_grad = False

    if net_name.startswith('resnet'):
        num_ftrs = net_model.fc.in_features
        net_model.fc = nn.Sequential(nn.Linear(num_ftrs, 256),
                                     nn.ReLU(),
                                     nn.Dropout(p=dropout_ratio),
                                     nn.Linear(256, len(class_names)))
        display(net_model)

    elif net_name.startswith('vgg'):
        num_ftrs = net_model.classifier[6].in_features
        net_model.classifier[6] = nn.Sequential(nn.Linear(num_ftrs, 256),
                                                nn.ReLU(),
                                                nn.Dropout(p=dropout_ratio),
                                                nn.Linear(256, len(class_names)))
        display(net_model.classifier)

    total_params = sum(param.numel() for param in net_model.parameters())
    print(f'{total_params:,} total parameters')

    total_trainable_params = sum(param.numel() for param in net_model.parameters() if param.requires_grad)
    print(f'{total_trainable_params:,} training parameters')

    return net_model
```

4- تعريف دوال plot_images_per_class و plot_grid_images لعرض الصور

```
def plot_images_per_class(images_path, mode, title):
    data_folder = images_path+'/'+mode+'/'
    item_dict = {root.split('/')[-1]: len(files) for root, _, files in os.walk(data_folder)}

    plt.figure(figsize=(20,8))
    plt.bar(list(item_dict.keys())[1:], list(item_dict.values())[1:], color='g')
    plt.title(title)
    plt.xticks(rotation=90)
    plt.xlabel('Class')
    plt.ylabel('Number of Images')
    plt.show()

def plot_grid_images(training_set, batch_size, class_names, mean, std, rows=3, columns=3, size=14):
    sampler = RandomSampler(training_set, num_samples=batch_size, replacement=True)
    train_loader = DataLoader(training_set, sampler=sampler, shuffle=False, batch_size=batch_size, num_workers=0)

    dataiter = iter(train_loader)
    #images, labels = dataiter.next()
    images, labels = next(dataiter)
    plt.figure(figsize=(size,size))
    for i in range(rows*columns):
        plt.subplot(rows, columns, i+1)
        plt.title(class_names[labels.numpy()[i]])
        img = images[i].permute(1,2,0)
        img = torch.tensor(std)*img + torch.tensor(mean)
        plt.axis('off')
        plt.imshow(img, interpolation='none')
        plt.tight_layout()
```

5- تعريف دالة train_model() لتدريب النموذج

```python
def train_model(results_path, model_name, model, train_loader, val_loader, lr, epoch, momentum, weight_decay, patience, n_epochs_stop):
    """
    """
    criterion = nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=lr)
    #optimizer = SGD(model.parameters(), lr=lr, momentum=momentum, weight_decay=weight_decay)
    scheduler = ReduceLROnPlateau(optimizer, patience=patience, factor=0.1, verbose=True)

    loaders = {'train': train_loader, 'val': val_loader}
    losses = {'train': [], 'val': []}
    accuracies = {'train': [], 'val': []}

    y_testing = []
    preds = []

    min_val_loss = np.Inf
    epochs_no_improv = 0

    if torch.cuda.is_available():
        if torch.cuda.device_count() > 1:
            model = nn.DataParallel(model)
        print(f'Using {torch.cuda.device_count()} GPUs')
        model.cuda()
    else:
        print('Using CPU')

    start = time.time()
    for epoch in range(epochs):
        for mode in ['train', 'val']:
            if mode == 'train':
                model.train()
            if mode == 'val':
                model.eval()

            epoch_loss = 0
            epoch_acc = 0
            samples = 0
```

```python
            for i, (inputs, targets) in enumerate(loaders[mode]):
                if torch.cuda.is_available():
                    inputs = inputs.cuda()
                    targets = targets.cuda()

                optimizer.zero_grad()
                output = model(inputs)
                loss = criterion(output, targets)

                if mode == 'train':
                    loss.backward()
                    optimizer.step()
                else:
                    y_testing.extend(targets.data.tolist())
                    preds.extend(output.max(1)[1].tolist())

                if torch.cuda.is_available():
                    acc = accuracy_score(targets.data.cuda().cpu().numpy(), output.max(1)[1].cuda().cpu().numpy())
                else:
                    acc = accuracy_score(targets.data, output.max(1)[1])

                epoch_loss += loss.data.item()*inputs.shape[0]
                epoch_acc += acc*inputs.shape[0]
                samples += inputs.shape[0]

                if i % (len(loaders[mode])//5) == 0:
                    print(f'[{mode}] Epoch {epoch+1}/{epochs} Iteration {i+1}/{len(loaders[mode])} Loss: {epoch_loss/samples:0.2f} Accuracy: {epoch_acc/samples:0.2f}')

            epoch_loss /= samples
            epoch_acc /= samples
            losses[mode].append(epoch_loss)
            accuracies[mode].append(epoch_acc)

            print(f'[{mode}] Epoch {epoch+1}/{epochs} Iteration {i+1}/{len(loaders[mode])} Loss: {epoch_loss:0.2f} Accuracy: {epoch_acc:0.2f}')
```

```
            if mode == 'val':
                scheduler.step(epoch_loss)

        if mode == 'val':
            if epoch_loss < min_val_loss:
                torch.save(model.state_dict(), str(model_name)+'.pth')
                epochs_no_improv = 0
                min_val_loss = epoch_loss
            else:
                epochs_no_improv += 1
                print(f'Epochs with no improvement {epochs_no_improv}')
                if epochs_no_improv == n_epochs_stop:
                    print('Early stopping!')
                    return model, (losses, accuracies), y_testing, preds
                model.load_state_dict(torch.load(str(model_name)+'.pth'))

    print(f'Training time: {time.time()-start} min.')
    return model, (losses, accuracies), y_testing, preds

def test_model(model_name, model, test_loader):
    model.load_state_dict(torch.load(str(model_name)+'.pth'))

    if torch.cuda.is_available():
        model.cuda()
    model.eval()

    preds = []
    trues = []

    for i, (inputs, targets) in enumerate(test_loader):
        if torch.cuda.is_available():
            inputs = inputs.cuda()
            pred = model(inputs).data.cuda().cpu().numpy().copy()
        else:
            pred = model(inputs).data.numpy().copy()

        true = targets.numpy().copy()
        preds.append(pred)
        trues.append(true)

        if i % (len(test_loader)//5) == 0:
            print(f'Iteration {i+1}/{len(test_loader)}')
    return np.concatenate(preds), np.concatenate(trues)
```

6- تنزيل dataset من Kaggle

```
#Set dataset from KAGGLE
import os
os.environ['KAGGLE_USERNAME'] = "aubaialkhabbaz"
os.environ['KAGGLE_KEY'] = "2715ab36f8842a62a77fc816d7f14851"
!kaggle datasets download -d moltean/fruits
```

```
Downloading fruits.zip to /content
 99% 1.27G/1.28G [00:14<00:00, 71.2MB/s]
100% 1.28G/1.28G [00:14<00:00, 96.6MB/s]
```

```
[ ] ! unzip /content/fruits.zip
```

```
Streaming output truncated to the last 5000 lines.
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/111_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/112_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/113_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/114_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/115_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/116_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/117_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/119_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/120_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/121_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/122_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/123_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/124_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/125_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/126_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/127_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/128_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/129_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/130_100.jpg
  inflating: fruits-360_dataset/fruits-360/Training/Tomato 3/131_100.jpg
```

7- نقوم بتحديد المسار للبيانات و تطبيق Load_transform على بيانات fruits 360 و طباعة أسماء Classes

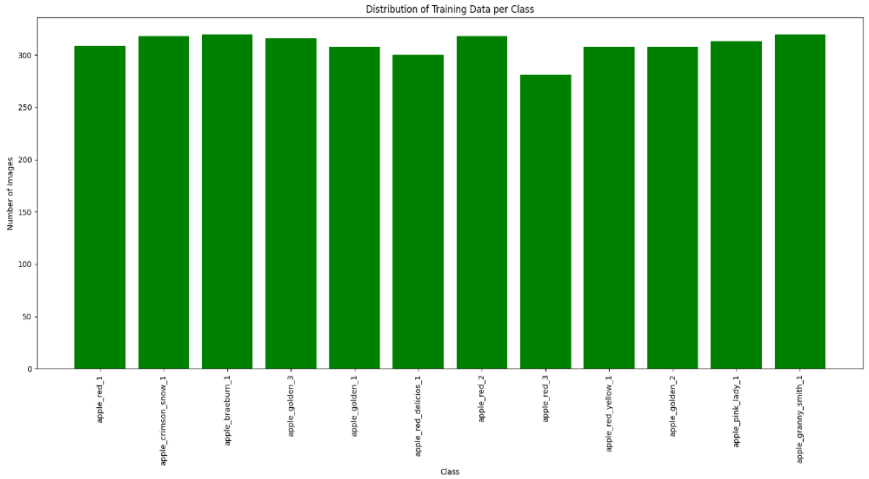
```
[ ] !rm -rf '/content/fruits-360-original-size/fruits-360-original-size/Validation/cabbage_white_1'
```

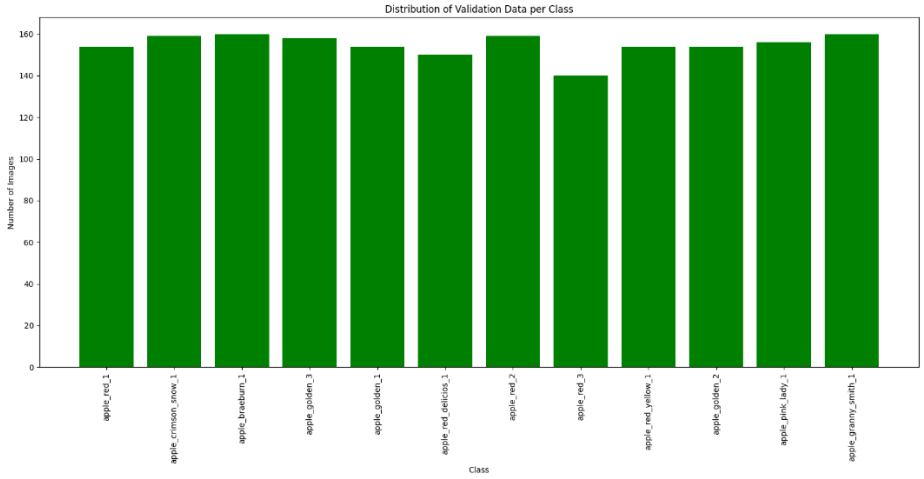
```
images_path = '/content/fruits-360-original-size/fruits-360-original-size'
results_path = images_path+'_results'
presplit = True
train_split = 0.5
val_split = 0.25
test_split = 0.25
batch_size = 256
threads = 0
mean = [0.485, 0.456, 0.406]
std = [0.229, 0.224, 0.225]

training_set_loader, testing_set_loader, validation_set_loader, dataset, training_set, testing_set, validation_set = \
                load_transform_images(images_path, presplit, train_split, test_split, val_split, batch_size, threads, mean, std)

class_names = dataset.classes
class_names = [classes for classes in class_names]
print(class_names)
```

```
['apple_braeburn_1', 'apple_crimson_snow_1', 'apple_golden_1', 'apple_golden_2', 'apple_golden_3', 'apple_granny_smith_1', 'apple_pink_lady_1', 'apple_red_1', 'apple_red_2', 'apple_red_3', 'apple_red_
```

8- طباعة توزع بيانات التدريب من خلال دالة plot_images_per_class

```
if presplit:
    plot_images_per_class(images_path, mode='Training', title='Distribution of Training Data per Class')
    plot_images_per_class(images_path, mode='Test', title='Distribution of Validation Data per Class')
```





9- استدعاء شبكة resnet 18 المدربة مسبقا

```
net_model = resnet.resnet18(pretrained=True)
net_name = 'resnet18'

dropout_ratio = 0.25

net_model = load_network(net_model, net_name, dropout_ratio, class_names)

print(f'Images in training set {len(training_set)}, validation set {len(validation_set)}, testing set {len(testing_set)}')
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' ins
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
100%|██████████| 44.7M/44.7M [00:00<00:00, 83.0MB/s]
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
11,310,924 total parameters
134,412 training parameters
Images in training set 3719, validation set 1858, testing set 1858
```

10- طباعة صور لعينات من dataset

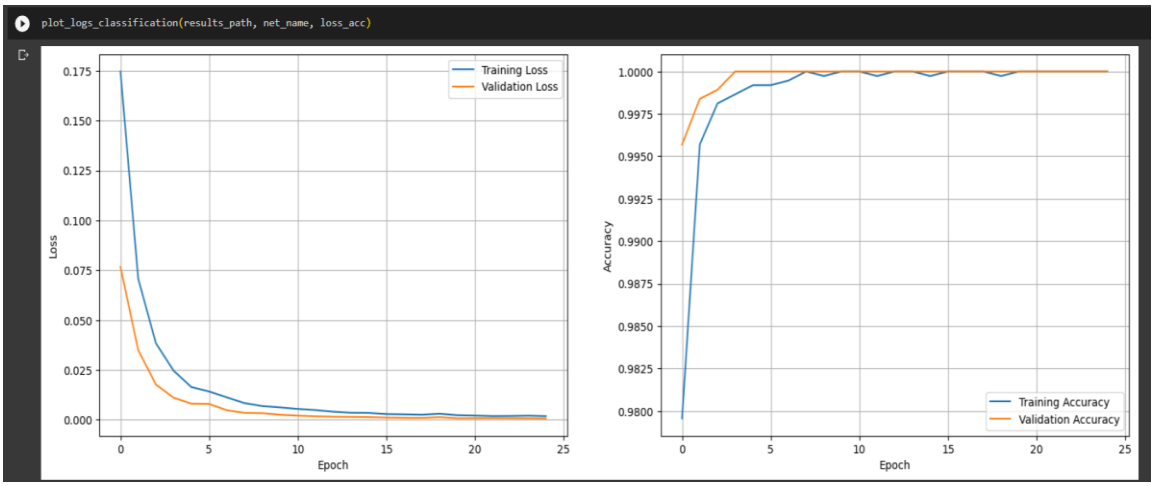
```
plot_grid_images(training_set, batch_size, class_names, mean, std, rows=3, columns=3, size=10)
```

11- بدء تدريب resnet18 من خلال train_model() و ضبط hyperparameter

```
[ ] learning_rate = 0.001
    epochs = 25
    momentum = 0.9
    weight_decay = 0
    patience = 3
    n_epochs_stop = 5

    net_model, loss_acc, y_testing, preds = train_model(results_path, net_name, net_model, training_set_loader, validation_set_loader,
                                                        learning_rate, epochs, momentum, weight_decay, patience, n_epochs_stop)

    Using 1 GPUs
    [train] Epoch 1/25 Iteration 1/15 Loss: 0.21 Accuracy: 0.98
    [train] Epoch 1/25 Iteration 4/15 Loss: 0.23 Accuracy: 0.97
    [train] Epoch 1/25 Iteration 7/15 Loss: 0.22 Accuracy: 0.97
    [train] Epoch 1/25 Iteration 10/15 Loss: 0.20 Accuracy: 0.97
    [train] Epoch 1/25 Iteration 13/15 Loss: 0.18 Accuracy: 0.98
    [train] Epoch 1/25 Iteration 15/15 Loss: 0.17 Accuracy: 0.98
    [val] Epoch 1/25 Iteration 1/8 Loss: 0.08 Accuracy: 1.00
    [val] Epoch 1/25 Iteration 2/8 Loss: 0.08 Accuracy: 0.99
    [val] Epoch 1/25 Iteration 3/8 Loss: 0.08 Accuracy: 0.99
    [val] Epoch 1/25 Iteration 4/8 Loss: 0.08 Accuracy: 1.00
    [val] Epoch 1/25 Iteration 5/8 Loss: 0.08 Accuracy: 1.00
    [val] Epoch 1/25 Iteration 6/8 Loss: 0.08 Accuracy: 1.00
    [val] Epoch 1/25 Iteration 7/8 Loss: 0.08 Accuracy: 1.00
    [val] Epoch 1/25 Iteration 8/8 Loss: 0.08 Accuracy: 1.00
```

12- طباعة مخطط الدقة و الخسارة لكل من training و val

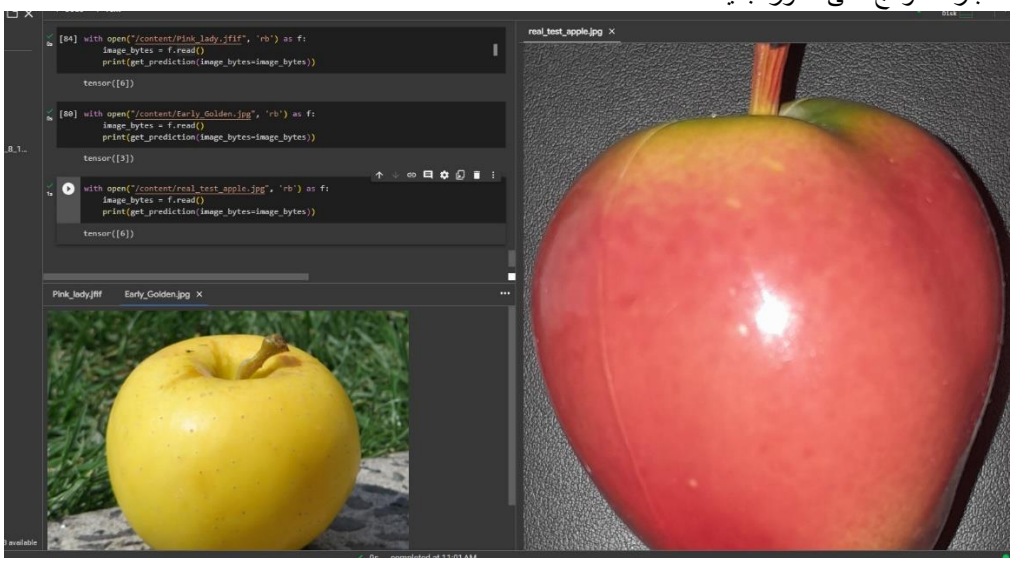
```
plot_logs_classification(results_path, net_name, loss_acc)
```

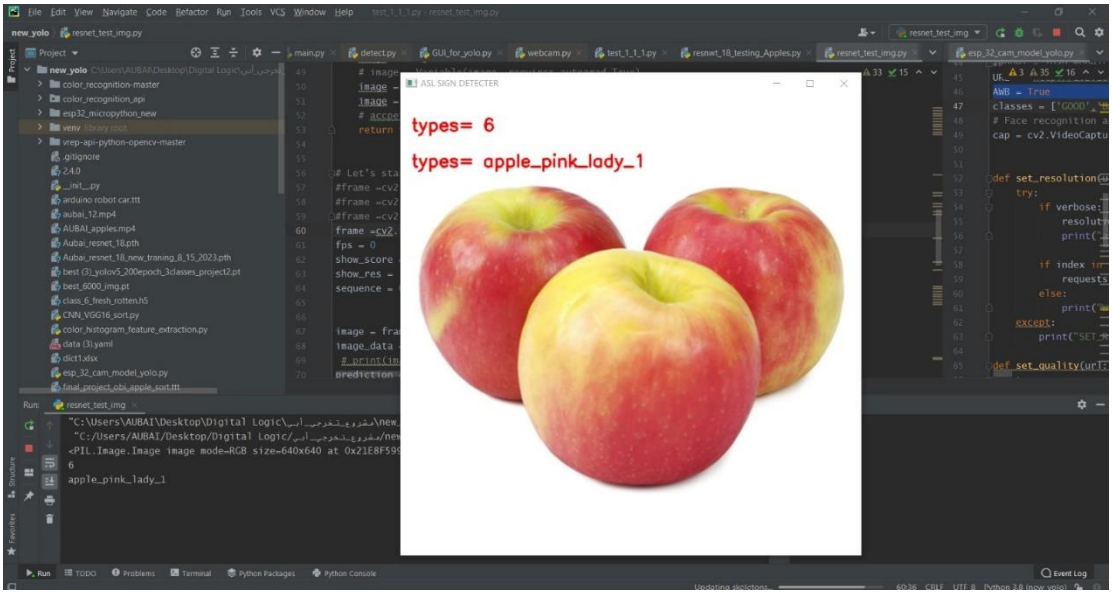


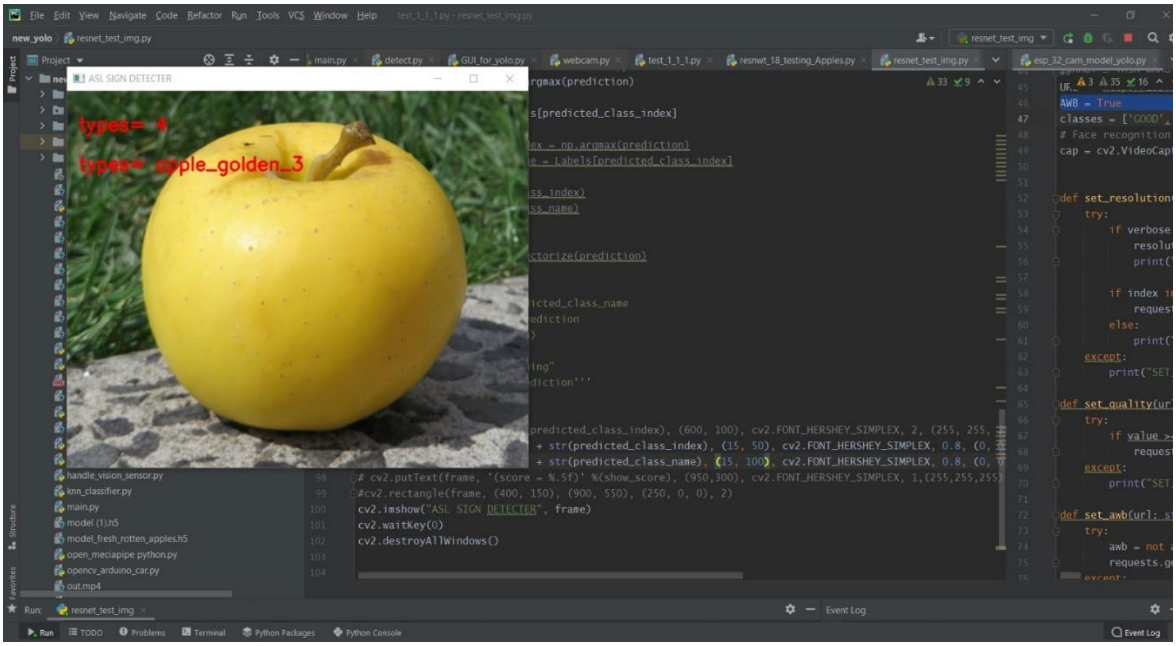
13- تخزين النموذج بعد تدريبه من خلال torch.save

```
[ ] torch.save(net_model, "Aubai_resnet_18_new_traning_8_15_2023.pth")
```

14- اختبار النموذج على صور جديدة




ASL SIGN DETECTER
types= 6
types= apple_pink_lady_1


ASL SIGN DETECTER
types= 4
types= apple_golden_3