5/27/2018

ML Project

Image Classification using deep neural networks

Done By:

Asma Hawari

Qamar Alzaman Hafez

Amena Samra

Nader Prveis

Supervised By:

ENG.Zeina AL-Dallal

فكرة المشروع :

التعرّف على الصور وتصنيفها إلى عدة صفوف .

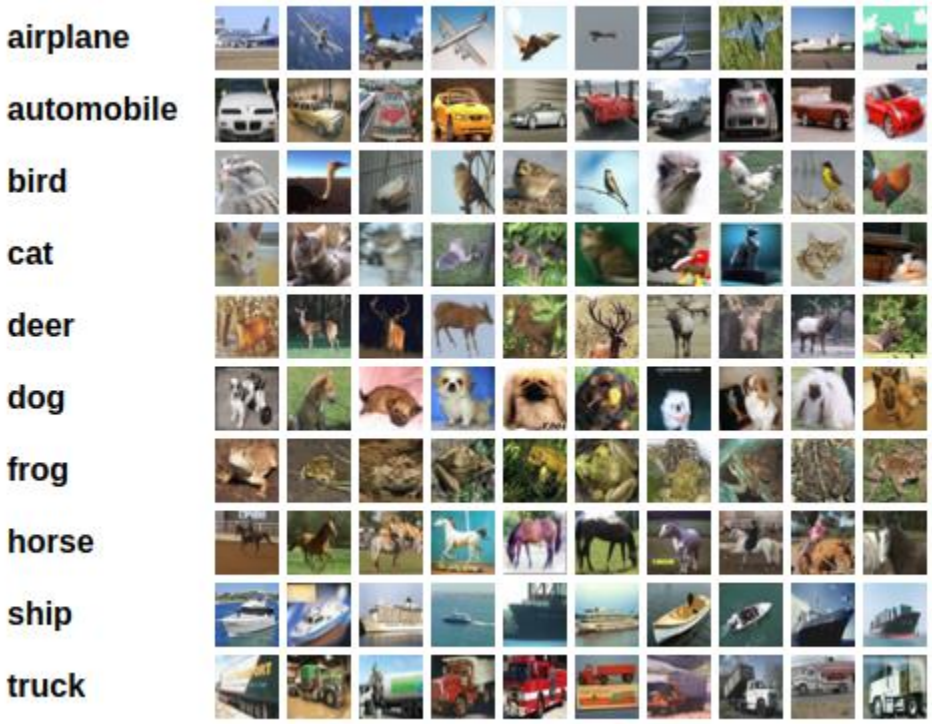
الـdataset :

قمنا باستخدام الـdataset من مكتبة keras الشهيرة والتي تسمى CIFR10 و CIFR100 مكونة من 80 مليون صورة صغيرة وتقسم إلى قسمين القسم الأول CIFR10 يكون للـdataset 10 كلاسات و CIFR100 لها 100 كلاس للتصنيف .

قمنا بالتدريب على CIFR10 بدايةً و المكونة من 32 * 32 60000 صورة ملونة ومصنفة في 10 صفوف في كل صف يوجد 6000 صورة .

تقسم الداتا إلى 50000 للـtraining images و 10000 للـtest images .

الداتا مقسمة إلى 5 أقسام للتدريب و قسم للاختبار في كل قسم يوجد 1000 صورة و قسم الاختبار مكون من 1000 صورة اختيرت بشكل عشوائي من كل صف .



CIFR10 dataset from keras

من الجدير بالذكر هنا أن هذه الداتا سيت أطلقها موقع Kaggle كـ competition منذ 4 سنوات .

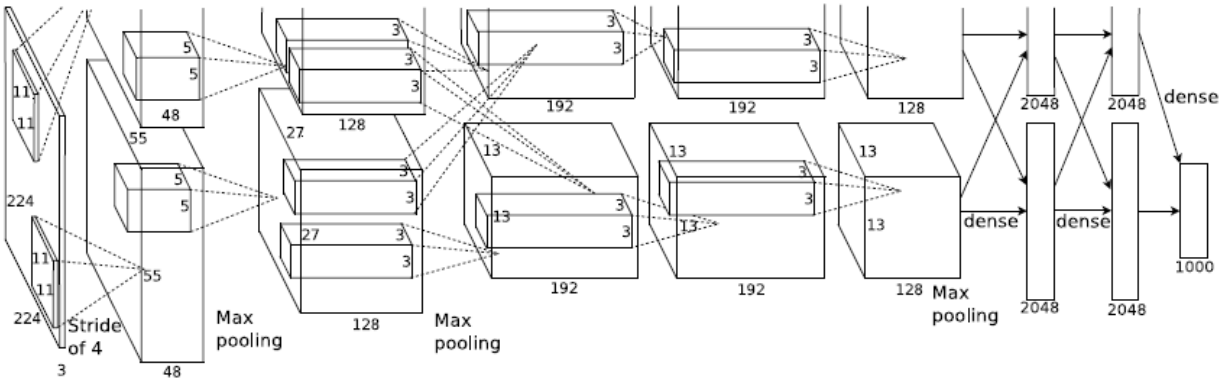
رابط المسابقة : https://www.kaggle.com/c/cifar-10/

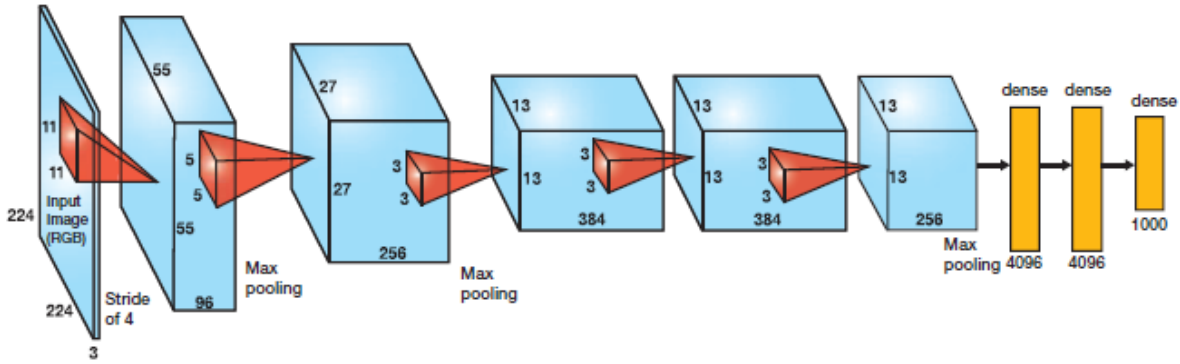
التقنيات المستخدمة في المشروع :

لهذه المشكلة العديد من الحلول التي لا تنتهي ولأنَّ كل حل سيعطي دقة مختلفة كان من أهداف المشروع الرئيسية تجريب عدة حلول لنصل إلى أعلى دقة ممكنة فبدأنا بتدريب الشبكات العصبونية من الصفر و هي الـ CNN و الـAlexNet و المقارنة بينهما عن طريق المقارنة بين نتائج كل خوارزمية و دقتها على نفس البيانات و اعتماد تقنيات أخرى مثل Transfer Learning لزيادة الدقة و الاستفادة من الشبكات العصبونية المدربة لاستخراج الفيتشرات من الصور و تصنيفها بشكل خطي عن طريق Logistic Regression , KNN، , SVM .

أولاً : AlexNet

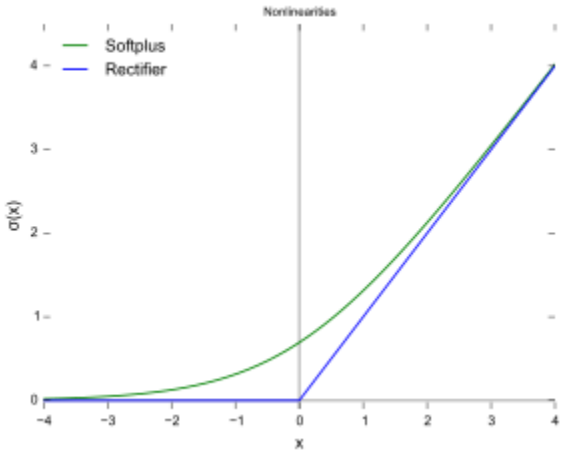
تحتوي هذه الشبكة العصبونية على 5 طبقات convolutional و 3 طبقات fully connected كما في الشكل التالي :





الـactivation function هو الـReLu عوضاً عن تابع الـSigmoid الذي في الشبكات العصبونية , يعتبر هذا التابع عماد الـ deep learning إذ أنه يضيف اللاخطية إلى العقد التي تكون أعلى من threshold معينة و هو أسرع من الـsigmoid في عملية التدريب ؛ لأن مشتق الـsigmoid صغير جداً و ذلك يؤثر على تعديل الأوزان و اختفائها تدريجياً وهذا ما يعرف بـ Vanishing gradient problem حيث يصبح الـmodel غير قار على تعديل الأوزان وبالتالي الشبكة أصبحت غير قادرة على التدريب .

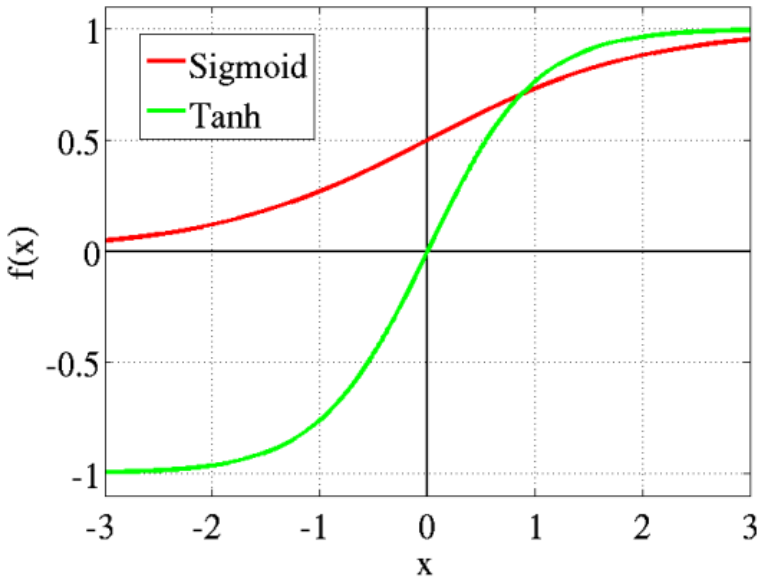
قيم المجال هنا بين الصفر إلى اللانهاية , يقوم بتقريب القيم السالبة فوراً إلى قيم صفرية .



ReLu non-linearity Activation Function

في هذه الشبكة العصبونية ReLu يكون متموضع بعد كل convolutional and fully-connected layers .

إضافة إلى ذلك يوجد tanh activation function في الطبقات الأخيرة و هو تابع مجاله بين -1 و 1 يقوم بتقريب قيم الدخل السالبة إلى قيم سالبة جداً و القيم الصفرية سيقوم بتقريبها لقيم قريبة من الصفر .



مقارنة بين Sigmoid و tanh

تحقيق الخوارزمية و تفسير النتائج :

1. تجهيز الداتا (Image Processing & Normalization) :

تم تغيير الـdatatype للصور من unit8 إلى float32 , وتخفيض مجال أرقام الصورة فكما نعلم أن كل صورة هي عبارة عن مصفوفة وبما أنها تحتوي على قيم RGB فإن مجالها بين 0 – 255 ولتسهيل العمل قمنا بـscale للقيم في الصورة و أصبحت بين 1- 0 وذلك بتقسيم الداتا على 255 , أما بالنسبة للخرج فهو مصفوفة تحوي رقم وحيد وهو رقم الصف التي تنتمي إليه الصورة فقمنا بتحويلها إلى شعاع من أصفار و واحدات وهو ما يعرف بـOne hot encoding .

2. تعريف الـModel :

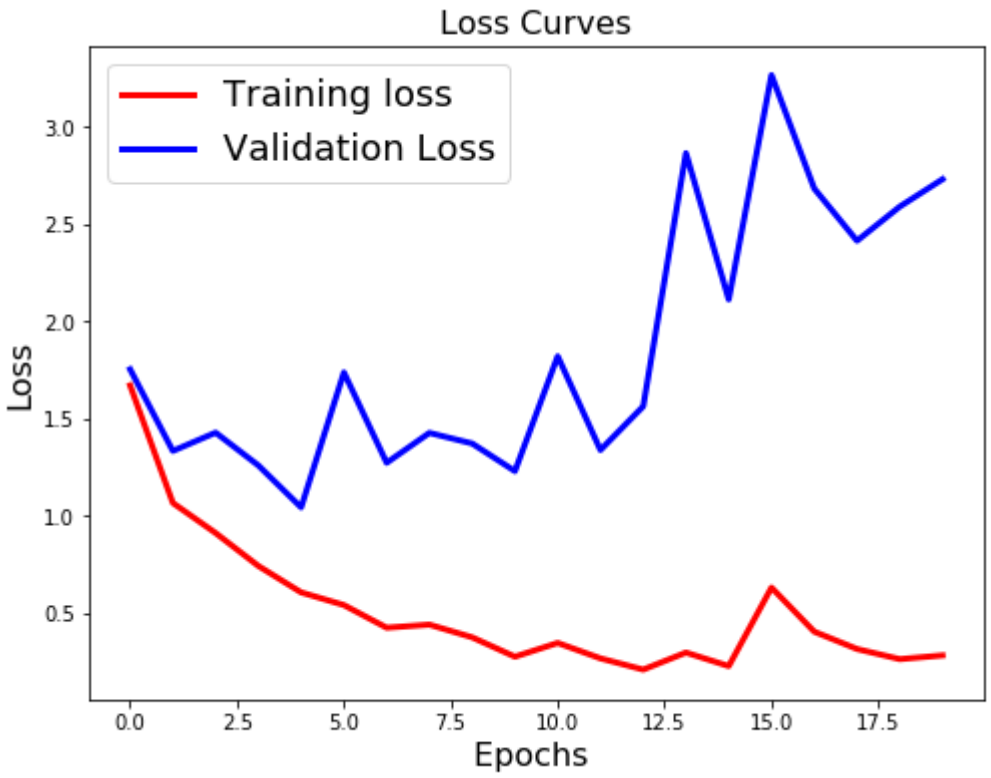
تم إنشاء الـmodel بالاستعانة بواجهة تؤمنها keras لتصميم الشبكات العصبونية ف قمنا باستدعاء التابع Sequential الذي يؤمن Model عبارة عن مكدس من الطبقات الخطية التي سنقوم بتصميمها تبعاً للبيبر المرفقة لخوارزمية الـAlexNet .

1) تم إنشاء أول طبقة convolution بـ(3*3) stride وليس (11*11) وذلك مراعاة لحجم الصور.

2) إضافة DropOut Layer
3) تم استخدام تابع الـReLu function كـ activation function بين الطبقات المخفية .
4) إضافة طبقتين تسمى الـDense Layer مع tanh activation function.
5) إضافة طبقة أخيرة من Dense ولكن باستخدام تابع يسمى Softmax function والذي يقوم بتوزيع الاحتمال على k فئة مختلفة خرجه شعاع يحوي احتمالية انتماء كل صورة لكل صف من الصفوف و الاحتمالية الأعلى ستكون التصنيف الصحيح .

3. نتائج التدريب :

تم تدريب الداتا على 50 عصر و بعد رسم الخطأ أو الـ Loss function تبين لنا أن قيمة الـvalidation loss أعلى بكثير من قيمة الـtraining Loss ولم يعطي الموديل نتائج مرضية وكانت الدقة 72%



تفسير الدقة السيئة للـ AlexNet:

تبين لنا أن الشبكة فضفاضة على الداتا سيت لذلك سنقوم بتجريب معمارية أخرى تقلل من عدد الطبقات المخفية و عدد العصبونات و طبقات الخرج وسنرى النتائج .

ثانياً : *(CNN) Convolution Neural Network*

لماذا الـ CNN وليس MLP ؟

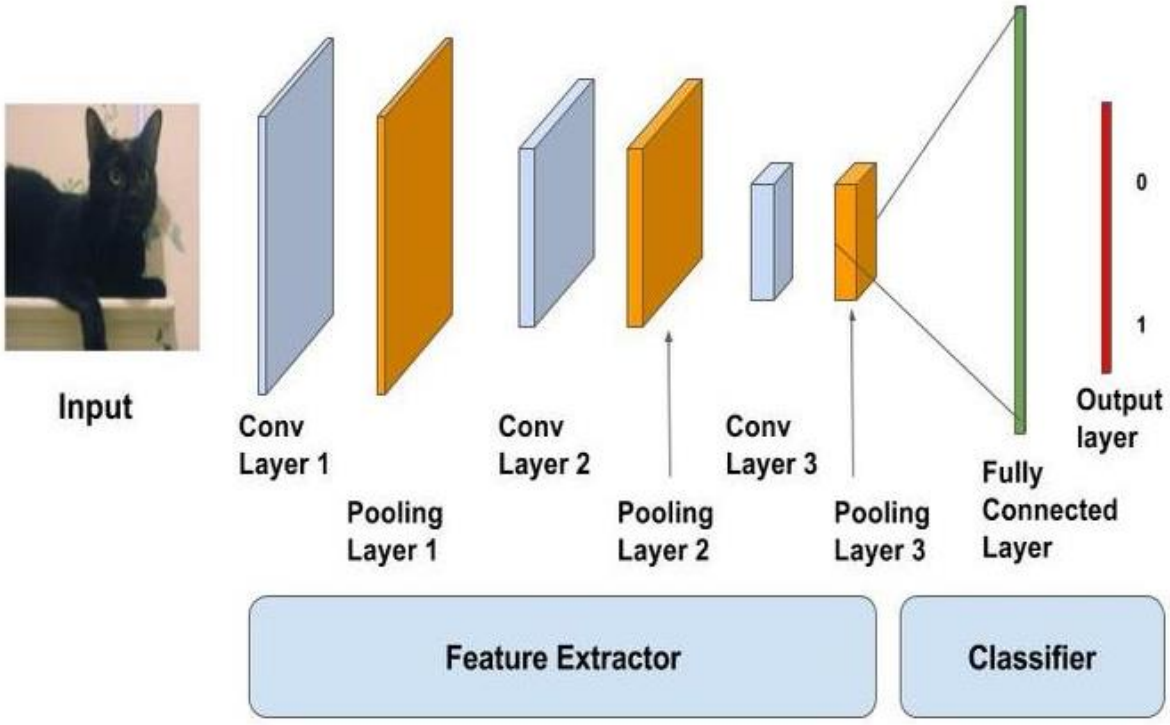
MLP تفشل في تصنيف الصور بدقة عالية نظراً لأنها لا تعطي نتائج صحيحة إذا كانت الصورة مثلاً مائلة إلى اليسار أو إلى اليمين بمعنى آخر تستطيع MLP تصنيف الصور المركزية بشكل صحيح في حين أن إزاحة بسيطة للغرض في الصورة قادرة على إفشال عمل الخوارزمية و هذا يمكن حله في الـCNN.

سبب آخر وهو أن الـMLP تقوم بزيادة الـparameters بشكل سريع جداً نظراً لترابط العقد بين الطبقات و بالتالي أصبح من الصعوبة بمكان أن نقوم ببناء شبكة عميقة deep Network .

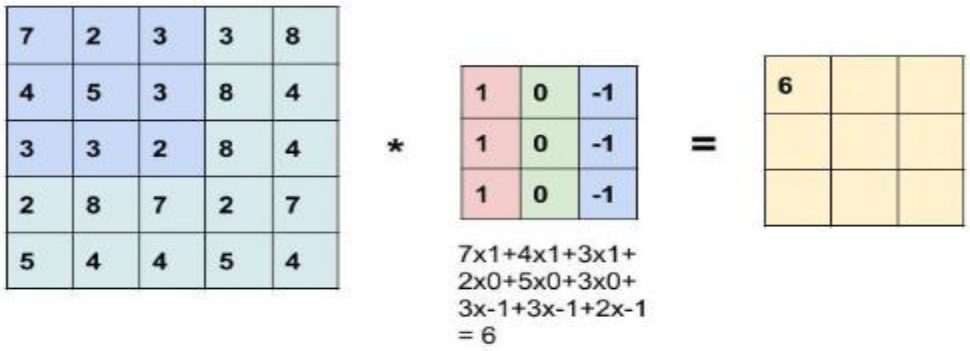
ما هي الـCNN ؟؟

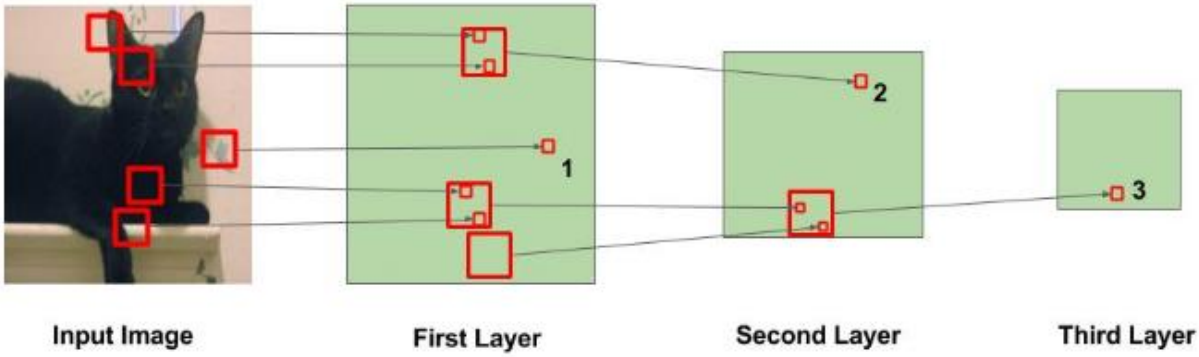
تلخص الـCNN بأنها Feed forward Neural Network

والتي تمثل بالصورة التالية :



أي هي مؤلفة من عدة طبقات من الـ Convolution layers تقوم باستخراج السمات من الصور عن طريق تمرير kernel بحجم نختاره على الصورة لاستخراج السمات منها كالتالي :

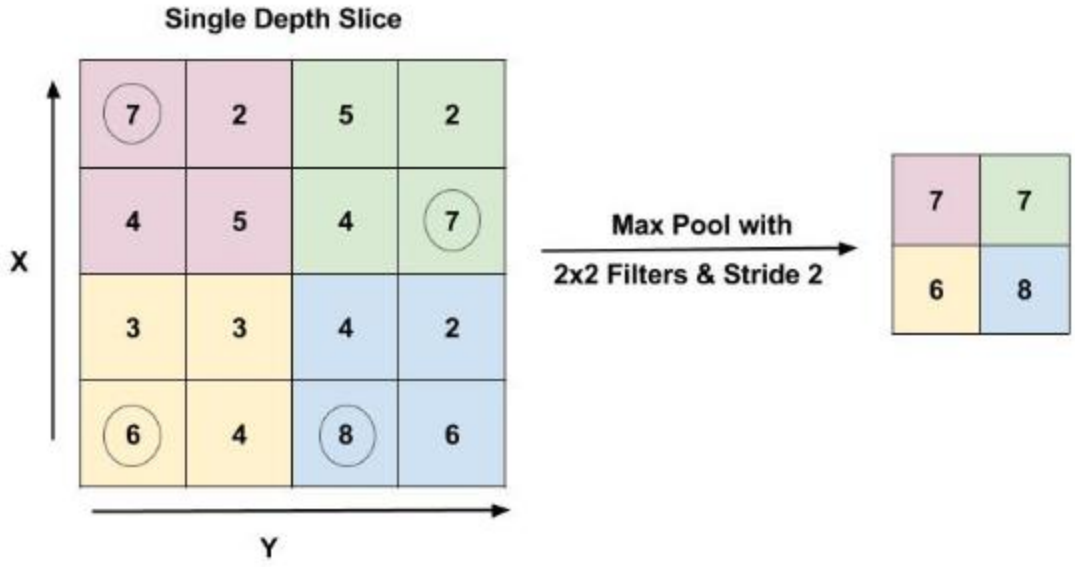




و بالتالي نرى أنه في الطبقات الأخيرة تقوم الطبقات باستخراج أهم السمات وتتجاهل السمات الغير مهمة .

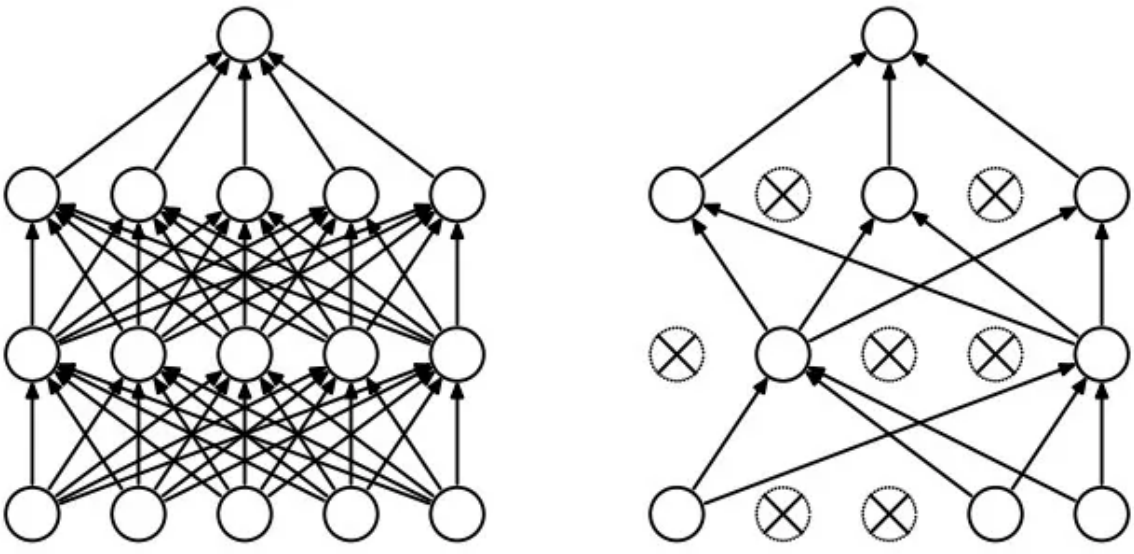
وظيفة الـmax pooling layer :

تطبق فوراً بعد طبقات الـconvolution لتقليل الـspatial size أي تقليل طول وعرض الصورة وليس العمق مما يؤدي إلى تقليل عدد الباراميترز في الـ Model و كما نعلم استخدام parameters أقل يقلل من احتمالية حدوث overfitting .



مشكلة أخرى نحلها هنا هو تقليل الـOverfitting عن طريق استخدام Dropout layer بعد كل fully-connected layer .

طبقة الـDropout لها احتمالية p مرتبطة بها و بكل عصبون , عشوائياً يتم الانتقال بين العصبونات عن طريق الاحتمال الأكبر p , كالتالي :



(a) Standard Neural Net

(b) After applying dropout.

الفكرة وراء الـDropOut Layer هو أن الطريقة هذه تمكننا من الاستفادة من السمات المهمة والتي تحوي على عصبونات لها احتمالية أكبر من غيرها وبالتالي سيقلل الـOverfitting و نحصل على سمات ذات معنى .

تحقيق الخوارزمية و تفسير النتائج :

1. تعريف الـModel :

تم إنشاء الـModel بـ 6 من طبقات الـConvolution و طبقة واحدة من طبقات الخرج التي تسمى fully-connected .

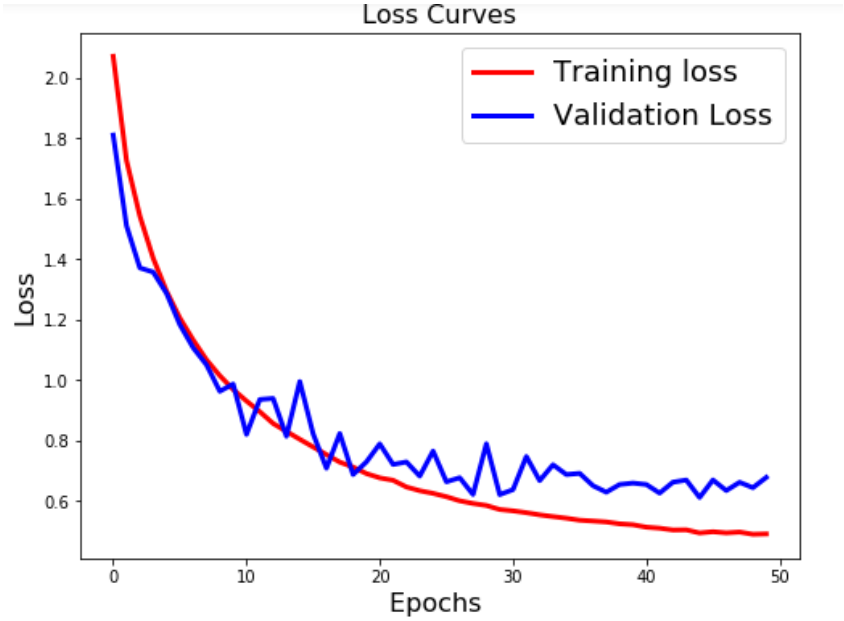
1) تم إنشاء أول طبقتين بـ32 filters و حجمها 3*3 , و الطبقة الثالثة لها 46 من فلاتر الصورة من نفس الحجم السابق
2) إضافة DropOut Layer و هي آلية لجلب الفيتشرات الأكثر أهمية و تقليل الـOverfitting بانتقاء العصبونات باحتمال P أو عدم انتقائهم باحتمال P-1 الـ ratio هنا هو الاحتمال.
3) تم استخدام تابع الـReLu function كـ activation function بين الطبقات المخفية .
4) إضافة طبقة تسمى الـDense Layer وهي التي ستتولى أمر تصنيف الصورة من بين 10 صفوف باستخدام تابع يسمى Softmax function والذي يقوم بتوزيع الاحتمال على k فئة مختلفة خرجه شعاع يحوي احتمالية انتماء كل صورة لكل صف من الصفوف و الاحتمالية الأعلى ستكون التصنيف الصحيح .

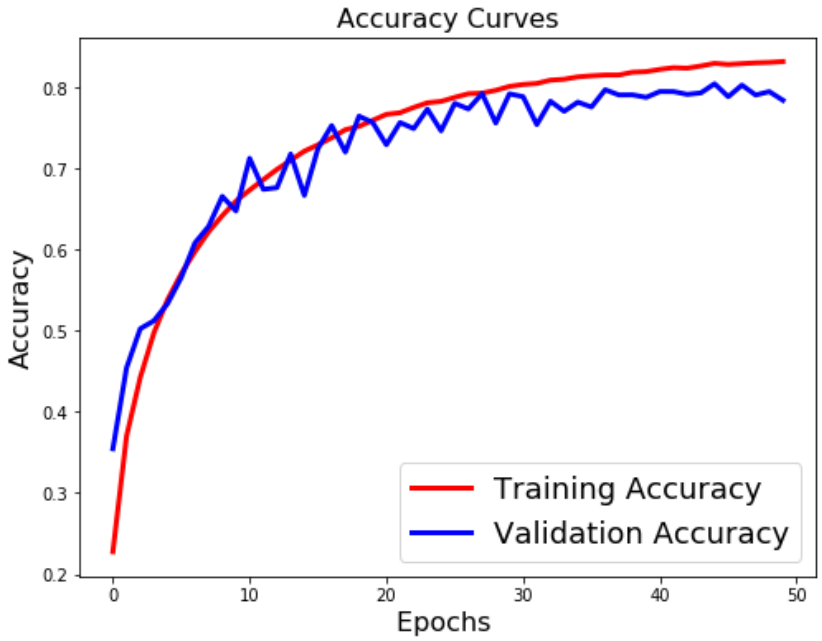
```
OPERATION          DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

      Input   #####     32   32    3
     Conv2D    \|/  -------------------      896     0.3%
       relu   #####     32   32   32
     Conv2D    \|/  -------------------     9248     3.3%
       relu   #####     30   30   32
MaxPooling2D   Y max -------------------        0     0.0%
               #####     15   15   32
    Dropout    | || -------------------        0     0.0%
               #####     15   15   32
     Conv2D    \|/  -------------------    18496     6.7%
       relu   #####     15   15   64
     Conv2D    \|/  -------------------    36928    13.4%
       relu   #####     13   13   64
MaxPooling2D   Y max -------------------        0     0.0%
               #####      6    6   64
    Dropout    | || -------------------        0     0.0%
               #####      6    6   64
     Conv2D    \|/  -------------------    36928    13.4%
       relu   #####      6    6   64
     Conv2D    \|/  -------------------    36928    13.4%
       relu   #####      4    4   64
MaxPooling2D   Y max -------------------        0     0.0%
               #####      2    2   64
    Dropout    | || -------------------        0     0.0%
               #####      2    2   64
    Flatten    ||||| -------------------        0     0.0%
               #####          256
      Dense    XXXXX -------------------   131584    47.7%
       relu   #####          512
    Dropout    | || -------------------        0     0.0%
               #####          512
      Dense    XXXXX -------------------     5130     1.9%
    softmax    #####           10
```

تمثيل للـ Model عن طريق keras2ascii

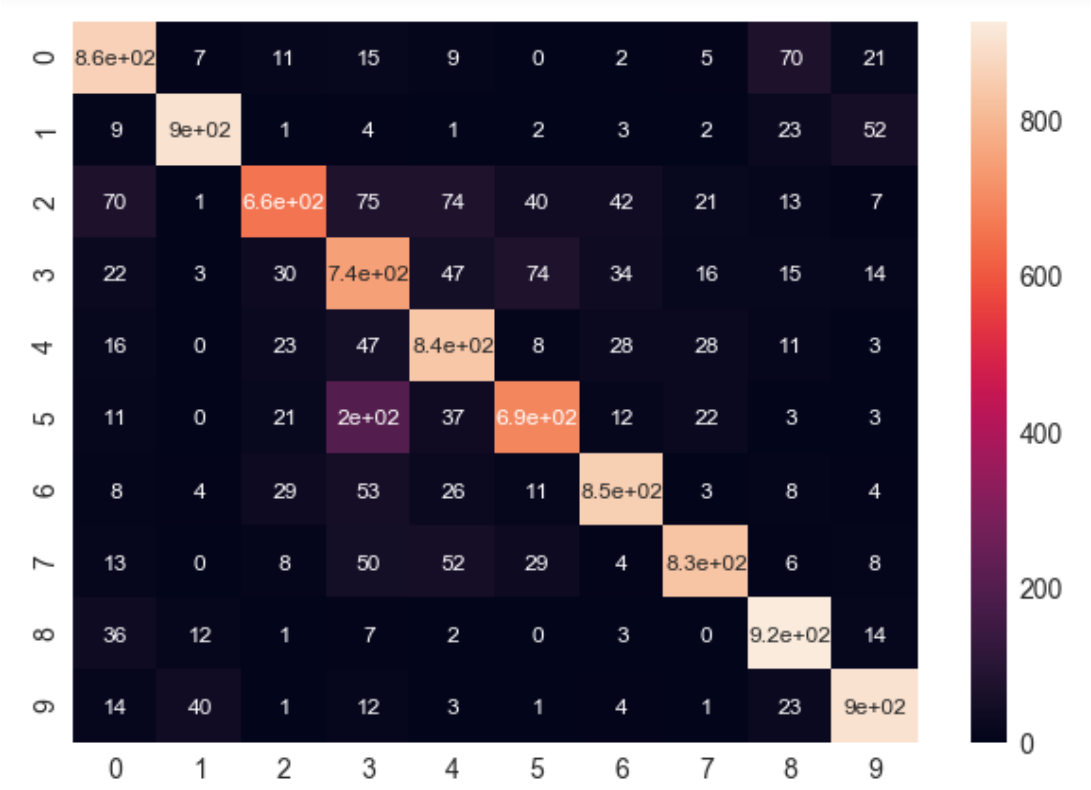
2. نتائج التدريب :

تم تدريب الداتا على 50 عصر و بعد رسم الخطأ أو الـ Loss function تبين لنا أن قيمة الـvalidation loss أعلى من قيمة الـtraining Loss وهذا يشير إلى أن الموديل لم يتمكن من التدريب على الداتا بشكل كافي و لم يتمكن من تعميم المسألة وهذا يدل على مشكلة الـ Overfitting .





نقطة ثانية و التي دلت على أن الـ Model السابق سيء هي الـConfusion Matrix :



نلاحظ من الصورة السابقة قيمة الخطأ من أجل كل صف وهذا يدل على أن الـModel غير قادر على التنبؤ بشكل صحيح .

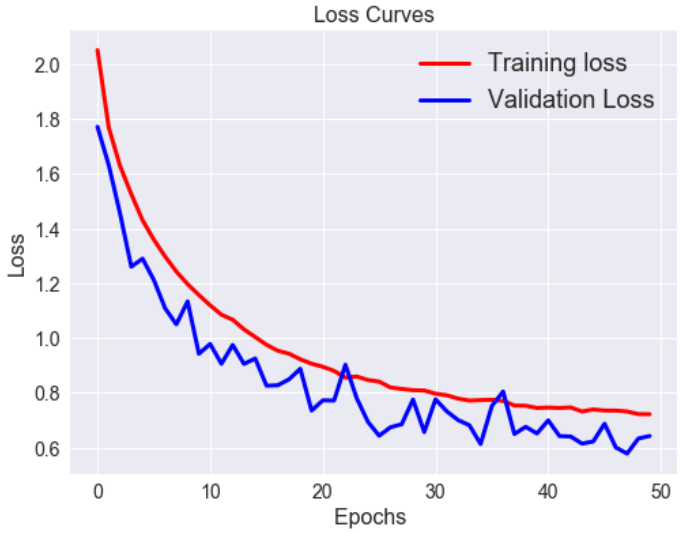
قمنا بإيجاد قيم الـ Precision & Recall & F-Measure & Specificity و هم شعاع يمثل القيمة لكل صف على حدى وليس قيمة وحيدة كما درسنا سابقاً في Binary Classification

```
In [14]: TP = np.diag(cm)
         TP
Out[14]: array([822, 950, 748, 686, 728, 722, 909, 806, 841, 804], dtype=int64)

In [15]: FP = np.sum(cm, axis=0) - TP
         FP
Out[15]: array([168, 183, 288, 388, 143, 227, 336, 112,  73,  66], dtype=int64)

In [16]: FN = np.sum(cm, axis=1) - TP
         FN
Out[16]: array([178,  50, 252, 314, 272, 278,  91, 194, 159, 196], dtype=int64)

In [17]: num_classes = 10
         TN = []
         for i in range(num_classes):
             temp = np.delete(cm, i, 0)    # delete ith row
             temp = np.delete(temp, i, 1)  # delete ith column
             TN.append(sum(sum(temp)))
         TN
Out[17]: [8832, 8817, 8712, 8612, 8857, 8773, 8664, 8888, 8927, 8934]

In [19]: precision = TP/(TP+FP)
         recall = TP/(TP+FN)
         print ("precision for CNN = " , precision)
         print ("Recal for CNN = " , recall)

         precision for CNN =  [0.83030303 0.83848191 0.72200772 0.63873371 0.8358209  0.76080084
          0.73012048 0.87799564 0.92013129 0.92413793]
         Recal for CNN =  [0.822 0.95  0.748 0.686 0.728 0.722 0.909 0.806 0.841 0.804]

In [20]: F_measure = 2  * ((precision * recall )/ (precision + recall ))
         F_measure
Out[20]: array([0.82613065, 0.89076418, 0.73477407, 0.66152363, 0.77819348,
                0.74089277, 0.80979955, 0.84045881, 0.87878788, 0.85989305])

In [21]: specificity = TN / (TN+FP)
         specificity
Out[21]: array([0.98133333, 0.97966667, 0.968     , 0.95688889, 0.98411111,
                0.97477778, 0.96266667, 0.98755556, 0.99188889, 0.99266667])
```

لحل هذه المشكلة وبما أن الداتا غير كافية و ازدياد العصور لن يجدي في مشكلتنا هذه اقترحنا أن نقوم بتغيير في الداتا ليصبح الموديل قادر على التعميم أكثر وهذا بتطبيق data augmentation أي نقوم بتغيير الصورة الواحدة نقربها و نزيحها إلى اليسار و إلى اليمين و هكذا نقوم بتوليد صور عديدة من كل صورة في الداتا سيت ونقوم بتدريب الموديل على صور صعبة التصنيف نوعاً ما ليتمكن من التصنيف بشكل أسهل و أسرع على بيانات الاختبار .

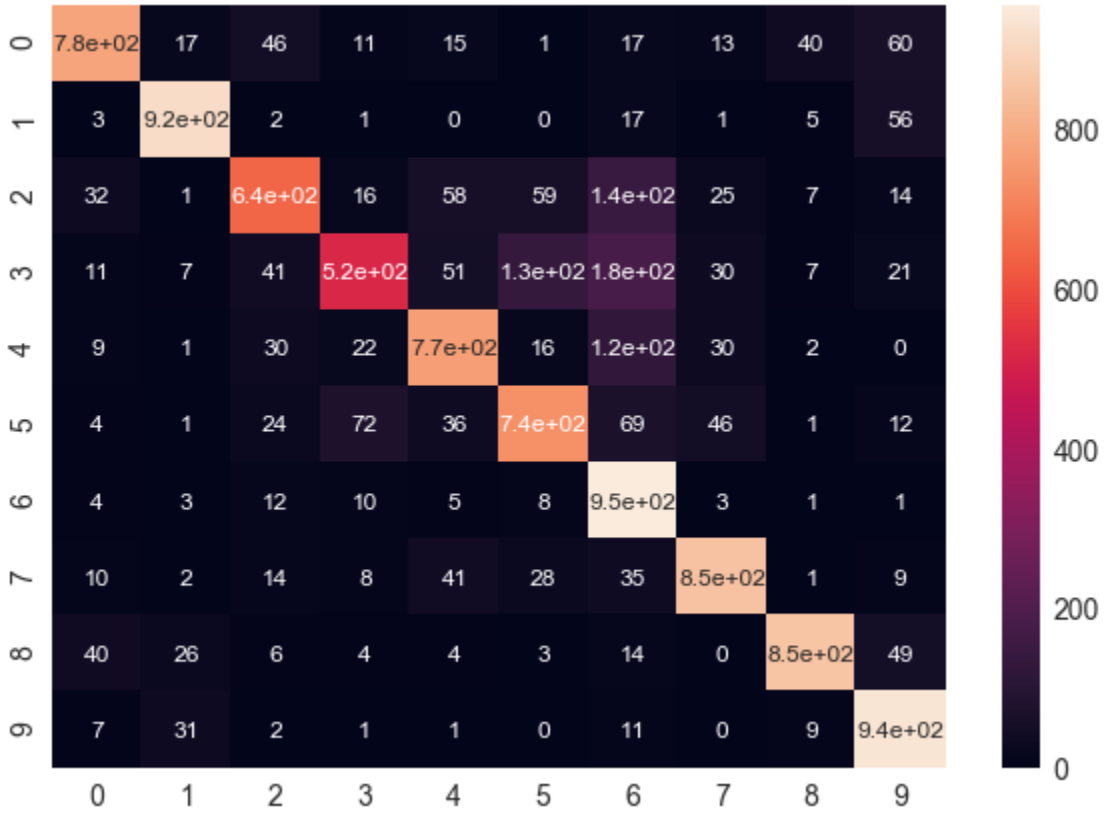
Data Augmentation

Loss curve after data augmentation



نلاحظ أن الـmodel أصبح قادر على التعميم و انخفاض الـvalidation Loss بهذا الشكل دليل على كفاءة الـ Model في عملية الـGeneralization .

وبمقارنة الـConfusion Matrix نلاحظ انخفاض ملحوظ في قيم أخطاء تصنيف الصور .



```
In [29]: TP = np.diag(cm)
         TP
Out[29]: array([780, 915, 645, 519, 766, 735, 953, 852, 854, 938], dtype=int64)

In [30]: FP = np.sum(cm, axis=0) - TP
         FP
Out[30]: array([120,  89, 177, 145, 211, 247, 611, 148,  73, 222], dtype=int64)

In [31]: FN = np.sum(cm, axis=1) - TP
         FN
Out[31]: array([220,  85, 355, 481, 234, 265,  47, 148, 146,  62], dtype=int64)

In [32]: num_classes = 10
         TN = []
         for i in range(num_classes):
             temp = np.delete(cm, i, 0)    # delete ith row
             temp = np.delete(temp, i, 1)  # delete ith column
             TN.append(sum(sum(temp)))
         TN
Out[32]: [8880, 8911, 8823, 8855, 8789, 8753, 8389, 8852, 8927, 8778]

In [34]: precision = TP/(TP+FP)
         recall = TP/(TP+FN)
         print ("precision for CNN = " , precision)
         print ("Recal for CNN = " , recall)
         precision for CNN =  [0.86666667 0.91135458 0.78467153 0.78162651 0.78403275 0.74847251
          0.60933504 0.852      0.92125135 0.80862069]
         Recal for CNN =  [0.78  0.915 0.645 0.519 0.766 0.735 0.953 0.852 0.854 0.938]

In [35]: F_measure = 2  * ((precision * recall )/ (precision + recall ))
         F_measure
Out[35]: array([0.82105263, 0.91317365, 0.70801317, 0.62379808, 0.77491148,
                0.74167508, 0.74336973, 0.852     , 0.88635184, 0.86851852])

In [36]: specificity = TN / (TN+FP)
         specificity
Out[36]: array([0.98666667, 0.99011111, 0.98033333, 0.98388889, 0.97655556,
                0.97255556, 0.93211111, 0.98355556, 0.99188889, 0.97533333])
```

ملاحظة :

بما أن مسألتنا MultiClassification فنحن أمام خيارين لطباعة الـROC Curve إما أن نقوم بتحويل المسألة لـBinary Classification Problem كما فعلنا في الجلسة السابعة من محاضرات العملي لأن sikit-learn لا تؤمن طباعة Multi Class Roc Curve أو نقوم باستيردا مكتبة من R Language تسمى Proc تقوم بطباعة الـROC Curve لنا إذ لم تفلح أي طريقة أخرى .

ثالثاً : GoogLeNet / Inception

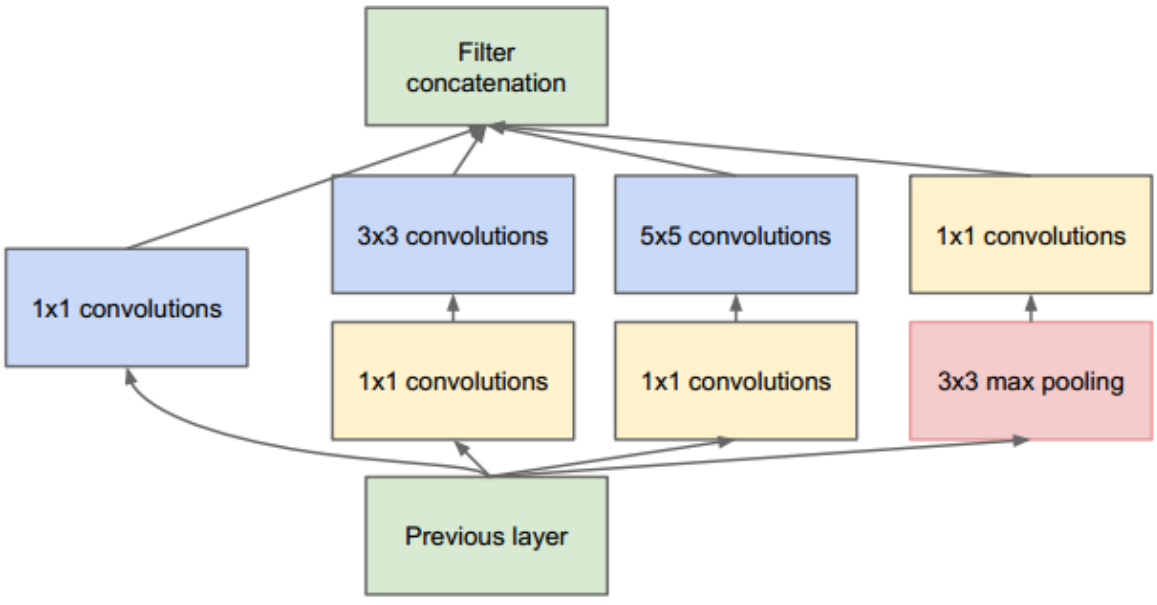
تعتمد فكرة هذه الشبكة على أنه بفرض هناك شبكة عصبونية لها 512 خرج مرتبط بالدخل وهو أيضاً 512 دخل .. إنّ أغلب الـactivations في هذه الشبكة غير مهمة (ذي القيمة صفر) أو مكررة نظراً للارتباطات بينهم ؛ وهذا يعطي أن أكثر معمارية ذات كفاءة عالية ستحتوي على sparse connection بين الـactivations .

أي معمارية الـGoogLeNet أو Inception module هو عبارة عن sparse CNN , وبما أن عدد صغير من العصبونات سيكون فعال في عملية انتقاء العصبونات المهمة في عملية التدريب فهذا سيؤدي إلى أن عرض و عدد الـ convolutional filters التابعة لـ kernal size محدد سيظل صغير , و أيضاً تستخدم الشبكة الطي بأحجام مختلفة لالتقاط التفاصيل التابعة لمقاييس متنوعة (1*1 , 3*3 , 5 * 5) .

نقطة مهمة أخرى في هذه المعمارية هو وجود ما يسمى bottleneck layer أي طبقة مؤلفة من

(1*1 convolutions) والتي تساعد بالحد بشكل كبير من العمليات الحسابية الكبيرة.

تغيير آخر في هذه الشبكة هو أنها استعاضت عن الـfully-connected layers في نهاية الشبكة بطبقة تسمى average pooling والتي تحتوي متوسط قيم القنوات السابقة , وتتموضع بعد آخر convolutional layer . وهذا يقلل من عدد الباراميترات الكلية للشبكة العصبونية . وكما أنها أسرع من الشبكات السابقة و الأكثر دقة بينهم .



رابعاً VGG 16 & VGG 19

هذه المعمارية طورتها Oxford وهي تطوير على AlexNet باستبدال الفلتر الكبير الحجم للـkernal (11*11)في أول طبقة وثاني طبقة بـ(3*3) Kernal size وكان يؤيد فكرة أنه إذا كانت المعمارية تكدس فلاتر للـkernal صغيرة الحجم فهذا يعطي أداء أعلى من حجم كبير للـkernal يستخدم مرة واحدة لأن الطبقات العديدة الغير خطية تزيد من عمق الشبكة وهذا يسمح لها بالتعلم على سمات أكثر تعقيداً و يقلل من الكلفة .

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

معمارية الـVGG

نلاحظ تطبيق أكثر من طبقة بنفس الـkernal size أكثر من مرة وهذا يؤدي إلى استخراج سمات مهمة ومعقدة , كما أن الشبكة تحوي على 3 fully-connected layers ودخلها صورة بحجم 224 * 224 .

خامساً: ResNet

كما رأينا إن زيادة عمق الشبكة من شأنه أن يزيد الدقة غالباً ولكن سنتعرض لمشكلة الـ vanishing gradient كلما كانت الشبكة أعمق ومشكلة ثانية هي الباراميترات الضخمة للشبكة فإن كل الطبقات المخفية في الشبكة العصبونية العميقة تحوي على باراميترات هائلة و حل هذه الخوارزمية قامت هذه الشبكة على اقتراح المعمارية التالية :

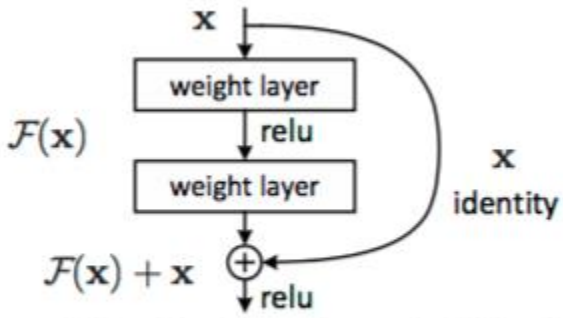


Figure 2. Residual learning: a building block.

ResNet (Residual Network)

يقوم الـModel على عمل identity map بين الدخل و الطبقات المخفية إذ تقوم الطبقات المخفية على التدريب على ما تبقّى من الدخل مع ناتج الـReLu activation function للطبقة السابقة .

هذه الخوارزمية حققت الدقة الأعلى بين كل الخوارزميات السابقة في مسابقة ImageNet .

Transfer Learning

هو آلية في علم تعلُّم الآلة حيث أن الـmodel يُطَوّر من أجل وظيفة معينة ويعاد استخدامه لتوفير الوقت و الحصول على أعلى أداء ممكن حيث نقوم باستيراد Pre-trained models قامت بالتدريب على داتا سيت شبيهة بالداتا سيت خاصتنا و استيراد أوزان الشبكة .

لماذا قررنا استخدام transfer learning ؟

نلاحظ من النتائج التي توصلنا إليها في تمثيل شبكة الـAlexNet & CNN من الـscratch والتدريب عليها أننا لم نحصل على دقة تتجاوز 80 ولرفع الدقة يجب أن نفكّر في طريقة أخرى و اتِّباع منهج آخر .

Feature Extraction From Keras

تم استخراج سمات الصور باستخدام الخوارزميات المذكورة أعلاه , باستخدام pre-trained models from keras و ذلك بحذف الطبقة الأخيرة و تطبيق الشبكة على الصور باستخدام الأوزان المرفقة بـ pre-trained models .

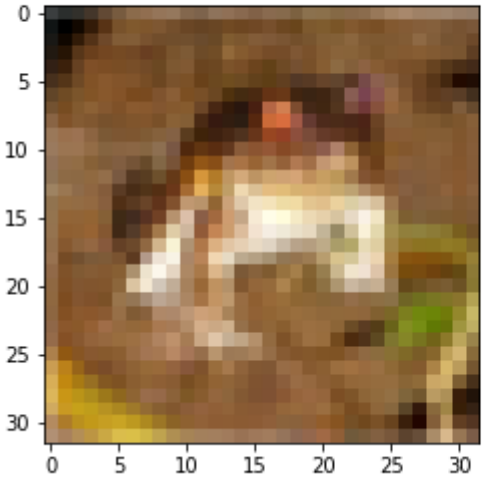
❖ الـpre-trained models دُربت على ImageNet Data set وهي مجموعة صور شبيهة بـCIFR10 ولكنها أكثر حجماً و عدداً و تحتوي 1000 كلاس للتصنيف .

تم استخراج السمات باستخدام VGG16 & incepv3 & VGG19

مثال :

استخراج السمات من الصورة التالية :

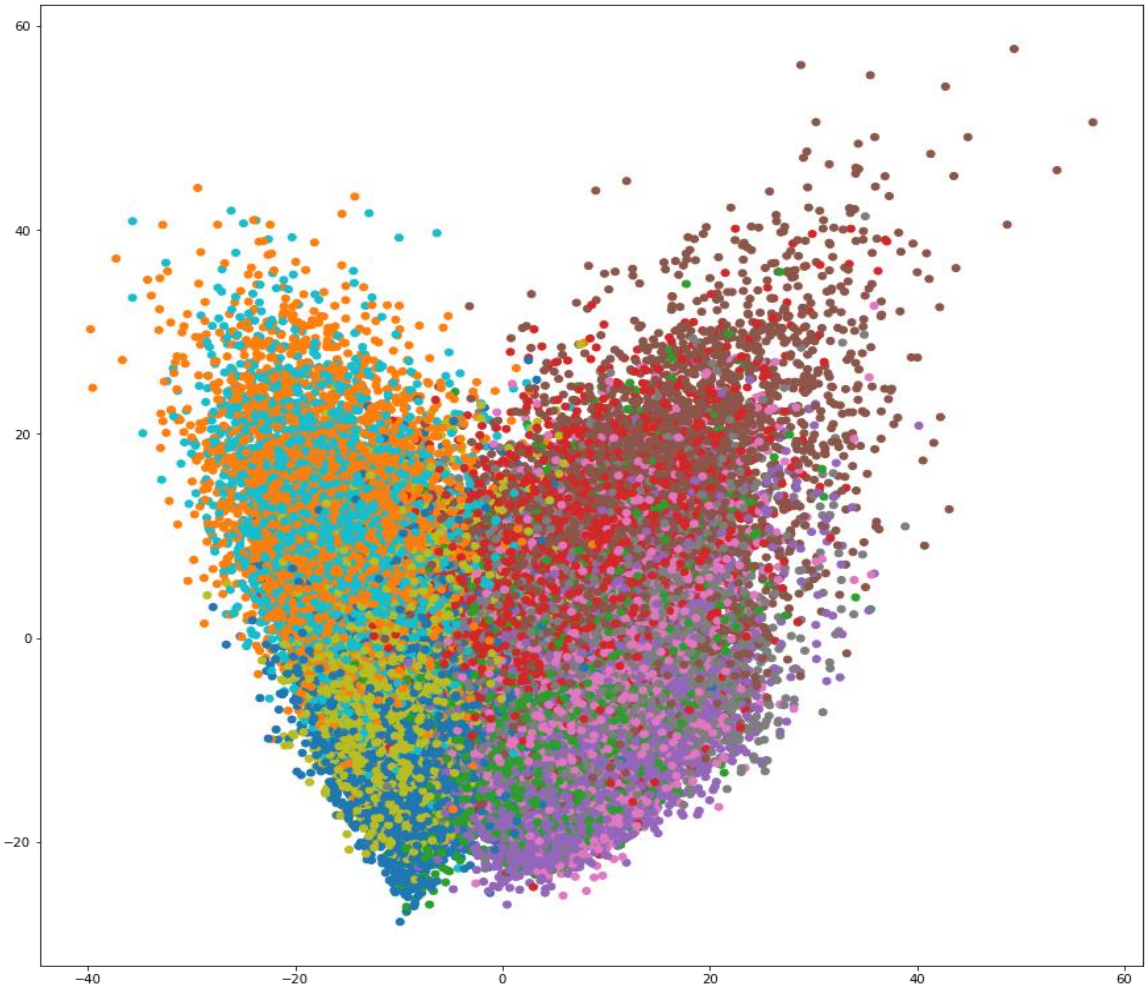
```
(50000, 32, 32, 3) (50000,) (10000, 32, 32, 3) (10000,)
```



Ten first features of X_train[0] (see figure above, with the frog)

```
'incv3':
    array([ 0.27140674,  0.52113253,  0.24185033,  0.20671989,  0.30322665,
            0.34751841,  0.36889949,  0.25563717,  0.27236217,  0.16386187], dtype=float32)
'vgg16'
    array([ 5.26894188,  1.65981817,  0.86902153,  6.91321135,  0.55422449,
            1.46428168,  2.49574828,  1.16215992,  3.32762074,  3.14453959], dtype=float32)
'vgg19'
```

تمثيل السمات باستخدام PCA على 2Dimentinal space كالتالي :

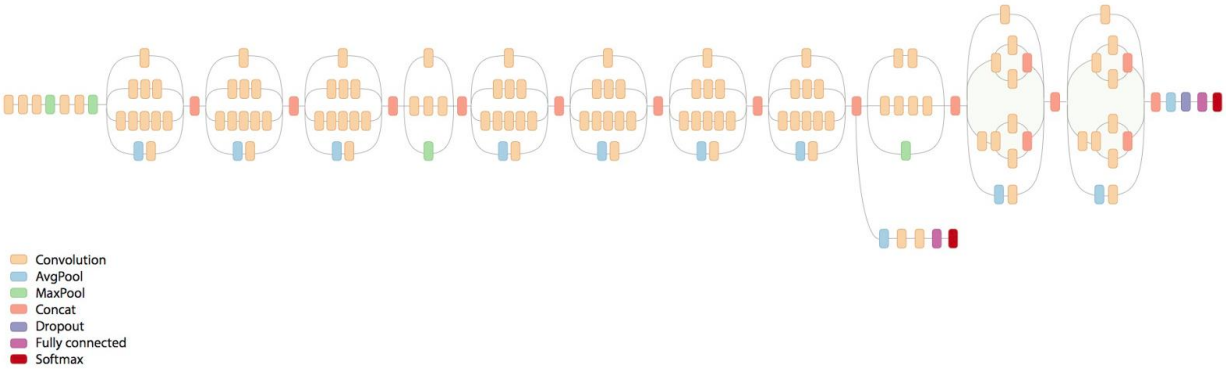


PCA For CIFR10 Pre-trained models

Feature Extraction From Tensor Flow

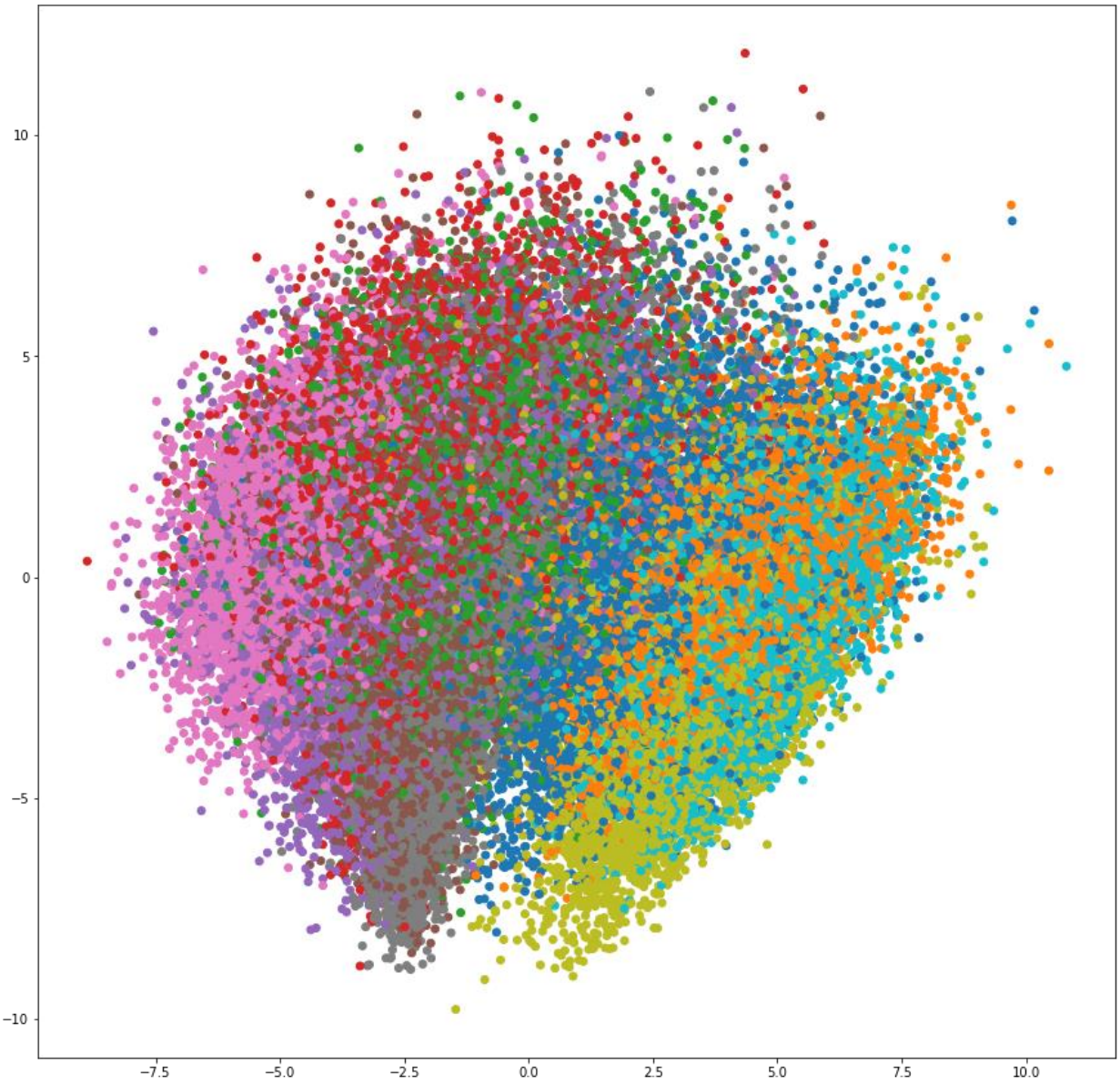
تم استخراج سمات الصور باستخدام Inception V3 pre-trained model

❖ الـpre-trained models دُربت على ILSVRC-2012-CLS



Inception v3 feature extraction

وكما فعلنا سابقاً بعد استخراج السمات من الموديل قمنا بتمثيل الفيتشرت بـPCA :



PCA For Inceptionv3 TensorFlow

Load Features and make an SVM classifier

Fine-Tuning the parameters for SVM classifier:

من أجل معرفة الباراميتر الأحسن لكل فيتشرات الخوارزميات قمنا بالتجريب على أكثر من قيمة لـ c و اعتمدنا على القيمة التي تحقق أعلى دقة من أجل كل خوارزمية كالتالي :

	Model				
C	vgg16-keras	vgg19-keras	resnet50-keras	incv3-keras	Inception_v3
0.0001	8515	8633	9043	7244	8860
0.001	8528	8654	9158	7577	9005
0.01	8521	8644	9130	7604	9061
0.1	8519	8615	9009	7461	8959
0.5	7992	8014	8858	7409	8834
1.0	8211	8225	8853	7369	8776
1.2	8156	8335	8871	7357	8772
1.5	8172	8022	8852	7318	8762
2.0	7609	8256	8870	7281	8736
10.0	7799	7580	8774	7042	8709

بعد تنفيذ الـSVM على الـC الأفضل من أجل كل خوارزمية كانت النتائج كالتالي :

```
model =  resnet50-keras
X_training size = (50000, 2048)
features=  resnet50-keras, C=0.001000 => score= 9158
model =  Inception_v3
X_training size = (50000, 2048)
features=    Inception_v3, C=0.010000 => score= 9061
model =  vgg16-keras
X_training size = (50000, 512)
features=     vgg16-keras, C=0.000100 => score= 8515
model =  vgg19-keras
X_training size = (50000, 512)
features=     vgg19-keras, C=0.001000 => score= 8654
model =  incv3-keras
X_training size = (50000, 2048)
features=     incv3-keras, C=0.001000 => score= 7577
```

Result of the SVM Classifier

نلاحظ أن أعلى دقة حققتها فيتشرات المستخرجة عن طريق الـResNet واستطعنا أن نقوم بإيجاد أفضل دقة ممكنة .

بطباعة الـ Confusion matrix يتبين لدينا الفرق الملحوظ بينها وبين المصفوفات السابقة و هذا يدل على أن البيانات قد صنفت بشكل أصح .

```
Confusion matrix:
 [[935   3   6   4   6   0   1   6  33   6]
 [  6 944   0   1   0   0   4   3   9  33]
 [ 20   0 878  28  34   9  21   6   2   2]
 [  6   3  19 834  22  68  23  14   4   7]
 [  3   0  23  10 918   8  14  18   3   3]
 [  2   0  11  71  18 875   6  14   3   0]
 [  4   1  12  19   6   2 954   0   2   0]
 [  6   1   6   9  32  17   0 929   0   0]
 [ 23   2   4   1   1   1   0   1 959   8]
 [ 11  45   1   2   0   0   1   1   7 932]]
['plane', 'auto', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

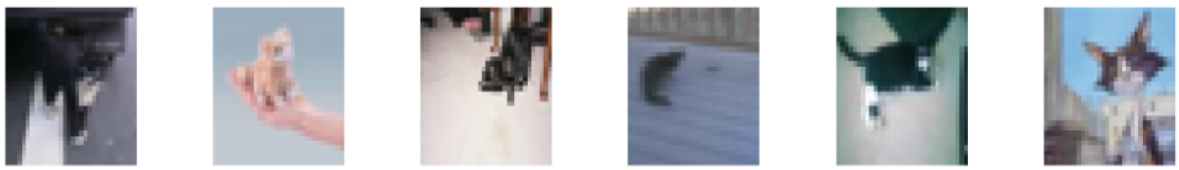
Confusion Matrix for SVM

لاختبار الموديل قمنا بالاستعانة بالـConfusion Matrix لطباعة مثال من التنبؤ .

In [12]:
```
i,j = 3,0
img_idx = [ k for k in range(10000) if y_testing[k]==i and y_predictions[k]==j ]
print( 'We have, e.g., {c} {iname}s predicted to be {jname}'.format(\
      c=conf_matrix[i,j], iname=labels[i], jname=labels[j]) )
# print(img_idx)
```

```
We have, e.g., 6 cats predicted to be plane
```

In [13]:
```
_, data_testing = myutils.load_CIFAR_dataset(shuffle=False)
from matplotlib import pyplot as plt
%matplotlib inline

fig = plt.figure(figsize=(18,2));
for _i in range(conf_matrix[i,j]):
    a=fig.add_subplot(1,conf_matrix[i,j],_i+1)
    plt.imshow(data_testing[img_idx[_i]][0])
    plt.axis('off')
```



نرى هنا أن الـModel تنبأ بـ6 قطط على أنها طائرات وقمنا بطباعة الصور عند الـindex المعين .

KNN Nearest Neighbor Classifier

k-nearest neighbors classifier

Let us note that simple kNN classifier (with k=10), trained with 5000 training features (CNN codes from Inception_v3) gives **83.45%** accuracy on whole 10000 testing images.

Remark that computing predictions with this classifier is very complex and it is not recommended for classificcation of images.

Here is the code to compute the score on testing dataset.

```
from sklearn.neighbors import KNeighborsClassifier
kNN_clf = KNeighborsClassifier(n_neighbors=10)
kNN_clf.fit(X_training, y_training)
print( 'Classification score = ', kNN_clf.score( X_testing, y_testing ) )
# Classification score =  0.8345
```

Logistic Regression

Logistic regression

Finally we used Logistic regression with default parameters. We trained the model with all the training data and obtained **90.37%** accuracy on testing dataset.

In [27]:
```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_training, y_training)
print( 'Linear regression accuracy = ', clf.score( X_testing, y_testing ) )
```

```
Linear regression accuracy =  0.9037
```

ملاحظة :

بما أن الداتا كبيرة نوعاً ما و العمل عليها يتطلب هاردوير بمواصفات عالية لاختصار الوقت و ليتسنى لنا التعديل على الـParameters بشكل أسرع تم تدريب الداتا سيت على gaming laptop مواصفات الهاردوير :

Alienware 15 r3 Cpu corei7 77oohq Gpu nvedia gtx 1060 Ram 16GB

مراجع :

1. AlexNet Paper : http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
2. GoogLeNet Paper : https://arxiv.org/pdf/1409.4842.pdf

https://hacktilldawn.com/2016/09/25/inception-modules-explained-and-implemented/

3. ResNet Paper : https://arxiv.org/abs/1512.03385
4. VGG paper :https://arxiv.org/pdf/1409.1556.pdf
5. Image Classification on CIFR10 Neural Network vs Support Vector Machines https://chahatdeep.github.io/docs/NNvsSVM.pdf
6. How to make feature extraction and use it in SVM : https://www.kaggle.com/craigglastonbury/using-inceptionv3-features-svm-classifier
7. Guide to fine-tuning deep learning models in keras : https://flyyufelix.github.io/2016/10/03/fine-tuning-in-keras-part1.html

https://flyyufelix.github.io/2016/10/08/fine-tuning-in-keras-part2.html

8. cifar-10-competition-winners-interviews-with-dr-ben-graham-phil-culliton-zygmunt-zajac : http://blog.kaggle.com/2015/01/02/cifar-10-competition-winners-interviews-with-dr-ben-graham-phil-culliton-zygmunt-zajac/