التعلم العميق Deep Learning

- **الدرس الأول : التعلم العميق و الشبكات العصبية**
  - الأسبوع الأول : مقدمة للتعلم العميق
  - الأسبوع الثاني : أساسيات الشبكات العصبية
  - الأسبوع الثالث : الشبكات العصبية المجوفة
  - الاسبوع الرابع : الشبكات العصبية العميقة
- **الدرس الثاني : تطوير الشبكات العميقة : المعاملات العليا**
  - الأسبوع الأول : السمات العملية للتعلم العميق
  - الأسبوع الثاني : الحصول علي القيم المثالية
  - الأسبوع الثالث : ضبط قيم الشبكات العميقة
- **الدرس الثالث : هيكلية مشاريع الـ ML**
  - الأسبوع الأول : استراتيجيات الـ ML - 1
  - الأسبوع الثاني : استراتيجيات الـ ML - 2

- **الدرس الرابع : الشبكات العصبية الملتفة CNN**
  - الأسبوع الأول : أساسيات الشبكات العصبية الملتفة
  - الأسبوع الثاني : حالات عملية من الشبكات العصبية الملتفة
  - الأسبوع الثالث : التعرف علي الأشياء
  - الاسبوع الرابع : التعرف علي الوجه
- **الدرس الخامس : الشبكات العصبية المتكررة RNN**
  - الأسبوع الأول : مفهوم الشبكات العصبية المتكررة
  - الأسبوع الثاني : المعالجة اللغوية الطبيعية NLP
  - الأسبوع الثالث : نماذج التتابع

درس 4: الشبكات العصبية الملتفة CNN

الأسبوع الثاني : حالات عملية من الـ CNN

==========

ما سنقوم به الآن هو قراءة و تحليل عدد من النماذج المكتوبة بالفعل للشبكات الملفتة CNN و ذلك للتعلم منها . .

و كما درسنا سابقا , فيمكن فحص و تحليل نموذج ناجح لأحد التطبيقات (تصنيف الصور) , لاستخدام نفس الفكرة لكن مع التعديل في تطبيق آخر (سيارة ذاتية القيادة)

فيه أنواع من الشبكات التقليدية زي :

- LeNet-5
- AlexNet
- VGG

وفيه شبكات أعمق زي "ResNet" اللي هي اختصار Convolutional Residual NN واللي عدد طبقاتها بيوصل لـ 152 طبقة

-

نبدأ بالـ LeNet

هي شبكة بسيطة و قديمة , و مخصصة لقراءة الصورة الصغيرة و باللون الأبيض و الأسود , لقراءة الحروف فيها , وتكون أبعادها غالبا 32 في 32 , مثل هذه الصورة :

نقوم بعمل فلتر عمقه 6 و بقيمة 5*5 و الخطوة تساوي 1, تتحول لمكعب 28*28*6 . .

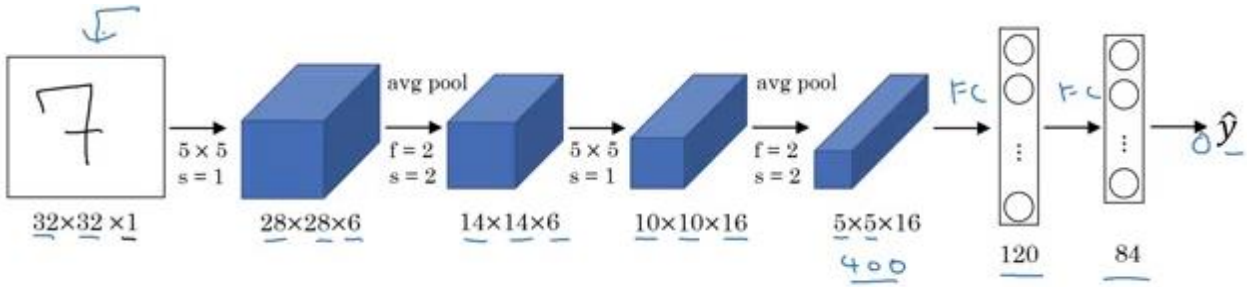
بعدها نعمل pooling حيث f=2 , s=2 فالأبعاد هتقل للنص و تكون 14*14*6

بعدها نعمل فلتر بعمق 16 , وابعاده 5*5 و بخطوة 1 , تتحول لمعكب 10*10*16

بعدها نعمل pooling تاني حيث f=2 , s=2 فالأبعاد ستقل للنص و تكون 5*5*16 , وستساوي 400 قيمة . .

(لاحظ ان هذه الشبكة تم تصميمها في التسعينات , فلم يكونو يستخدمو الحشو, لذا الابعد قلت و العمق زاد)

ثم نقوم بعمل اتصال كامل fully connecting مع طبقة NN بـ 120 خلية (حيث تتصل كل قيمة من الـ 400 بكل خلية من الـ 120) , ثم طبقة تالية بـ 84 قيمة , ثم المخرج y , والتي ستكون بعشر قيم محتملة (من 0 إلي 9) لأن هذا تصنيف لقراءة الأرقام



وبعدها قامو بعمل تحديث أن يكون المخرج بتكنيك softmax بعشر مخارج .

ولاحظ أن هذه الشبكة لها 60 ألف parameter وهو رقم قليل نوعا , فالشبكات الحديثة يتجاوز عدد الـ parameters فيها 10 مليون . .

و أيضا لاحظ أن هذه الشبكة كلها ازدادت عمقا , كلما قلعدد الصفوف و الأعمدة فيها , وزاد العمق (لعدم استخدام الحشو) .

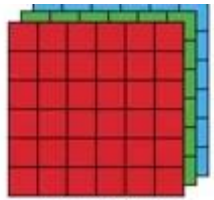
و يكون التكوين العام لها :

شبكة ملتفة + pooling + شبكة ملتفة + pooling + طبقة FC+ طبقة FC + المخرج

و غالبا ما يستخدم هذا النوع من الشبكات دالة سيجمويد او تانش , حيث أن دالة ReLU لم تكن قد ابتكرت بعد

النوع الثاني من الشبكات التقليدية , هو ما يسمي AlexNet

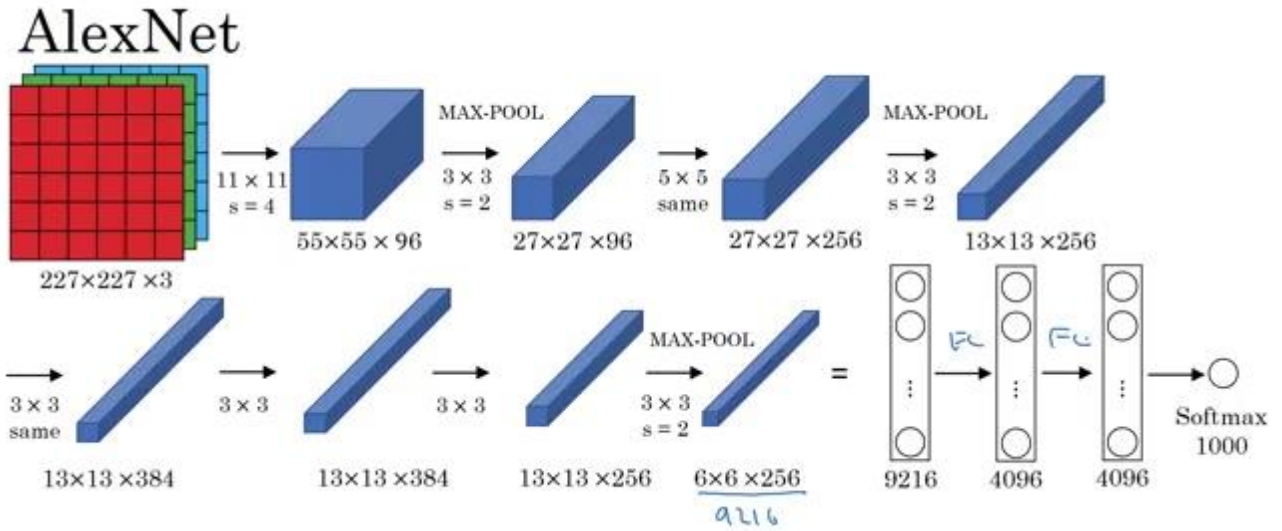
والتي تمت تسميتها علي اسم أليكس كريزيفسكي , الذي قام ابتكر فكرتها .



ونبدأ هنا بصورة بأبعاد 3*227*227 , حيث هي صورة ملونة .

ثم نقوم بعمل فلتر بعمق 96 , وبأبعاد 11*11 و بخطوة 4 , فتتحول لمكعب 96*55*55 , ثم max pool بقيم 3*3 بخطوة 2 , فتتحول لمكعب 96*27*27 .

ثم فلتر 5*5 , ويكون باستخدام الحشو الذي يؤدي لنفس الأبعاد , فتتحول لـ 256*27*27 , ثم max pool 3*3 و خطوة 2 , تتحول لـ 256*13*13 , ثم فلتر بحشو مماثل 3*3 تتحول لـ 384*13*13 . ثم بتكرار العملية بأرقام مختلفة , تتحول لـ 256*6*6 , والتي تعني 9216 قيمة , ومنها ثلاث طبقات FC بقيم 9216 ثم 4096 ثم 4096 , ثم softmax بألف قيمة .



و مع تشابه هذا النموذج قليلا مع LeNet لكن هو أكثر عمقا , حيث الـ LeNet لها 60 ألف parameter بينما هنا 60 مليون , وهو ما يعني كفاءة أعلي , لكن وقت أكبر

و غالبا يستخدم هذا النوع دالة ReLU بدلا من سيجمود او تانش .

و قديما (حينما ابتكرو هذه الشبكة) بدأو باستخدام أكثر من GPU معا للمساعدة علي تسريع عملية المعالجة .

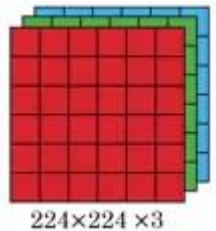
و وقتها تم استخدام تكنيك يسمي LRN وهو اختصار (local response normalization) و والذي يعني , أن نمسك بلأحد المكعبات , ليكون 13*13*256 , ثم نمسك بكل قيمة من قيم الوجه الأمامي (13*13) , ونأخذ كل القيم الـ 256 في ذات العمق لها ,ونقوم بعمل normalization لها كما في الصورة :

وكانت الفكرة تقوم علي جعل عملية حساب الارقام اسهل , حينما تكون الارقام قد تمت تسويتها , لسهولة عمل ارقام activation في الـ NN

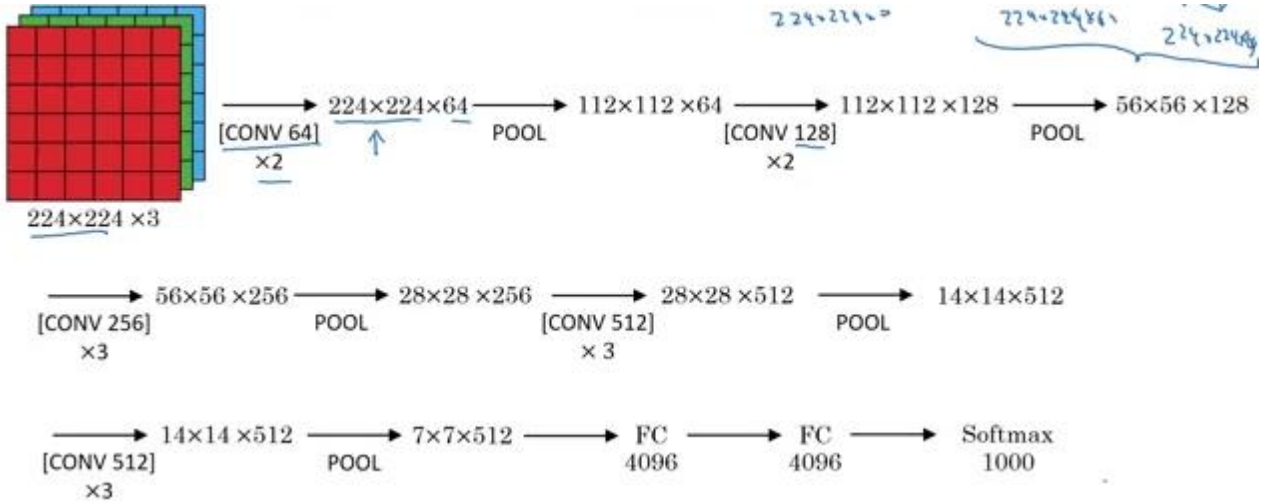
لكن فيما بعد , وجد الباحثون أن هذه الخطوة لا تقوم بتسريع المعالجة , فتم ايقاف استخدامها .

ننتقل للنوع الثالث وهو : VGG او يسمي VGG-16

و قامت فكرتها , علي تجنب التعامل مع عدد كبير من الـ parameters مثل المستخدمة في أليكسنت , فتم التركيز فقط علي فلتر 3*3 , وبخطوة 1 , وان يتم استخدام maxpool 2*2 بخطوة 2 , وهي التي تقوم بعمل تبسيط كبير في الشبكة , وتقليل عدد قيمها . . .



و نقوم بهذه التطبيقات بتلك الأرقام , مع مراعاة أن الفلتر دائما 3*3 و الخطوة 1



و نلحظ أن عدد الـ parameters فيها يزيد عن 130 مليون , وهو رقم كبير حتي بمقاييس العصر الحالي , كما نري أن مع كل خطوة فإن الطول و العرض يقل للنصف , بينما العمق يزيد للضعف , وهو من تأثير استخدام الفلتر و الماكس بهذه الأرقام .

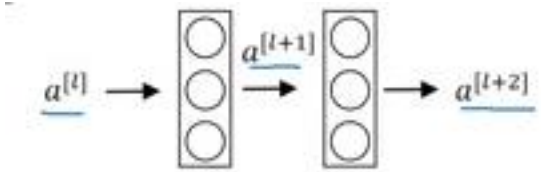
كما أن هناك نسخة أعمق منها هي VGG-19 و هي أكبر و أعمق , لكن ليس لها زيادة كبيرة في الفائدة عن هذه الشبكة .

-

مع الشبكات العميقة ذات الطبقات العديدة , تكون لدينا مشكلة التدريب مع عدد هائل من الأوزان , لذا تم ابتكار فكرة الشبكات المتبقية Residual NN والتي تتمكن من تدريب عدد كبير من الـ Parameters بشكل سريع

بداية علينا تذكر المسار الطبيعي الذي نتعامل معاه بين الطبقات . .

لو كان لدينا طبقتين بهذا الشكل :



فهناك مدخل لطبقة الأول a(l) يقوم بإخراج a(l+1) والتي تدخل الطبقة الثانية و تخرج a(l+2) . .

والتي ستكون بالمسار:

حيث نقوم بتطبيق المعاجلة z=wa+b لإيجاد z ثم دالة ReLU لإيجاد a ثم التكرار . .
حيث المعادلات كالتالي :

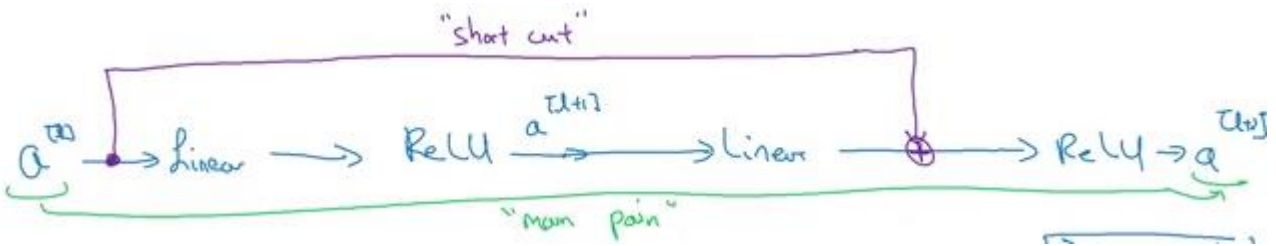
$$z^{[l+1]} = W^{[l+1]}\, a^{[l]} + b^{[l+1]}$$

$$a^{[l+1]} = g(z^{[l+1]})$$

$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$

$$a^{[l+2]} = g(z^{[l+2]})$$

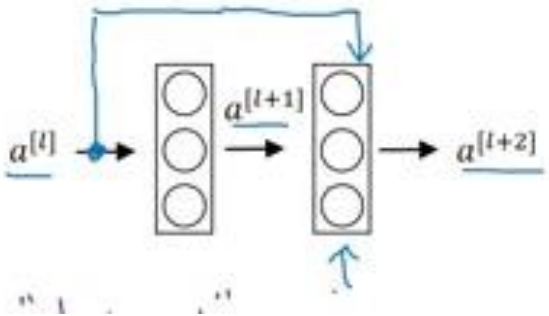
وفكرة الـ residual block تقوم علي إضافة قيمة al مع قيمة z(l+2) قبل تطبيق دالة الـ ReLU و كأن المسار الأخضر (main path) قم تم استبداله بالمسار البنفسجي (shortcut)



وتكون المعادلة :

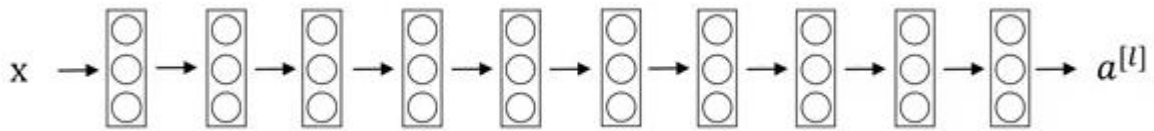
$$a^{[l+1]} = g(z^{[l+2]} + a^{[l]})$$

ويكون شكل البلوك الأصلي :

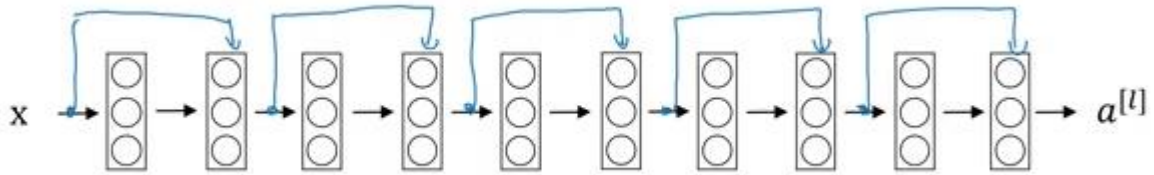


مع مراعاة أن قيمة al قد تم إضافتها قبل تطبيق دالة الـ ReLU , وتسمي العملية أحيانا Skip connection

وفكرة الـ ResNet تقوم علي ضم عدد من الـ Residual Blocks معا , حيث كل بلوك يقوم بقفز طبقتين مها , لتكون الشبكة الأصلية قبل تطبيق الـ ResNet هكذا :



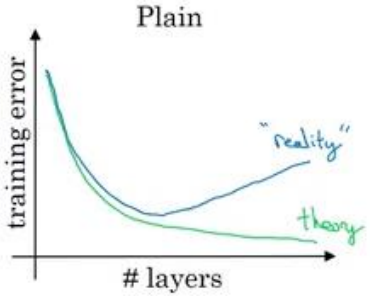
لتتحول هكذا :



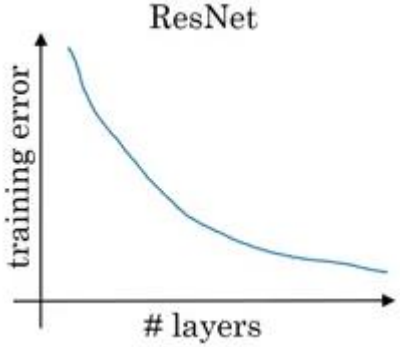
بعدما تم ضم 5 بلوكات معا .

ونري الفائدة الكبري للـ ResNet حينما نقوم بفحص مدي كفاءة الشبكات العميق للغاية .

فنجد هنا في الشبكات العادية أنه مع زيادة عدد الطبقات , فنظريا ستقل نسبة الأخطاء (الخط الأخضر) , لكن فعليا ستزيد نسبة الأخطاء بعد عدد عميق من الطبقات (الخط الأزرق) , مما يؤدي لتدهور الكفاءة .



بينما مع تطبيق الـ ResNet فيها , نجد أن الكفاءة تقل كلما زاد عدد الطبقات بهذا الشكل :



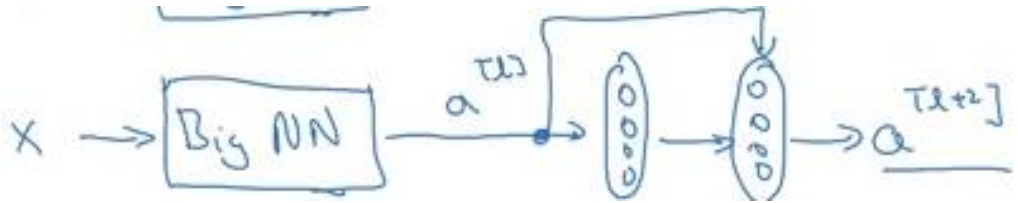
ولكن بعد عدد محدد من الطبقات , نجد أن الخط الأيمن يكون مستوي , مما يعني أن الكفاءة ثابتة مع ازدياد عدد الطبقات , وهو ما يجعل حتي الـ ReNet غير مناسبة للشبكات العميقة جدا .

-

إذن كيف تعمل الشبكات المتبقية ResNet ؟

بفرض أن لدينا شبكة عميقة , لها مدخل x و مخرج a(l)

فسنقوم بإضافة طبقتين لها علي أن تكون بنظام ResNet, نعم ستكون أكثر عمقا لندرس التأثير .



وهو ما معناه أن دالة الـ ReLU الأخيرة التي ستقوم بإنتاج a(l+2) ستأخذ قيمتي z(l+2) و a(l)

$$a(l+2) = g(\ z(l+2)\ +\ \ a(l)\ )$$

وإذا قمنا بفك z(l+2) لقيمتها التفصيلية :

$$a(l+2) = g(\ w(l+2)\ a(l+1) + b(l+2) +\ a(l)\ )$$

وحينما نقوم بعمل تنعيم البيانات regularization لقيم z(l+2) فهذا سيؤدي إلي تقليل قيمة w(l+2) وهو ما معناه زيادة قيمة a(l+1) و بالتالي تقليل قيمة b(l+2)

و مع اقتراب w(l+2) من الصفر , تزداد قيمة a(l+1) بشكل كبير مما يجعل قيمة b(l+2) هي الاخري تقل للصفر , مما يجعل كل قيمة + w(l+2) a(l+1) b(l+2) تقترب من الصفر . .

و هنا تكون كل الـ g() فقط لقيمة a(l) . .

و لأن دالة ReLU تساوي نفس قيمة المدخل لو كانت موجبة , فيكون المخرج النهائي a(l+2) يساوي a(l) , وهو ما يعني أن زيادة طبقتين باستخدام ResNet لن يغير تقريبا من الناتج , ولن يقوم بعمل ضرر في البيانات .

وهذا الأمر يفسر سبب زيادة او ثبات كفاءة اي طبقات إضافية من نوع الـ ResNet

ولاحظ أنه يشترط أن يكون حجم مصفوفة a(l) متساوي مع حجم مصفوفة z(l+2) , حتي نقوم بعملية الجمع بسهولة داخل دالة الـ ReLU, ولا تنس أن اي مصفوفة z تكون مساوية لمصفوفة a السابقة لها تبعا للقانون :

w(l+2) a(l+1) + b(l+2)

وبالتالي اذا كان عدد parameters المصفوفة a(l) مساوي لعدد a(l+1) فيمكن الجمع بسهولة . .

لكن ماذا لو كان عدد الـ parameters مختلف ؟

وقتها بدلا من الصيغة التقليدية للقانون هذا :

$$a(l+2) = g(\ w(l+2)\ a(l+1) + b(l+2) +\ a(l)\ )$$

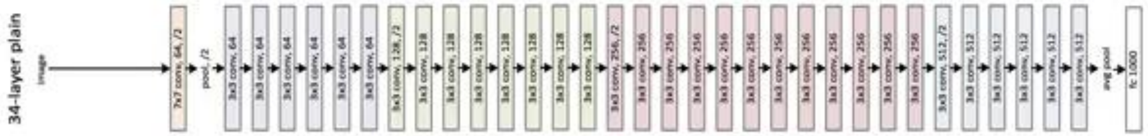
سنقوم بإنشاء مصفوفة جديدة تسمي Ws حتي يتم ضربها في a(l) لتكون بنفس حجم z(l+2)

فإذا كان حجم a(l) هو 1*128 و حجم z(l+2) هو 1*256 , وقتها يكون حجم Ws هو 128*256 حتي اذا ما تم ضربها في a(l) تكون 1*256

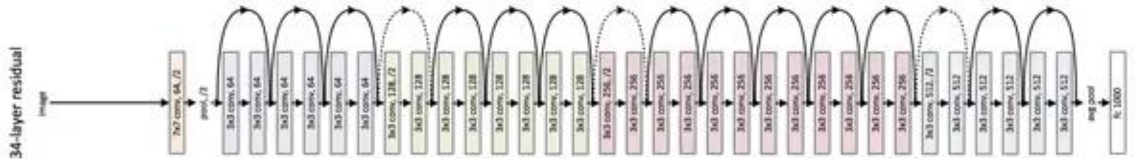
$$Ws \times a(l) = 256*128 \times 128*1 = 256*1$$

واثناء التدريب , سيقوم الخوارزم باختيار و ضبط ارقام مناسبة لمصفوفة Ws

وبالتالي إذا كان لدينا شبكة عميقة هكذا . .



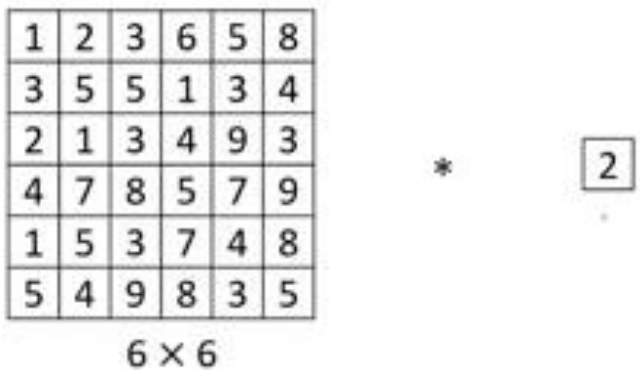
يمكن تحويلها لـ ResNet هكذا



-

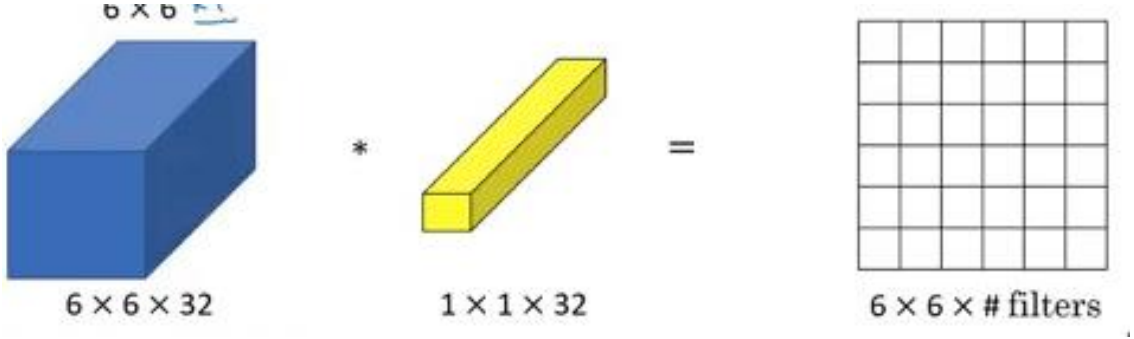
نتناول الآن طريقة هامة , وهي الشبكة الملتفة 1*1

اذا قمنا بترجمة المعني حرفيا , فهو جعل الفلتر الذي يقوم بالالتفاف حول الصورة , يكون قيمة واحدة 1*1 , وهو ما يعني هذا الأمر :



سيبدو الأمر كأنه ضرب لقيم المصفوفة الـ 36 في رقم 2 , وهو بالفعل شئ ليس له استخدام .

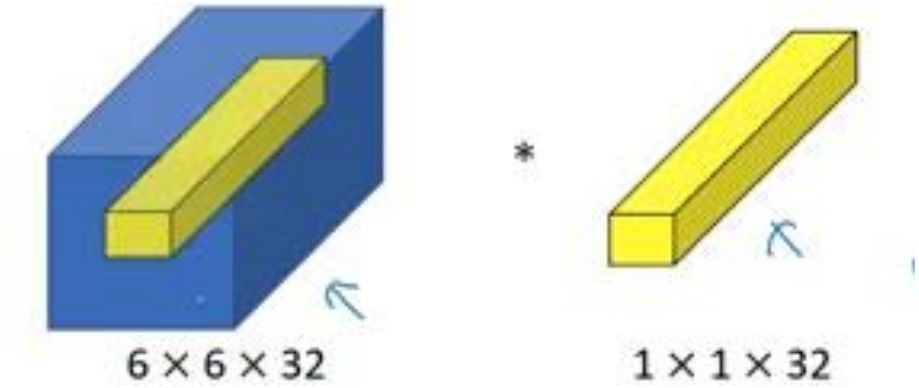
لكن فكرة 1*1 , تكون مفيدة في المصفوفة المجسمة هكذا :



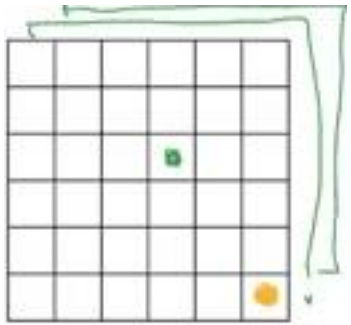
حيث أن الـ 1*1 هي الصفوف و الأعمدة بالفعل , لكن يكون لها عمق يساوي عمق المصفوفة المجسمة , مما يعني أن كل قيمة من قيم الفلتر الـ 32 , سيتم ضربها في القيمة المناظرة لها في المصفوفة الاصل , و من ثم عمل دالة ReLU لها , وايداعها في مصفوفة الناتج . .

وكأنه تم تحويل المصفوفة ثلاثية الأبعاد لثنائية الأبعاد ,و هو ما يسهل كثيرا عملية المعالجة

بحيث تكون قيم الارقام التي سيتم ضربها في المصفوفة الأم هكذا :



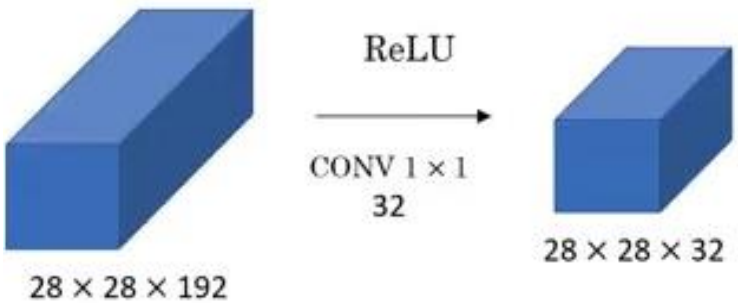
ويمكن أن يكون هناك عدد من الطبقات للفلترز 1*1 , فلو كان الفلتر الأصفر هذا , له عدد من الطبقات المتتالية , نقوم بضرب كل طبقة فيهم في كل قيم العمق الـ 32 , وايداعها في الجدول ثنائي الأبعاد , بحيث يكون له عدد طبقات يساوي عدد طبقات الفلتر المستخدم .



لذا يتم وصف أبعاد مصفوفة الناتج بـ :

$$6 \times 6 \times \#\,\text{filters}$$

وبالتالي لو كان لدينا مصفوفة مكعبة 192*28*28 و نريد تقليلها لـ 32*28*28



فنقوم باستخدام فلتر 1*1 و بالطبع يكون عمقه 192 حتي يقوم بتغطية العمق الكامل للمكعب , و أن يكون له 32 طبقة . .

بحيث أن يقوم الفلتر 192*1*1 بالطبقة الأولي منه , أن يقوم بانشاء الطبقة الأولي من المكعب الجديد , عبر ان يلتف علي كل قيم المكعب الأصلي , ويتم تكرار الأمر مع باقي الطبقات .

-

و هنا يأتي سؤال هام , حينما أقوم بتصميم شبكة , هل اختار الفلتر بابعاد 3*1 , ام 3*3 ام 5*5 ام ماذا ؟ ؟

هنا نستخدم تكنيك : شبكة البداية inception network لحل المشكلة عبر دمجهم معا بطريقة ما . .

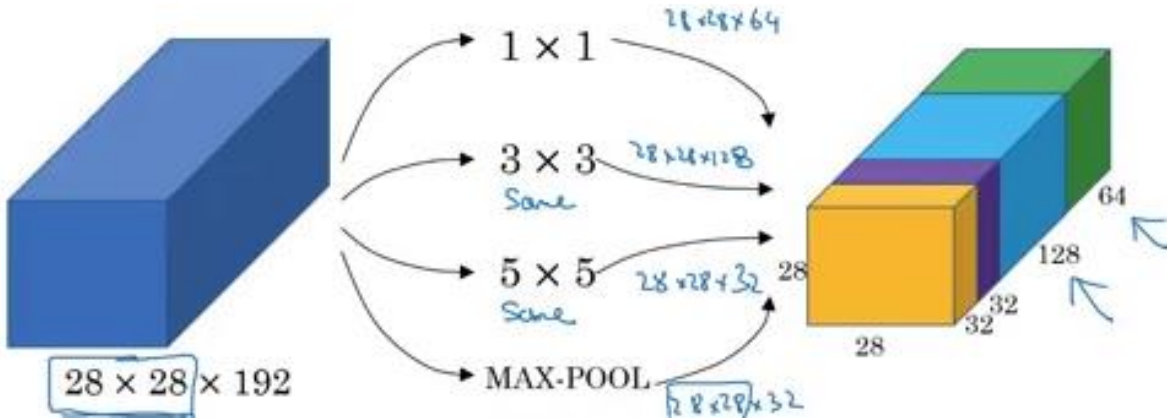
و قد تمت تسميته بناء علي فيلم inception بسبب ان الفيلم اعتمد علي الأعماق في الحلم

فلو كان لدينا مصفوفة مجسمة بابعاد 192*28*28 هكذا :

و نريد ان نطبق عليها اربع تكنيكات للشبكات الملتفة , وهي :

- شبكة 1*1
- شبكة 3*3
- شبكة 5*5
- شبكة max pooling

فيمكن تطبيق الأربعة معا هكذا :



أولا يتم استخدام الشبكة 1*1 , وأن يكون لها 64 طبقة (مثلا) , وهو ما سينتج مكعب 28*28*64 , وهو الموجود في المؤخرة باللون الأخضر . .

ثم استخدام شبكة 3*3 , بعمق 128 طبقة (مثلا) , ومع عمل حشو لها حتي يكون عدد صفوف و اعمدة الخارج بنفس عدد الداخل , يكون الناتج 28*28*128 باللون اللبني .

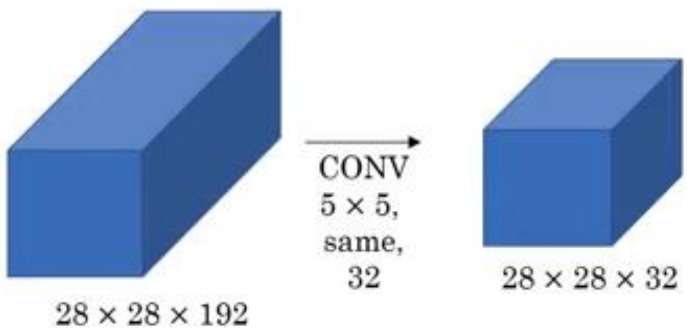
ثم شبكة 5*5 , بعمق 32 طبقة , وايضا استخدام حشو للمحافظة علي الأبعاد , فتكون 28*28*32 , باللون الأزرق .

ثم استخدام تكنيك maxpool بعمق 32 طبقة , فيكون الناتج 28*28*32 باللون البرتقالي .

و بالطبع يجب المحافظة علي ثبات عدد الصفوف و الأعمدة في المخارج (باستخدام الحشو) , حتي يتم ضمهم معا كما في الشكل .

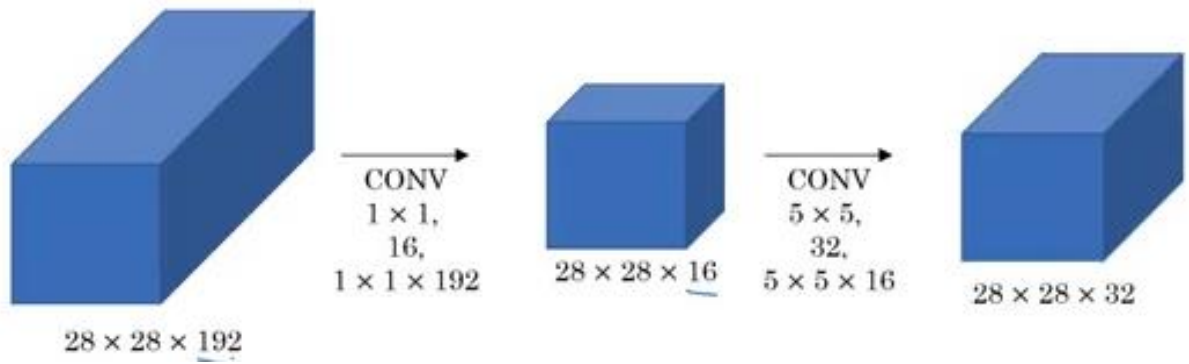
فكانت مصفوفة الداخل الكلية هي 28*28*192 , والخارج هي : 28*28*256 .

ولكن هناك مشكلة ستظهر في ما يسمي تكلفة الحساب computational cost , في الطبقة الزرقاء 28*28*32 , والتي أتت من استخدام فلتر 5*5 . .



و المشكلة تكمن انه في حالة حساب عدد الارقام التي ستنشأ من ضرب 32*28*28 , مضروبة في 192*5*5 , وهي عدد ارقام المصفوفة الملتفة , فسيزيد الناتج عن 120 مليون قيمة , وهو رقم ضخم حتي بمقاييس العصر الحديث .

و سنري الان كيف يمكن تقليل هذا الرقم لمقدار العُشر تقريبا , باستخدام مصفوفة 1*1



الفكرة تقوم علي عمل مرحلة وسيطة بين مصفوفة المدخل : 192*28*28 و مصفوفة المخرج 32*28*28 , وهي استخدام مصفوفة ملتفة 192*1*1 , بـ 16 طبقة , والتي ستصنع مصفوفة وسيطة 16*28*28 . .

و بعدها سنستخدم مصفوفة الـ 16*5*5 و لكن ستكون بـ 32 طبقة فقط . .

فعبر تقسيم العملية لعمليتين , سنجد أن الحساب الإجمالي أقل .

فالعملية الأولي ستكون :

28x28x16 x 192 = 2.4 M

والثانية :

28x28x32 x 5x5x16 = 10 M

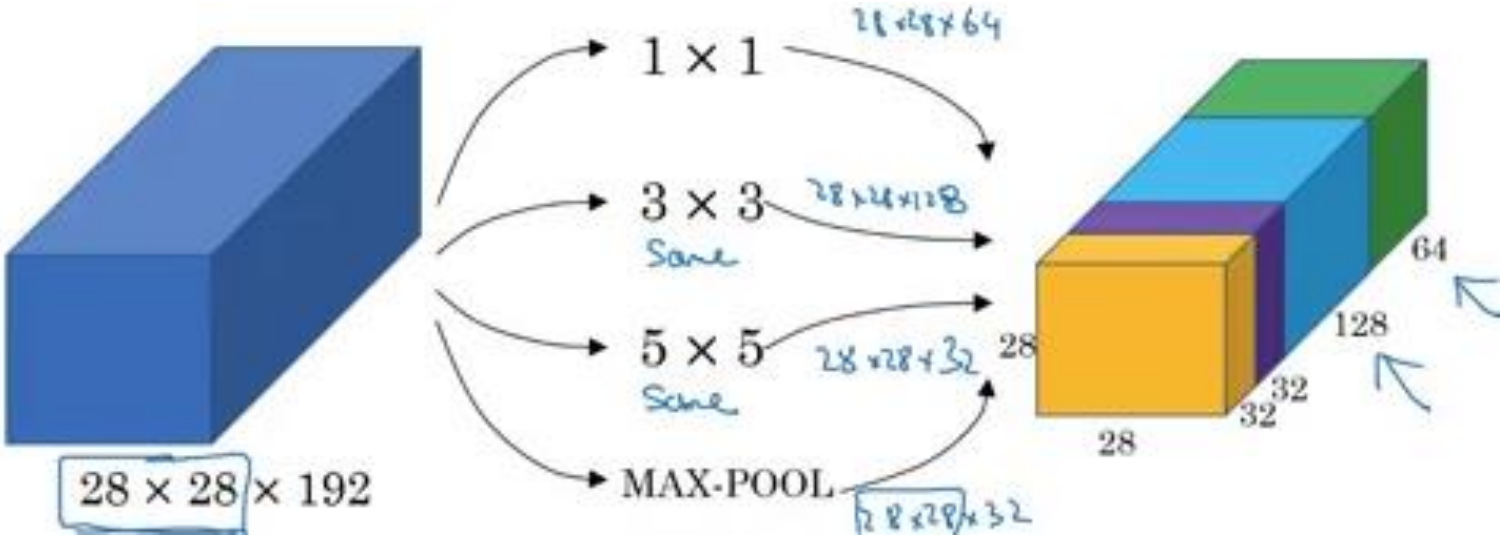
فيكون المجموع 12 مليون , وهو ما يساوي 10% من العملية المباشرة .

و تسمي المرحلة الوسطي : عنق الزجاجة bottleneck و ذلك لأنها مصفوفة ضيقة و صغيرة بين مصفوفتين كبيرتين .

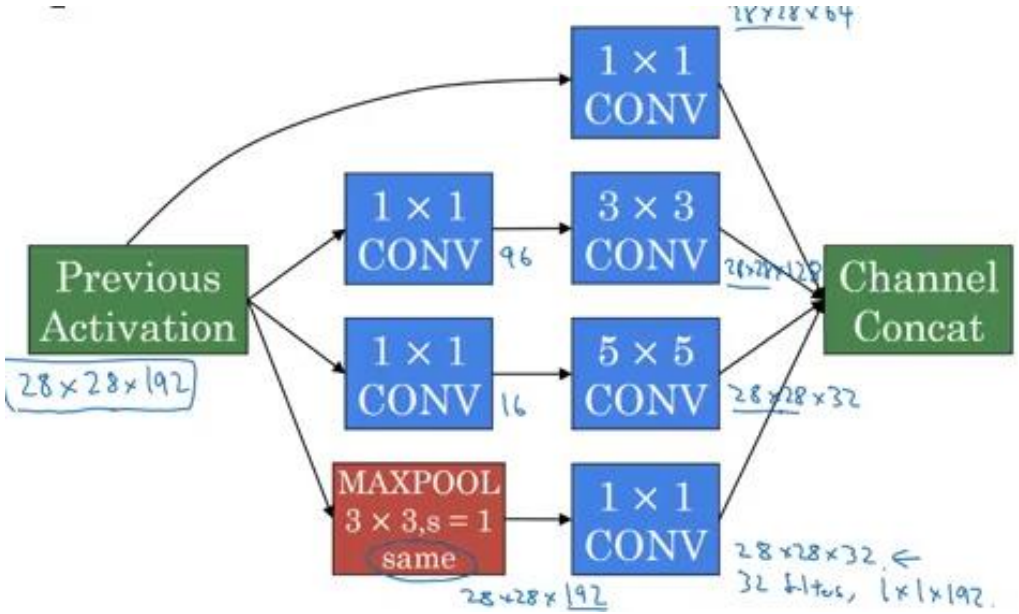
-

بعد ان رأينا التكوين العام لنموذج البداية inception model , فنتعرف علي اسلوب بنائها .

كان هذا هو الشكل العام , قبل استخدام فكرة الشبكة الوسيطة 1*1

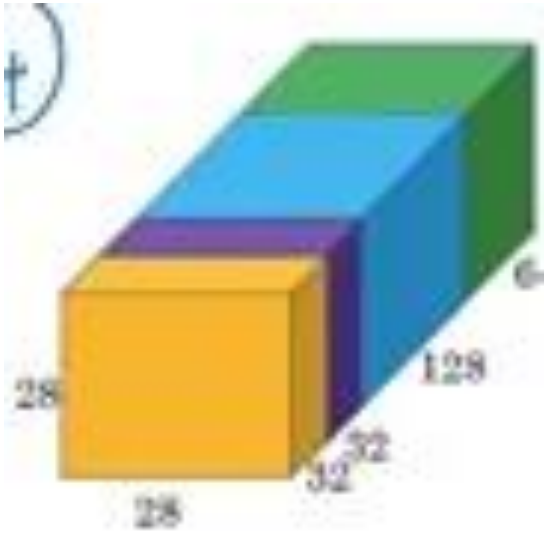


بعد استخدامها سيكون هكذا :

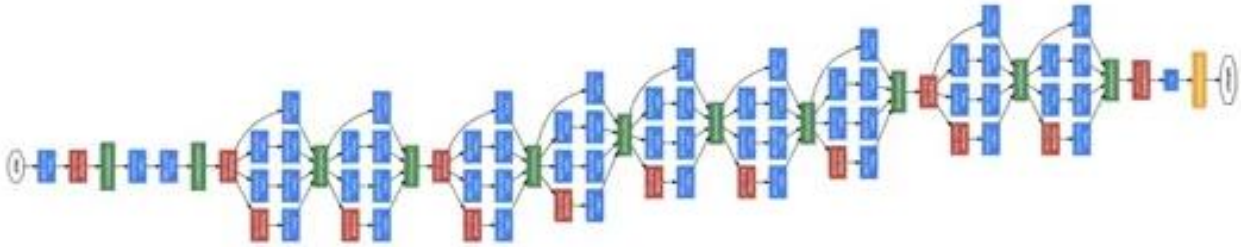


حيث يتم استخدام 1*1 في المرحلة الثانية 3*3 , والثالثة 5*5 , ولا حاجة لاستخدامها في الأولي لانها اصلا بسيطة , وتستخدم في الرابعة , لكن بعد مرحلة الماكسبول .

وأخيرا تأتي مرحلة concatenate أي ضم القيم معا في مصفوفة كبيرة 256*28*28 لتكون هكذا :

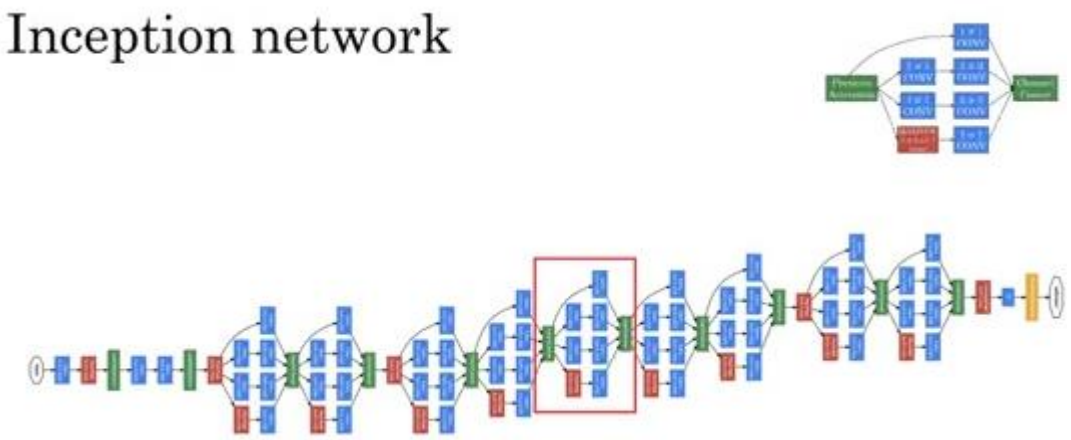


وإذا اردنا القيام ببناء الشبكة كاملة ستكون كالتالي :

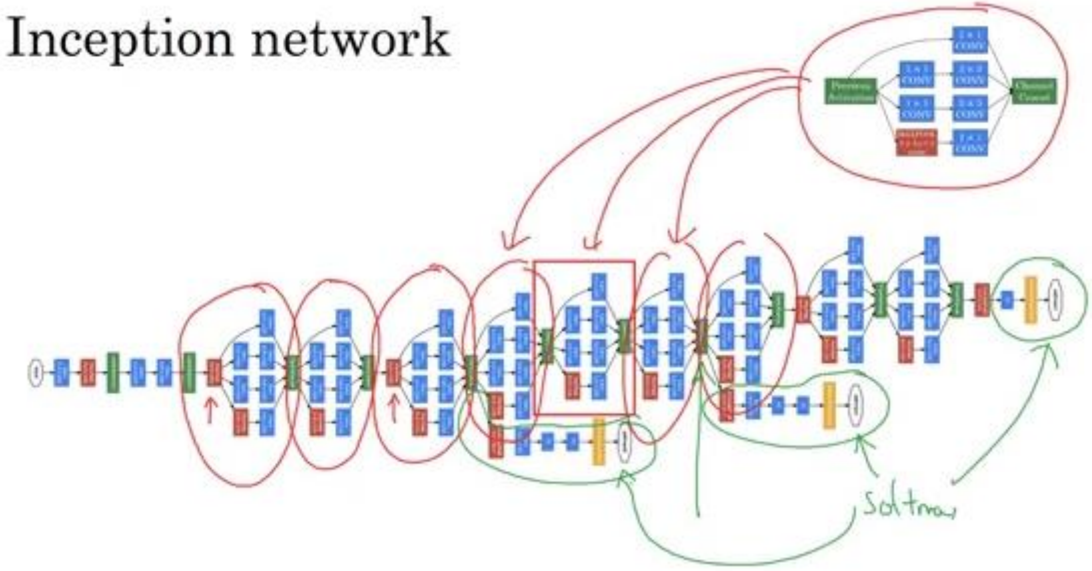


حيث ان كل وحدة يتم عمل الإضافات المذكورة فيها , قبل ضمها للتالية و هكذا . .

و تكون كل وحدة فيهم كالتالي :



كما أنه بالقرب من نهاية الشبكة , نري أن هناك وصلات فرعية لتتصل بشكل مباشر مع النهاية و مما يزيد من كفاءة الشبكة :


Inception network
Softmax

-

بعد ان فهمنا كيفية بناء تلك الشبكات المعقدة , سنتكلم الآن عن التعامل مع الأكواد المتاحة , بأسلوب "المصادر المفتوحة" open source

وذلك لأن مع الصعوبة المتزايدة في بناء الـ NN باختلاف انواعها ,فليس من الذكاء تجاهل الأعداد المتراكمة من المشاريع الموجودة بالفعل , بل عليك أن تُحسن تعلمها و الاستفادة منها و ومن أهم المواقع التي بها مشاريع جاهزة : github

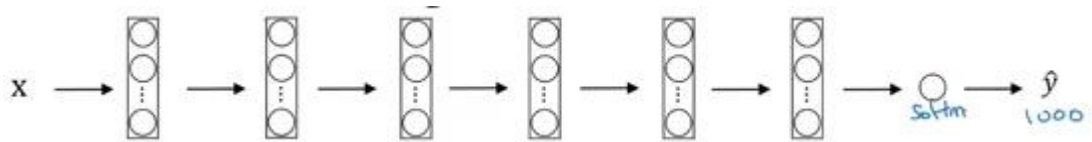
وحتي التواجد الالكتروني لا يقتصر علي الخوارزميات المستخدمة , لكن أيضا علي قواعد البيانات datasets الضرورية لتدريب الخوارزم .

و أيضا بدلا من عمل تدريب يأخذ وقتا طويلا , فقد تقوم بتحميل النتائج الخاصة بتدريب علي آلاف الصور بالفعل , وتبدأ انت من حيث انتهي هو .

وكل هذه الأمور تسمي التعلم المتنقل Transfer Learning

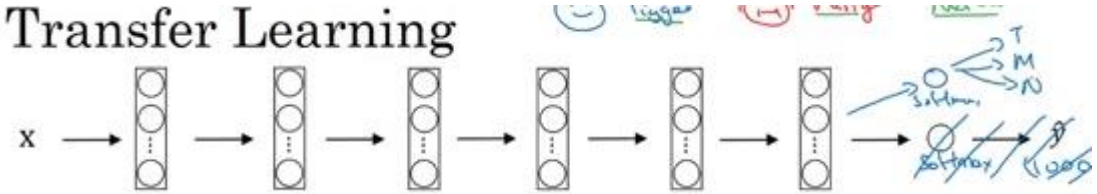
فبفرض أنك تريد أن تقوم بعمل خوارزم لتصنيف صور القطط , بين نوعين من أنواع القطط tigger or misty وبالتالي سيكون لديك كلاسيفير به 3 اختيارات , تيجر او ميستي او آخر

فيمكن البحث عن خوارزم تصنيف جاهز , وليكن هذا الخوارزم



وستلاحظ ان هناك في النهاية softmax و الذي يقوم بعمل تصنيف لألف اختيار , علي فرض أن الخوارزم الاصلي كان للتصنيف بين الف اختيار . .

فيمكنك وقتها ان تقوم بإلغاء أخر أجزاء (سوفتماكس + المخارج الألف) , وبناء سوفتماكس خاص بك , بثلاث مخارج فقط



مع ملاحظة هامة . .

ان الخوارزم الذي قمت بتحميله , اذا ما كان تم تدريبه قبل ذلك , فهذا معناه ان له قيم اوزان الـ parameters جاهزة , ولا تحتاج لتدريب آخر , فيمكن وقتك ان تقوم بعمل freeze لقيم parameters الشبكة الأصلية , حتي لا يقوم الخوارزم بتغييرها , وان يركز فقط علي الطبقات التي قمت انت بإنشائها (الأخيرة) . .

و إذا رأيت ان الخوارزم الذي قمت بنسخه بعيد قليلا عن الخوارزم المطلوب لك , فيمكن الغاء طبقات اكثر من الخوارزم الأصلي , او إنشاء طبقات جديدة أخري , او عدم freeze للاوزان حتي تتغير

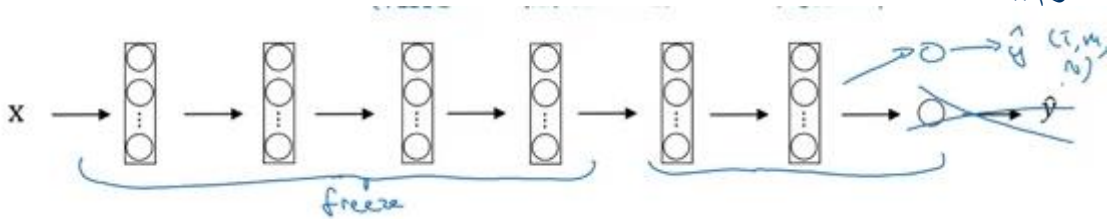
وأكبر ميزة في هذا الأمر , ليس فقط اختصار الوقت في بناء الشبكة , ولا أيضا اختصار الوقت في تدريبها , ولكن الأهم أنك لا تحتاج الي العديد من البيانات dataset لتدريب الخوارزم الخاص بك , فيمكن عبر عدد قليل من صور القطط , ان تجعله يدرب الطبقات الأخيرة فقط .

و غالبا ما يكون في قلب البرنامج متغير يسمي freezeparameter والذي يكون بقيمة صفر للطبقات التي تريد للخوارزم ان يقوم بتدريبها , وبقيمة 1 للطبقات التي تريد للخوارزم ان يتركها و لا يدربها لانها جاهزة .

كذلك من الخطوات التي تقوم بتسريع العملية , ان تقوم بتجميع كل الطبقات التي سيتم تجميدها (وهي الجاهزة في البناء و في الأوزان) , وان تضمها في دالة function بحيث يكون لها مدخل الصور x و المخارج و هي قيم الـ activations التي ستخرج منها . .

و من ثم .., يكون بناء الخوارزم الخاص بك اسهل , الصور تدخل في دالة (جميع الطبقات المجمدة) , وتخرج لتدخل في الطبقات الاخيرة التي قمت ببناءها .

وفي حالة كان لديك كمية كبيرة من البيانات (صور القطط) , فيمكن أن تقوم بتجميد عدد اقل من الطبقات و تدريب الطبقات الاخيرة , مع دالة السوفتماكس التي ستقوم ببناءها



و غالبا ما يتم استخدام الاوزان الموجودة في الطبقات الغير مجمدة كقيم مبدأية للخوارزم لكي يقوم بالتدريب لها .

فكلما زادت البيانات الموجودة , كلما قل عدد الطبقات المجمدة , وزاد عدد الطبقات التي سيتم اعادة تدريبها . حتي يمكن ان نقوم بتدريب الخوارزم من البداية , واعادة تعيين الاوزان من البداية .

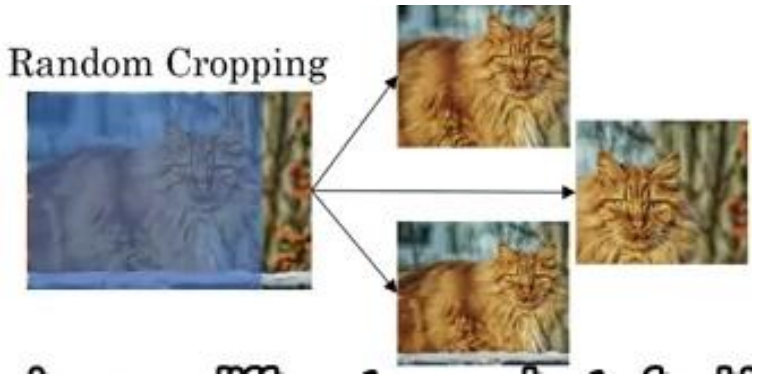
-

زيادة البيانات : Data Augmentation

وهي عملية هامة و مفيدة , لتدريب اي خوارزم يختص بالتعامل مع الميديا (صور , صوت , فيديو) , حيث أن زيادة بيانات الميديا شديدة الأهمية لرفع كفاءة اي خوارزم لتدريبها .

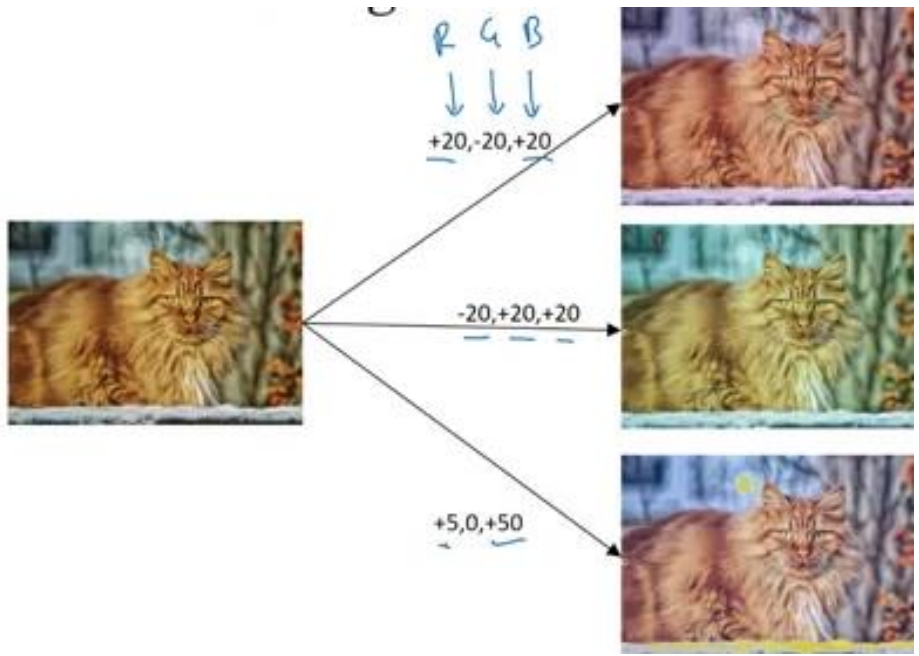
فمن الطرق المستخدمة : طريقة المرآة المعكوسة حيث نقوم بعكس الصور المتواجدة لدينا

كذلك يمكن استخدام طريقة الإجتزاء العشوائي random crop , والتي تقوم باجتزاء اجزاء من الصور , لتكوين صور جديدة



كذلك يمكن استخدام الدوران , او الإمالة (لتكون شبه متوازي الأضلاع) , او المط , أو أي طريقة تنتج المزيد من الصور المنطقية

كذلك يمكن استخدام تكنيك : تغيير الألوان



و ذلك عبر عمل إضافات محددة علي ألوان RGB لتغيير اللون العام للصورة , فإذا أضفنا 20 علي الأحمر , و20 علي الأزرق , وقمنا بنقص 20 من الأخضر , فهذا يجعل الصورة أكثر بنفسجي , و هكذا في باقي الألوان . .

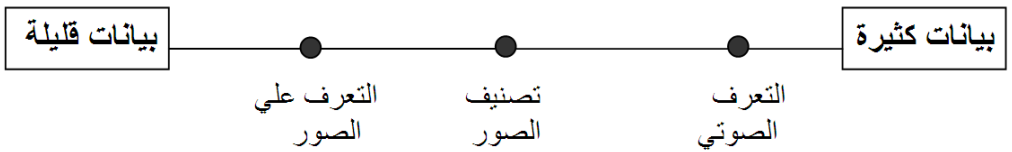
و الهدف الاساسي , هو جعل الخوازم يتدرب علي صور بظروف لونية مختلفة , فقد يكون هناك صور وقت الشروق حينما تكون الصورة أكثر ازرق , او وقت الغروب حينما تكون بها أحمر اكثر .

و غالبا ما يستخدم تكنيك PCA الذي سبق دراسته في الـ unsupervised لموازنة الألوان , ولعمل إضافات دقيقة .

-

و في نهاية هذا الأسبوع , سنقوم بسرد عدد من النصائح و الأمور الهامة , الخاصة بتطبيقات الرؤية عبر الحاسب computer vision والتي تختلف نوعا عن باقي تطبيقات الـ DL

إذا أردنا ان نقوم بعمل مقياس عام , لنري مدي توافر البيانات المتاحة لنا في عدد من تطبيقات الـ DL , سنري ان التقسيم كالتالي :



نري أن التعرف الصوتي speak recognition له كمية كبيرة من البيانات المتاحة (نظرا لرخص ثمنها) , بينما تصنيف الصورة image recognition له كمية متوسطة , أما التعرف علي الصور object detection فله كمية اقل لارتفاع ثمنها . .

و المقصود بالتعرف علي الصور object detection هو حينما يتم ادخال صورة ما , و يقوم الخوارزم لعمل أطر حمراء اللون مثلا حول كل العناصر الموجودة . .

و كقاعدة عامة , فكلما كانت البيانات اكثر , كلما كان الخوارزم اسهل و ابسط , وكلما قل الاختياج الي ما يسمي الهندسة اليدوية hand engineering , وهو التدخل البشري في الأكواد و تصميم الشبكات , وتحديد العوامل المختلفة و هكذا .

و العكس , كلما كانت البيانات أقل , كلما كان الخوارزم اصعب , وتم الاعتماد علي الهندسة اليدوية اكثر لتعويض النقص في البيانات المتاحة .

ولأن بصورة عامة , التعامل مع الرؤية عبر الحاسب computer vision به نقص حاد من الصور المطلوبة , و ارتفاع في تكلفتها

و من الطرق المستخدمة لتطوير الكفاءة , ما يسمي ensembling

وهي ان تقوم باستخدام البيانات (صور القطط) في اكثر من شبكة NN (مثلا 5 شبكات بالتوازي) , ثم تقوم بايجاد جميع المخارج و تحسب المتوسط الخاص بهم .

و هذه الطريقة تزيد الكفاءة لكنها تستهلك وقت اكثر .

-