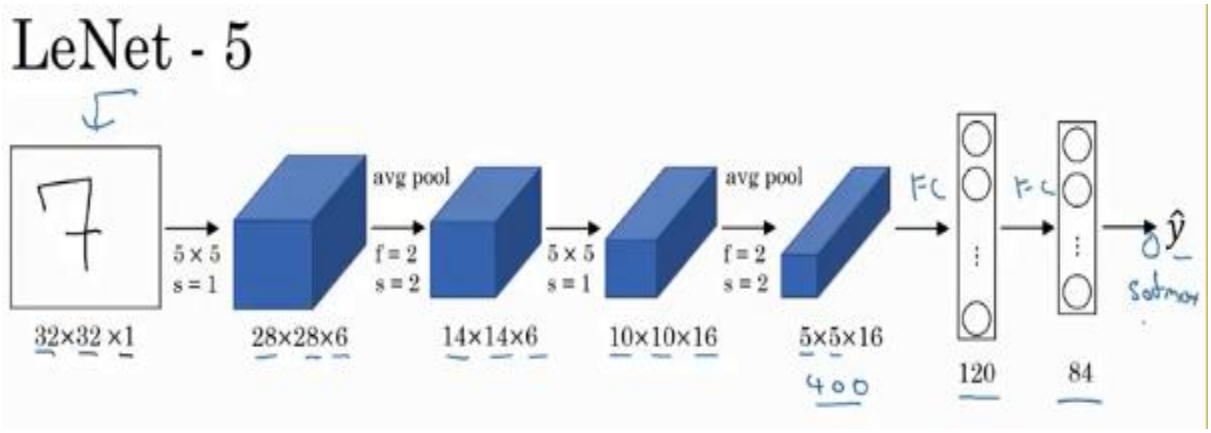
Course 4 Week 2

We have studied the basic layers that form together convolutional neural networks and over the few past years the research was on how to put those layers together to form effective one. Like we have learned coding by seeing other people's code, we can build convnets (convolutional networks) by seeing other examples of effective convnets. It turns out that convnets which work well on one computer vision task often work well on other computer vision tasks. Like if your network works well on a task of recognizing cats, dogs and people so you can apply the same network on other tasks as recognizing cars by applying it to its dataset and most probably it will work well on it. So let us start of some famous classic neural networks that were a state of art.

Classic networks

LeNet-5 By Yann-LeCun

LeNet was trained on greyscale images on MNIST dataset concerning handwritten digits recognition and Here is the architecture:



Notes:

1-This neural network is pretty small compared to recent neural networks as it has only 60K parameters while now we have reached from 10 mils to 100 mils parameters so it is small.

2-As we go deeper in this network, the height and width shrinks (32→28→14→10→5…).

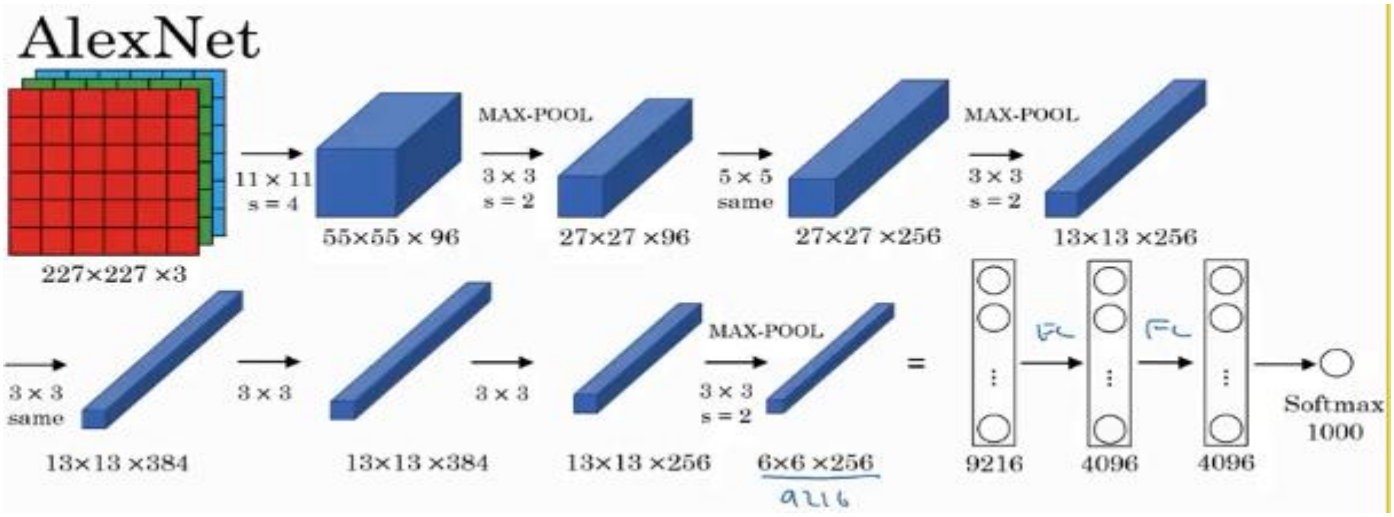
3-As we go deeper, the number of channels increases (1→6→16).

4-This network was using sigmoid/Tanh activation function not ReLU as it wasn't introduced yet.

5-Sigmoid non-linearity was used after pooling layer but now it is not used anymore.

6-In case you will read its paper just focus on section II and III as recommended form Andrew.

AlexNet By Alex Krizhevsky



Notes:

1-This neural network similar to leNet-5 but much bigger with 60 mil parameters.

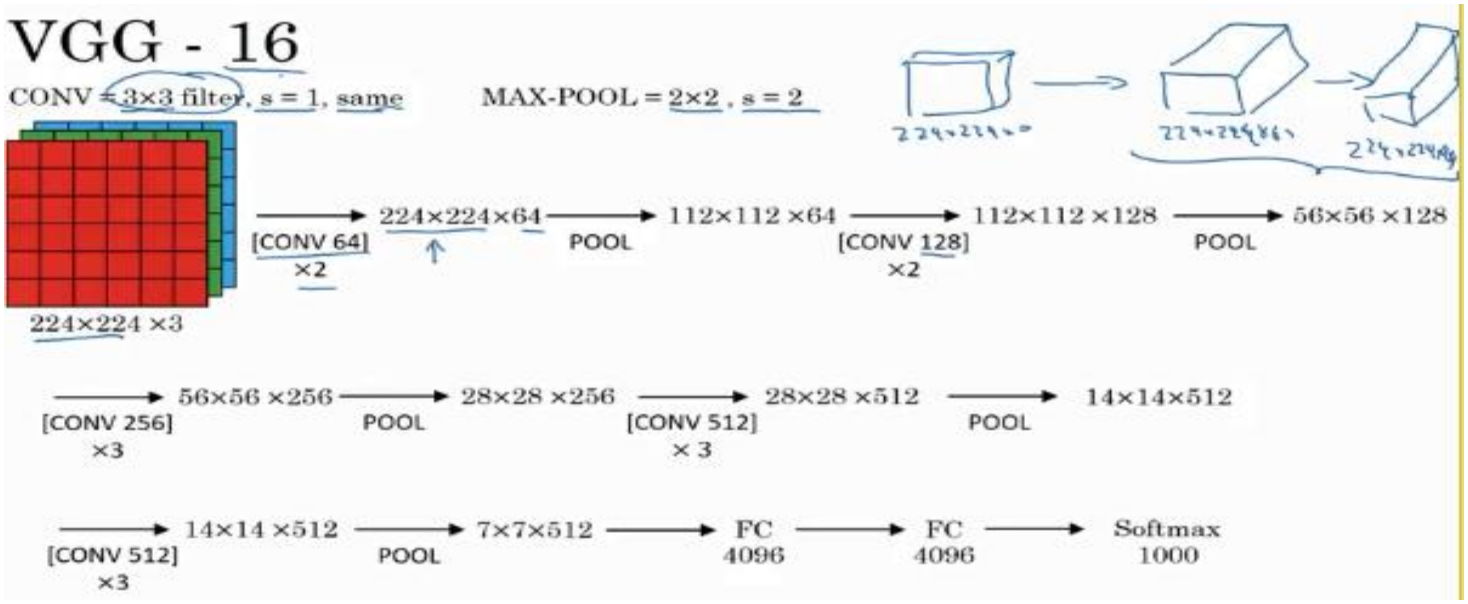
2-This network used ReLU activation function instead of sigmoid/Tanh functions.

3-This architecture was introduced in a time when GPU was slower than now so it was trained on multiple GPUS.

4-This network uses layer called local response normalization but this type of layer is not used much recently.

VGG-16 By Simonian & Zisserman

It has all conv layers of size 3×3 with S=1 and same padding in addition to all max-pooling layers of size 2×2 and S=2. As it is so deep so we represent n similar layers by (×n).



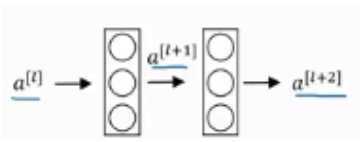
Notes:

1-This network is composed of 16 layers of a total number of learning paramters≈138 mil.

2-There is another VGG network composed of 19 layers instead of 16 called VGG-19.

Residual networks (ResNets)

Very deep neural networks are hard to be trained due to vanishing and exploding problems. Here comes the idea of skip connections which would make activations from one layer feeds a deeper layer in the network and it is the basic idea resnets relies on.



Residual block

To go from $a^{[l]}$ to $a^{[l+2]}$ you have to go through the main path as follows:

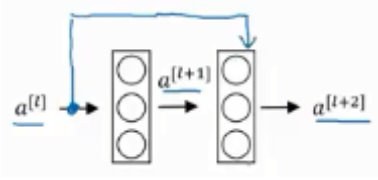
$a^{[l]} \rightarrow z^{[l+1]}=w^{[l+1]}.a^{[l]}+b^{[l+1]} \rightarrow a^{[l+1]} = g(z^{[l+1]}) \rightarrow z^{[l+2]}=w^{[l+2]}.a^{[l+1]}+b^{[l+2]} \rightarrow a^{[l+2]} = g(z^{[l+2]})$

How to modify this ?

we are going to take $a^{[l]}$ and add it to output of $z^{[l+2]}$ using shortcut path as follows:

$a^{[l]} \rightarrow z^{[l+1]} = w^{[l+1]}.a^{[l]}+b^{[l+1]} \rightarrow a^{[l+1]} = g(z^{[l+1]}) \rightarrow z^{[l+2]} = w^{[l+2]}.a^{[l+1]}+b^{[l+2]} \rightarrow a^{[l+2]} = g(z^{[l+2]}+ a^{[l]})$

Notes:



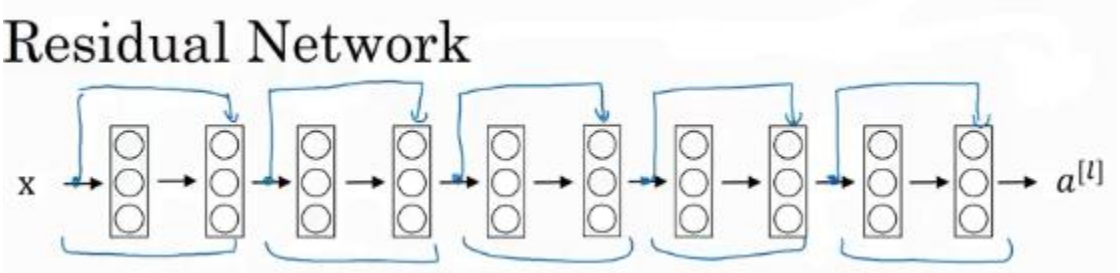
1-The path made above is called either shortcut or skip connection.

2-Shortcut is connected before applying activation function g().

3-You can make the path added to any following layer $z^{[l+2]}$ or $z^{[l+3]}$ or … or $z^{[L-1]}$.

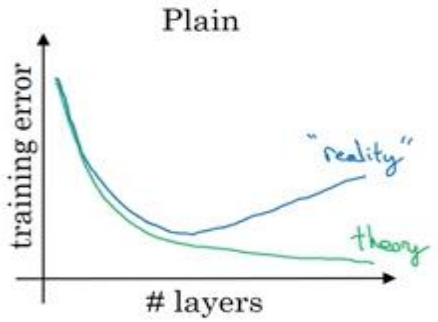
4-As we are adding $z^{[l+2]}$ to $a^{[l]}$ so they should have the same sizes so same padding (conv) is used more often than valid padding (conv) in resnets. But in case $z^{[l+2]}$ has different size so we make it $g(z^{[l+2]}+ w_s.a^{[l]})$ where w_s will match the size of $a^{[l]}$ to $z^{[l+2]}$ and it could be zero padding or a weight matrix to be learned along the learning process.

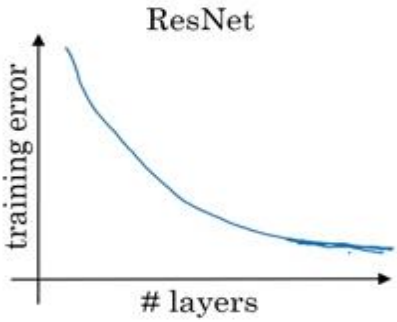
Residual network



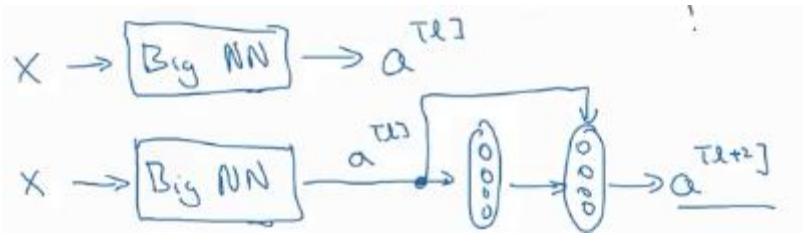
Did residual networks affect the performance compared to plain networks (no residuals)?

Yes, it did make the performance better as theoretically when we have more layers in the network it should make training error decrease but practically with plain networks it didn't help that much as it will reach to a certain number of layers then the training error will increase again. ResNets solved this to a high extent and make it stand further with increasing number of layers of the network.





Why ResNets work?



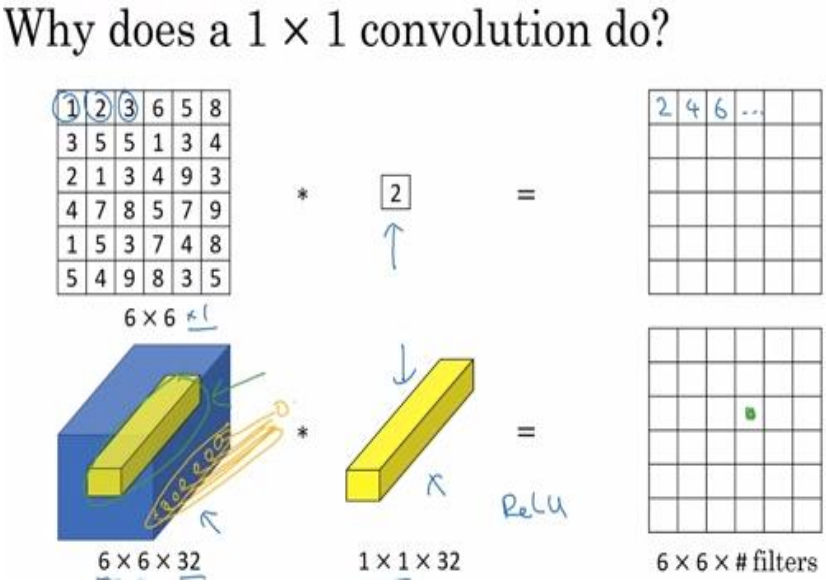
Assume we have those two networks where one of them is a big neural network and the other is a big neural network followed by residual block, so by writing the equation for the 2nd one ➔ $a^{[l+2]}= g(z^{[l+2]}+ a^{[l]})= g(w^{[l+2]}.a^{[l+1]}+b^{[l+2]}+a^{[l]})$

So if $w^{[l+2]} = b^{[l+2]} = 0$ and we are using ReLU activation function so this will lead to that $a^{[l+2]}$ will be equal to $a^{[l]}$ which means that residual block is acting now as just an identity function which means that with the presence of residual block we can add more deep network and if there is nothing more to learn, the network will have the ability to just copy previous output again and not hurt the neural network by making it doing similarly to simpler networks unlike plain networks which has to choose values for parameters as we go deeper which may result in worse result if it couldn't learn. Of course it is not only for not hurting your neural network but also for improving by making this residual blocks learn more if there is something to be learned. To summarize, the residual block will allow the network to either learn something new or act as identity so not affect the neural networks as it goes deeper.

Network in Network (1×1 convolution)

In fact if I told you that I'm doing 1×1 convolution you would laugh as 1×1 convolution means just multiplying each feature (pixel) by number !! so it is just multiplication !! this is true for 1 channel but if we are convolving several channels then now it is more logical.

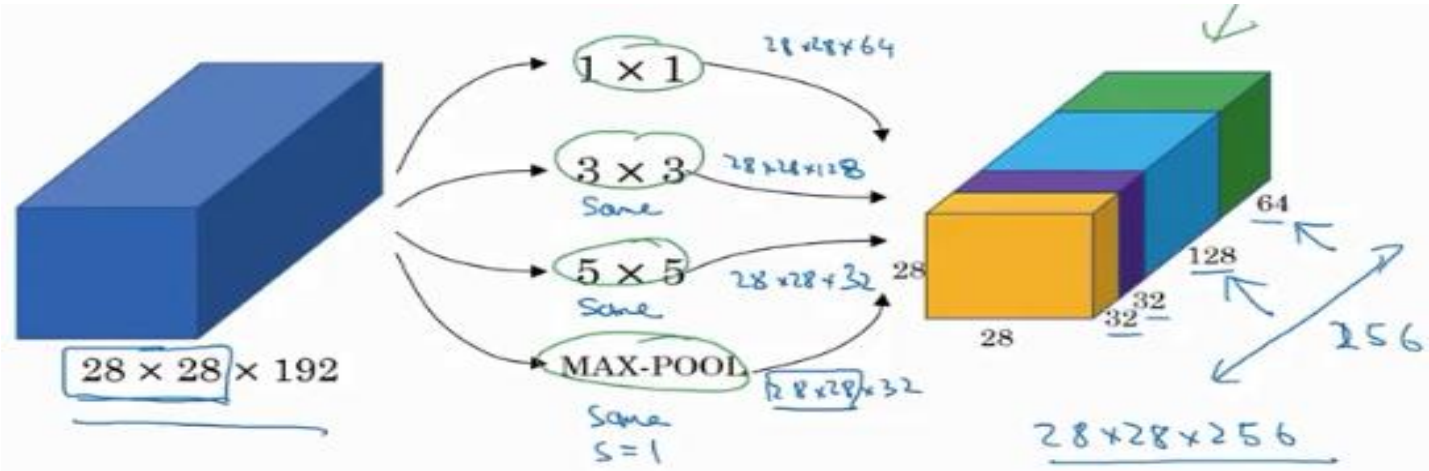
If you are convolving 6×6×1 image with 1×1 filter so this is a direct multiplication as shown but if you are convolving 6×6×32 image for an example with 1×1×32 filter and now let's do convolution so we are now convolving 1×1×32 (slice) with filter 1×1×32 (slice) and end up with a single number to be an output for this place and then repeat for every slice of the 36 slices (6×6) to end up with 6×6 output but as we said before that we are not using only 1 filter but multiple filters so we end up with 6×6× #filters.



But what is the effect of using such convolution? in fact it could affect in adding more non-linearity to your system as we apply activation function (say ReLU) after this step as usual so we are making more complex function. It can also help in shrinking number of filters if the input to this conv layer was of large number of filters and you want to restrict it to reduce computations.

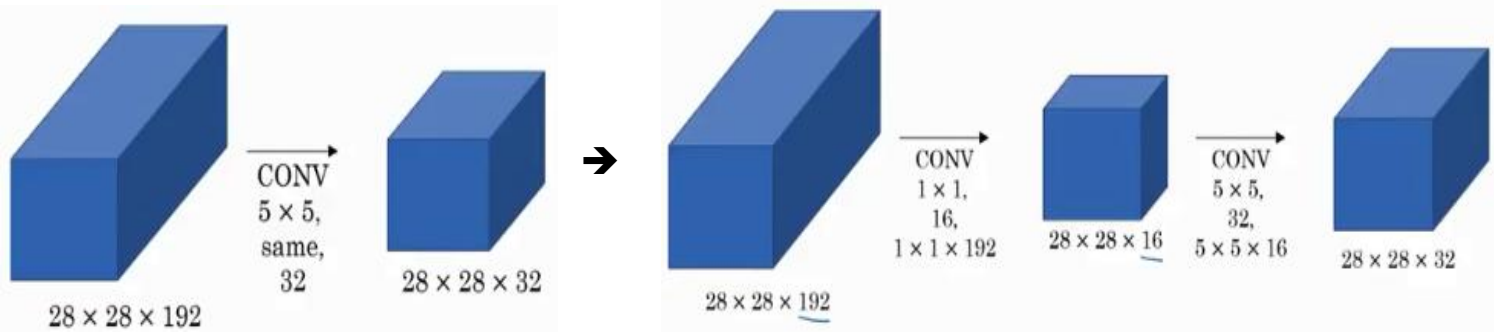
Inception Network

Inception network relies on the idea of network in network in its architecture. Inception network makes more complicated architecture but remarkably doing well as you always should pick the type of layer when constructing one so you have to decide this either 3×3 conv layer or 5×5 conv layer or 7×7 conv layer or pooling layer but inception says that why should you do them all? let us see how to construct inception module then use several modules to construct network: Assume having 28×28×192 input and now you have to pick a conv layer with specific size or pooling layer with specific size but instead of picking one, we will do them all by applying n_i filters of 1×1 & 3×3 & 5×5 & … & Max pooling filters, all of them, to the input taking into consideration that the output of each are of the same size (28×28) so we use same convolution in conv layers and we do padding in max pooling with stride 1 to make sure that all outputs are of same size then stack all outputs together. This style of constructing networks is doing well as we don't need to pick some sort of filters but letting all the parameters of all types of filters learn and do the combination they need for better performing network.



The problem is obvious here in the inception module which is the computational cost ☹ as it is so high compared to other architecture. So for an example if we just estimate the number of operations done in the 5×5 filters only so we just have 28×28×192 input and have 28×28×32 output so this means that we have 28×28×32 values each value is computed using 5×5×192 operations using the filter so this ends up to (28×28×32) × (5×5×192) =120 mil operations !!!

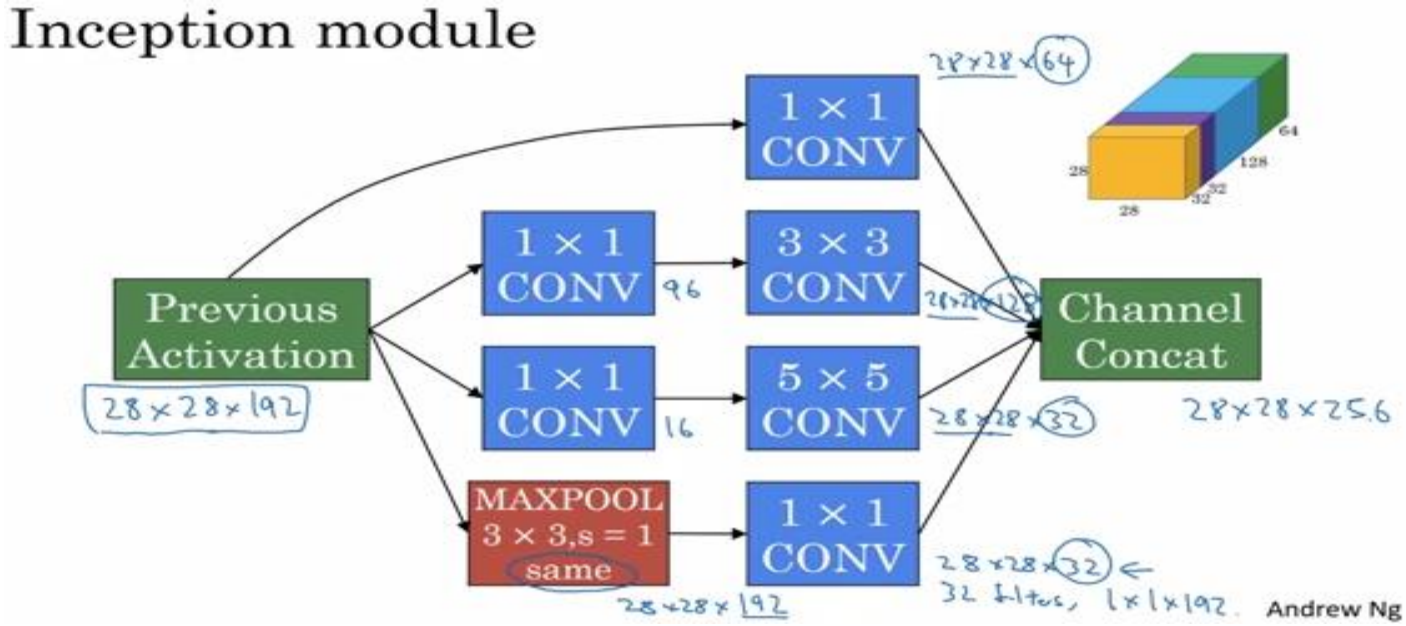
So here comes the idea of introducing 1×1 conv to perform 5×5 conv in about 1 tenth the operations needed in the original method (instead of 120 mil →12mil) . This will be done as shown : (left is the original method and right is the modified one to reduce computations)



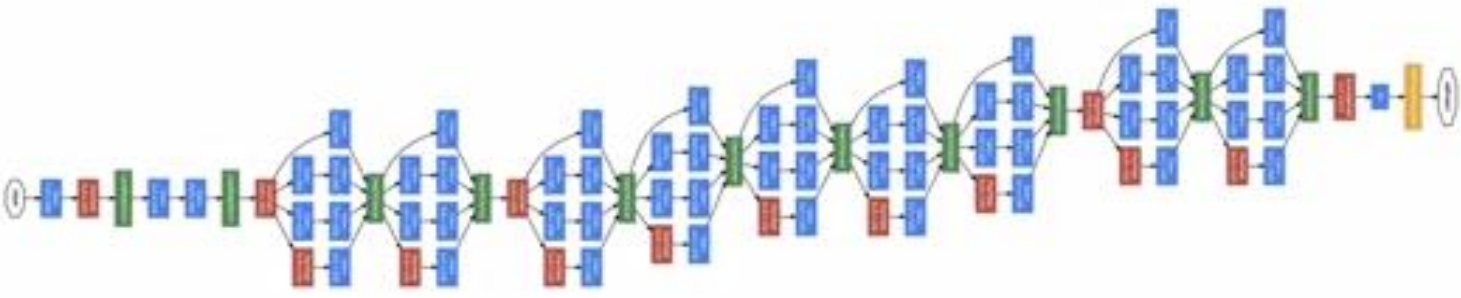
The new method still preserve the input and output dimensions but with less computations.
Computation operations: [(1×1×192) × (28×28×16)] + [(5×5×16) × (28×28×32)]=12.4 mil
So the computational cost decreased to approximately 1/10 of the operations needed in the original method with preserving the dimensions of the input and output.
Note: The layer added before the 5×5 layer is called bottle neck layer.

The last thing to say before showing the final inception module is that we have said before that 1×1 conv helps in shrinking number of channels which will be used after the pooling layer to decrease number of channels from 192 to any less number we need by using n number of filters each having size of 1×1×192.

Final inception module



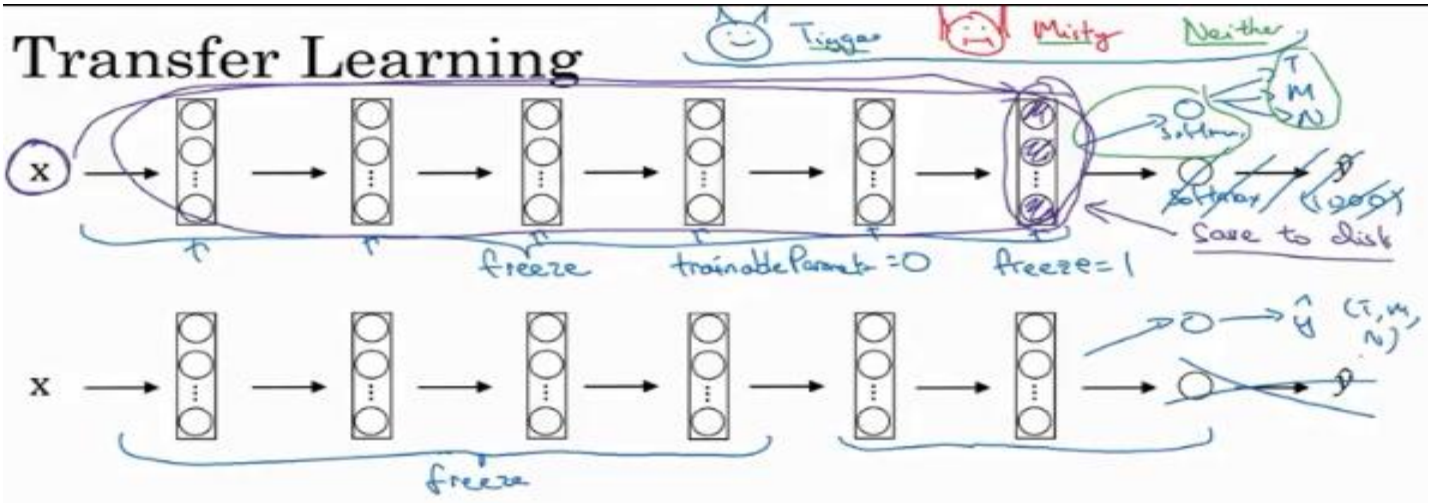
Now to construct an inception network we will add some inception modules to form it as follows:



Transfer learning

Rather than making your network learns from scratch with randomly initialized weights, you can make much faster progress if you started from good weights that somebody else has ended to after training his network (of course both should be using same architecture ☺). These weights are usually reached by training on well-known datasets as COCO, ImageNet or Pascal for weeks or may be months so starting from these weights is a very good initialization. Transfer learning is important in a case like having small dataset. In fact transfer learning is almost always used except if you have exceptionally large dataset that doesn't need this method.

Important Note: Change the number of classes of softmax layer to match your task and freeze all the layers of the network except for softmax layer and start training in case of small dataset but for larger dataset you can freeze some layers of network and train the others and as you have larger dataset you increase the number of layers to be trained such that if the dataset is large enough you can train the whole network so the pre-trained weights are just initialization to the network.



Remember: For small dataset, data augmentation is a good option for increasing your data as discussed before.