Week #2:- Deep Convolutional Models: Case Studies

Case Studies:- This week we will look at some case studies where CNNs worked better.

- ## Why Case Studies?

  - Good way to see or get intuition about implementation of ConvNets is by looking at other people's work.

Classic Networks:

- LeNet-5
- AlexNet
- VGG

} Revolutionized modern DL in CV. You'll see that, these ideas will probably be useful for your own application.

⇒ ResNet ⇒ Residual Networks (152 layers)

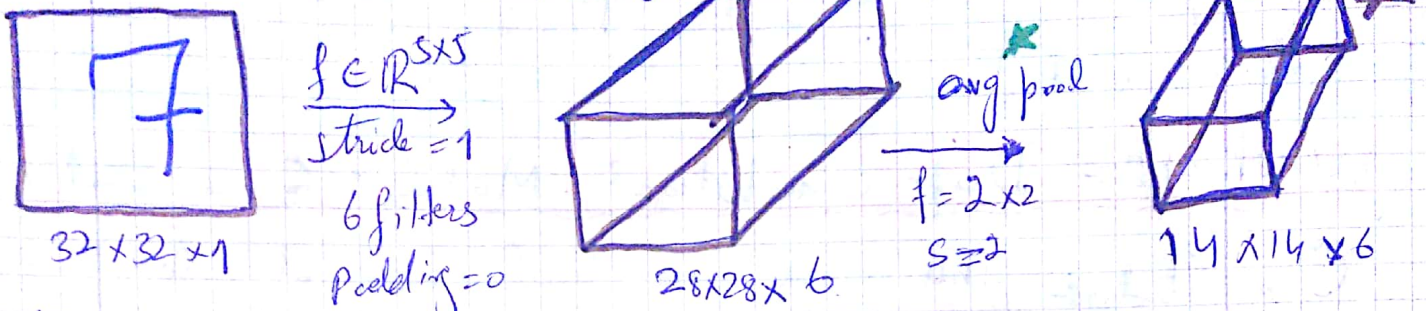
⇒ Inception Neural Network.

Classic Networks:-

Let's go over some classic neural network architecture.

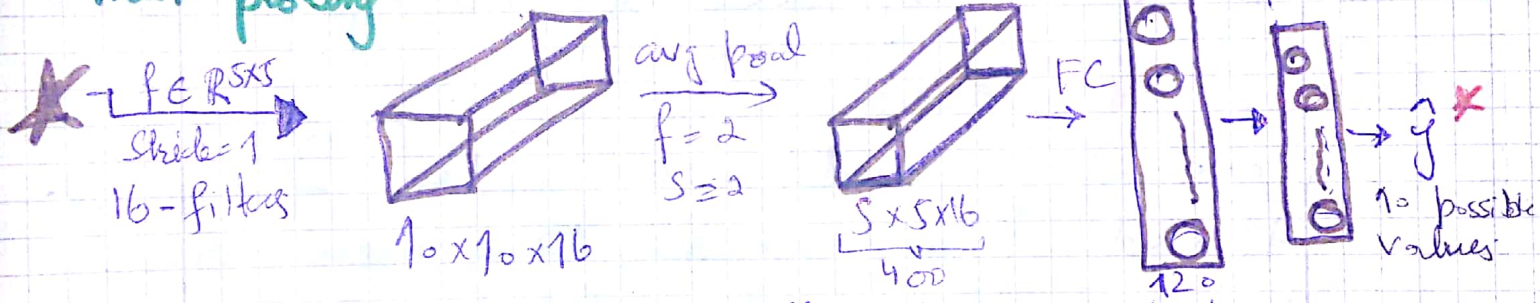
LeNet - 5 :-

Used to recognize hand written digits.



trained on grayscale

* back then when it was developed, developers used average pooling, now we will use max pooling.



→ When this paper was written, people didn't use padding that's why everytime you apply convolution the dimension shrinks.

* Modern version of this architecture would use softmax layer as an output.

→ As we go deep into the network, height and width tend to decrease while the number of channels increase

→ Pattern in this architecture somewhat resembles the modern ones. like

Conv → Pool → (1 or more) Conv → Pool → fc → fc → o/p

Paper:- Gradiant-based learning applied to document recognition. [LeCuN et al., 1998].

→ Back then when this paper was written people were using Sigmoid and tanh non-linearities and weren't using Relu non-linearities

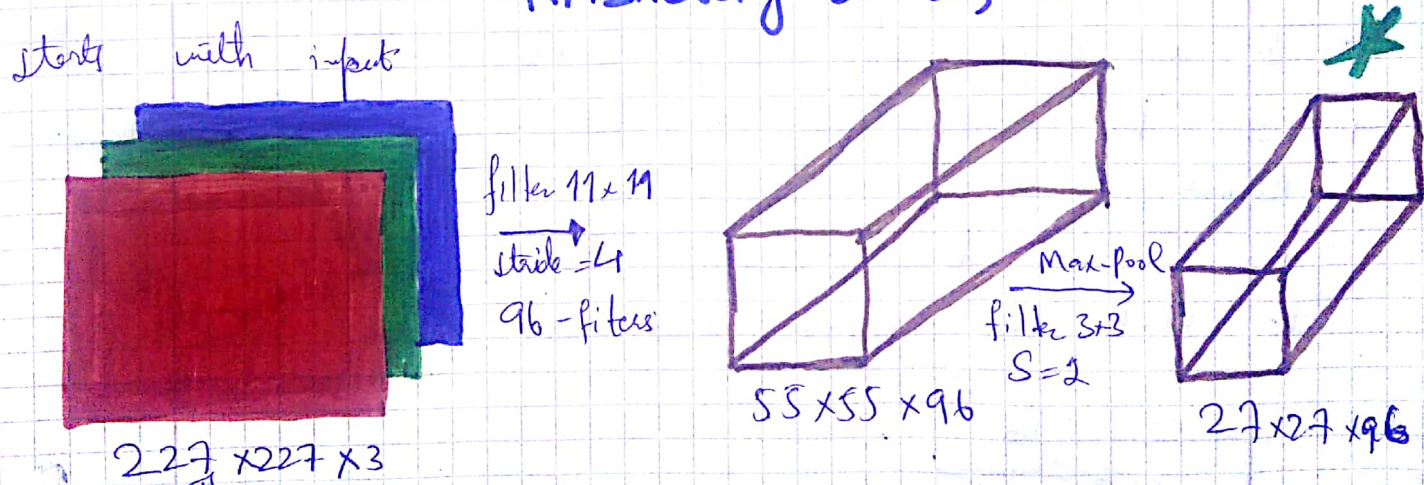
→ There were some funny wiring in the network if you look at the paper (atleast by modern standards).

e.g now we use $(f \times f \times n_c)$ filter for $(n_H \times n_w \times n_c)$ network where every filter looks at every one of these channels. Back then computers were much slower so, different filters were used assessing different channels of the input block.

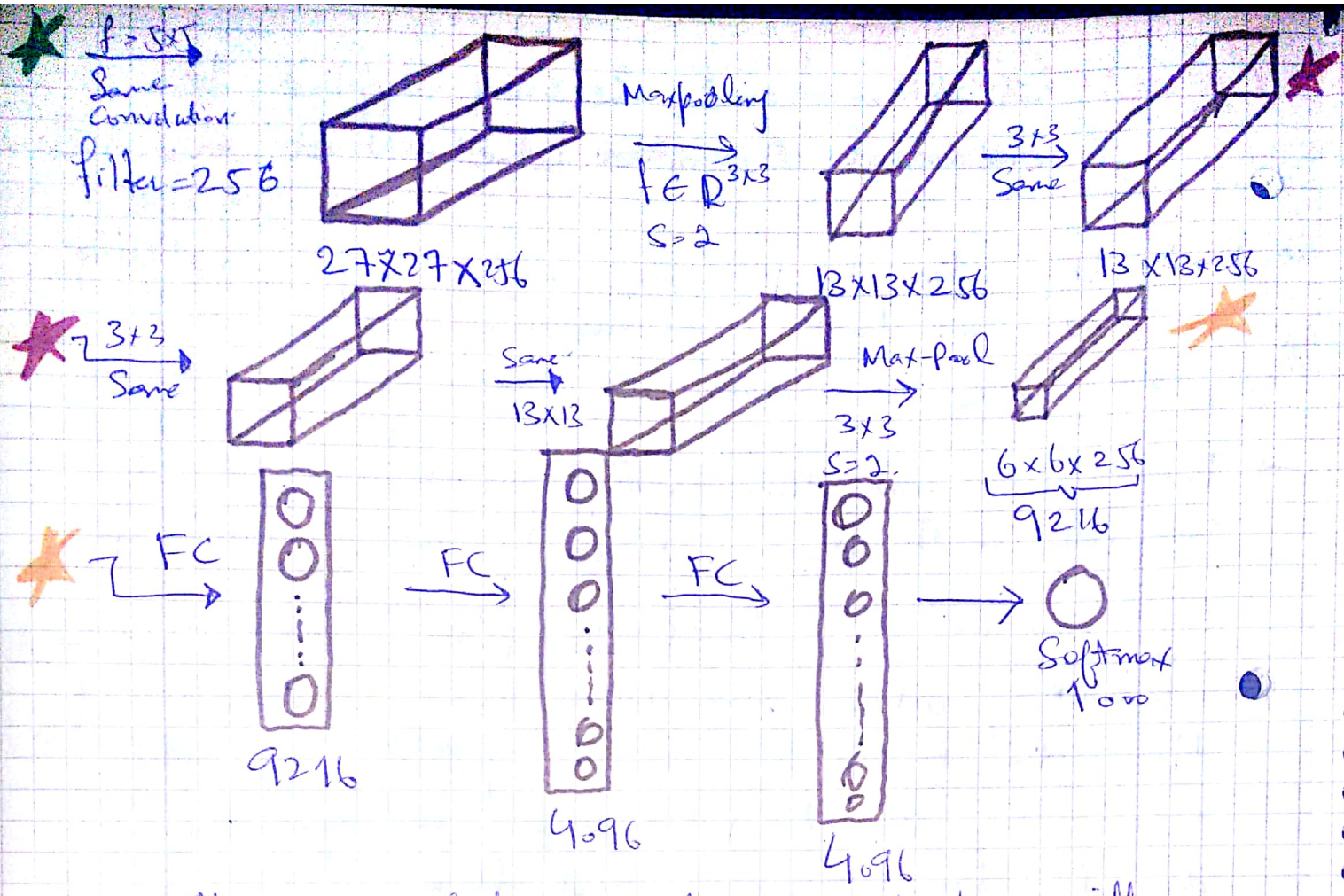
One more detail:-

Original LeNet-5 had non-linearity after pooling. It uses Sigmoide non-linearity after the pooling layer.

AlexNet:- ImageNet classification with deep Convolutional neural networks. Krizhevsky et al., 2012.



Paper refers to 224 x 224 x 3 img. But 227 makes sense.



→ This NN had a lot of similarities with LeNet, but it was much bigger.

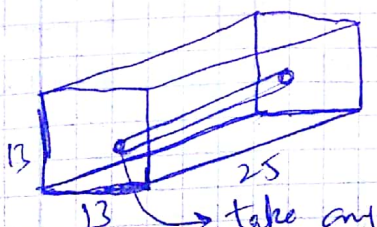
→ AlexNet had 60K parameter, this one had 60M parameters.

→ One aspect that made it much better to LeNet was using **ReLU non-linearity**.

→ It was trained on multiple GPUs.

→ It had another type of layer called **Local Response normalization** → Not used much.

But the basic idea is that with anyone of the block take anyone like let's take 13x13x256.



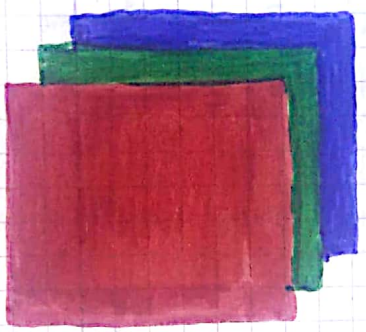
→ take any of the number in nH and nW book all the numbers in that row and normalize them.

→ Motivation for this is that, for each neuron may be you don't want high activation for some. But this doesn't help much

* This was the paper which motivated DL community to take this area seriously.

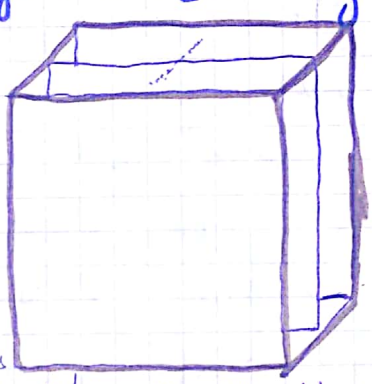
→ Complex Network, lots and lots of hyperparameters.

VGG-16:- Very deep CNNs for large-scale image recognition. [Simonyan & Zisserman] 15



224 x 224 x 3

first 2 layers → filters = 64 [Conv 64] x2 means 2 layers of 64 filters Convolution



224 x 224 x 64

Pool → f=2

112 x 112 x 64 *

Remarkable thing:- Instead of having a lot of hyperparameters, let's use a simpler network with Conv f = 3x3 and padding always the 'Same' i.e 1. and all Max pooling layers → $f \in \mathbb{R}^{2x2}$ with stride of 2.

* → [Conv 128] x2 Same → 112 x 112 x 128 → f=2 Pool → 56 x 56 x 128 → [Conv 256] x3 → 56 x 56 x 256

→ f=2x2 Pool → 28 x 28 x 256 → [Conv 512] x3 → 28 x 28 x 512 → f=2 Pool → 14 x 14 x 512

→ [Conv 512] x3 → 14 x 14 x 512 → Pool → 7 x 7 x 512 → FC → FC 4096 → FC 4096 → Softmax 1000

VGG-16 → refers that it has 16 layers that has weights.

→ 138 M parameters

→ Relative uniformity makes it quite appealing.

→ Main downside is that, it has quite large number of parameters.

ResNets:- [Residual Networks].

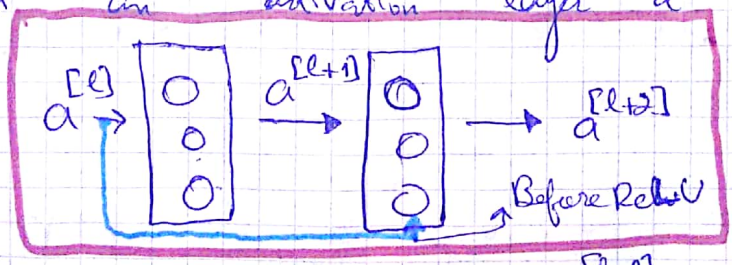
→ Very deep neural networks are difficult to train due to vanishing and exploding gradients.

→ Let's go through the process of **Skip Connections** which allows you to take the activation from one layer and suddenly feed it to another layer even much deeper in the neural network and using that to build ResNets which enable you to train very deep neural networks, sometimes even networks of 100 layers.

→ **Resnets are built over residual block.**

Residual Block:-

Here are 2 layers of N.N where you start of with an activation layer $a^{[l]}$



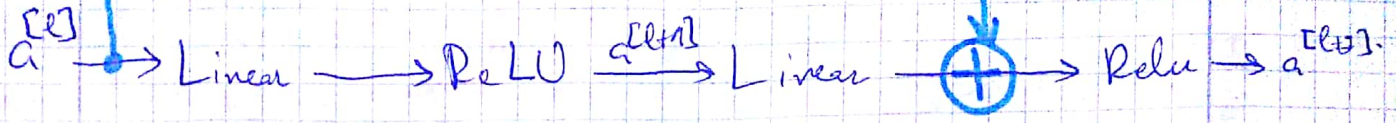
$a^{[l]} \longrightarrow$ linear operator $\longrightarrow$ Relu $\xrightarrow{a^{[l+1]}}$ linearity $\longrightarrow$ Relu

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

In order to go from $a^{[l]}$ to $a^{[l+2]}$, we'll have to go through all of aforementioned steps.

→ In residual network, we are gonna make change to this.

"Short cut"/Skip connection



So that means for the last layer instead of using ~~$a^{[l+2]} = g(z^{[l+2]})$~~ we are gonna use:

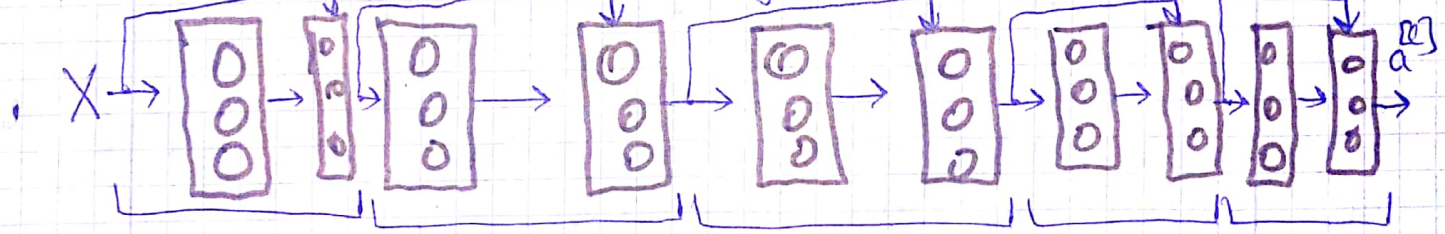
$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

Residual block.

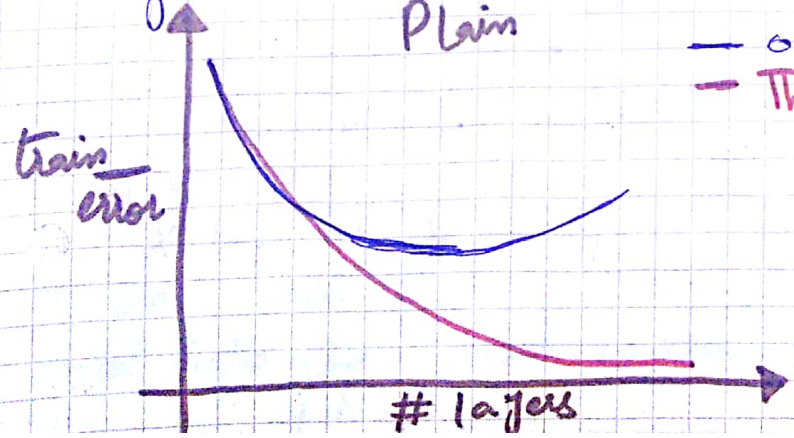
→ What developers of ResNets found is that, using resnets allow you to train very deep neural networks

→ The way you build ResNets is that, you take many of these residual blocks and stack them together to form a deep network.

Let's look at the following (not residual but "Plain network"



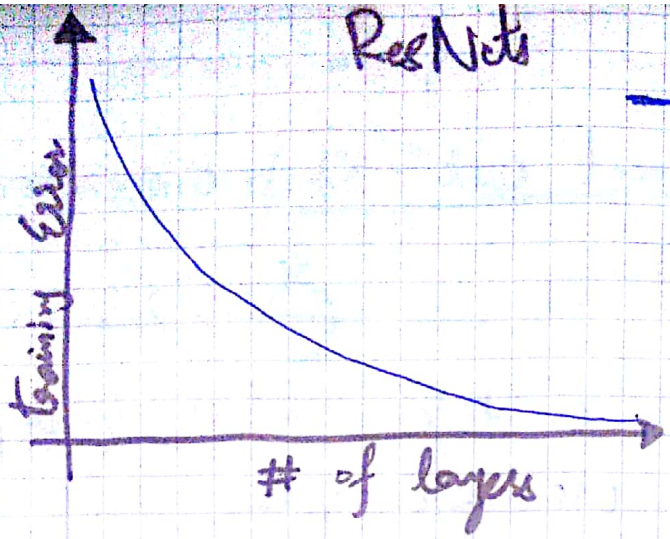
→ In order to turn this into residual networks you add all those Skip Connections.



— original
— Theoretical

Optimization algorithm will have much harder time reducing the cost for very deep network.

ResNets



- Reality

* ResNets' skip connection really helps with Vanishing and exploding gradients and allow you to train very deep networks.

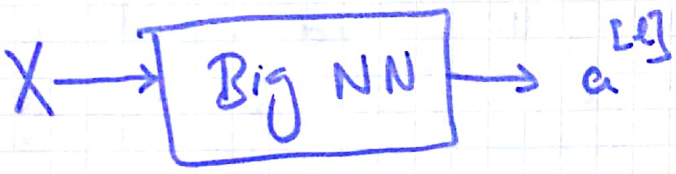
Why ResNets Work?

Let's see an example :-

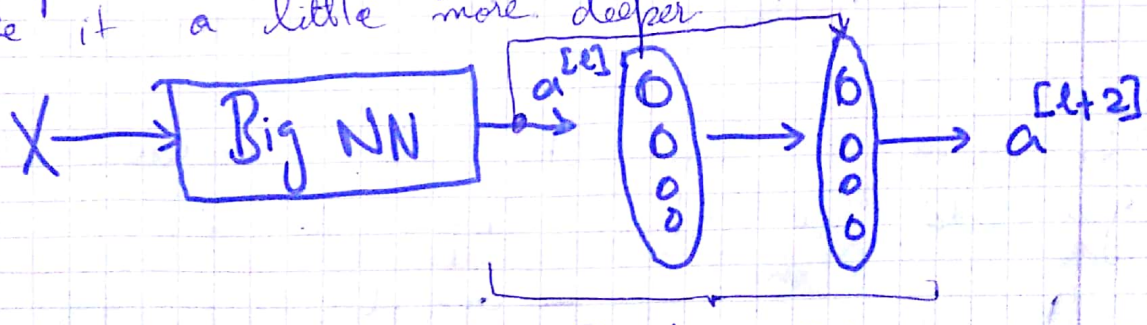
→ As we have seen that, if you make a network deeper, it can hurt your ability to train the network to do well on the training set.

→ But, this is not the case when you are training a ResNet.

e.g.:-



For example, let's say you modify the neural network to make it a little more deeper



all the activations are ≥ 0 cause we are using Relu, with the possible exceptions of the input X.

Let's look at $a^{[l+2]}$.

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

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

→ If you are using L2- regularization or weight decay that would tend to shrink the value of $w^{[l+2]}$.

→ If you apply weight decay to b, that will also shrink then.

• But W is the term to pay attention here.

Suppose $w^{[l+2]} \approx 0$ and $b \approx 0$ so

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

because we are using ReLU activation function here.

→ This means that, identity function for residual block is easy to learn, and it is easy to get $a^{[l+2]} = a^{[l]}$ because of this skip connection.

→ So what this means is that, adding block to your neural network doesn't really hurt your neural network's ability as well as this simpler network without these two extra layers, because it is easy for it to learn the identity function.

→ So, adding this block in the middle of your big network doesn't really hurt your performance.

→ But our goal is not just to "not hurt performance" but help these units to learn something better rather than identity.

→ What goes wrong in these very big nets (Plain) is that, it is actually very difficult for the network to choose parameters that learn even the identity function which is why a lot of layers end up making your result worse rather than better.

→ I think, the main reason ResNets work is that, it can learn identity function which doesn't hurt performance and a lot of times you get lucky and it even helps performance.

In ResNets our baseline is considered as "not hurting performance" and optimizer can only improve from there.

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

We are assuming that, $z^{[l+2]}$ and $a^{[l]}$ have the same dimensions, that's why there is a lot of usage of "Same" convolution.

→ Suppose $z^{[l+2]}$ and $a^{[l]}$ have dimension clash then you can add another matrix 'Ws' as a multiple with $a^{[l]}$

Suppose $z^{[l+2]} \in \mathbb{R}^{j}$ $a^{[l]} \in \mathbb{R}^{x}$

then $W \in \mathbb{R}^{j \times x}$

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

W_s can be learned or remain there as a fixed matrix which adds zero padding in $a^{[l]}$.

Paper: Deep residual networks for image recognition. [He et al]

→ Add residual layers to plain network. [Look up this paper online].

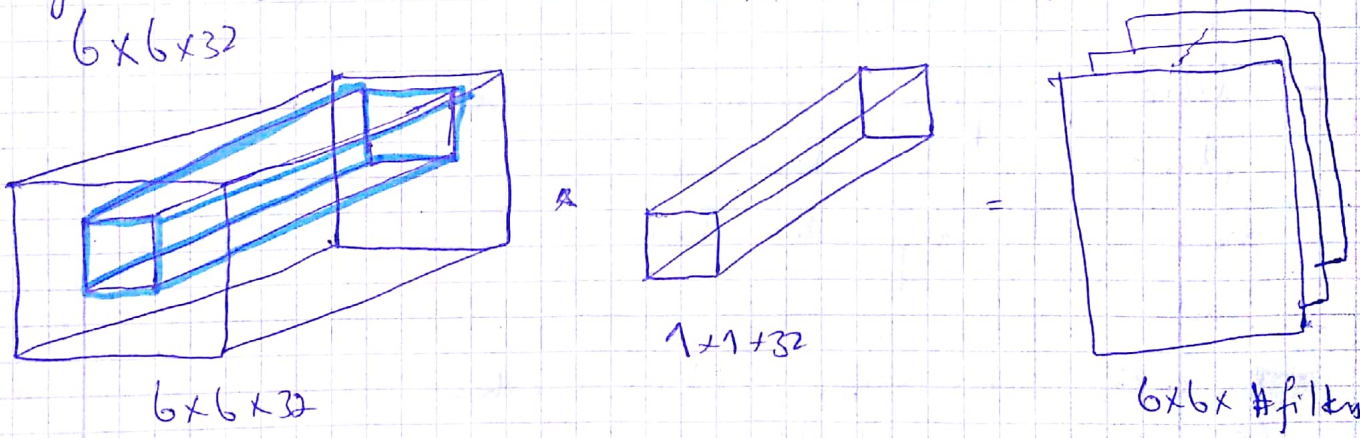
→ whenever there is fading dimensions are gonna change then add 'W_s' there as a multiple of 'a' with W_s.[1]

Network in Network and 1×1 Convolutions:-

→ In terms of design ConvNet one of the ideas that might help is 1×1 Convolution.

What does a 1×1 Convolution do?

6×6 image convolve with 1×1 filter is just like multiplying 6×6 image with a number. But imagine if we had an i/p block of 6×6×32



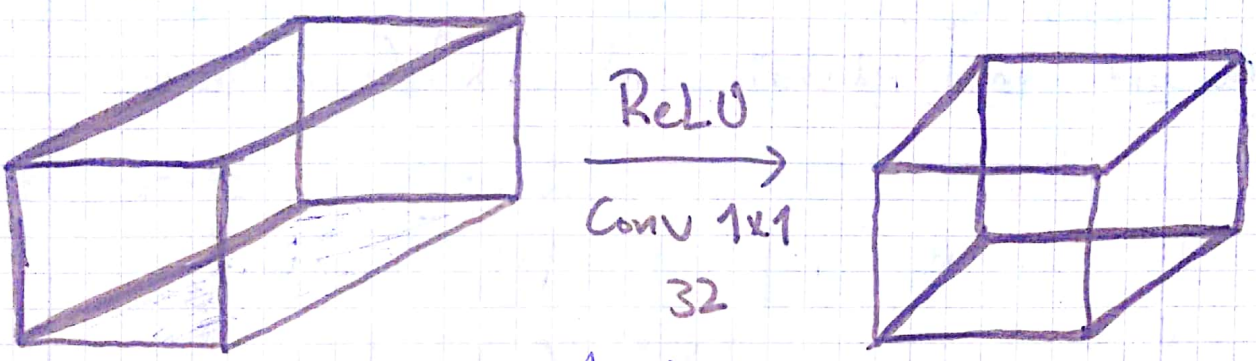
So, in this case multiply element wise and add them to get a raw number.

→ It can be thought of as one neuron taking 32 number processing it applying ReLU and output non-linearity.

This idea is sometimes called Network in Network.

Here is an example when 1×1 Convolution is useful

if your height and width of image get too big you can use pooling what if channels increase, then use 1×1 Convolution



Inception Network motivation :-

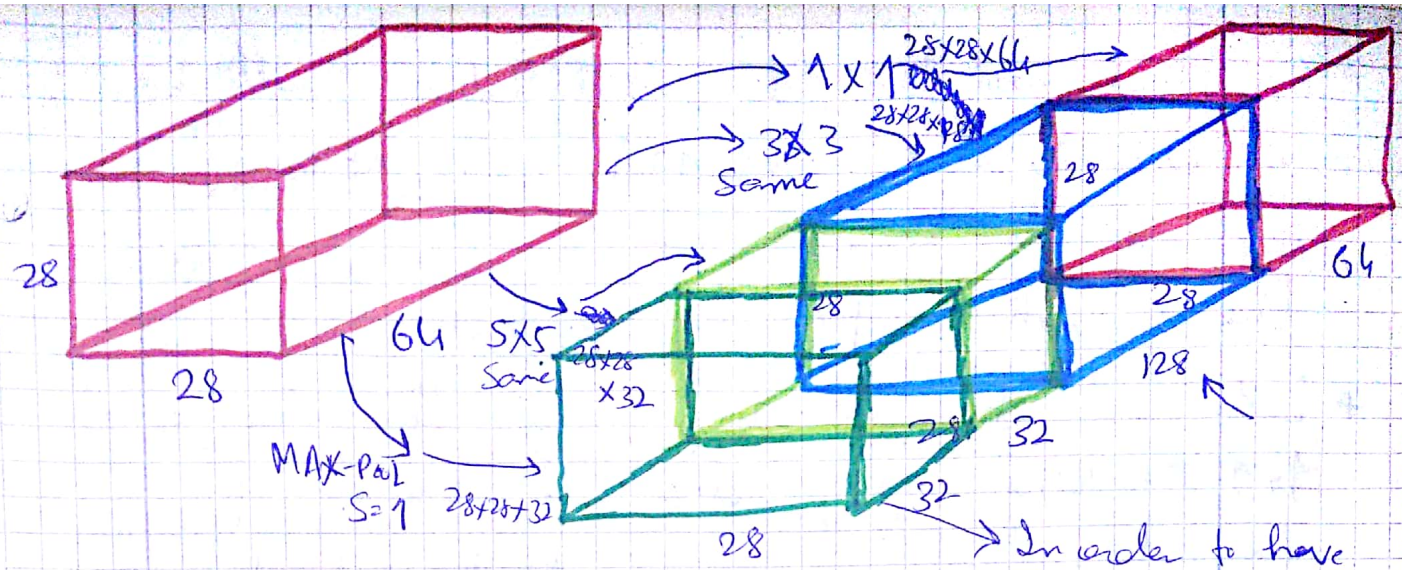
When designing a ConvNet you have to pick which filter you want, 3×3, 5×5 ----- e.t.c

→ What inception network suggests is that, why don't you try them all, this makes network more complicated but it works remarkably well.

→ Lets see how this works.

(Paper: Szegedy et al. 2014. Going deeper with convolutions)

Suppose, we have 28×28×192. Now what filter size to use, whether to do Convolution or pooling, inception Network: Well let's do them all and see which one works best.



→ You had 28x28x64 i/p and you o/p 28x28x256 by trying different things. This is the heart of Inception Networks.

"Try them all and stack up the results then see and let the network learn whatever combination it want to go with.

- There is a problem with inception networks and that is,

Computational Cost.

From above example let's figure out what problem the 5x5 convolution could pose interms of Computational Cost.

28x28x192 → (CONV 5x5, same, 32) → 28x28x32

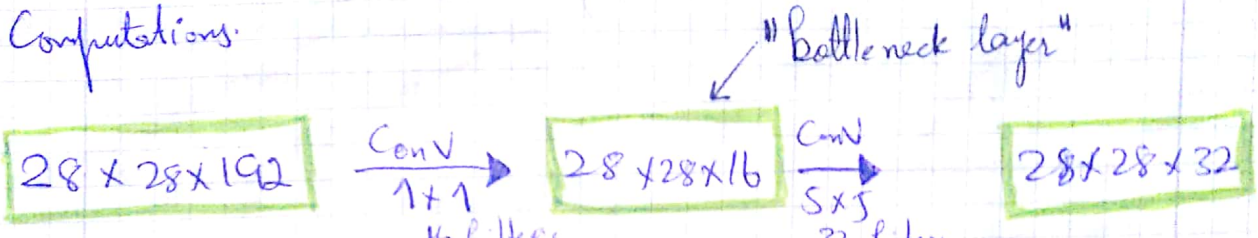
→ You have 32 filters each filter is 5x5x192

You need to compute 28x28x32 numbers. for each of them you need to do 5x5x192 multiplications.

So, total multiplications will be
$(28 \times 28 \times 32) \times (5 \times 5 \times 192) \approx$ 120 Million.

Seems like a normal task for today's computer this is still a pretty big operation.

→ Let's see how can we utilize the idea of 1×1 convolution and reduce the amount of computations.



Observe that the i/p and output are still the same as previous example.

Let's look at the computational cost involved.

For computing bottleneck layer
$(28 \times 28 \times 16) \times (1 \times 1 \times 192)$ multiplications = 2.4 M.

How about the second layer
$(28 \times 28 \times 32) \times (5 \times 5 \times 16)$ multiplications = 10.0 M.

So Total Multiplications are 12.4M.

Woo hu! (:P), Right ;)

from 120 M → 12.4 M round about 90% decrease

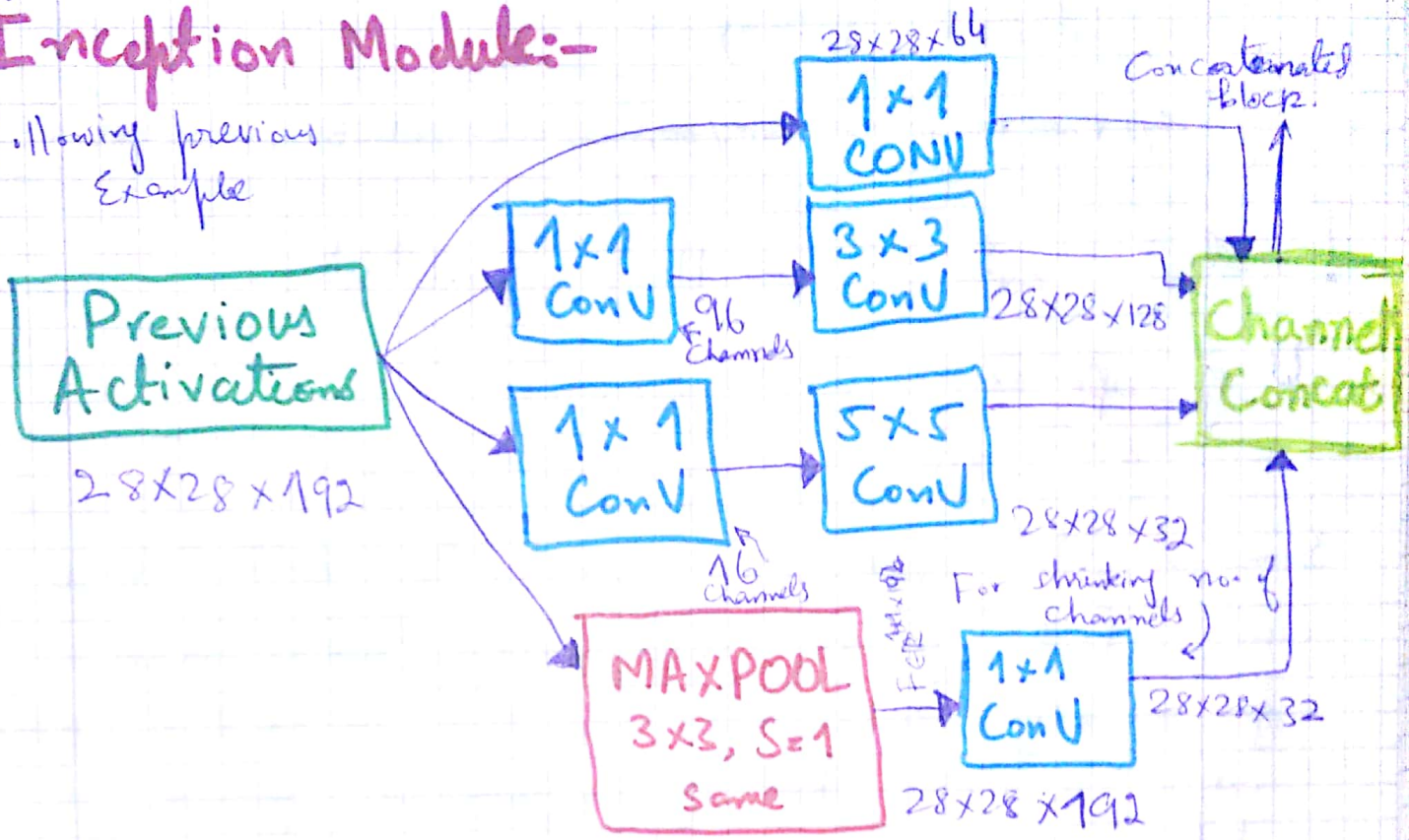
→ You must be thinking that, whether your performance remains the same or not by shrinking the size so drastically. It turns out that, so long as you implement this strategy reasonably it doesn't hurt your performance.

Inception Network:-

We have seen the building block for Inception network, let's see how can we utilize all of them to build a full network.

Inception Module:-

following previous Example



This is one inception module.

What inception network does is that, take a lot of these modules together.

In full big Inception Network, along the network to the output, in between we by to Compute/predict the output through the layers in between.

→ Just another detail of Inception network. What it does is that, it helps ensures that the feature Computed even in the intermediate layers, that they are not too bad for predicting the output Cost of the Image.

This appears to have regularization effect which helps prevent this layer from overfitting.

Where does the word Inception come from?

Have you seen "Inception" ;). I think you can understand lol :)

http://knowyourmeme.com/memes/we-need-to-go-deeper

Practical advices for Using ConvNets:-

Using open-source Implementation:-

git clone "_____"

Some of the open source material's authors have used high power GPUs and massive datasets for training their models. So, you might not have access to that kind of computational power or data, so replicating the network might do the trick for you (Transfer Learning).

Transfer Learning:-

If you are building a CV application, it is often a good way to start by looking someone else's implementation and use those parameters/hyperparameters in order to sort your work and start learning from there.

Let's take a deeper look and start with an example:-

Let's say you are building a Cat classifier. According to internet Tigger and Misty are two common Cat names.

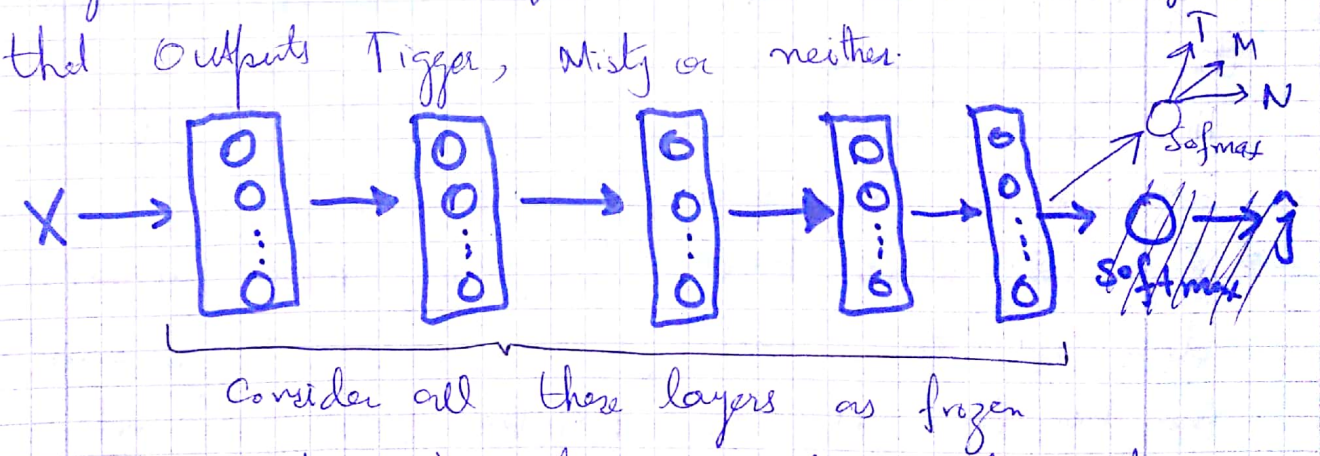
→ You have a classification problem with 3 classes, i.e. is it a Tigger, Misty or None in the image?

→ Now probably you don't have a lot of pictures of Tigger or Misty, so your training data would be small, now what can you do?

→ Go online and download some open source implementation of Neural Networks, not just the code but also the weights.

There are lot of online NN architecture which you can download eg image net data has thousand different classes so network will have 1000 softmax outputs.

→ what you can do is that, get rid of that layer and create you own softmax layer that outputs Tigger, Misty or neither.



→ only just train parameters associated with softmax layer.

→ By using some one else's work, you might be able to get pretty good performance even with small dataset.

→ Deep learning frameworks support this sort of things. like by setting trainable parameters = 0, freeze = 1 you specify whether or not to train the weights associated with a particular layer

→ **One neat trick** that might help with some implementations is that, because all of these early layers are frozen, there is some fixed function that doesn't change because you are not changing it, so, it can be seen as you input any image and maps it to output Softmax layer

- One thing that could speed up your training is that, Just compute the values for the layer before Softmax for all the training examples you have then start training it.
This way you won't need to recompute those activations on every optimization iteration.

→ **On the other hand:-** If you have massive data set then what you can do is freeze some layers and train the rest or keep the frozen layers and reject the rest and attach your own setting of layers there.
Either ways are fine.

Bottom Line:- If you have big enough dataset, no. of layers frozen could be smaller and unfrozen greater and the opposite is true if you have small dataset.

→ For big enough dataset you can also train the whole thing.

Data Augmentation:-

- Most computer vision tasks could use more data.
- Data augmentation is one of the techniques to improve the performance of computer vision system.
- This is true regardless of the approach you are following i.e. ↑ Transfer learning or building everything from scratch. either

→ Most common method is mirroing the image.

- Another is Random cropping
  - ↳ Won't work in some cases, you have to make sure that cropping settings are legible.

→ Rotation.
→ Shearing.
→ Local warping.

} In practice they are used less, perhaps due to the complexity.

Color Shifting:

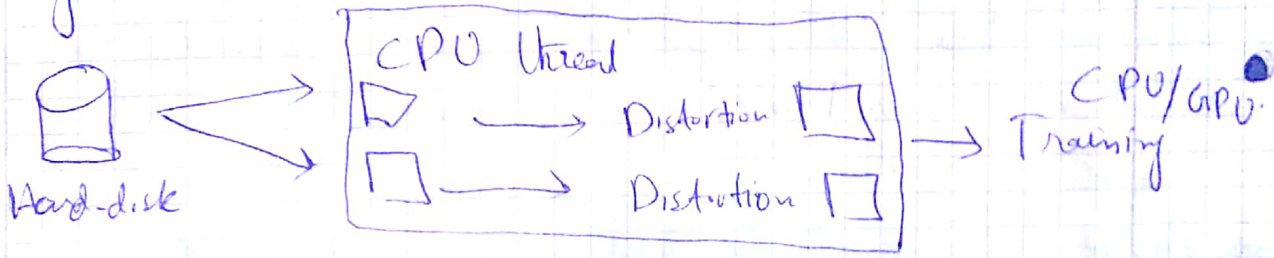
Let's say you have an image and you add some distortion to every channel like [(R+20), (G-20), (B+20)]
You can set other settings like [(R+5), G, (B+50)].

→ In practice these values are drawn from some probability distribution.

There are different ways to tample R, G, and B.

One of the algorithm is called PCA ⇒ Principle component analysis.

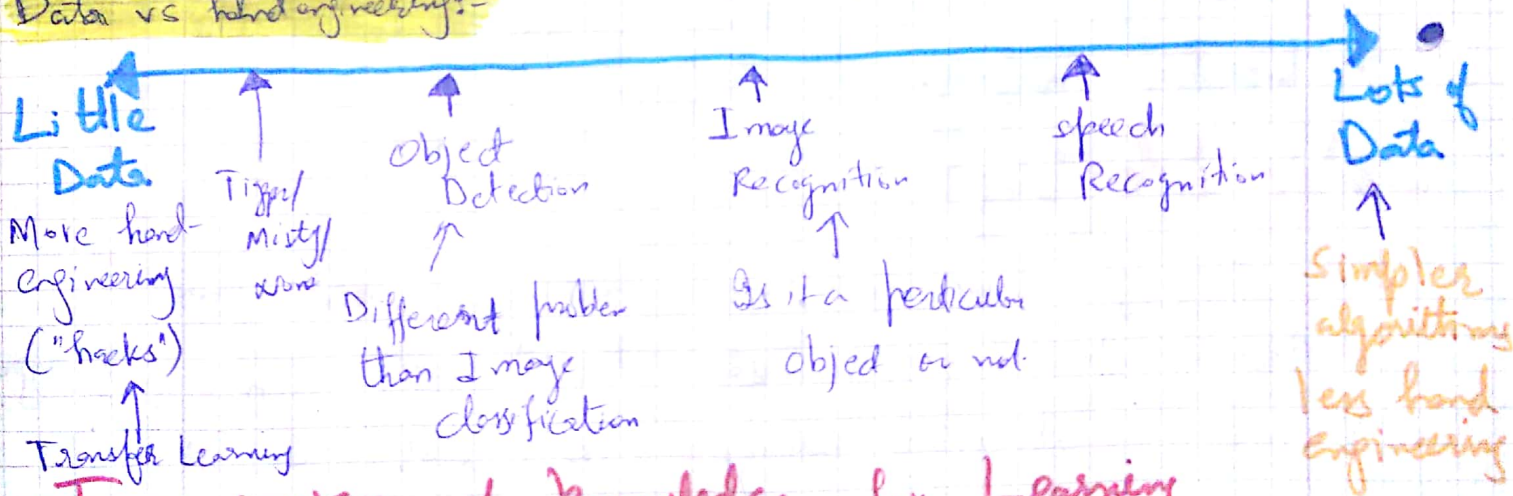
PCA Color augmentation ⇒ Rough idea is that, if your image is mainly purple ie mainly red or blue pixels then PCA will add or subtract a lot of Red or blue images in order to give overall color the same.



State of Computer Vision:-

Deep learning has a lot of applications, but there are few things about deep learning that are unique just for Deep learning, let's see.

Data vs hand engineering:-



Two sources of Knowledge for Learning Algorithm:-

- Labeled Data (x, y)
- (Hand engineering / network architecture / other components).

→ Since we don't have much data for CV applications that's why we use more complex networks.

Tips for doing well on benchmarks / winning Competitions

- **Ensembling:**
  ↳ Train several networks independently and average their outputs (not weights) augments 1-2% performance.

- **Multi-Crop at test time:**
  ↳ Run classifiers on multiple versions of test images and average results.
  → It is a form of providing data augmentation at test time to your test images.
  like 10-Crop Suppose you have an image and its mirror reflection. take 1 each image of a cropped part from both. Now take 4 images each from both of them with a pre-set setting from 4 sides. That makes up your 10 images.
  Right top, bottom and left top, bottom
  Run them through your classifier then average the results.

→ These tips are used for benchmarking not in production.

Use open source Code:-

In CV. application, transfer learning works.

- Use architecture of networks published in the literature
- Use open source implementations if possible.
- Use pretrained models and fine-tune on your dataset.
  ↳ Because lot of little details like learning rate, decay rate are tuned.