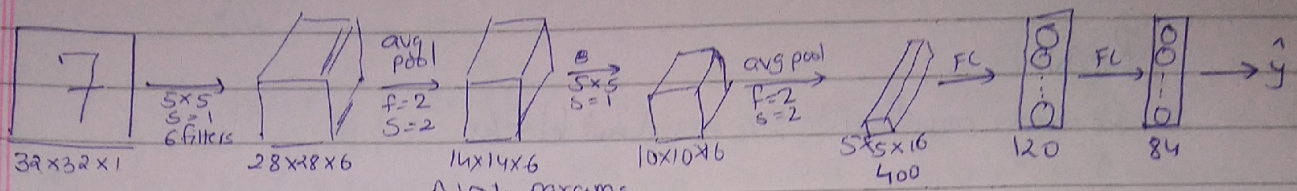
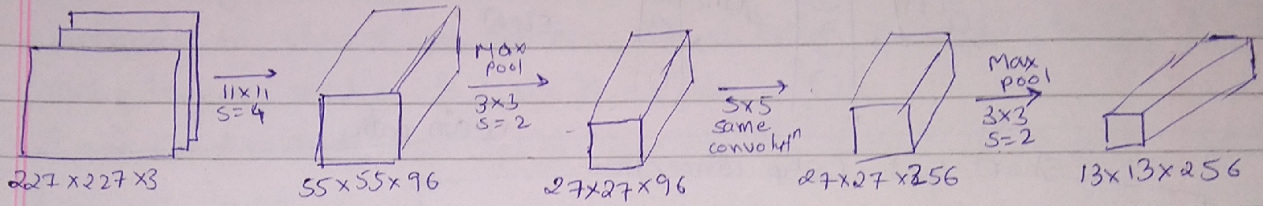
Week-2 Classic Networks

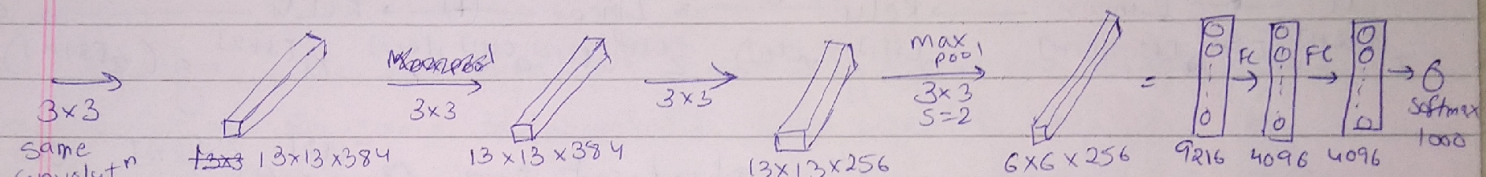
LeNet-5



~10k params

AlexNet

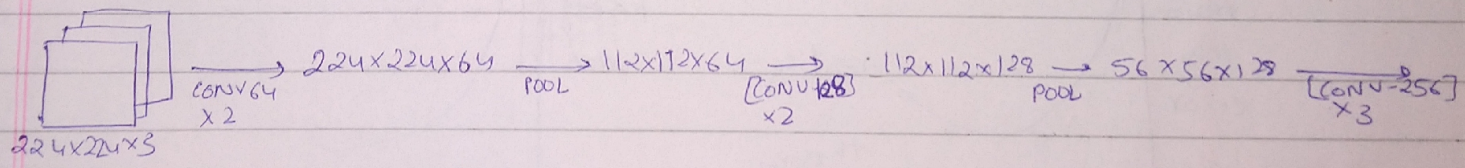


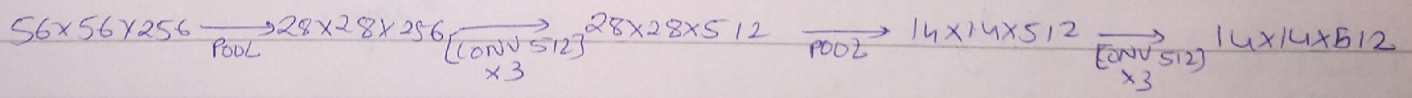


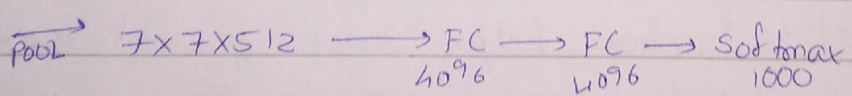
~60m params

VGG-16

CONV = 3×3 filter, S=1, same Max Pool = 2×2, S=2







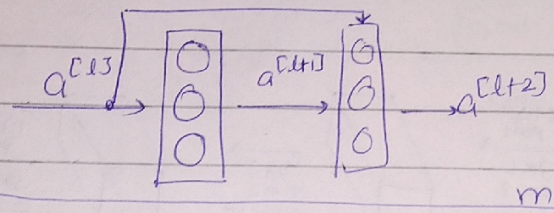
~138M parameters

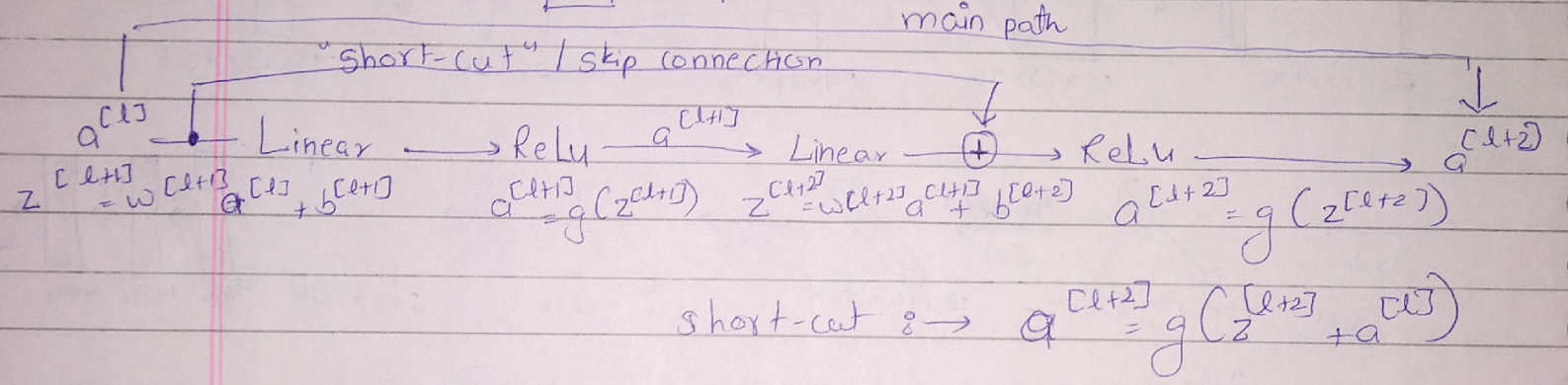
Residual Networks (ResNets)

Very big NNs are difficult to train, due to vanishing and expanding gradient descent problem.

ResNets

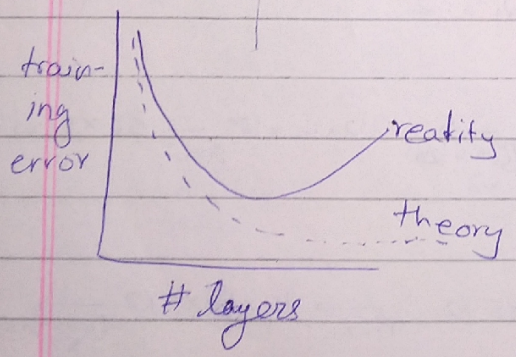
Resnets are built on residual blocks

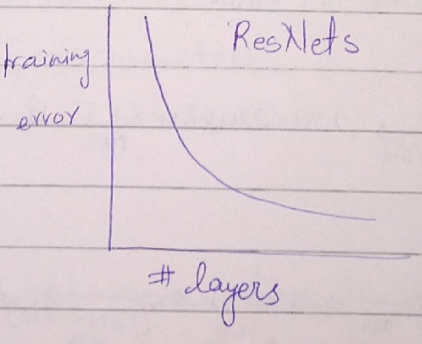




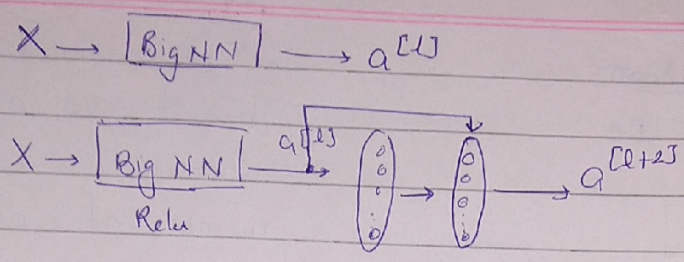
short-cut :→ $a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$

In theory, training error should decrease as we make our NN bigger





This skipping of connections helps solve gradient descent problems. But ResNets are not ~~very~~ great at very big NN



Identity f^n is easy for Residual block to learn.

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

due to L2 regularizat^n, $w^{[l+2]}$ and $b^{[l+2]}$ will be close to zero.

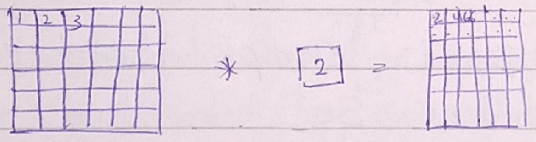
Reason why ~~of~~ big (deep) NN don't work, is because, it is very difficult to choose parameters that learn even the identity f^n. And so, many layers make the result worse.

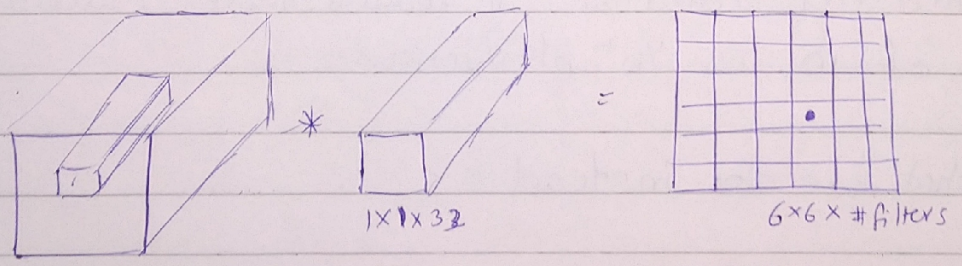
Since ~~red~~ resNets can learn identity f^n easily, it is guaranteed that (extra layers) they wont hurt.

Here, we assume that $z^{[l+2]}$ and $a^{[l]}$ have same dimension. So, a lot of ~~to~~ same ~~v~~ Convolut^n's are used in resNets

In case they are not of same size, one could add W_s matrix to balance ~~the~~ it. (Padding is used.)

1×1 Convolution





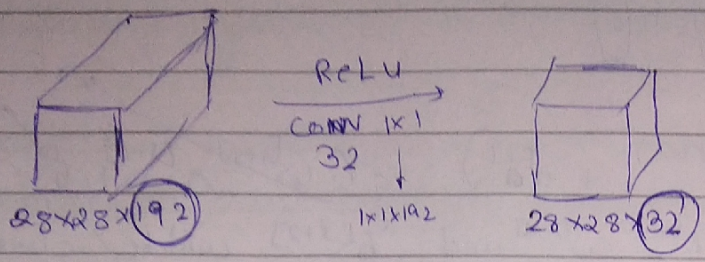
each element of 36 (channel) matrix is multiplied with each chanel element of 1×1×32 and then summed to get a value on o/p

Called One by One Convolution or Network in Network

Using 1×1 Convolutions

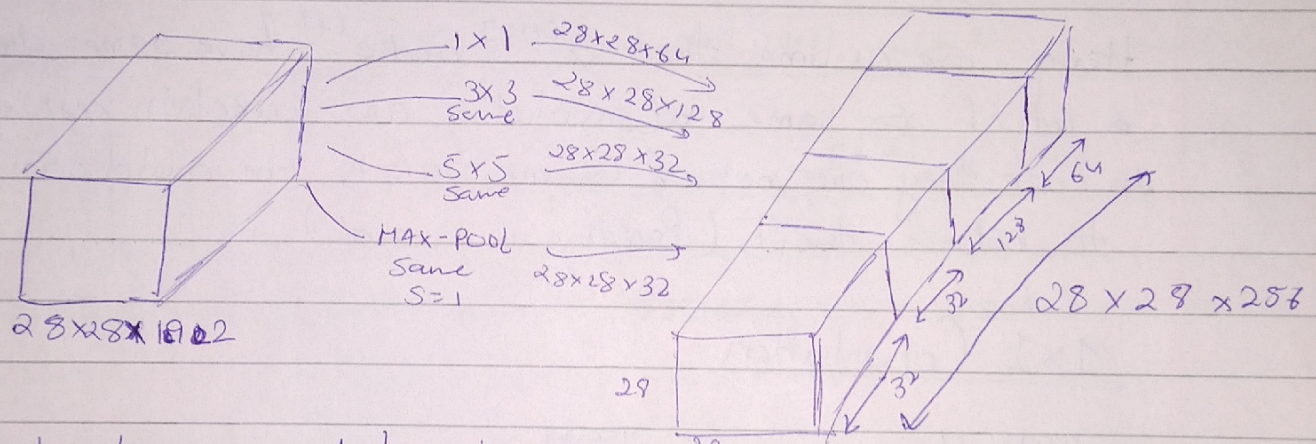
In order to shrink nc

Thus, it can help to increase or decrease, number of channels.



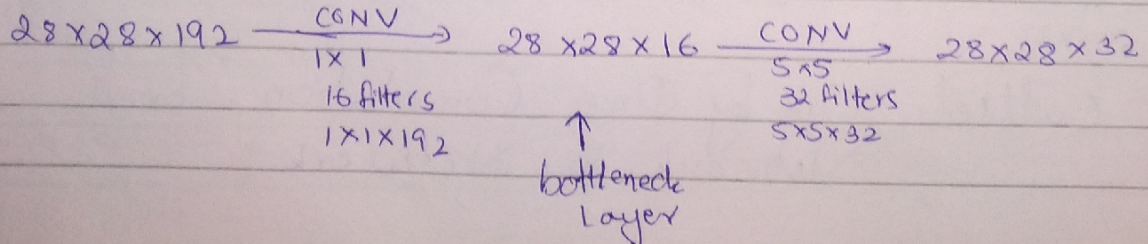
Incept^n Network

When you are confused which layer to use, you use all, and stack them up. This stacked thing is your i/p to model. so that the model can choose whichever parameters it wants to use.



But, the problem here, is computational cost. For eg., for 5×5, we have 32 filters. So, the total no of multiplicat^n's done will be $28 \times 28 \times 32 \times 5 \times 5 \times 192 = 120M$.
But in 1×1 convolut^n, it reduces these calculat^n by a factor of 10. i.e 1/10th of 120 Million.

So, what we do instead is:



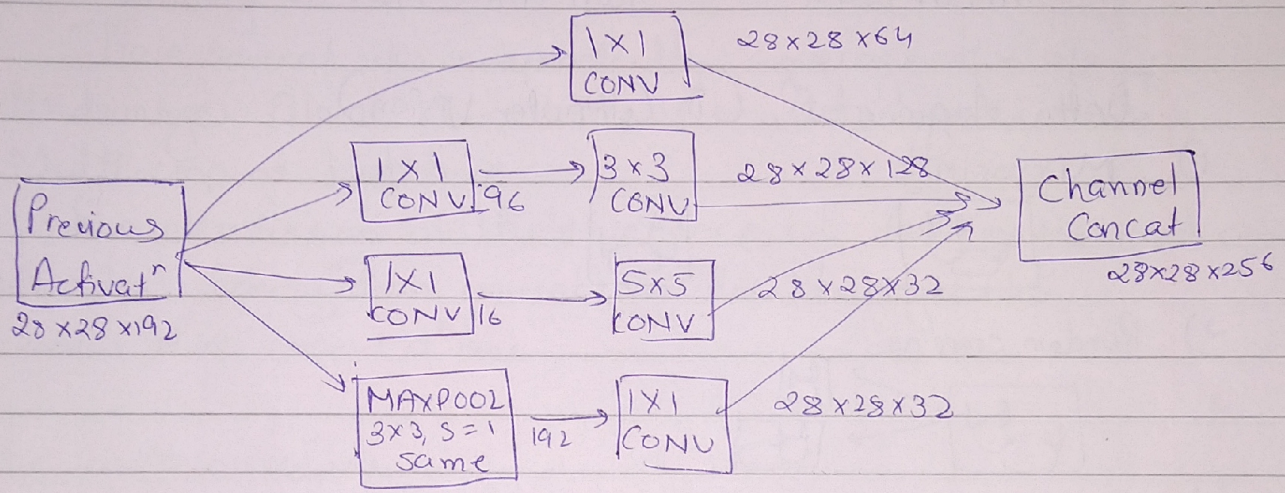
Computation cost:

$28 \times 28 \times 16 \times 192 = 2.4$ Million

$+ (28 \times 28 \times 32) \times (5 \times 5 \times 16) = 10.0$ Million $\Big\} =$ 12.4 Million

This shrinking does not hurt the ~~layer, in~~ performance but does save a lot of computation.

So, what we actually do is:



These inception blocks can be placed inside the n/w.

There are certain side branches to the inception blocks. These are actually softmax o/ps they try to predict the o/p ~~layer~~ label. They are present in hidden layers. Thus, even hidden layers have a ~~chance to~~ & contribut^n to o/p. This appears to have a regularizing effect on incept^n n/w and helps prevent overfitting

Transfer learning

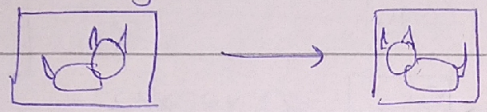
More the data, more the layers you freeze.
By freeze, we mean to not change any parameters of that already trained layers.

data size	what to do
small	remove last 1-2 layers, add yours & train yours
~~more~~ big	" " few " add yours & train yours
lot of data	train entire n/w. Previously ~~use~~ work as initialization

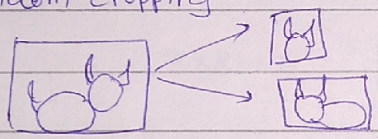
trained weights

Data Augmentation (in Computer Vision)

1) Mirroring



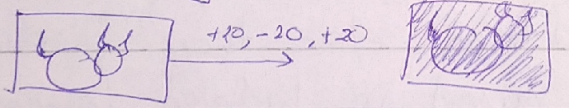
2) Random cropping



3) Rotation
4) Shearing
5) Local warping

} Not used in practically world

6) Color shifting

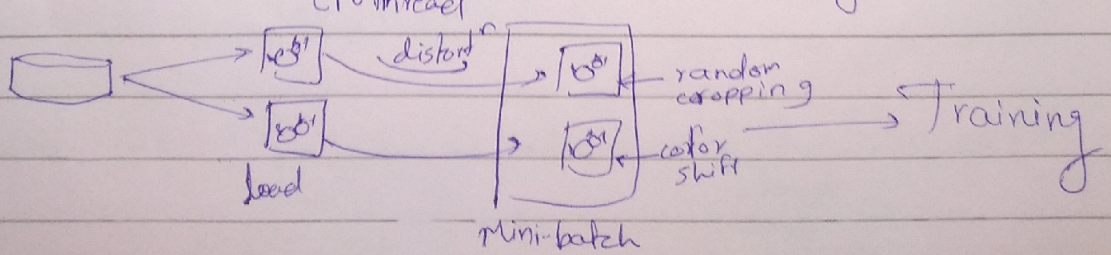


In practice, R, G, B are from a distribution that should be quite small

PCA color augmentation :- Finds which is more out of R, G, B and then adds more of it ~~wit~~ while subtracts the other.

ref: AlexNet paper

⇒ ~~8~~) Implementing distortions during training



In ~~the~~ NN

We have two sources of knowledge

1) Labelled data
2) Hand engineered features / n/w architecture / other components

If you have less of (1), you need to do (2)

Tips for doing well on benchmarks / winning competitions

1) Ensembling

Train several n/w independently, average their o/p

disadvantage: Slows computat^n, not used in production

2) Multi-crop at test time

Run classifier on multiple versions of test images and avg. results

adv.: less memory used than ensembling due to single n/w

disadv.: More computat^n time

Use open source code

Use architectures of n/ws published in literature

" Open src. implementat's if possible

" pretrained models & fine-tune on your dataset