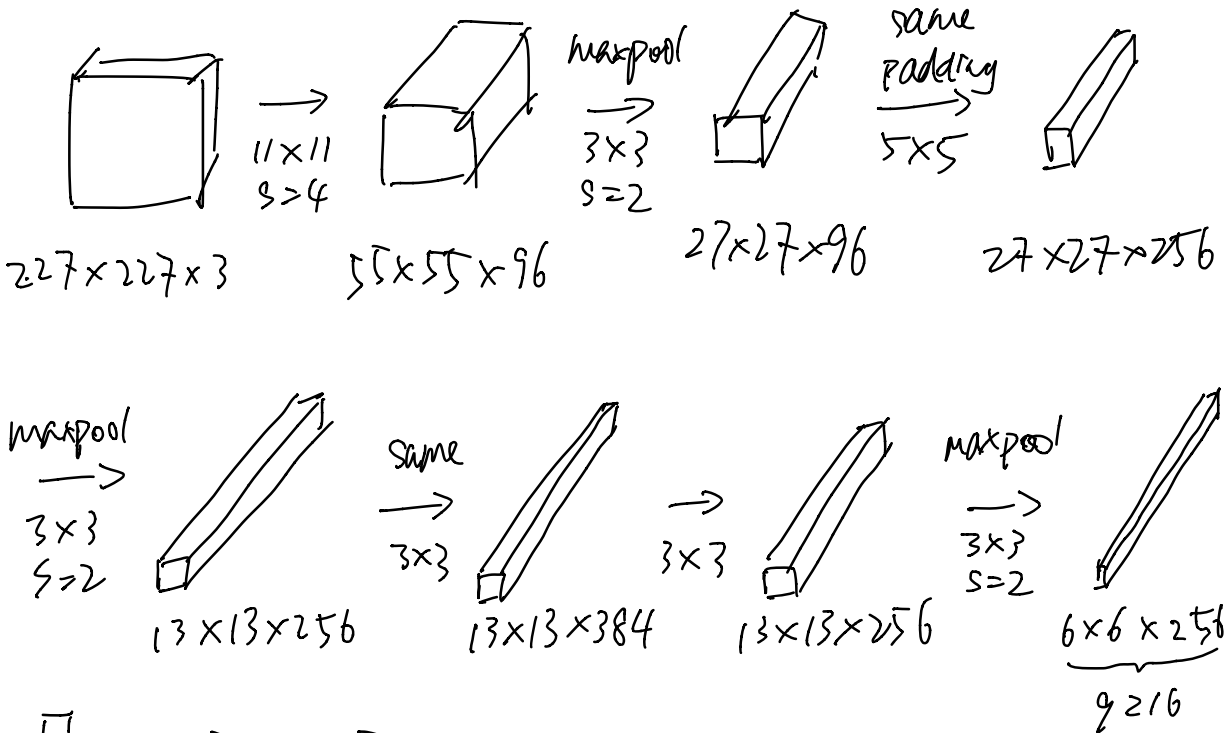
AlexNet



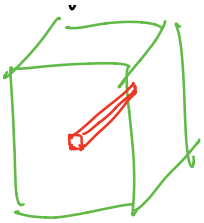
Note :

✓ - similar to LeNet, but much deeper.

✓ - while LeNet uses sigmoid/tanh, this uses Relu, which is much better.

X - optimized for multiple GPU (due to computation constraint at the time)

X - Local Response Normalization (LRU)



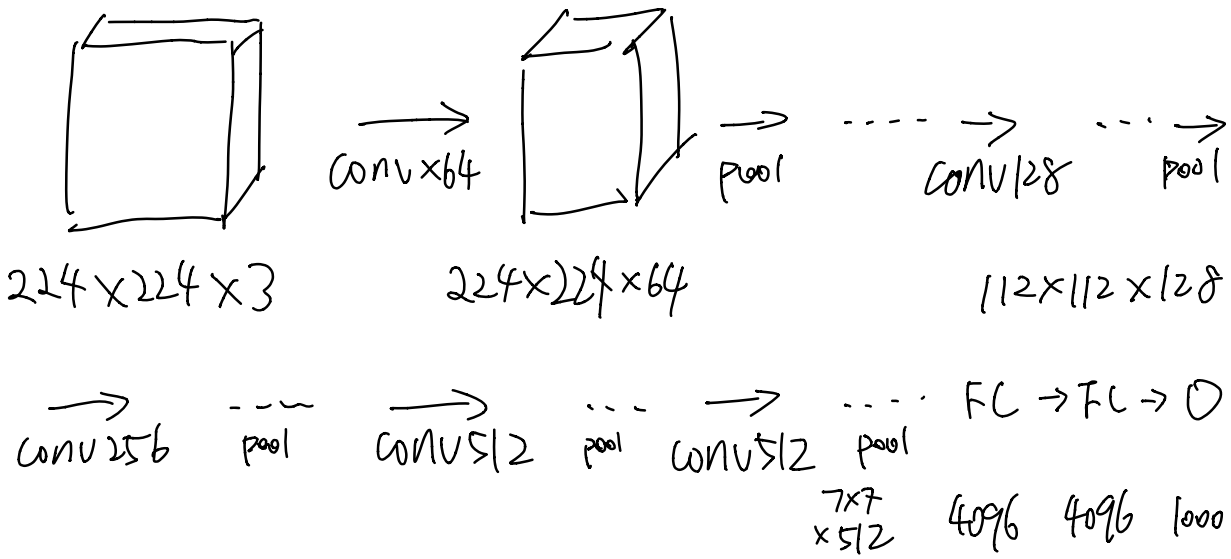
Normalize across diff channels.
(Not used anymore, proven ineffective)
IDEA was do not want too many neuron with high activation

VGG - 16

16 layer, ~138 M parameters

Simplified NN architecture.

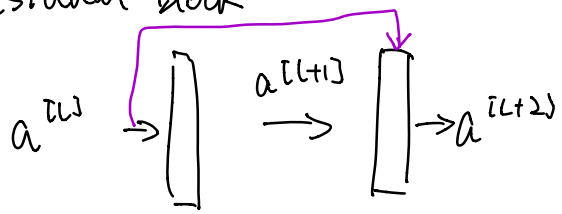
- Conv = 3×3 filters, S = 1, same padding
- max-pool = 2×2, S = 2



IDEA: use relatively uniform layers.

ResNet

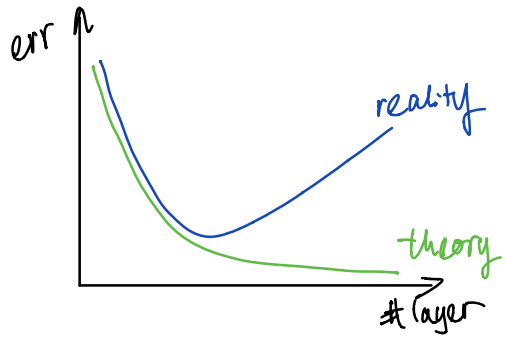
Residual Block



a[l] → linear → relu → a[l+1] → linear → relu → a[l+2]

skip connection

ResNet : Stack of residual Block



Deep Network is hard to train

because of vanishing / exploding gradient

ResNet allow us to train much deeper NN.

Why ResNet Works ?

$x \rightarrow$ Big NN $\rightarrow a^{[l]}$

$x \rightarrow$ Big NN $\rightarrow a^{[l]} \rightarrow [] \rightarrow [] \rightarrow a^{[l+2]}$

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

if $w^{[l+2]}, b^{[l+2]} = 0$

$$= g(a^{[l]})$$ assume g is relu.

$$= a^{[l]}$$ positive unchanged

From above, we see that the bottom line is that the added residual do not hurt the performance of the Big NN (at least do as well).

Note on Dimension matching:

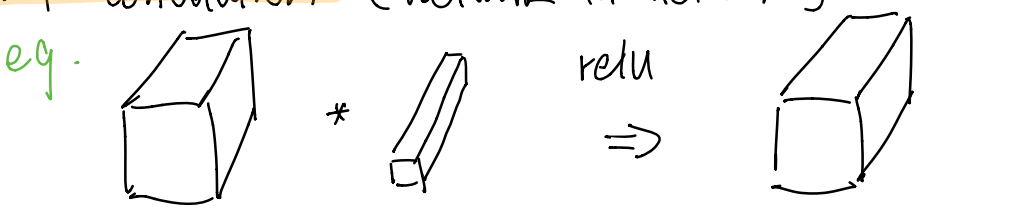
{ same dimension padding.
or add W_s (eg. pooling layer)

eg. $a^{[l+2]} = g(\cdots W_s a^{[l]})$

256 (for $a^{[l+2]}$), 256 × 128 (for W_s), 128 (for $a^{[l]}$)

W_s: params or just padding

1 × 1 convolution (network in network)

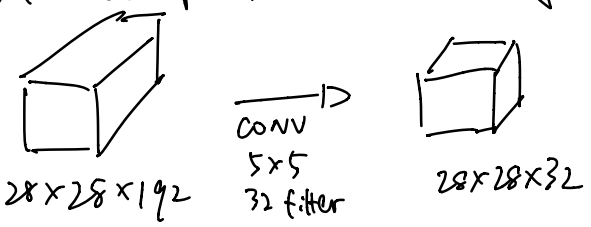


Goal: Shrink the # of channels

Inception Network

Motivation: instead of choosing which kind of layer to use, use them all and stack the output (with padding to make Dimension match for stacking) and let the NN Learn when / what layer to use

Problem: computation cost is high



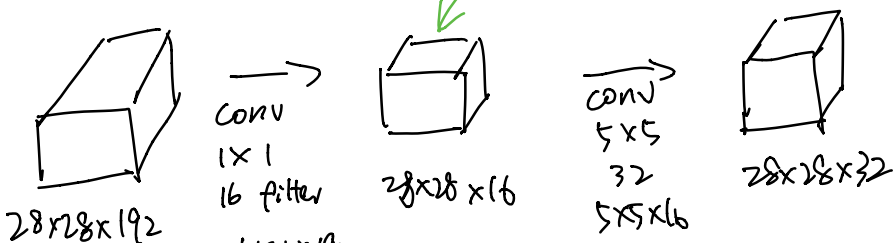
32 filters on 5×5×192

$\Rightarrow 28 \times 28 \times 32 \times 5 \times 5 \times 192$

$= 120M$

solution:

"bottleneck layer": shrink representation

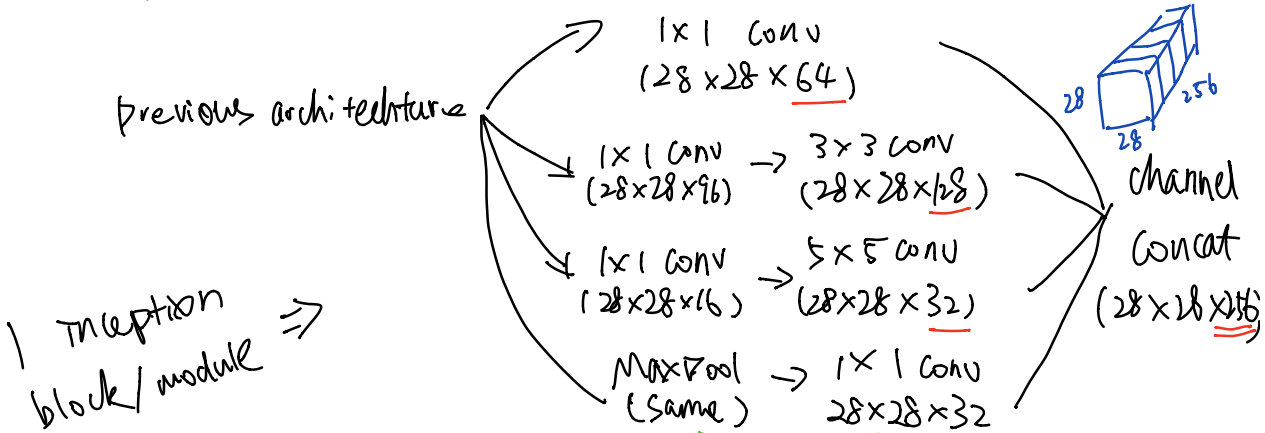


$28 \times 28 \times 16 \times 192 = 2.4M$

$28 \times 28 \times 32 \times 5 \times 5 \times 16 = 10.0M$

$12.4M \approx \frac{1}{10} \cdot 120M$

Inception Block.



Inception block/module $\Rightarrow$

we don't want to have 192 pooling channel (we don't want pooling to dominate.

(use 1x1 conv to shrink channel)

Inception Network

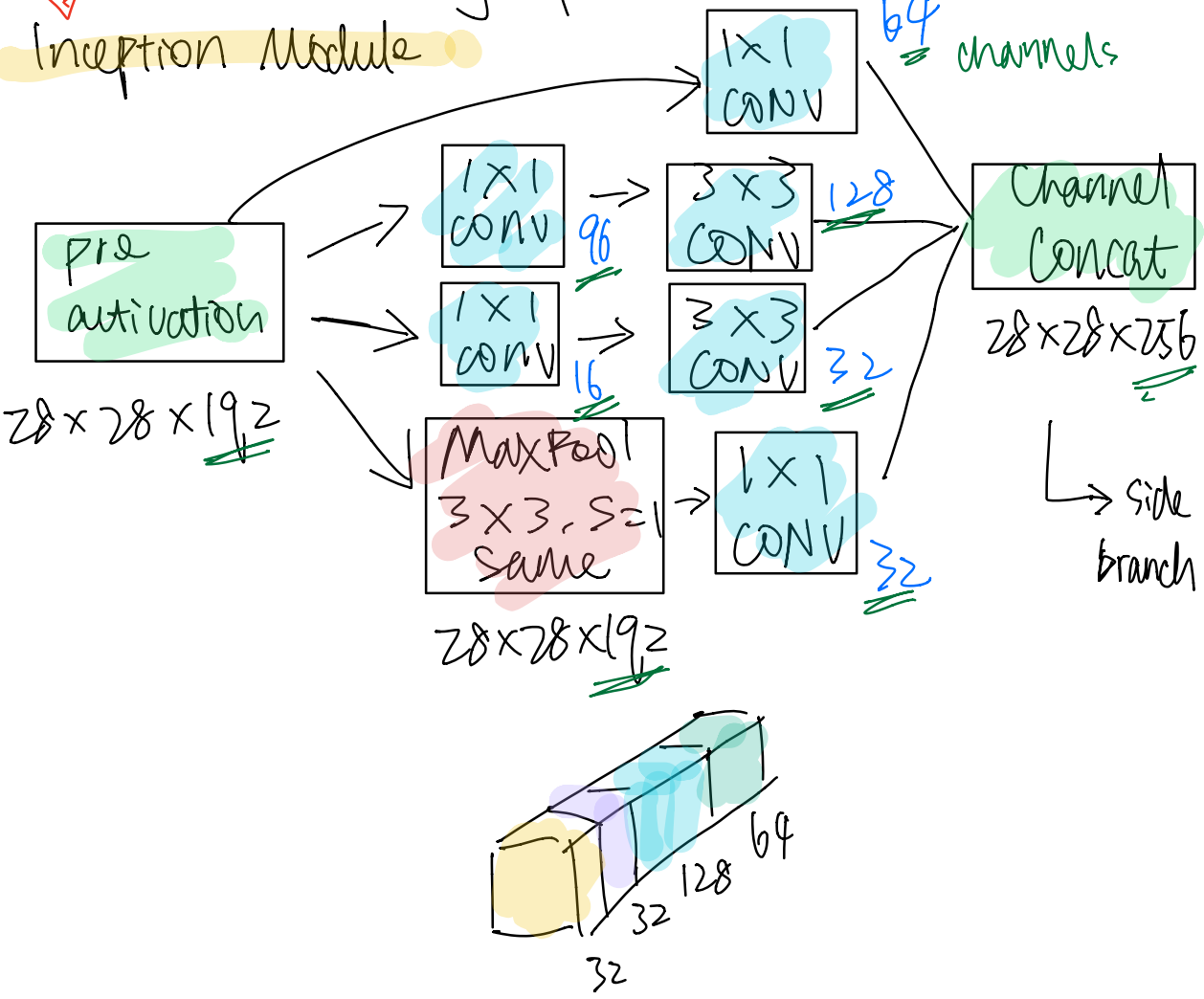
NN — Inception Block — Inception block ···· inception Block → softmax
↳ softmax ↳ softmax

The softmax in the intermediatery is to ensure that its not too bad of generalization in the middle, therefore prevent overfitting (ie. no useless blocks)

[Szegedy et al. 2014 Going Deeper with Convolutions]

Side Branch: a few fully connected layer followed by softmax

Inception Module



Transfer Learning.

freeze previous layers and train softmax ...

Data Augmentation