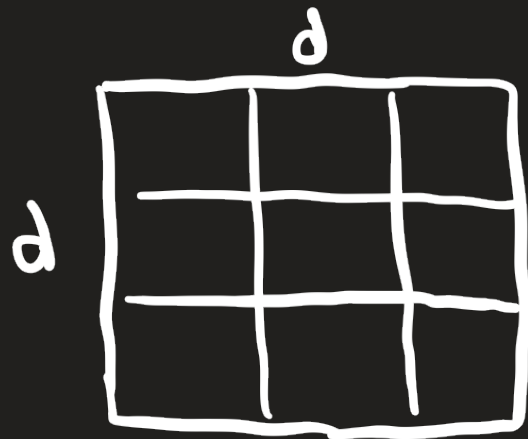


ML and Opt lecture 18

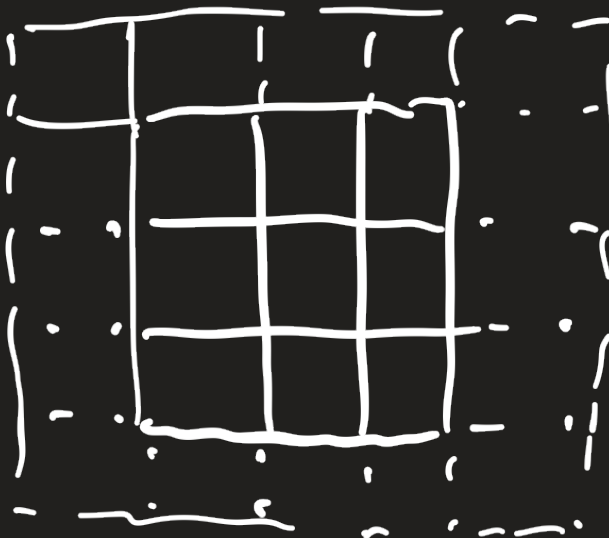
- Transposed Convolutions
- Backprop for Convolutional Layers (sketch)
- Vanishing and Exploding Gradients
- Google Inception architecture (v1); auxiliary losses

Transposed Convolutions

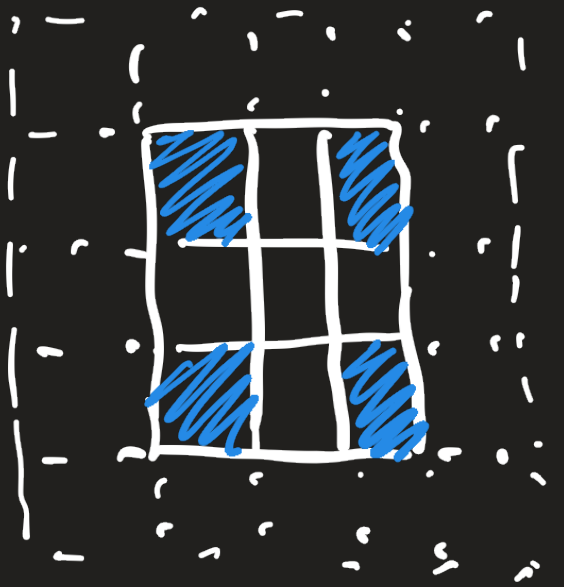
Convolution



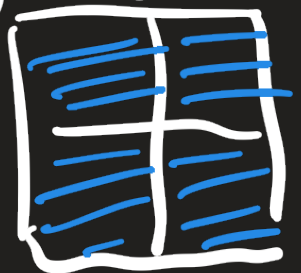
pad P



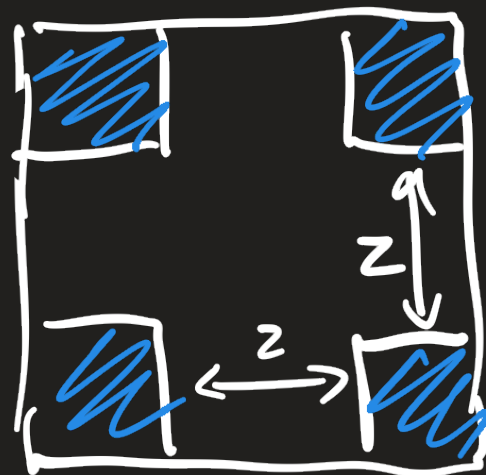
convolve
w/ stride S
filter of size K



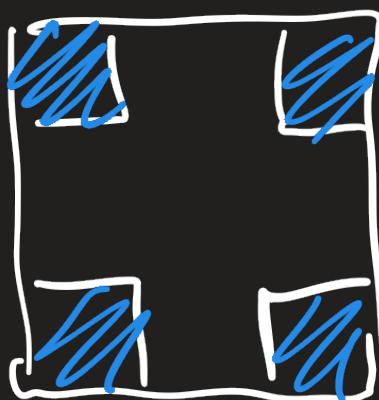
Transposed Convolution



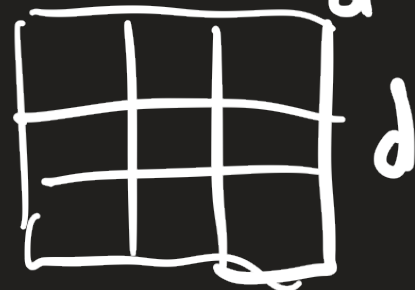
insert Z
zeros b/w
each row & column



pad P'



convolve w/ stride 1
filter of size K



To "invert" a convolutional filter K , we learn via BP, a filter $\bar{K}$. We need choose Z, P' so that the images from K when passed through $\bar{K}$ have dimension d

Let o be the size of the image from the forward convolution

pytorch:
Conv2D

$$o = \frac{d + 2p - k + 1}{s}$$

Then d' be the size of the image from the transposed convolution when the input is size o

$$d' = o + (o - 1) \cdot z + 2p' - k + 1$$

Want to choose z and p' so $d' = d$

Claim is: $z = s - 1$

$$p' = k - p - 1$$

pytorch:
Conv2DTranspose

$$d' = \left(\frac{d + 2p - k + 1}{s} + 1 \right) + \left(\frac{d + 2p - k}{s} \right) \cdot (s - 1) + 2(k - p - 1) - k + 1$$

$$= 1 + d + 2p - k + 2k - 2p - 2 - k + 1$$

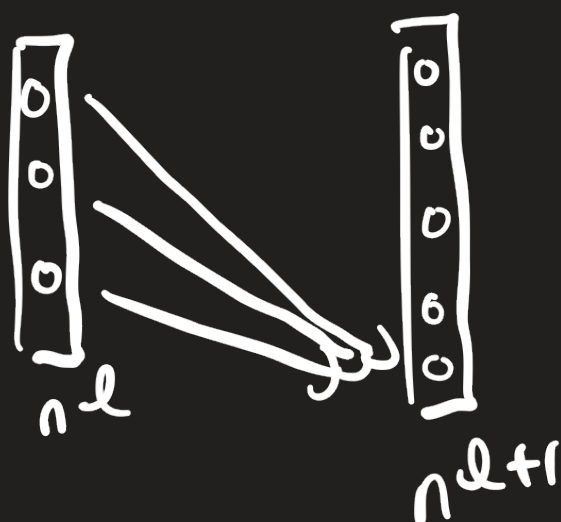
$$= d$$



Efficient Convolutions & Backprop

Parallels b/w MLPs and ConvNets

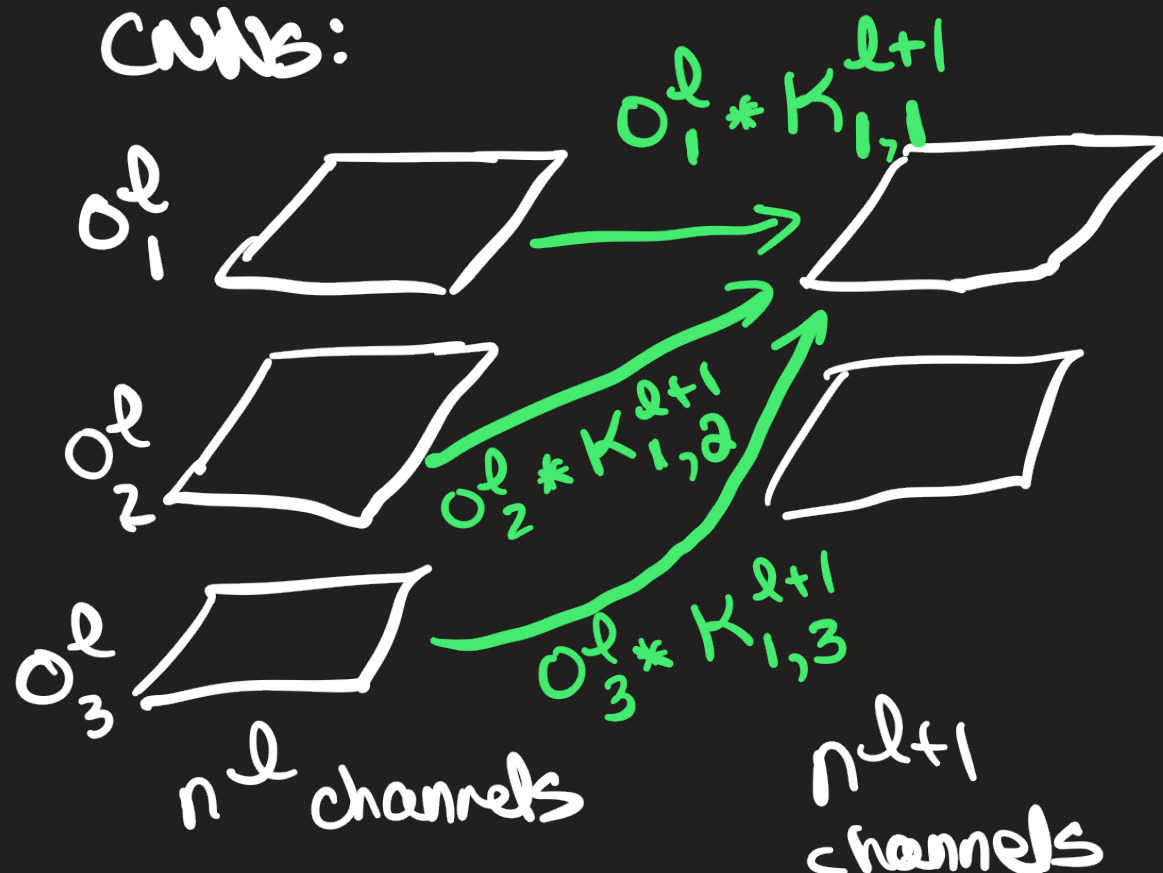
MLP:



$$a^{l+1} = W^{l+1} o^l + b^{l+1}$$

$$o^{l+1} = \sigma(a^{l+1})$$

CONV:



$$A_1^{l+1} = o_1^l * K_{1,1}^{l+1} + o_2^l * K_{1,2}^{l+1} + o_3^l * K_{1,3}^{l+1} + b_1^{l+1}$$

A scalar that is the same for each pixel

$$o_1^{l+1} = \sigma(A_1^{l+1})$$

If layer l has n_l channels, then

$$A_i^{l+1} = \left(\sum_{j=1}^{n_l} O_j^l * K_{i,j}^{l+1} \right) + b_i^{l+1}$$

$$O_i^{l+1} = \sigma(A_i^{l+1})$$

The number of parameters for layer $l+1$ is

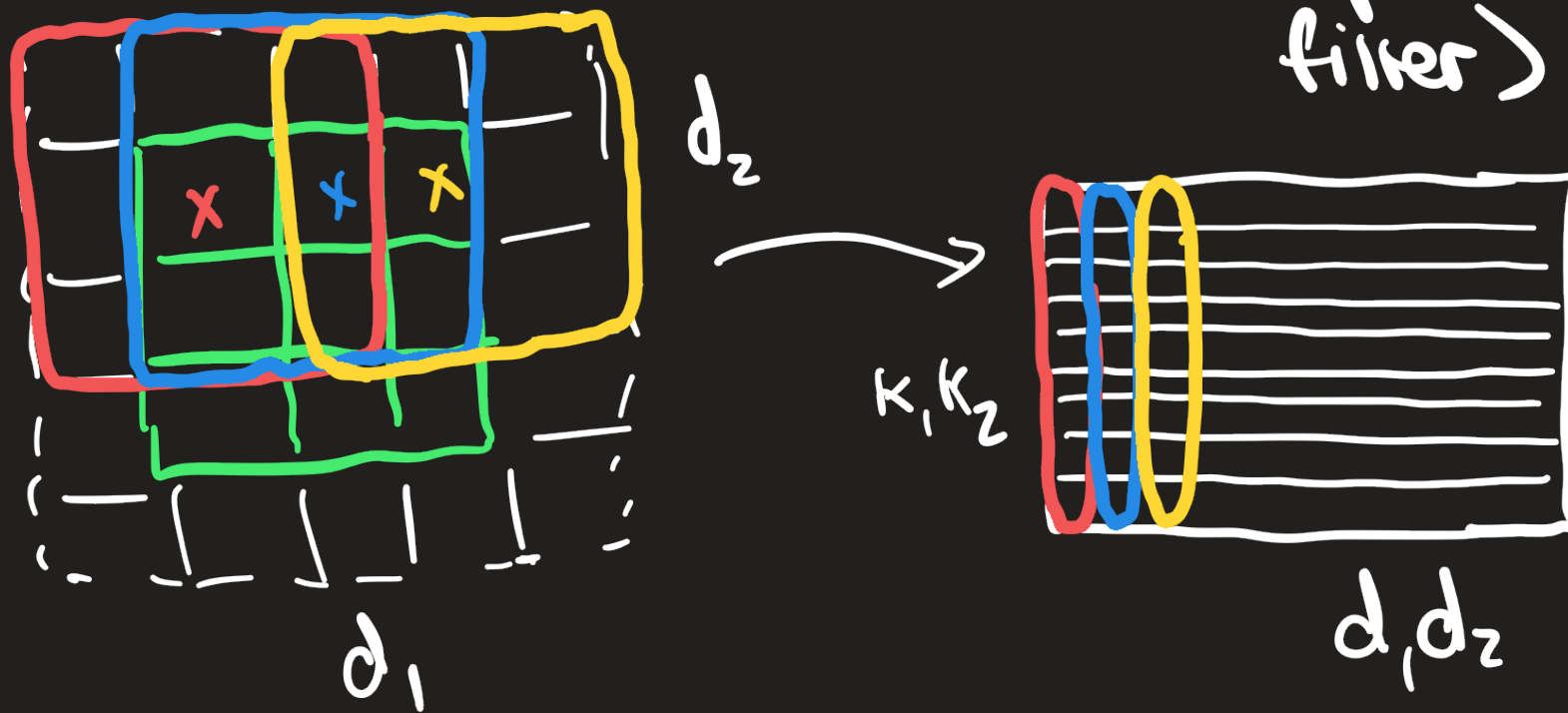
$$n_l n_{l+1} k_1 k_2 + n_{l+1} \quad (\text{assuming all the filters connecting layer } l \text{ to layer } l+1 \text{ are } k_1 \times k_2)$$

(Sketch) Efficient Computations

cf Manas Sahni "Anatomy of a High-Speed Convolution"

idea: reduce convolution to the **indcol** operation and GEMMs

- **indcol** takes an image in $\mathbb{R}^{d_1 \times d_2}$ and maps to a matrix $\mathbb{R}^{k_1 k_2 \times d_1 d_2}$ (corresponding to zero-padding the input and convolving by a $k_1 \times k_2$ filter)



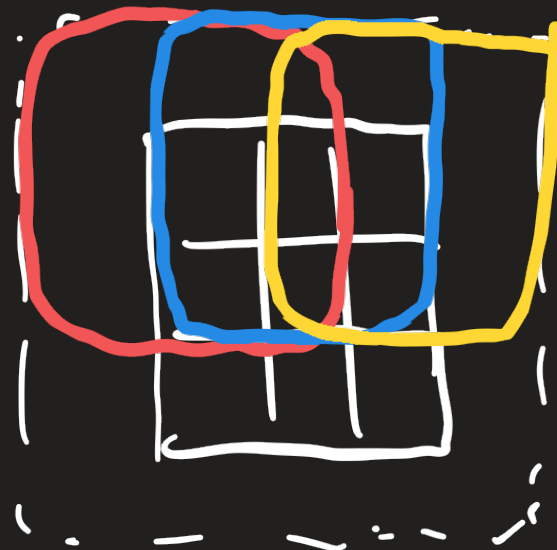
note that we want to compute
and this implies

$$O_j^l * K_{i,j}^{l+1}$$

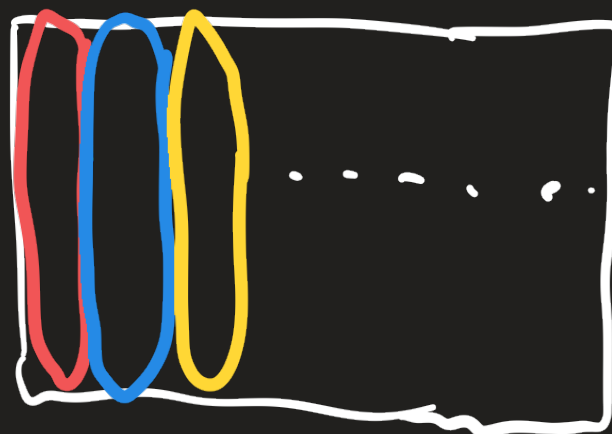
$$\text{vec}(O_j^l * K_{i,j}^{l+1}) = \text{vec}(K_{i,j}^{l+1}) \cdot \text{index}(O_j^l)$$

1	2	3
4	5	6
7	8	9

*



1	2	3	...	9
---	---	---	-----	---



Issues with deep NNs (not just CNNs):

- overfitting (too much capacity for the amount of training data)
- vanishing & exploding gradients $\Rightarrow$ slow learning!
- hyperparameter selection, e.g.
 - kernel sizes?
 - # channels per layer?
 - # layers?
 - type of pooling & locations?
 - stride & dilation & padding?
 - learning rates? algorithm? minibatch size?
 - weight decay?
 -

Vanishing & Exploding Gradients

Phenomenon that as $L \rightarrow \infty$

$$\text{as } l \rightarrow 1: \|\nabla_{w^l} f\|_2 \rightarrow \begin{cases} 0 & \text{vanishing gradients} \\ \infty & \text{exploding gradients} \end{cases}$$

Why? Because of the chain rule.

E.g. for MLPs:

$$\begin{aligned} \nabla_{o^l} f &= \text{diag}(\sigma'(a^{l+1})) (w^{l+1})^T \nabla_{o^{l+1}} f \\ &= \left[\prod_{i=l+1}^L \text{diag}(\sigma'(a^i)) (w^i)^T \right] \nabla_{o^L} f \end{aligned}$$

Two considerations:

1) How $\text{diag}(\sigma'(a^{l+1}))$ behaves



so if a^l far from 0, then this looks like a zero matrix, so

$$\|\nabla_{\theta^l} f\|_2 \ll \|\nabla_{\theta^{l+1}} f\|_2$$

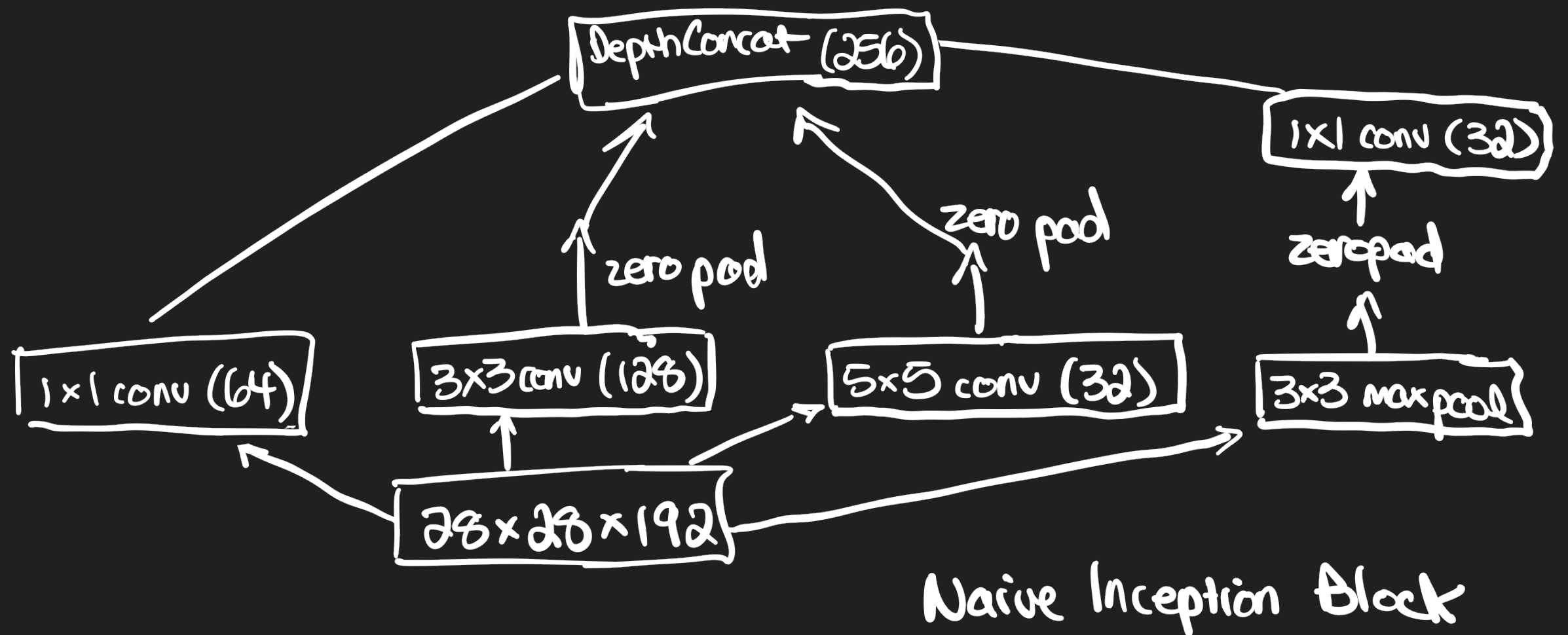
2) If the norm of our weight matrix W^{l+1} is large, then $\|\nabla_{\theta^l} f\|_2 \gg \|\nabla_{\theta^{l+1}} f\|_2$

if it is small, then $\|\nabla_{\theta^l} f\|_2 \ll \|\nabla_{\theta^{l+1}} f\|_2^2$

Consequence:

naively training deep NN architectures
either fails or gives poor performance

GoogLeNet (22 layer) CNN Inception v1



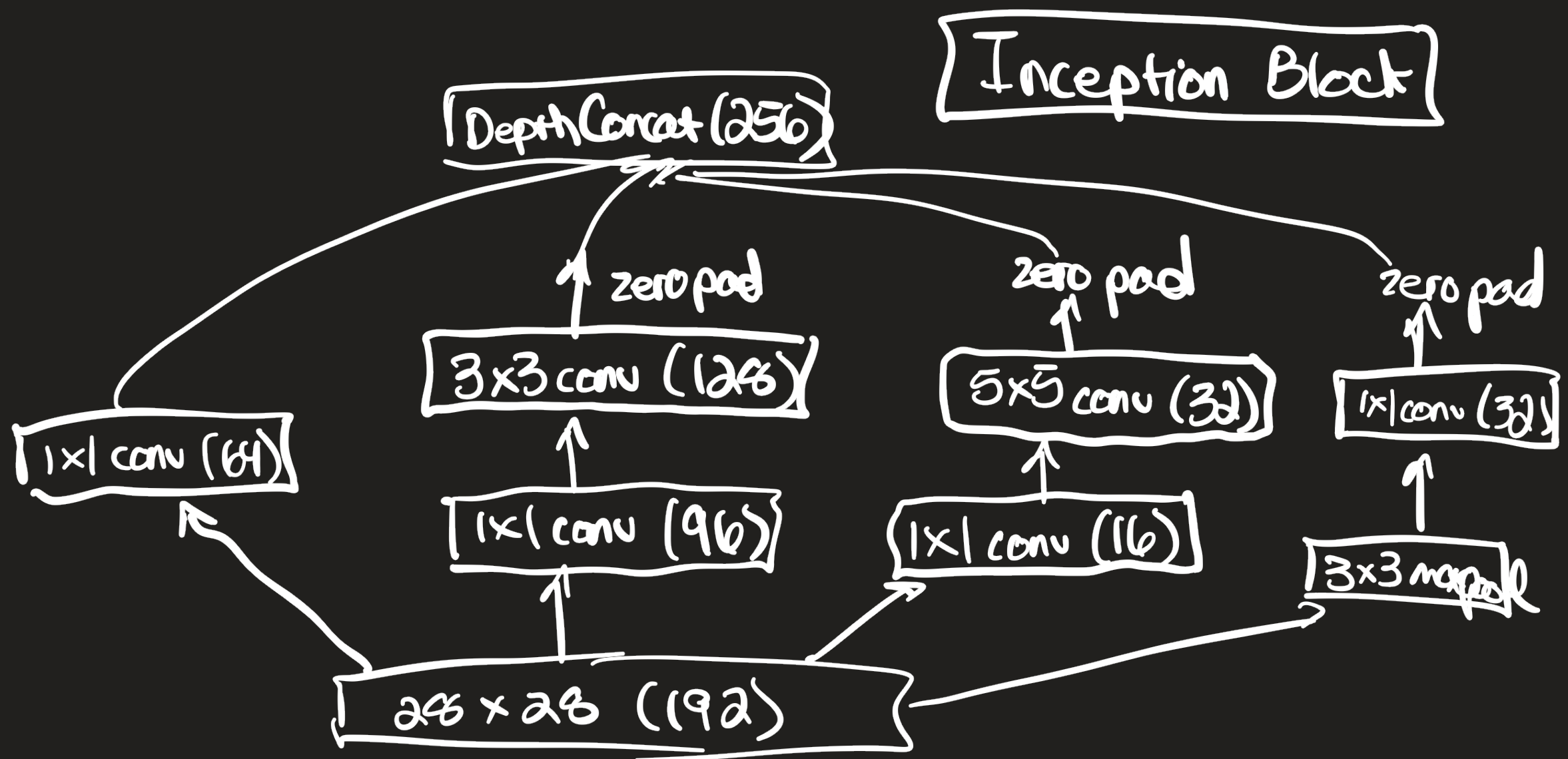
Downside: lots of parameters

E.g. for the 3×3 conv features, we have

$$192 \times 128 \times 3 \times 3 + 128 = 221,312 \text{ parameters}$$

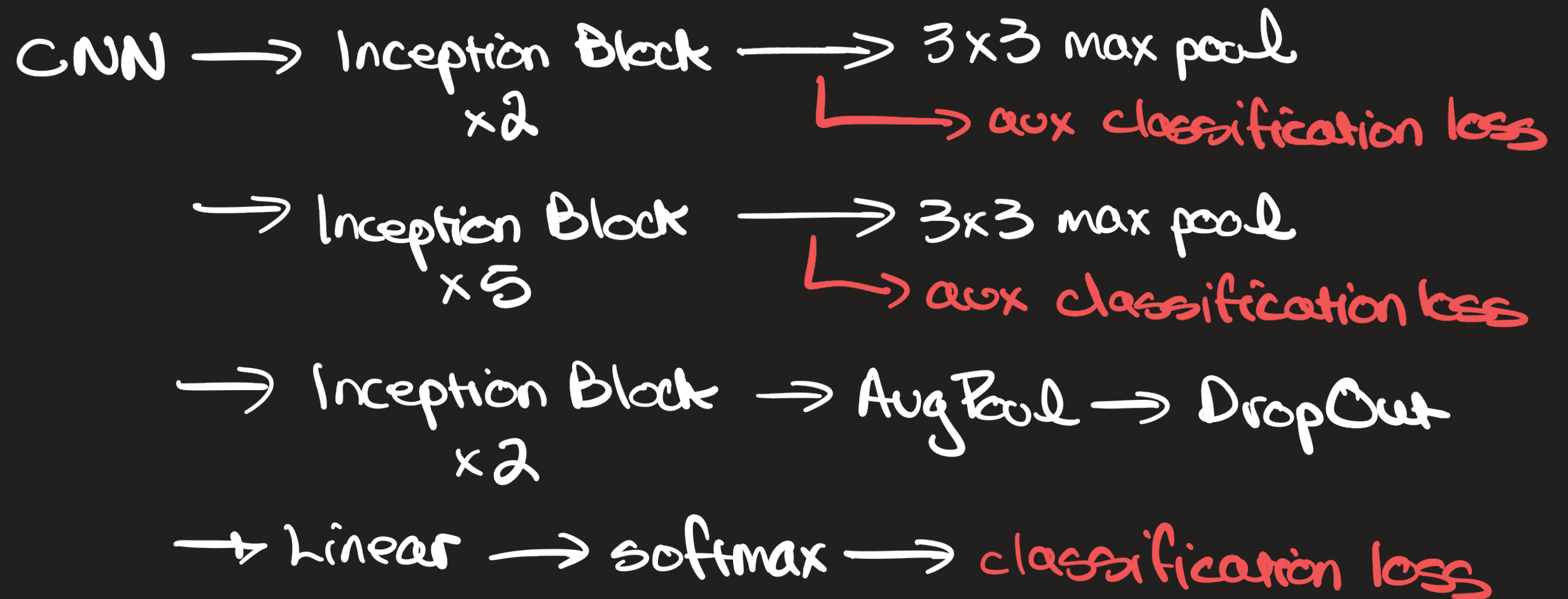
Issue: too many parameters

Soln: use 1×1 conv layers for dimensionality reduction



Check: # of 3×3 conv parameters (including the 96 1×1 conv)
= 129,248 parameters

Original Inception (v1) architecture



Train this architecture to minimize the weighted sum of the three losses. Injects gradient information at intermediate layers to mitigate vanishing/exploding gradients.