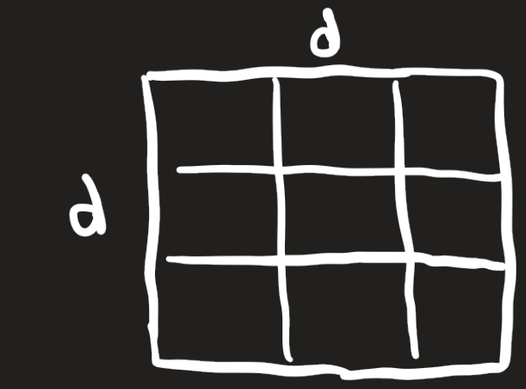
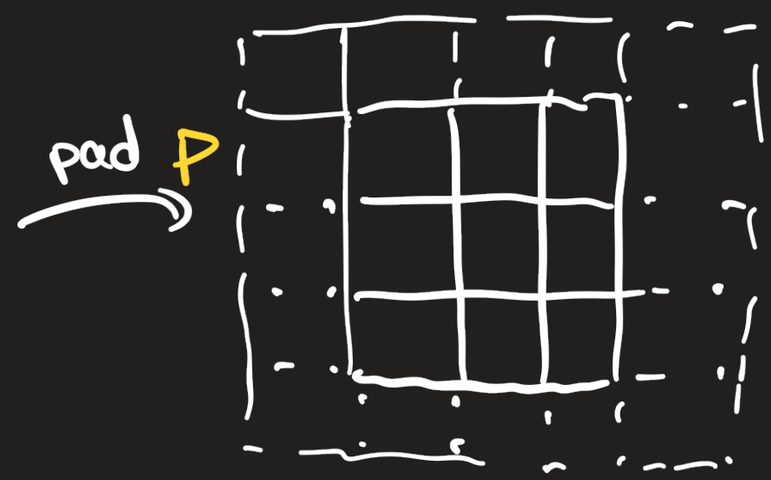
ML and Opt lecture 19

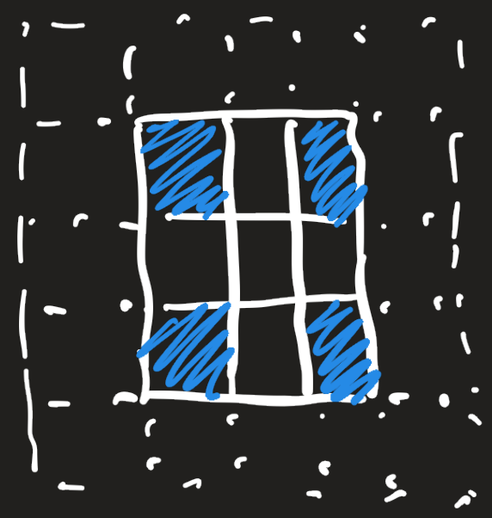
- Transposed Convolutions
- Backprop for Convolutional Layers (sketch)
- Vanishing and Exploding Gradients
- Batch Normalization
- Google Inception architecture (v1) ; auxiliary losses

Transposed Convolutions

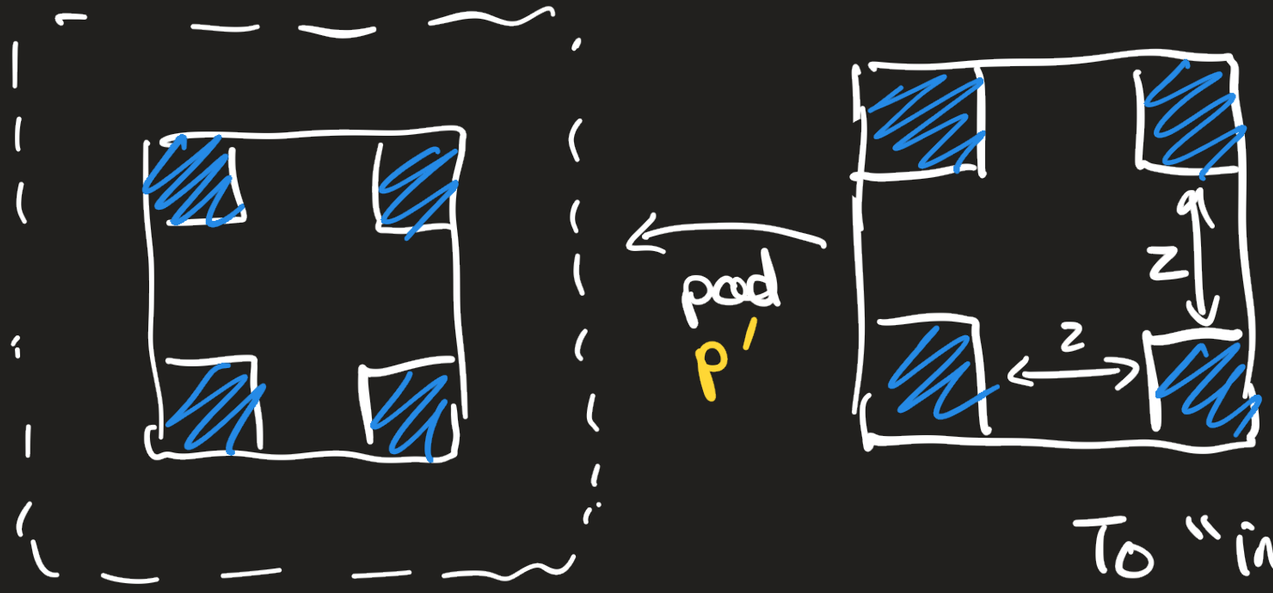
Convolution

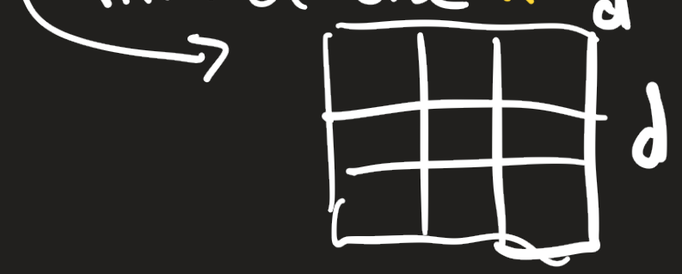






Transposed Convolution





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

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

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

pytorch: Conv2D

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

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

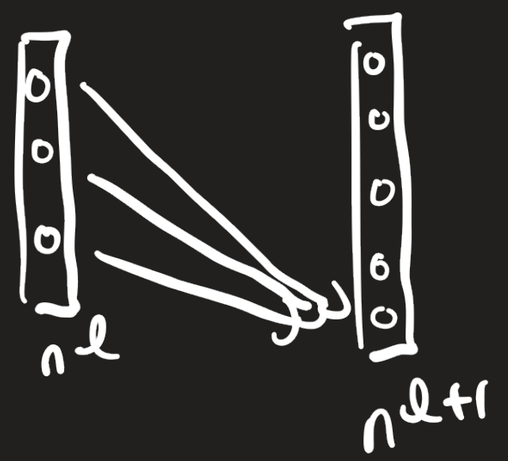
$$= 1 + d + 2p - k + 2k - 2p - 2 - k + 1$$

$$= d \quad \checkmark$$

Efficient Convolutions & Backprop

Parallels b/w MLPs and ConvNets

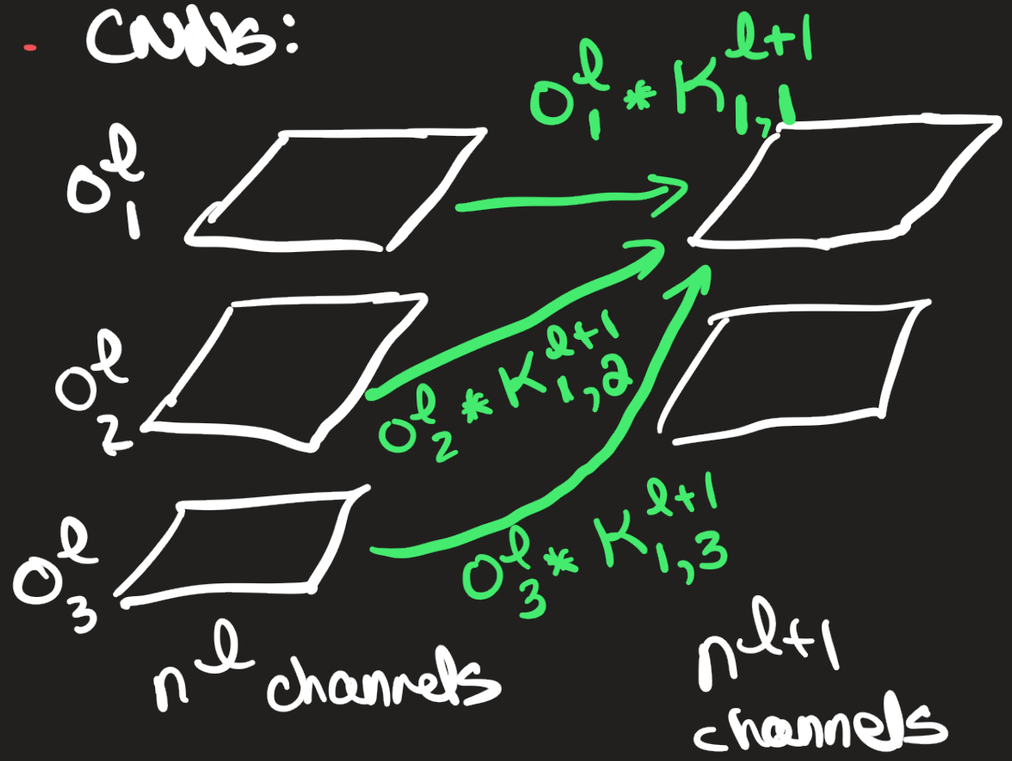
MLP:



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

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

- CNNs:



a scalar

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

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

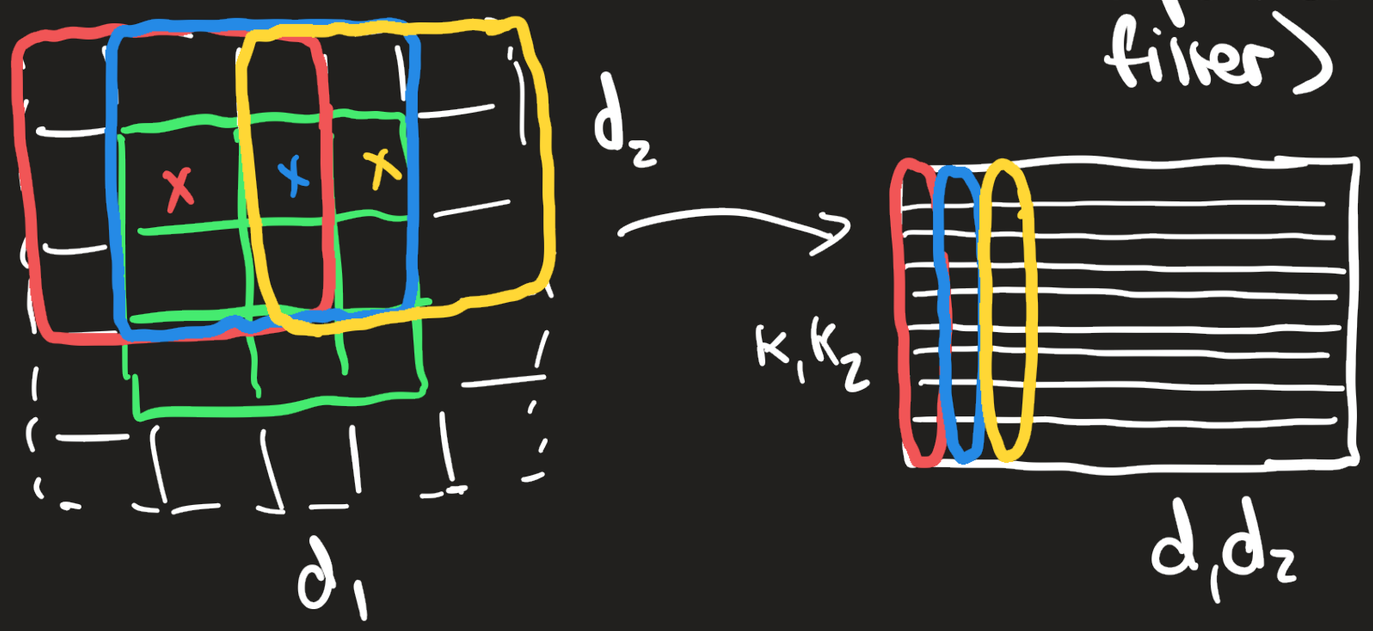
$n_l n_{l+1} k_1 k_2 + n_{l+1}$ (assuming all the filters connecting layer l to layer $l+1$ are $k_1 \times k_2$)

(Sketch) Efficient Computations

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

idea: reduce convolution to the **im2col** operation and GEMMs

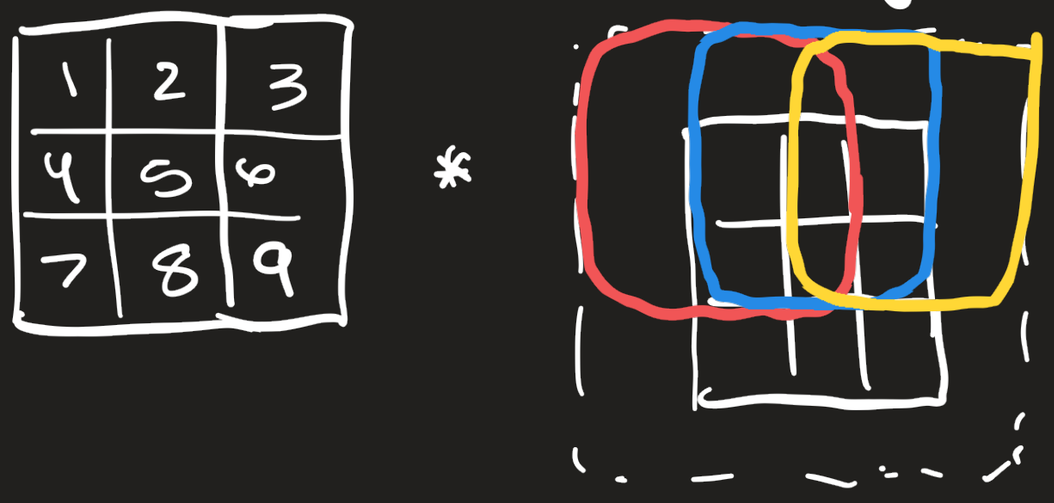
- im2col takes an image in $\mathbb{R}^{d_1 \times d_2}$ and maps to a matrix $\mathbb{R}^{k_1 k_2 \times d_1 d_2}$ (corresponding to zero-padding the input and convolving by a $k_1 \times k_2$ filter)



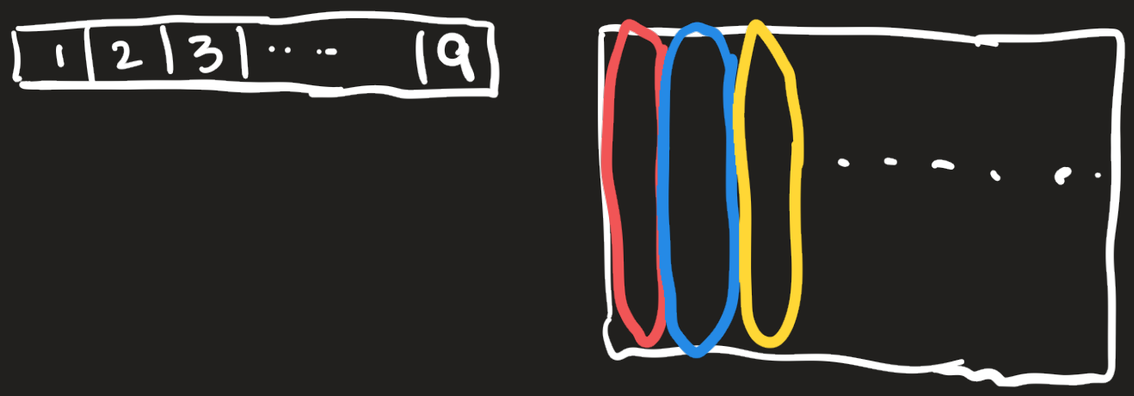
note that we want to compute $O_j^l * K_{i,j}^{l+1}$

and this implies

$$\text{vec}\left(O_j^l * K_{i,j}^{l+1}\right) = \text{vec}\left(K_{i,j}^{l+1}\right) \cdot \text{im2col}\left(O_j^l\right)$$



$\Leftrightarrow$



Issues with deep NNs (not just CNNs):

- overfitting (too much capacity for the amount of training data)
- vanishing & exploding gradients ⇒ 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 $L \to \infty$

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

Why? Because of the chain rule.

E.g. for MLPs:

$$\nabla_{o^l} f = \operatorname{diag}\left(\sigma'(a^{l+1})\right)(w^{l+1})^T \nabla_{o^{l+1}} f$$

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

Two considerations:

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

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

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

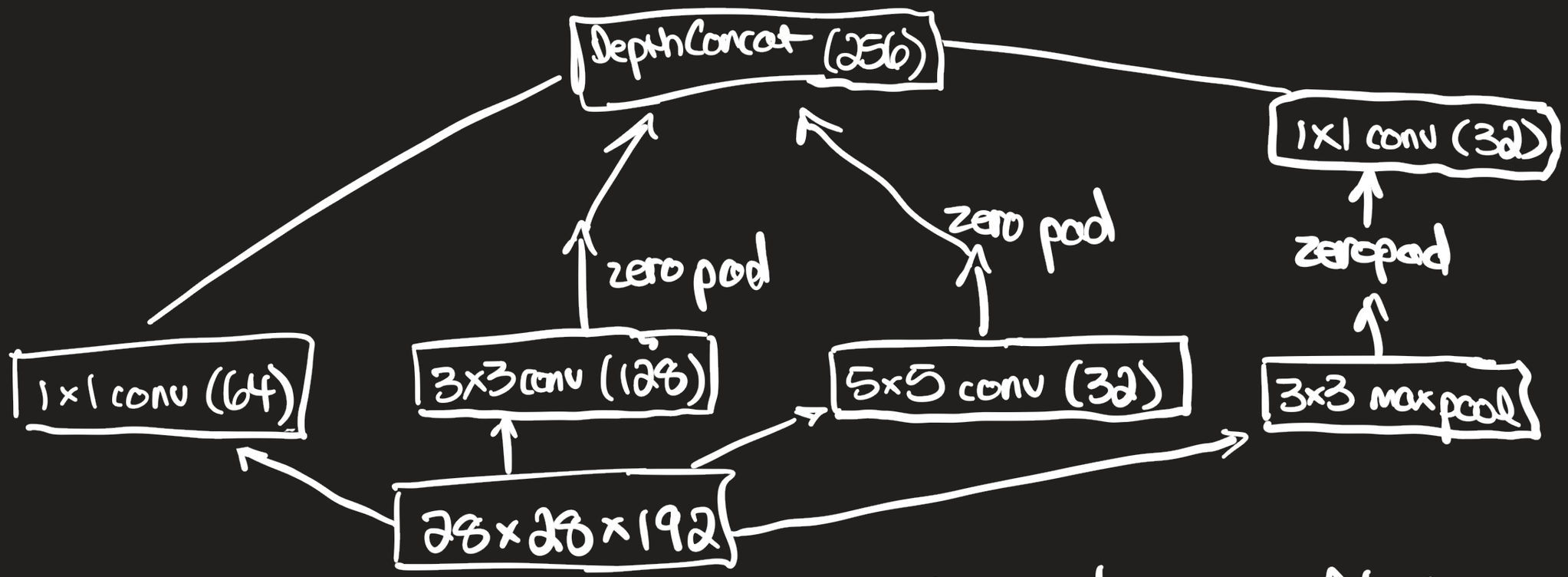
2) If the norm of our weight matrix w^{l+1} is large, then $\|\nabla_{a^l} f\|_2 \gg \|\nabla_{a^{l+1}} f\|_2^2$

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

Consequence:

naively training deep NN architectures either fails or gives poor performance

GoogLeNet (22 layer) CNN Inception v1



Naive Inception Block

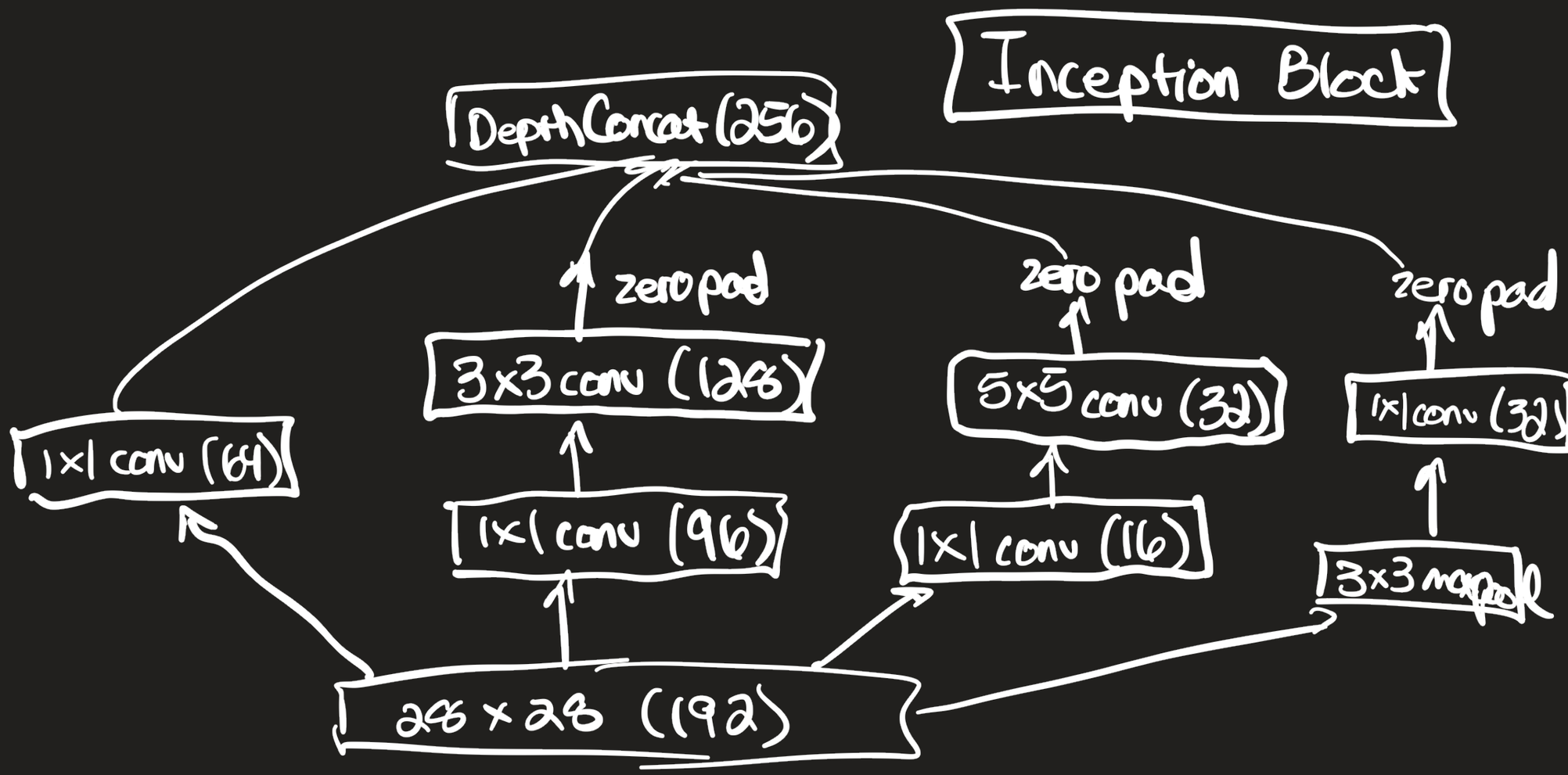
Downside: lots of parameters

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

$192 \times 128 \times 3 \times 3 + 128 = 221,312$ 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

CNN → Inception Block ×2 → 3×3 max pool → aux classification loss

→ Inception Block ×5 → 3×3 max pool → aux classification loss

→ Inception Block ×2 → AvgPool → DropOut

→ Linear → softmax → classification loss

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