Data Augmentation (overfitting)

CNNs automatically deal well w/ translational invariance because of their architecture design:

- shared parameters (using the same filter at each pixel location)
- pooling

help to detect the same features in different regions

They are not very good at rotational invariance or scale invariance, so we build these in by augmenting the training dataset: add randomly rotated & cropped images to the dataset.

Inception architecture state-of-the-art perf. on ILSVRC 2014

22 layers deep initially

ideas: max pooling loses spatial details
CNNs as we've seen them have issues w/ detecting the same feature at different scales

trend: increase # layers & # filters per layer, using dropout to prevent overfitting

problem w/ this approach is you see a quadratic increase in the # of parameters in our model:

if n_l and n_{l-1} (# of filters in each layer) increase by a factor of c

initially for each of the output channels on layer n_l you had to learn n_{l-1} kernel

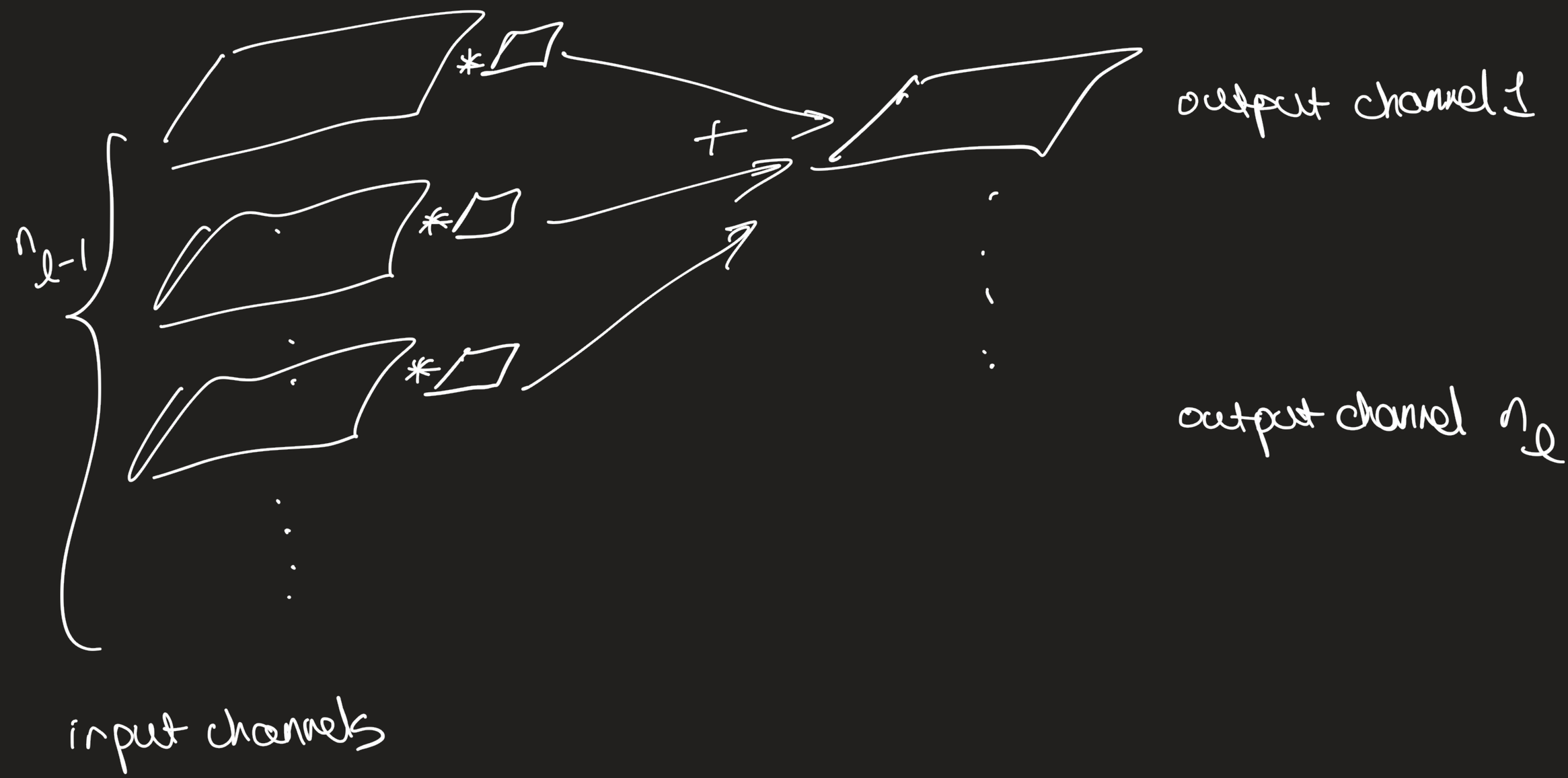
so now you have to learn

$$(cn_l)(cn_{l-1}) = c^2 n_l n_{l-1} \quad \text{kernels}$$

This is computationally expensive

Recall convolutional layers w/ multi-channel inputs:

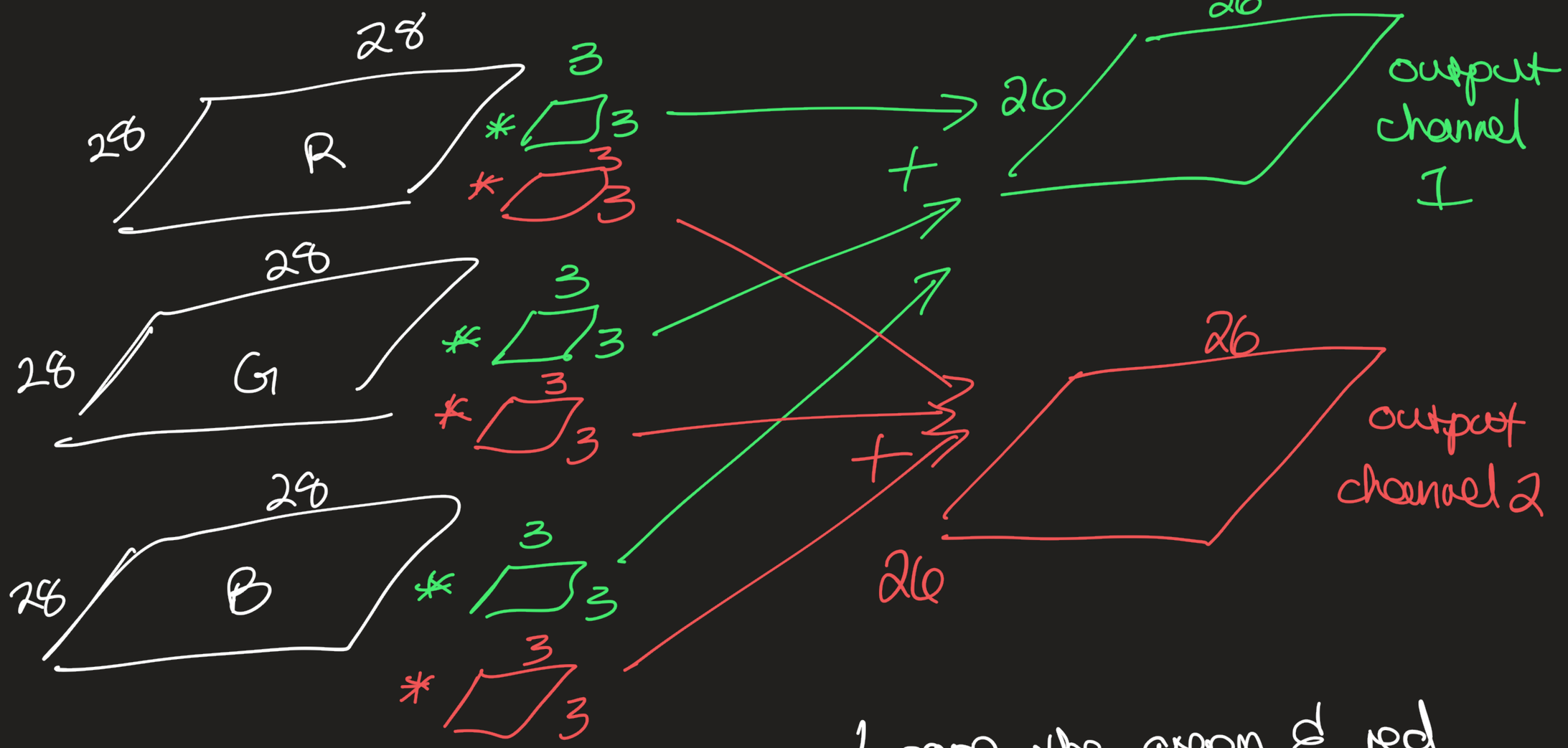
layer l



Ex: passing a single image through my CNN

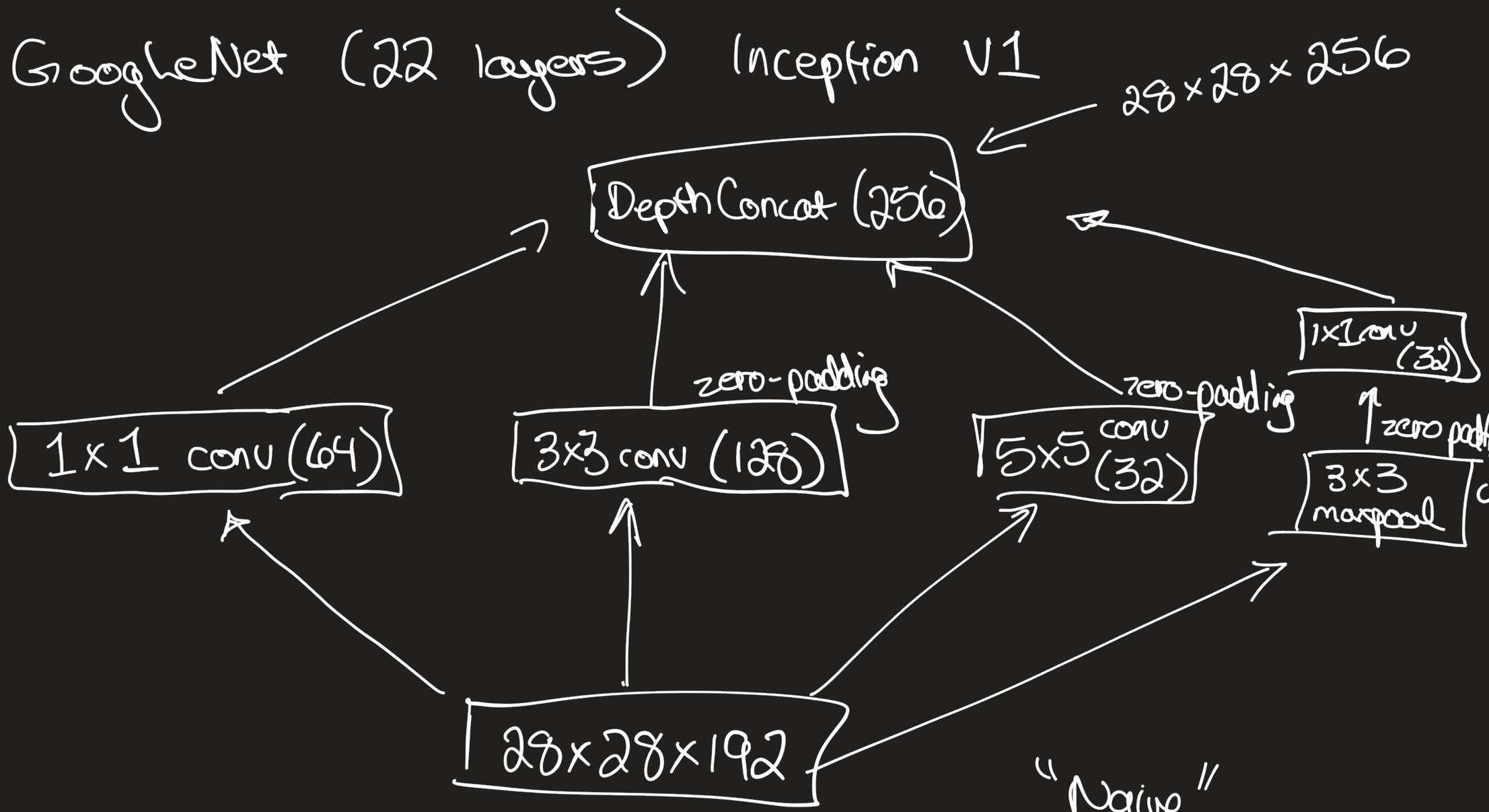
$x \in \mathbb{R}^{28 \times 28 \times 3}$

layer 1 has 2 output channels



Learn the green & red filters via backprop

GoogLeNet (22 layers) Inception V1

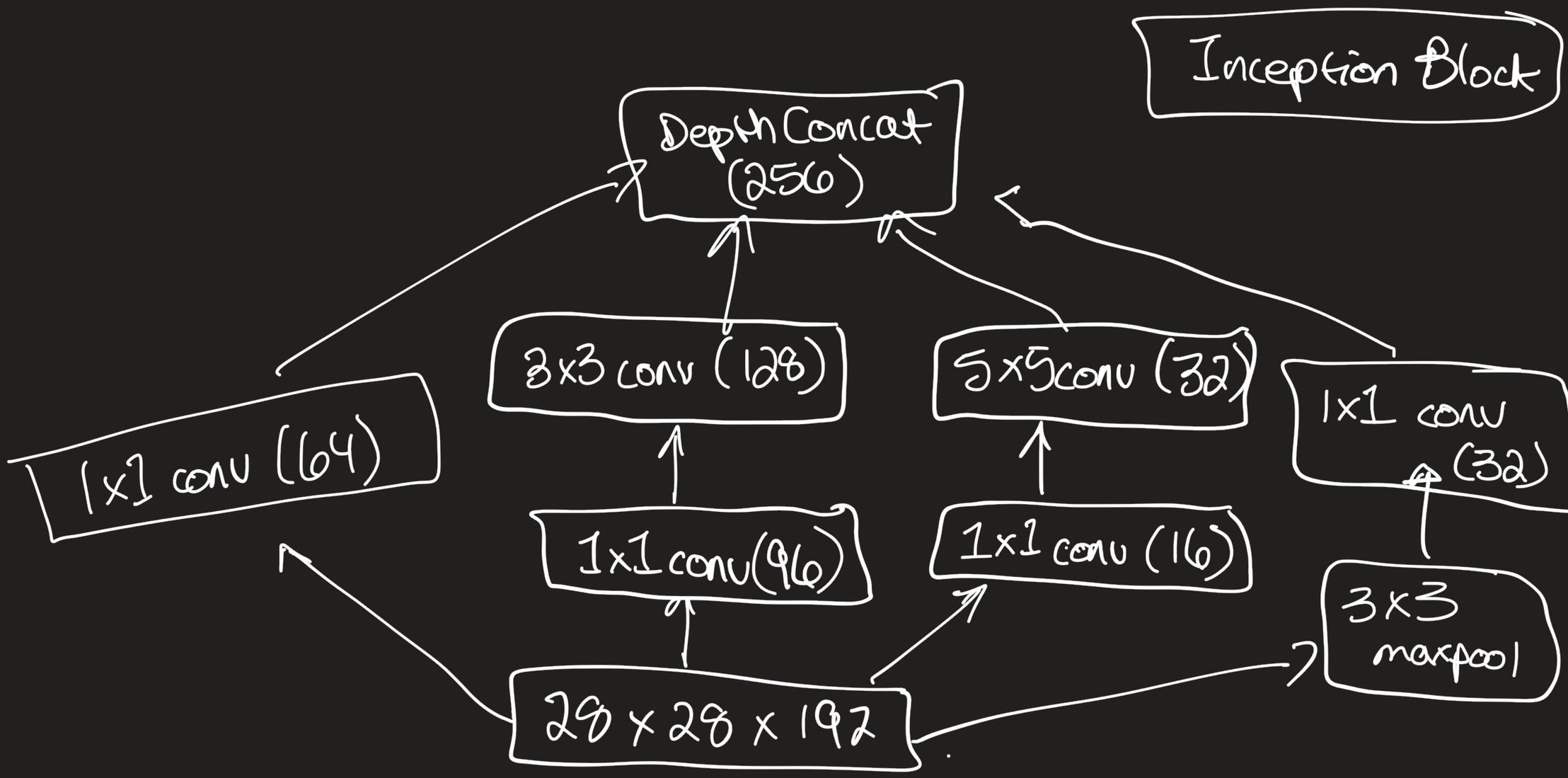


"Naive" Inception Block

For example: for 3×3 conv block we have

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

Issue: increased number of parameters
Soln: use some dim reduction via 1x1 convs



With the 1×1 convs for dim reduction, the # of parameters for the 3×3 conv features now becomes

$$\underbrace{(192 \times 96 \times 1 \times 1 + 96)}_{\text{for the 1×1 conv dim red.}} + \underbrace{(96 \times 128 \times 3 \times 3 + 128)}_{\text{for the 3×3 convs}}$$

$$= 129,248 \text{ parameters}$$

significantly less.

Original Inception (v1) architecture

CNN → Inception Block x2 → 3x3 max pool → aux class loss

→ Inception Block x5 → 3x3 max pool → aux class loss

→ Inception Block x2 → AvgPool → Dropout → Linear (1000) → softmax → classification loss

train this architecture to minimize the sum of the classification losses (weighted)

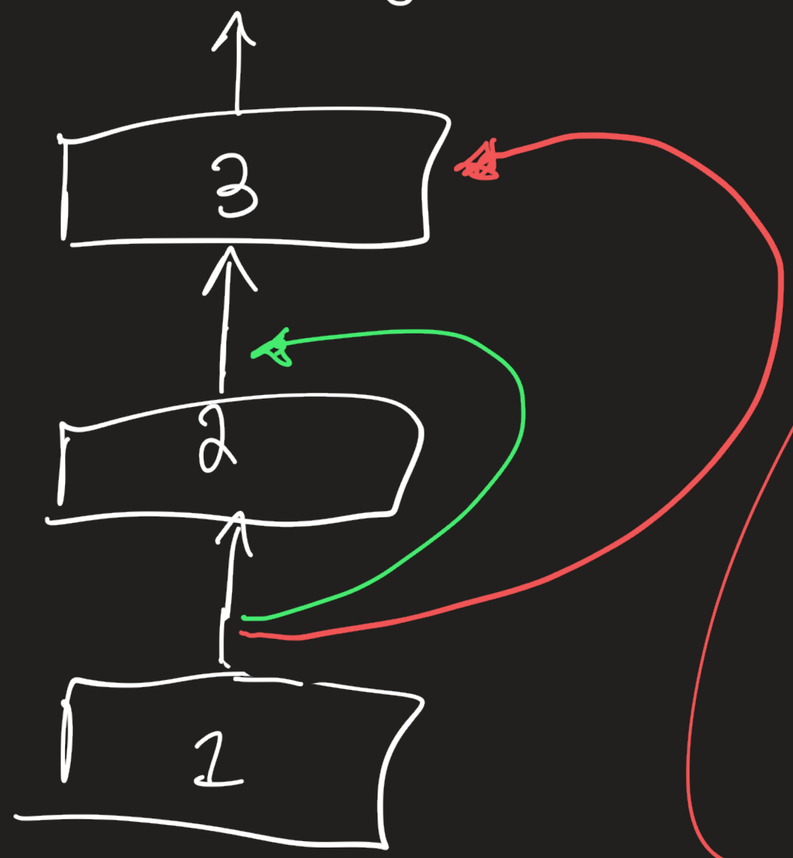
Later versions of Inception:

- replace each 5×5 conv layer w/ two sequential 3×3 conv layers
- use Batch Normalization so can go deeper & train faster (and remove dropout)

Skip Connections

Introduce skip connections b/w non-consecutive layers.

This helps w/ gradient propagation.



$$o_3 = \sigma(w^3 o_2 + b^3) + w^{1\to 3} o_1$$

$$o_3 = \sigma(w^3 o_2 + w^{1\to 3} o_1 + b^3)$$

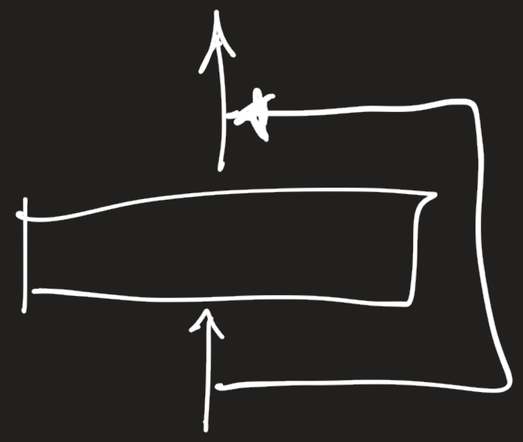
skip connect from output of layer 1 to input of layer 3

skip connect from output of layer 1 to output of layer 3

Residual Network

Idea: it could be easy to learn arbitrarily deep networks if all the last layers could learn the identity function

Residual block



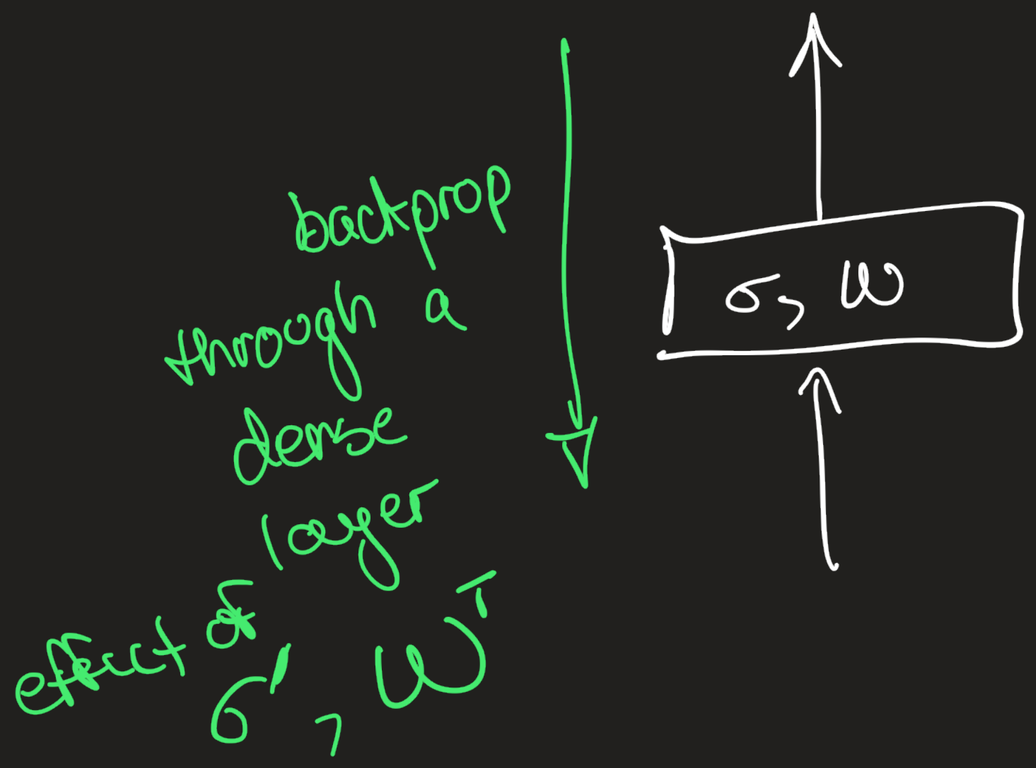
$$o^l = \sigma(w^l o^{l-1} + b^l) + o^{l-1}$$

this means $n_l = n_{l-1}$

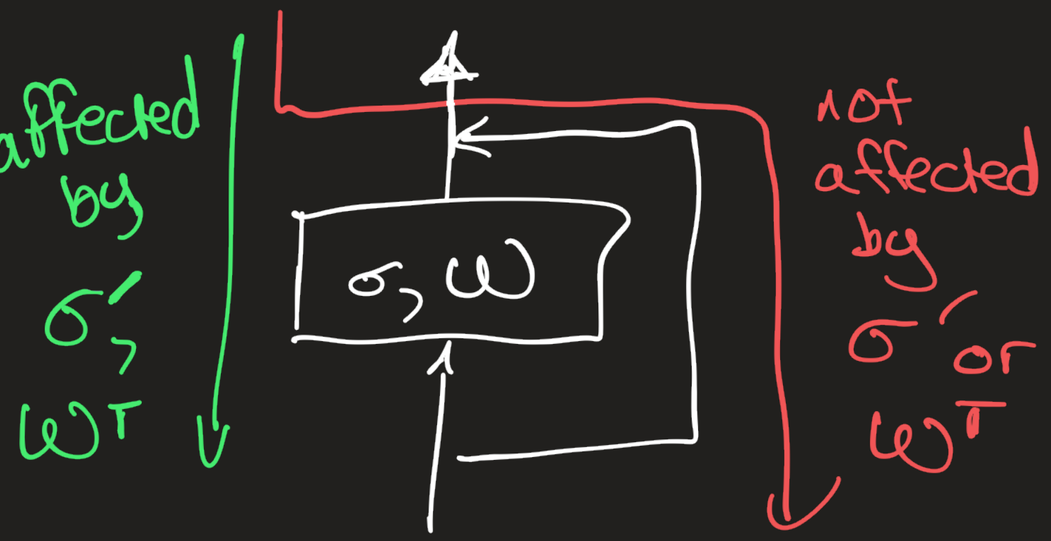
for example if $\sigma = \text{ReLU}$ then learning the identity is easy: $w^l = 0 = b^l \Rightarrow o^l = o^{l-1}$

Recall vanishing & exploding gradients caused by the deriv of the activation and the norm of the weight matrix

backprop through a dense layer



backprop through a resid block



With residual blocks we can get very simple, very deep architectures

ResNet-152