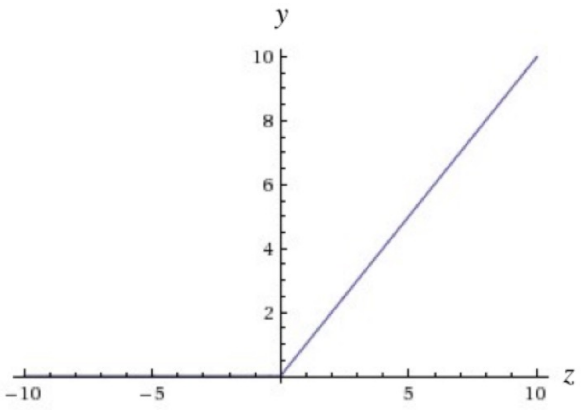
SEIS 764 Artificial Intelligence

Lecture 3: Convolutional Neural Networks

Andrew Van Benschoten, Ph.D.
Adjunct Professor, Department of Software Engineering & Data Science
University of St. Thomas
vanb2168@stthomas.edu

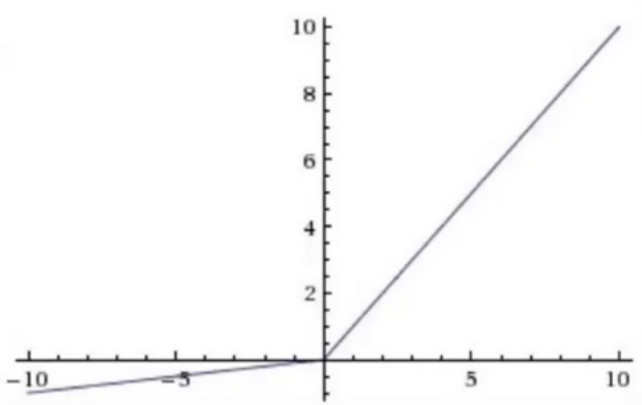
ReLU fixes the problem

Regular



f(z) = z if z >0, 0 elsewhere

"Leaky"



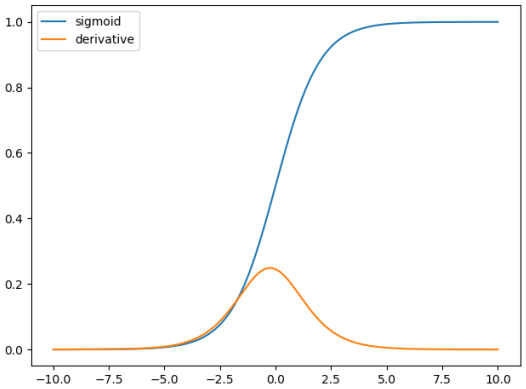
*f(z) = z if z >0, 0.01*z elsewhere*

ReLU is nearly always used for hidden layers


I LIED

The vanishing gradient returns

- The derivative of the sigmoid & tanh is ~ zero at most points.
- Max value of derivative is only 0.25!!
- Backpropagation collapses to zero



$$\frac{\partial C_0}{\partial w^{(L-1)}} = a^{(L-2)} \cdot \sigma'(z^{(L-1)}) \cdot w^{(L)} \cdot \sigma'(z^{(L)}) \cdot 2(a^{(L)} - y)$$

What about the weights?

ReLU doesn't handle the weights being small or large

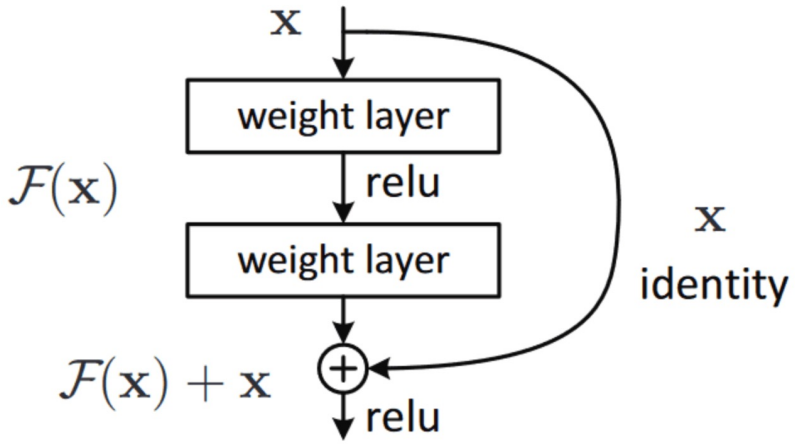
$$\frac{\partial C_0}{\partial w^{(L-1)}} = a^{(L-2)} \cdot \sigma'(z^{(L-1)}) \cdot w^{(L)} \cdot \sigma'(z^{(L)}) \cdot 2(a^{(L)} - y)$$

$$w^{(L)}\, w^{(L-1)}\, w^{(L-2)}$$

...

If $w < 1$, gradient collapses
If $w > 1$, gradient explodes

Skip connections



$$y = \text{Sublayer}(x) + x$$

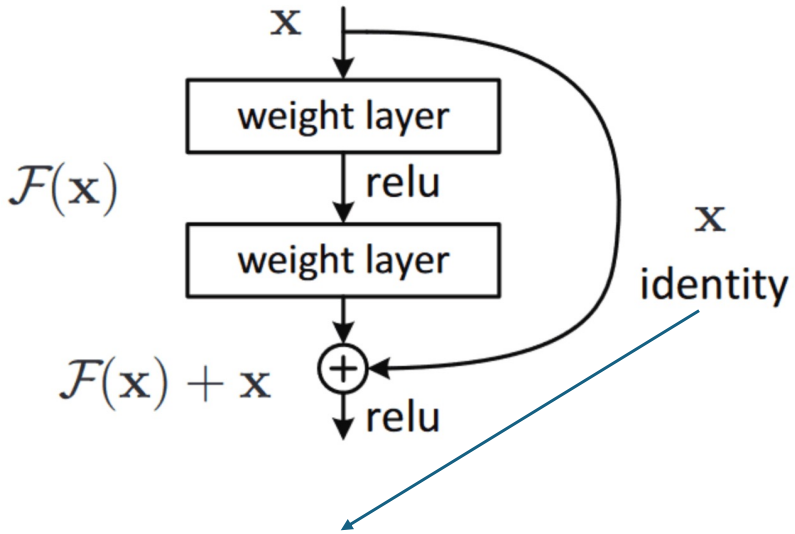
1) Allows the model to "fall back" to identity if the sublayer is unhelpful

2) Proven to stabilize deep architectures

3) Prevents vanishing gradients

$$\prod W^{(l)} \quad \Rightarrow \quad I + \text{small perturbations}$$

Skip connections



$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma\left(z^{(L)}\right)$$

$$C_0 = \left(a^{(L)} - y\right)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = a^{(L-1)} + \sigma\left(z^{(L)}\right)$$

$$C_0 = (a^{(L)} - y)^2$$

$$\prod W^{(l)} \quad \Rightarrow \quad I + \text{small perturbations}$$

The end state

$$\frac{\partial C_0}{\partial w^{(L-1)}} = a^{(L-2)} \cdot \sigma'(z^{(L-1)}) \cdot w^{(L)} \cdot \sigma'(z^{(L)}) \cdot 2(a^{(L)} - y)$$

$$|w^{(L)}\sigma'(z^{(L)})| \ll 1$$

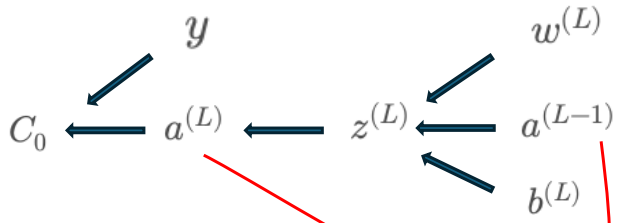
$$\text{gradient factor} \approx 0$$

$$\frac{\partial C_0}{\partial w^{(L-1)}} = a^{(L-2)}\ \sigma'(z^{(L-1)})\ \left(1 + w^{(L)}\sigma'(z^{(L)})\right)\ 2(a^{(L)} - y)$$

$$|w^{(L)}\sigma'(z^{(L)})| \ll 1$$

$$\text{gradient factor} \approx 1$$

Skip connections for L



$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma\left(z^{(L)}\right)$$

$$C_0 = \left(a^{(L)} - y\right)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

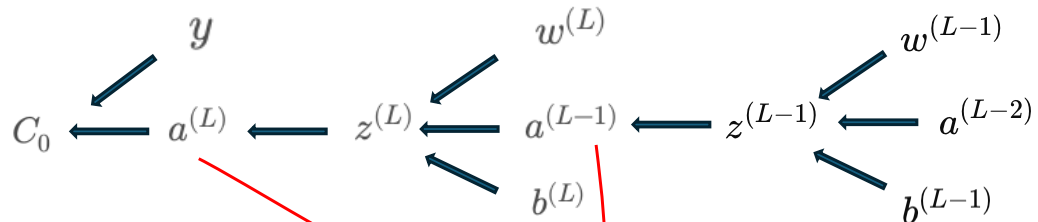
$$a^{(L)} = a^{(L-1)} + \sigma\left(z^{(L)}\right)$$

$$C_0 = (a^{(L)} - y)^2$$

$$\frac{\partial C_0}{\partial w^{(L)}} = a^{(L-1)}\, \sigma'(z^{(L)})\, 2(a^{(L)} - y)$$

Nothing changes for w(L) – which is good!

Skip connections for L-1



$$C_0 = (a^{(L)} - y)^2$$

$$\frac{\partial C_0}{\partial a^{(L)}} = 2(a^{(L)} - y)$$

$$a^{(L)} = a^{(L-1)} + \sigma(z^{(L)})$$

$$\frac{\partial a^{(L)}}{\partial a^{(L-1)}} = \underbrace{1}_{\text{skip path}} + \underbrace{\frac{\partial \sigma(z^{(L)})}{\partial a^{(L-1)}}}_{\text{through weights}}$$

$$z^{(L-1)} = w^{(L-1)}a^{(L-2)} + b^{(L-1)}$$

$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial a^{(L)}}{\partial a^{(L-1)}} \cdot \frac{\partial C_0}{\partial a^{(L)}}$$

$$= (1 + w^{(L)}\sigma'(z^{(L)})) \cdot 2(a^{(L)} - y)$$

$$\frac{\partial \sigma(z^{(L)})}{\partial a^{(L-1)}} = \sigma'(z^{(L)}) \cdot \frac{\partial z^{(L)}}{\partial a^{(L-1)}} = \sigma'(z^{(L)}) \cdot w^{(L)}$$

$$\frac{\partial C_0}{\partial w^{(L-1)}} = \frac{\partial C_0}{\partial a^{(L-1)}} \cdot \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \cdot \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}}$$

$$z^{(L)} = w^{(L)}a^{(L-1)} + b^{(L)}$$

$$\frac{\partial C_0}{\partial w^{(L-1)}} = \frac{\partial C_0}{\partial a^{(L-1)}} \cdot \sigma'(z^{(L-1)}) \cdot a^{(L-2)}$$

$$\frac{\partial a^{(L)}}{\partial a^{(L-1)}} = 1 + w^{(L)}\sigma'(z^{(L)})$$

$$\frac{\partial C_0}{\partial w^{(L-1)}} = a^{(L-2)}\sigma'(z^{(L-1)})(1 + w^{(L)}\sigma'(z^{(L)}))2(a^{(L)} - y)$$

The end state

$$0.1^1 = 0.1$$
$$0.1^2 = 0.01$$
$$0.1^3 = 0.001$$
$$0.1^4 = 0.0001$$
$$0.1^5 = 0.00001$$
$$0.1^{10} = 0.0000000001 = 10^{-10}$$

$$I + \text{small perturbations}$$

$$1.1^2 = 1.21$$
$$1.1^4 = 1.21^2 = 1.4641$$
$$1.1^8 = 1.4641^2 \approx 2.14358881$$
$$1.1^{10} = 1.1^8 \cdot 1.1^2 \approx 2.14358881 \cdot 1.21 \approx 2.59474246$$

$$(1.1)^{50} \approx e^{50 \ln 1.1}$$

$$\ln 1.1 \approx 0.09531$$

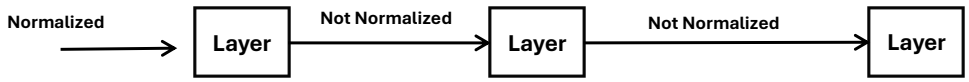
$$50 \cdot 0.09531 \approx 4.7655$$

$$e^{4.7655} \approx 117.6$$

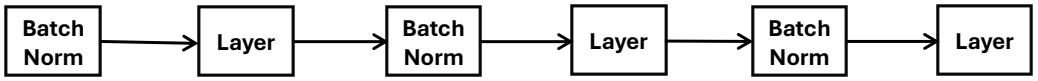
$$a^{(L)} = a^{(L-1)} + \sigma\left(z^{(L)}\right)$$

What if our gradient explodes?

Batch Normalization

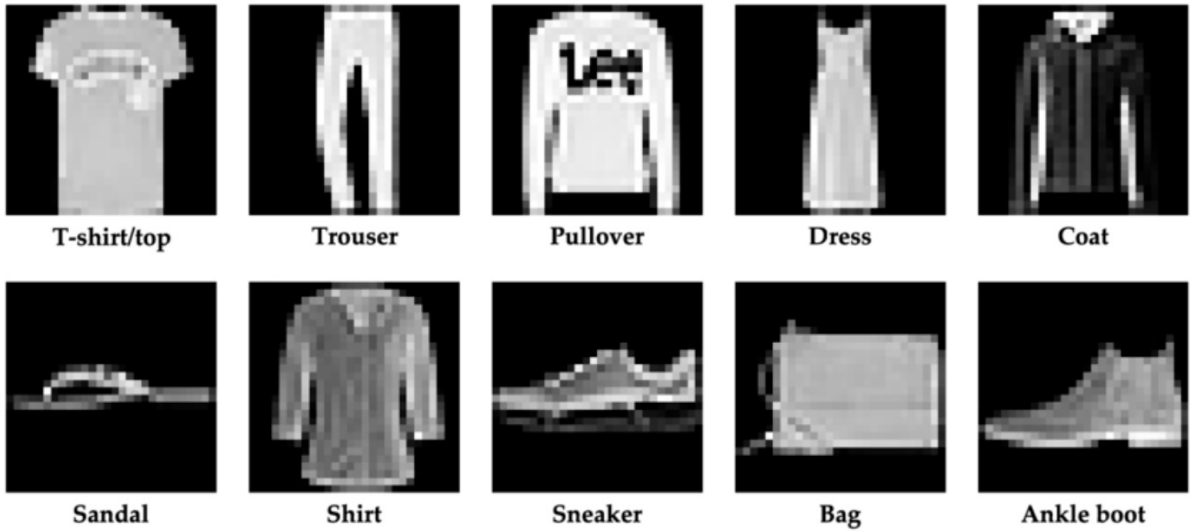


What if we normalized in between layers?



$z_1, z_2, \ldots, z_m$ $\quad \mu_B = \frac{1}{m}\sum_{i=1}^{m} z_i$ $\quad \hat{z}_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$ $\quad \mathbb{E}[\hat{z}] \approx 0 \quad Var[\hat{z}] \approx 1$

Example: FashionMNIST



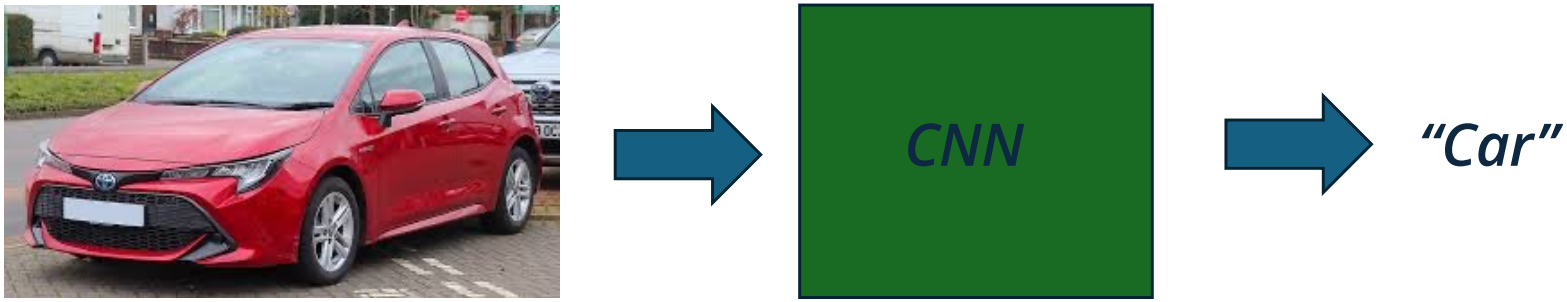
How do we “see” images?

When we see, we don’t look at pixels…



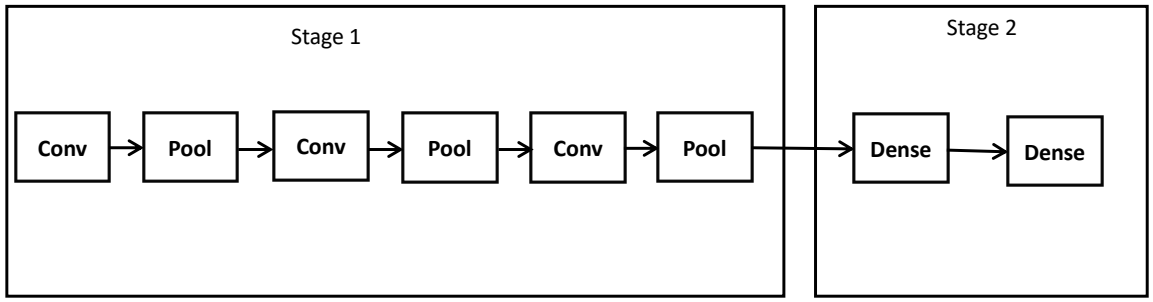

…we identify features

CNN architecture (high level)



We don't use raw pixels directly. We transform the pixels through a feature detection process called "convolution"

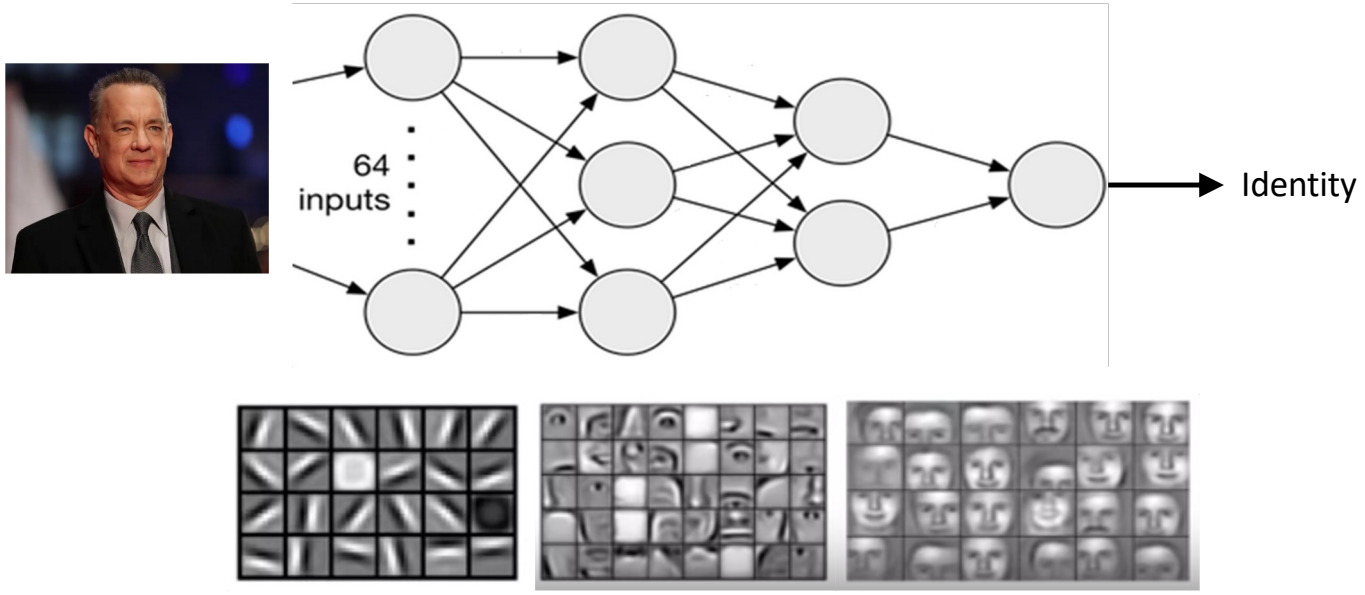
CNN architecture (low level)



Feature identifier *Classifier*

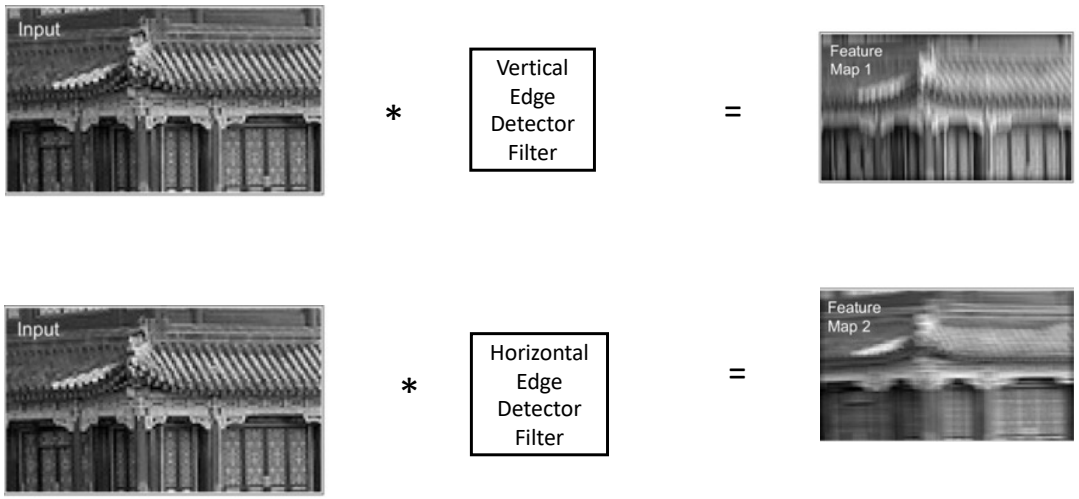
Why is deep learning so powerful?

3) Hierarchical feature learning

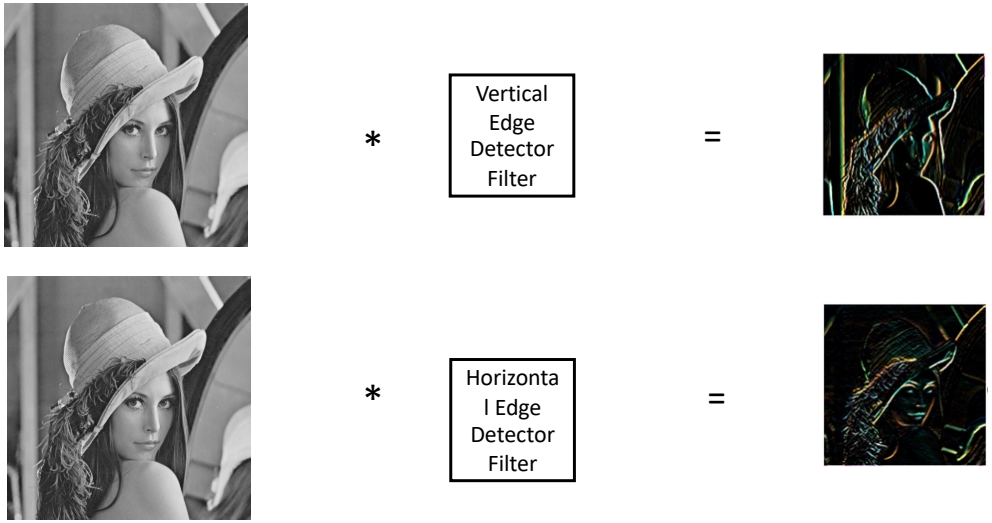


Convolution

The convolution operation is the "feature detector" that filters for certain kinds of information.

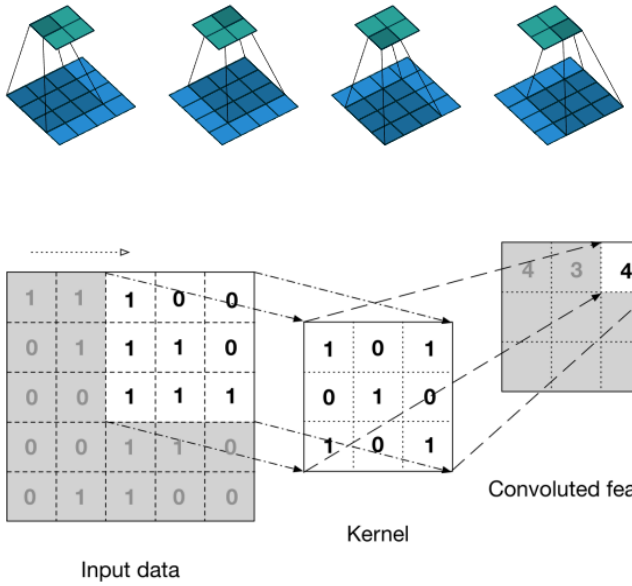


Convolution



Kernel = feature detector
Convoluted feature = feature map

Convolution example #1



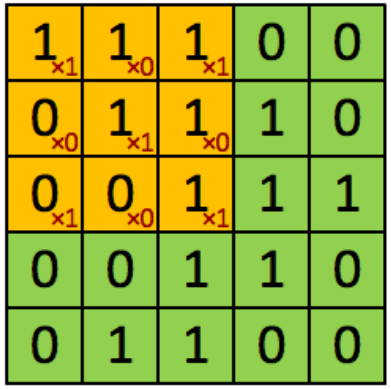
Stride = 1

Deep Learning, Oreilly
https://arxiv.org/pdf/1603.07285.pdf

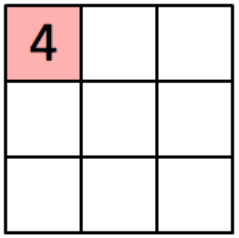
The output image size changes! (”valid convolution”)

Input (N x N) + Filter (F x F) = output (N-F+1) x (N-F+1)

Convolution example #2



Image



Convolved Feature

Vertical Edge Filter

-1	0	1
-2	0	2
-1	0	1

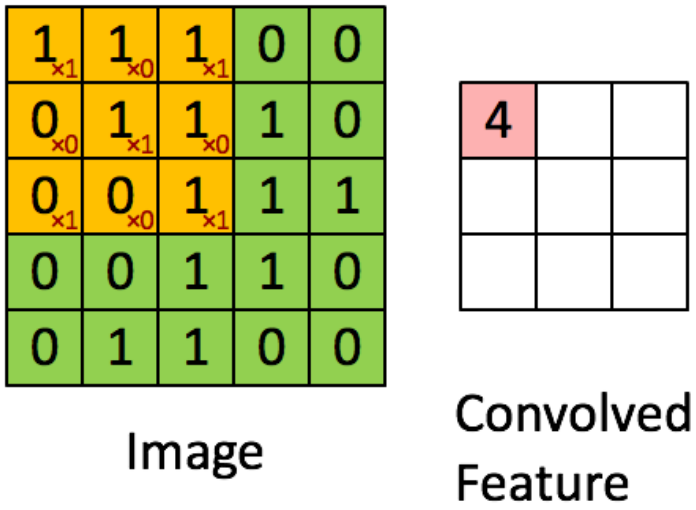
Horizontal Edge Filter

1	2	1
0	0	0
-1	-2	-1

https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

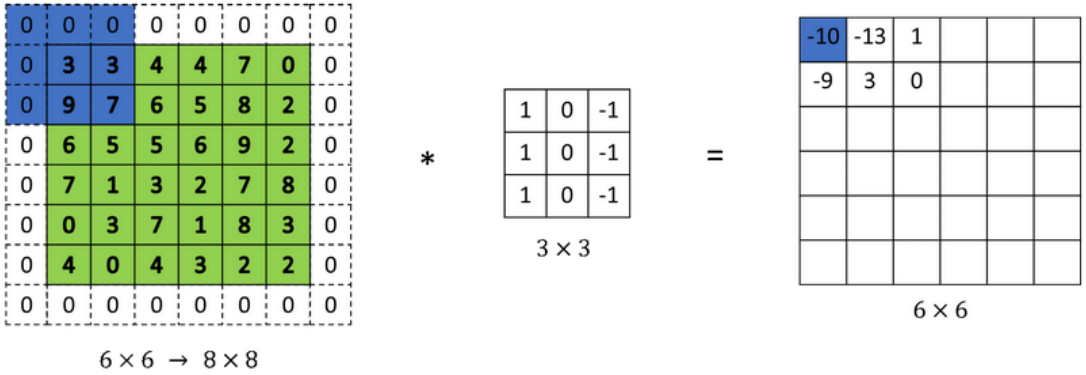
Convolution with padding

How do we avoid losing information at the edges due to underrepresentation?



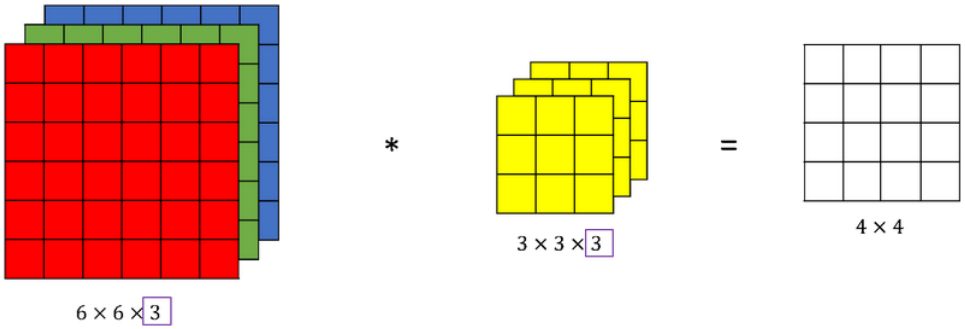
Convolution with padding

How do we avoid losing information at the edges due to underrepresentation?



"Same convolution" keeps the image size constant

What about multiple filters?

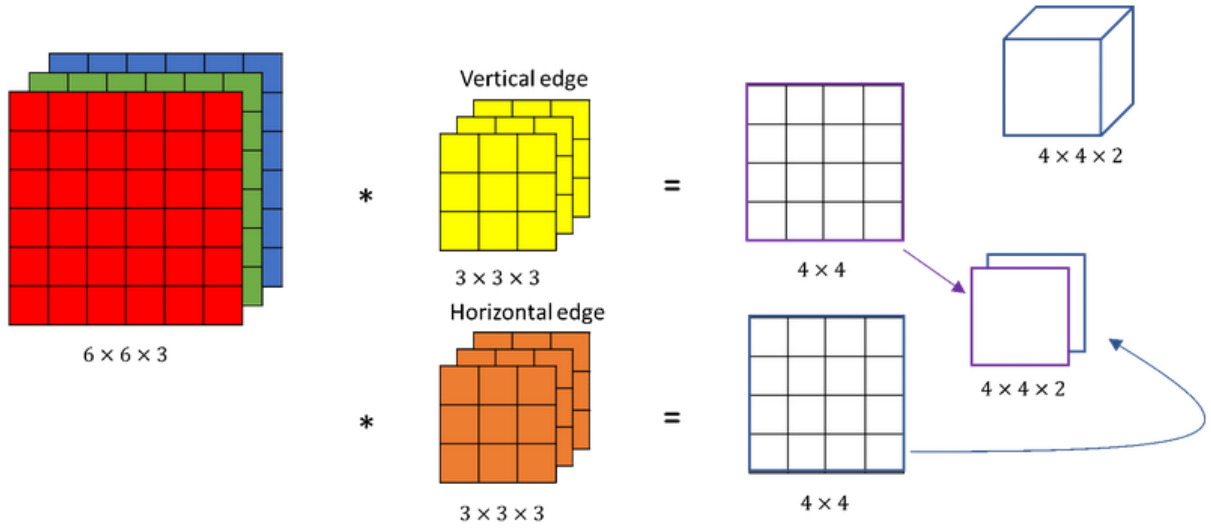


We simply need to match the number of channels and do a bit more multiplication

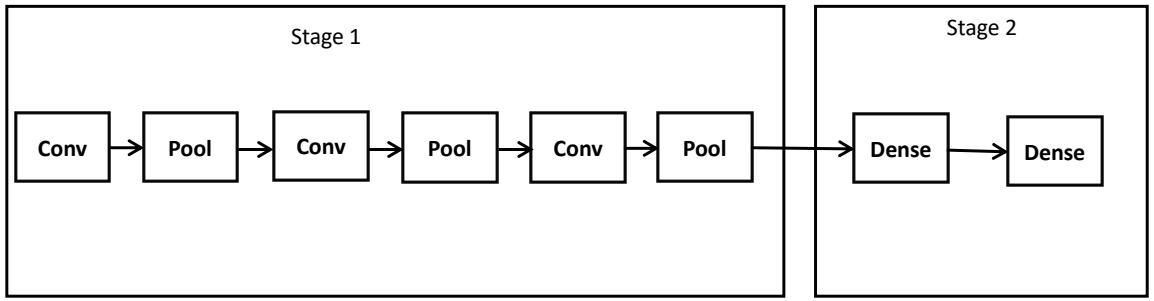
Make sure your matrix/tensor sizes match!

What about multiple filters?

We can expand this process to multiple filters per channel



CNN architecture (low level)

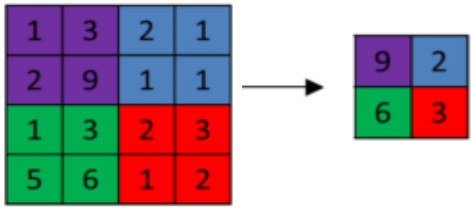


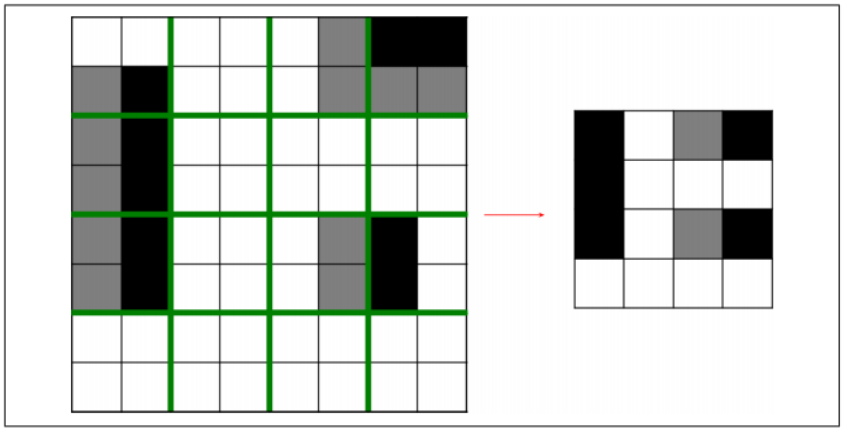
Feature identifier *Classifier*

Usually just a "regular" neural network

Pooling

Pooling enables robust feature recognition through translational invariance





1) Overlay a 2x2 grid

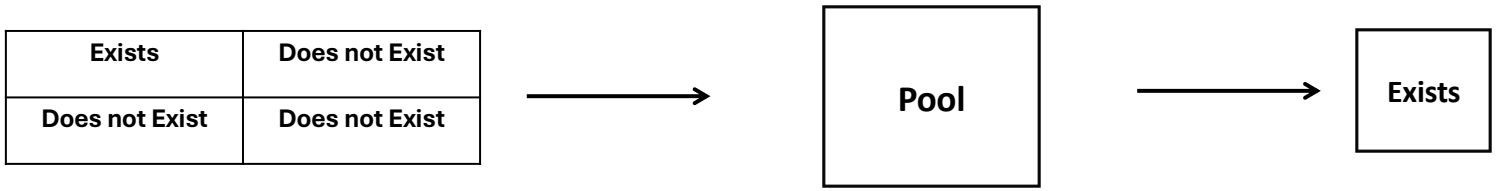
2) Find the max value

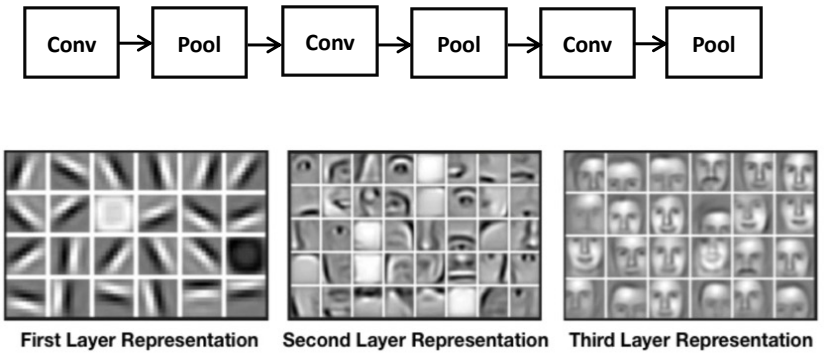
3) Pass that value through

Pool size = 2, stride =2 (non-overlap is most common)

Convolution & Pooling

Convolution detects the feature and the highest number is the best matching location





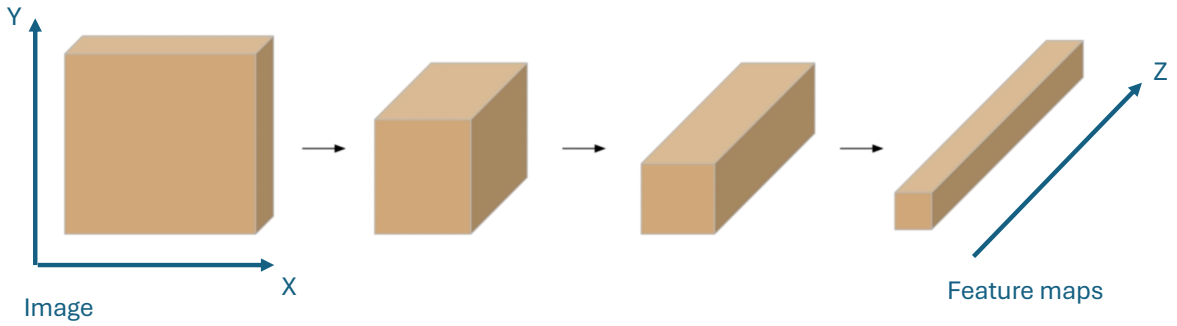
Why do we repeat convolution & pooling?

The image shrinks and we lose spatial information, right??

Remember the feature maps

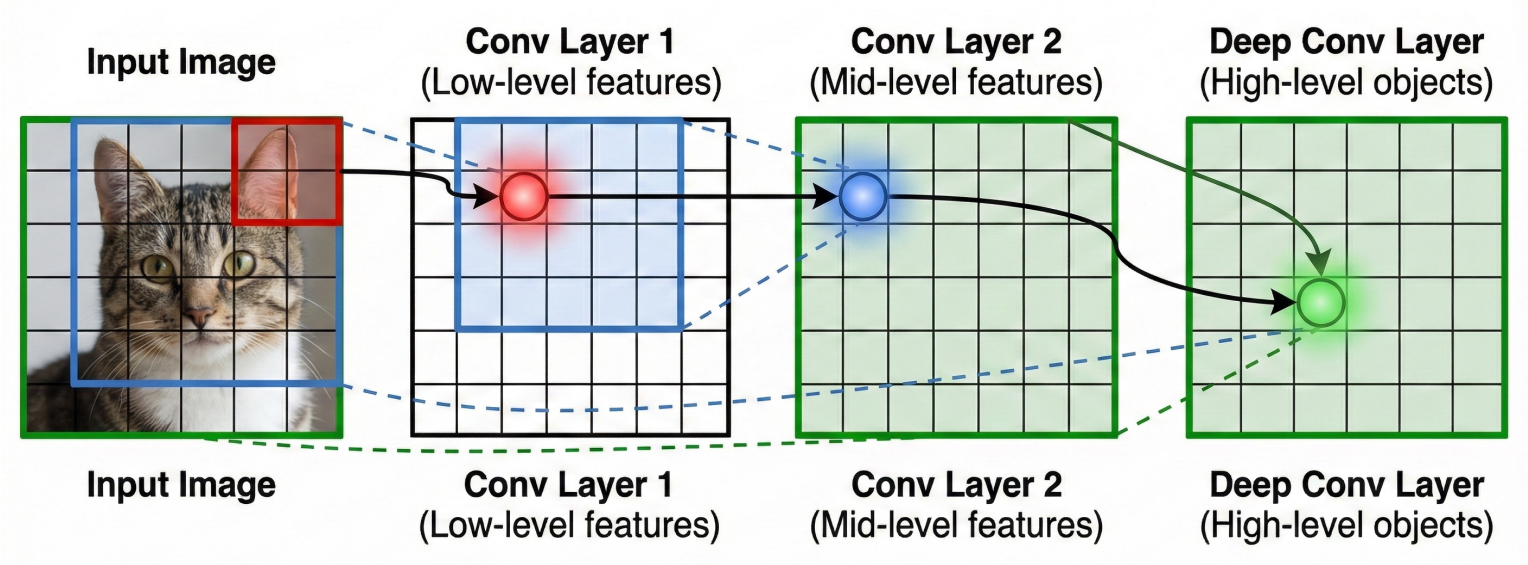
Why do we repeat convolution & pooling?

We add more feature maps (convolutions) at each layer … and we don't really care <u>where</u> a feature is found!

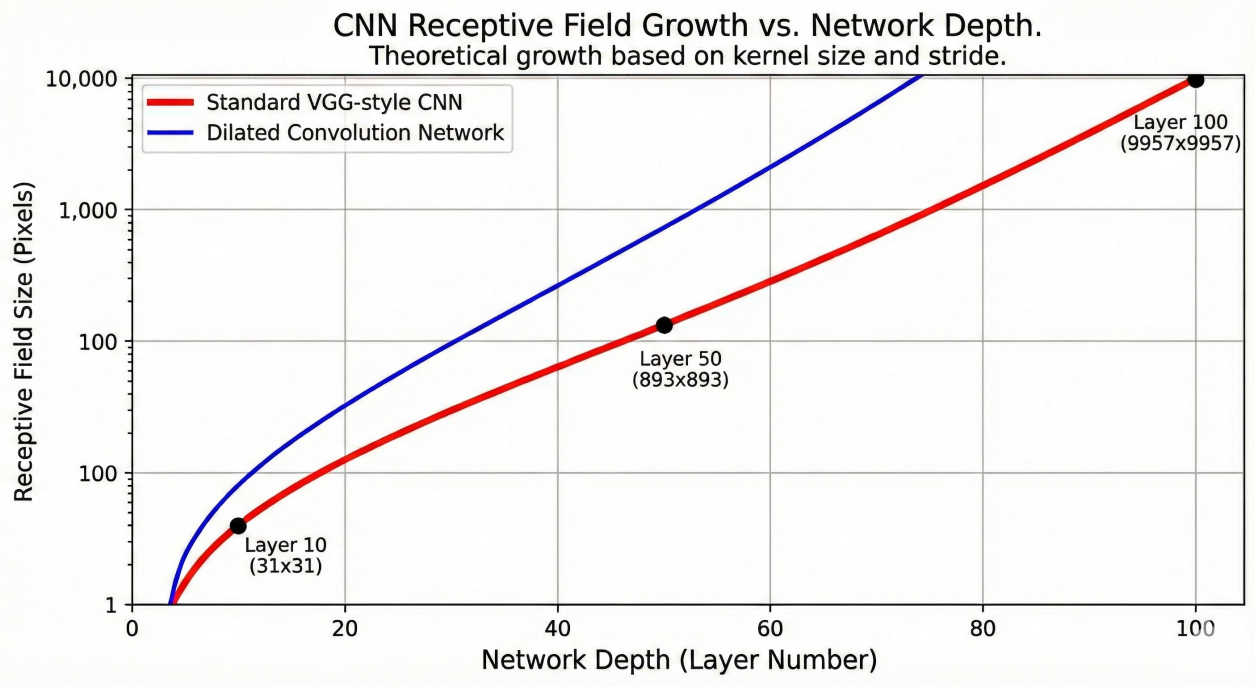


We trade spatial information for feature information!

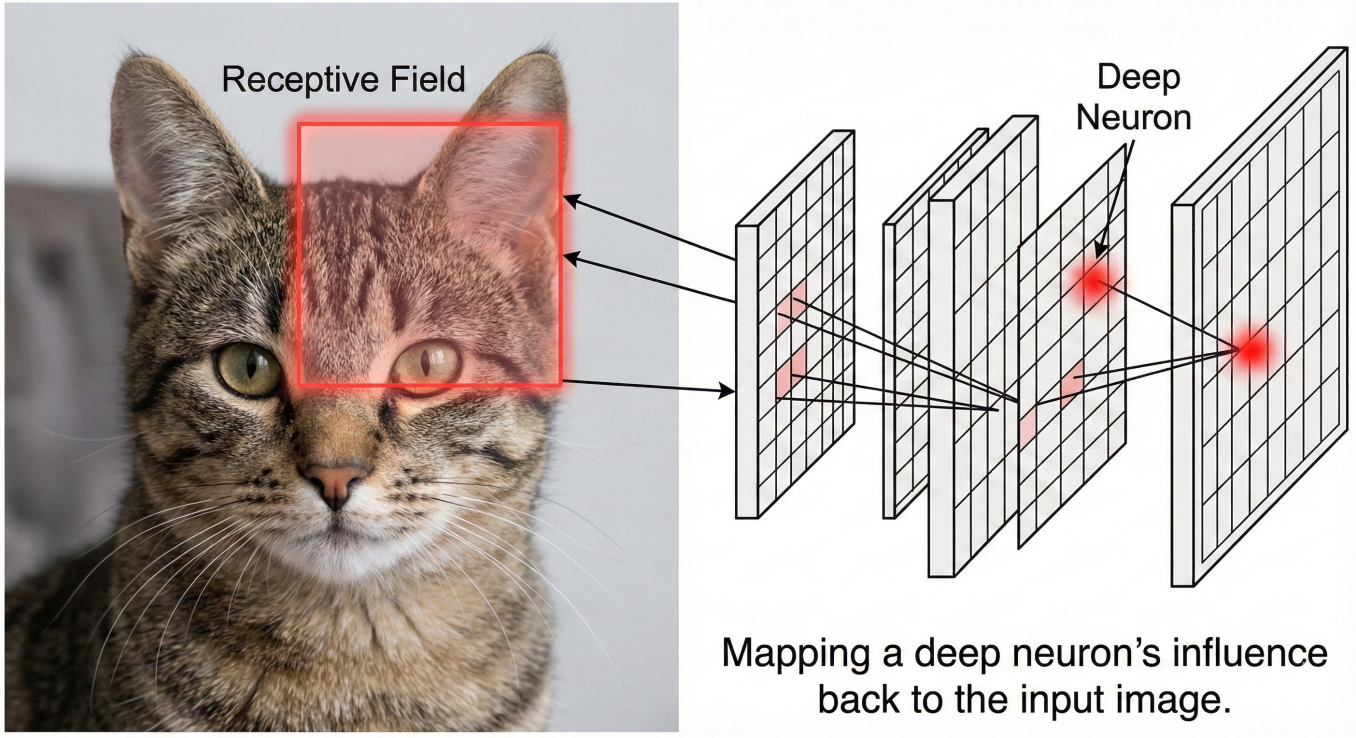
The receptive field



The receptive field



The receptive field



CNN hyperparameters

1) Feature maps per layer
   Increase per layer (32 -> 64 -> 128 -> 128)

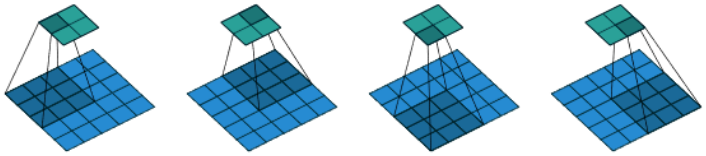
2) Convolution filter size
   Hold size constant (3x3, 5x5, 7x7)

3) Convolution/Pooling pattern
   Repeat the conv -> pool -> conv -> pool pattern

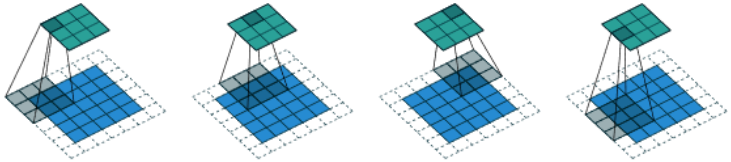
Also need learning rate, # of hidden layers, # of units in the hidden layer, dropout, etc

Strided convolution: another possibility

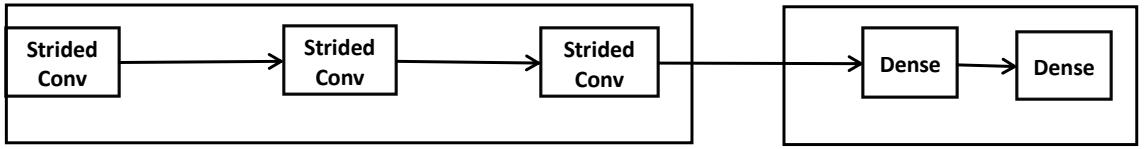
Avoid pooling by shifting the kernel more than one step



Convolving a 5x5 with a filter of 3x3 with a stride of 2

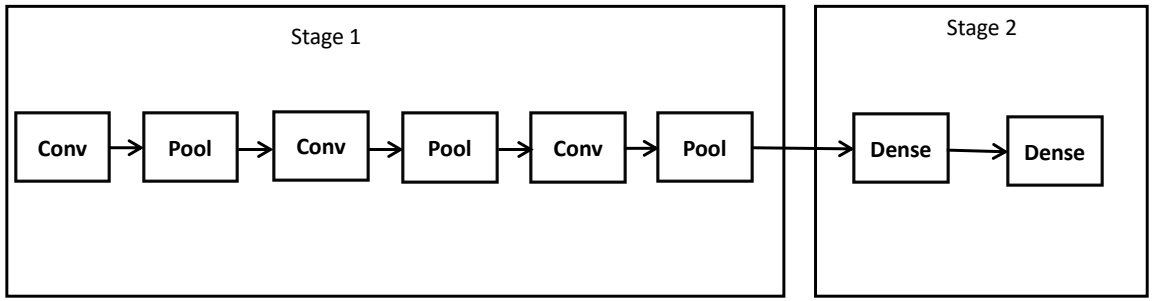


Convolving a 5x5 with a filter of 3x3, stride of 2 and padding of 1



Output size: $\lfloor \frac{n-f+2p}{s} \rfloor + 1 \times \lfloor \frac{n-f+2p}{s} \rfloor + 1$

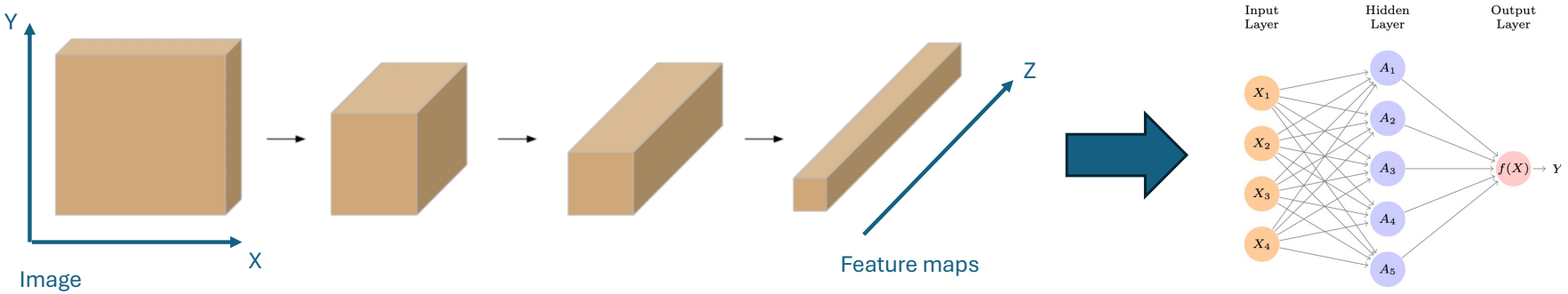
CNN: stage 2



Feature identifier *Classifier*

Build the classifier like a ”regular” neural network

Re-shape stage 1 inputs

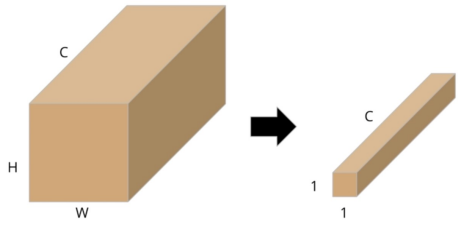


1) Flatten the whole image (same size images)

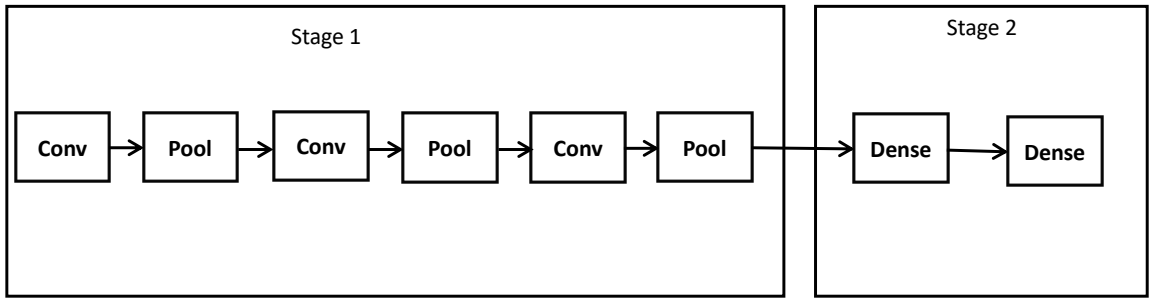
100 x 4 x 4 output becomes a 1600 dimension 1D vector
100 x 8 x 8 output becomes a 6400 dimensional vector

2) Global max pooling

Use the max value from each feature map!



Putting it all together



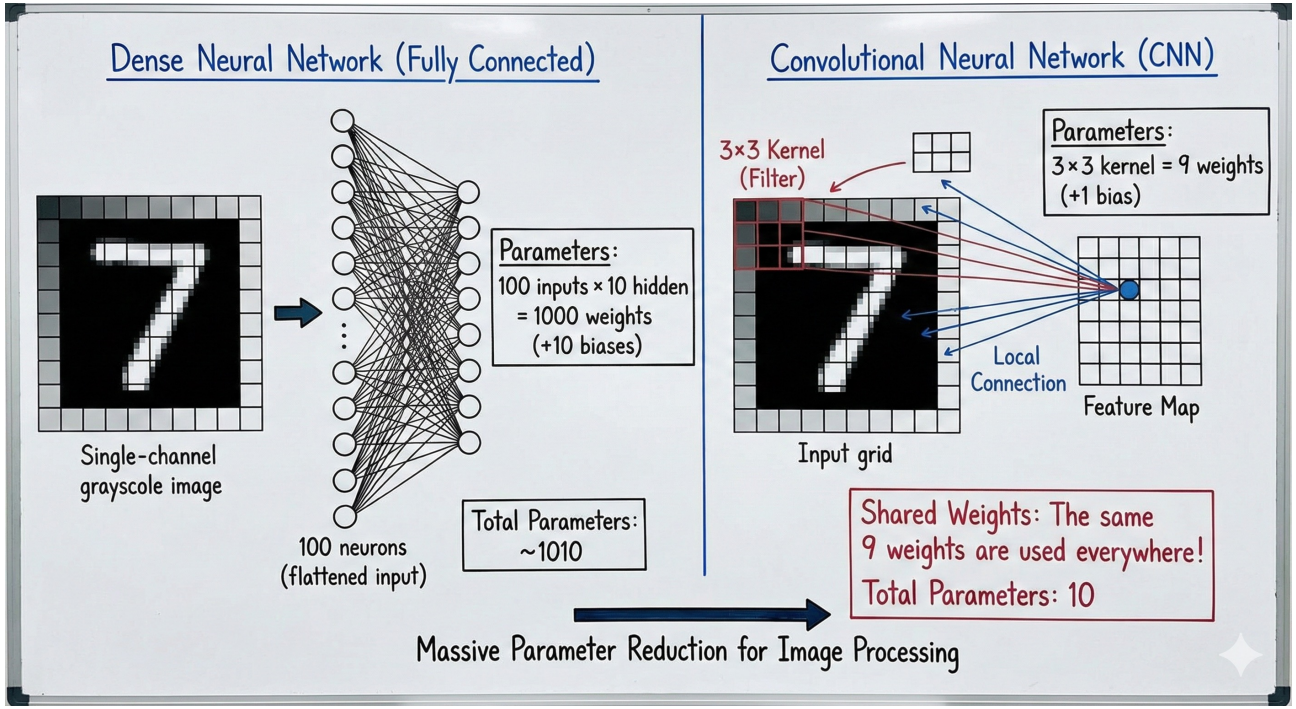
Conv -> Pool -> Conv -> Pool
Strided Conv -> Strided Conv

Flatten
Global Max Pool

Dense -> Dense -> …

Parameter efficiency

*CNNs have *substantially* fewer parameters than standard NNs*



Parameter efficiency

Local connectivity: *instead of every neuron connecting to every pixel (DNN), CNN neurons only connect to local pixels (one kernel overlap)*

Weight sharing: *instead of every neuron having its own weights, we reduce to *just* the kernel parameters*

This efficiency is why CNNs are still popular, even in the transformer era

Code example

NOTE: depending on time, perhaps push GANs to next week?

Generative Adversarial Networks

GANs are the original generative deep learning model
Diffusion models are now SOTA, but GANs are still key due to speed (and are often part of diffusion/GAN hybrids)

The 2-step GAN structure

1) Generator (creates “fake” image)
2) Discriminator (attempts to determine if an image is fake)

The 2-step GAN structure

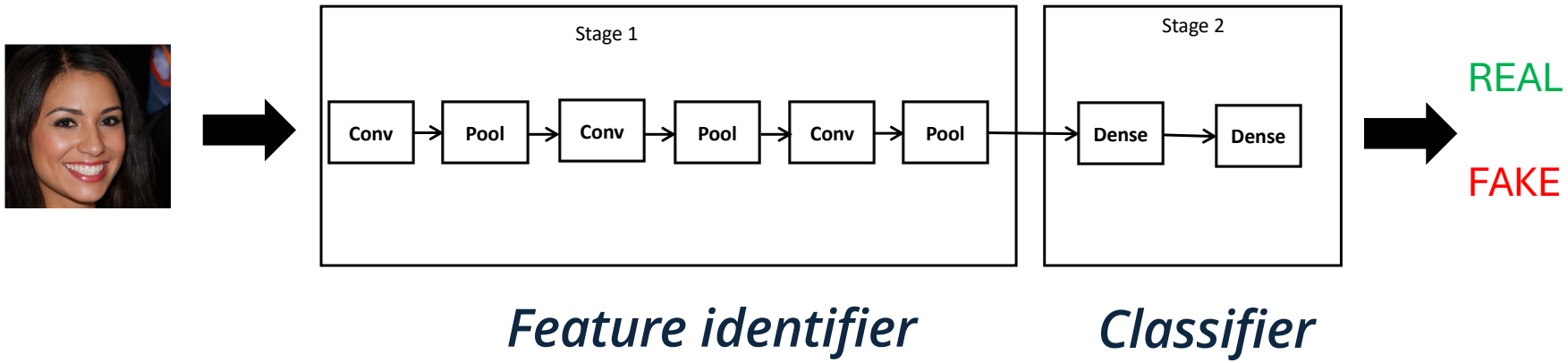
1) Generator (creates “fake” image)
2) Discriminator (attempts to determine if an image is fake)

VS

The discriminator

We can use a CNN to classify images as “real” or “fake” (just like “dog”, “cat”, etc)

GANs with convolutions are called “deep convolutional networks”.
GANs can also simply be densely connected layers!

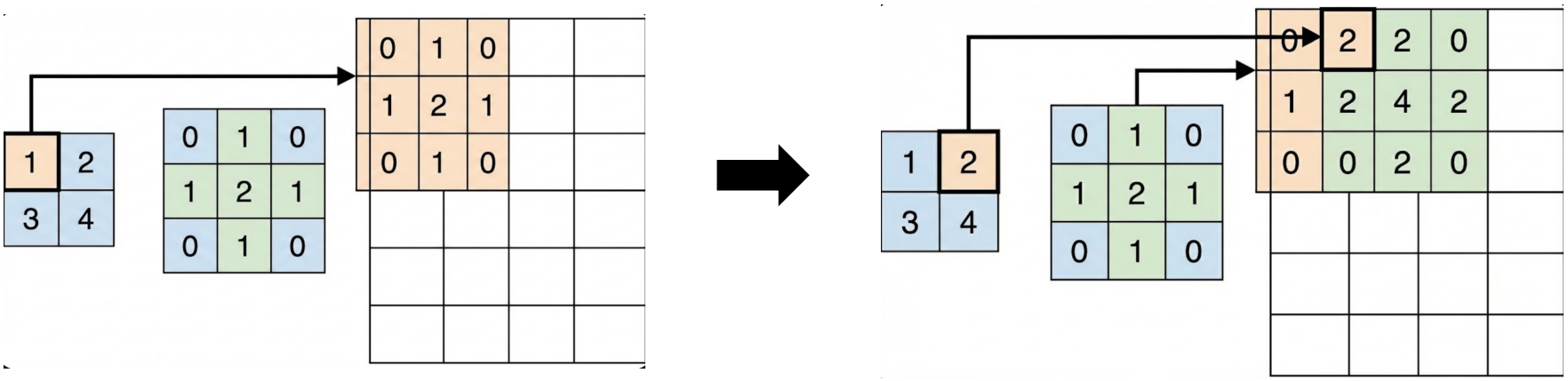


What about the generator?

We perform reverse convolution, otherwise known as “transposed convolution”

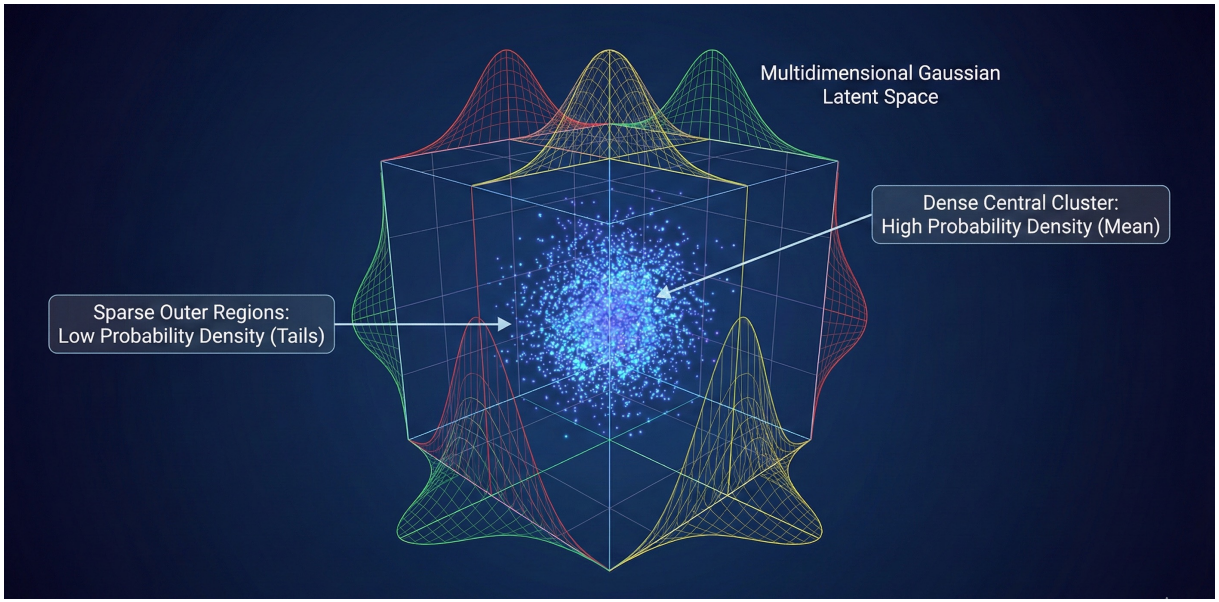
Instead of using a kernel to downscale, we use a kernel to upscale

We start with a random number in “latent space”, and then expand it to an image



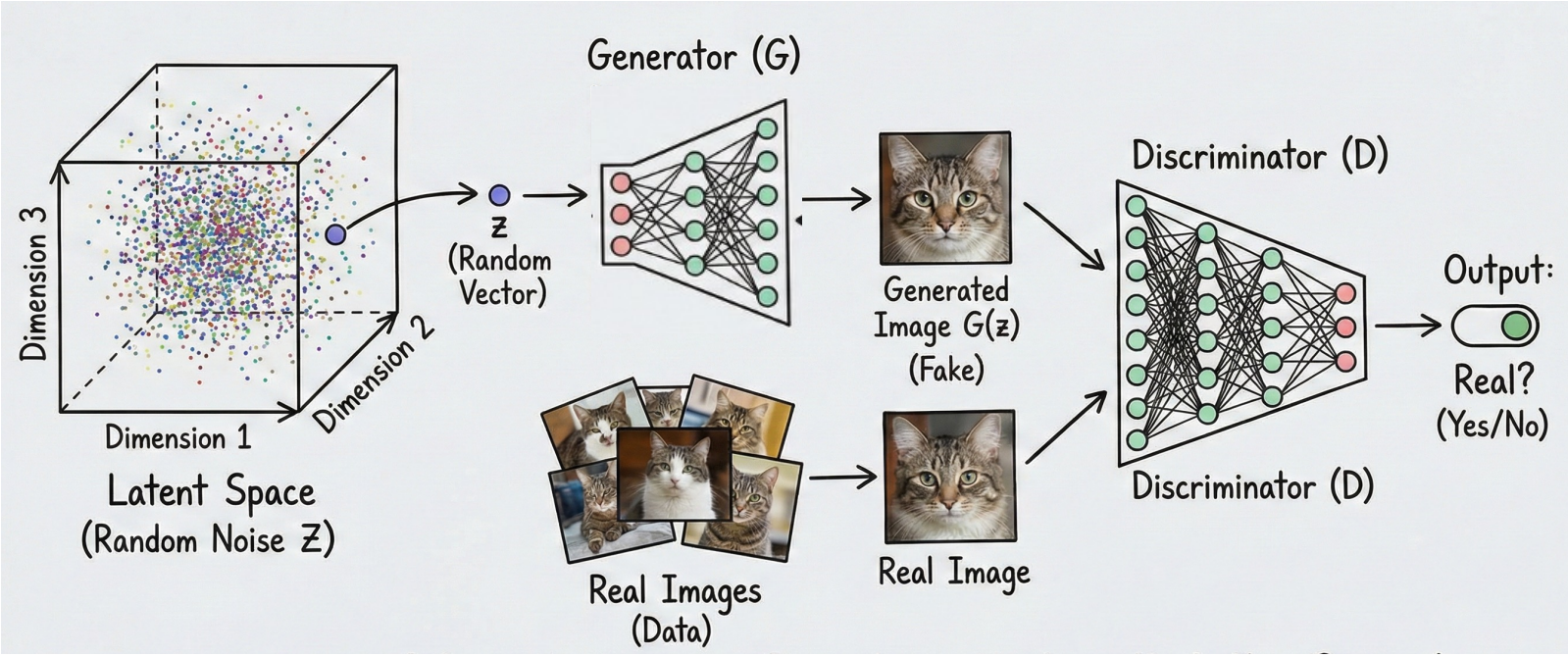
The latent space

We need to start with *something* in our generation process

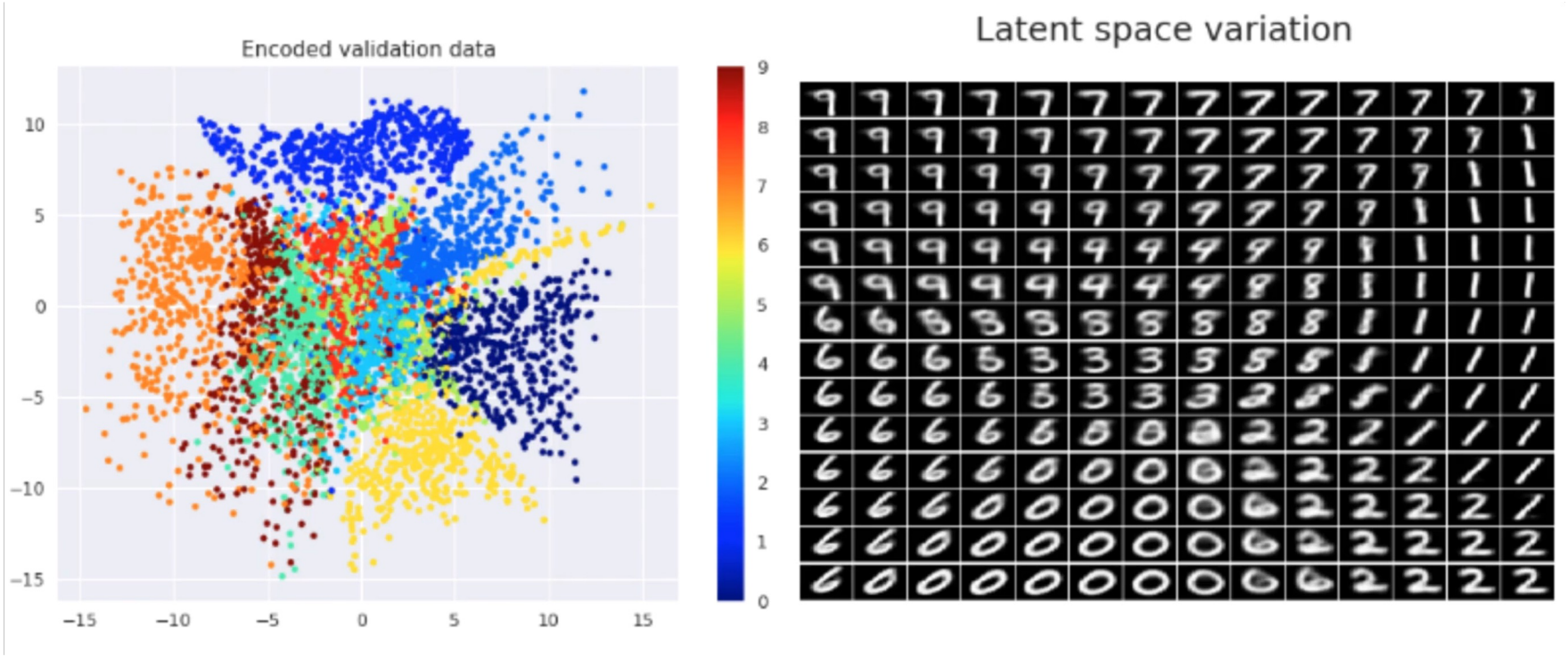


The latent space

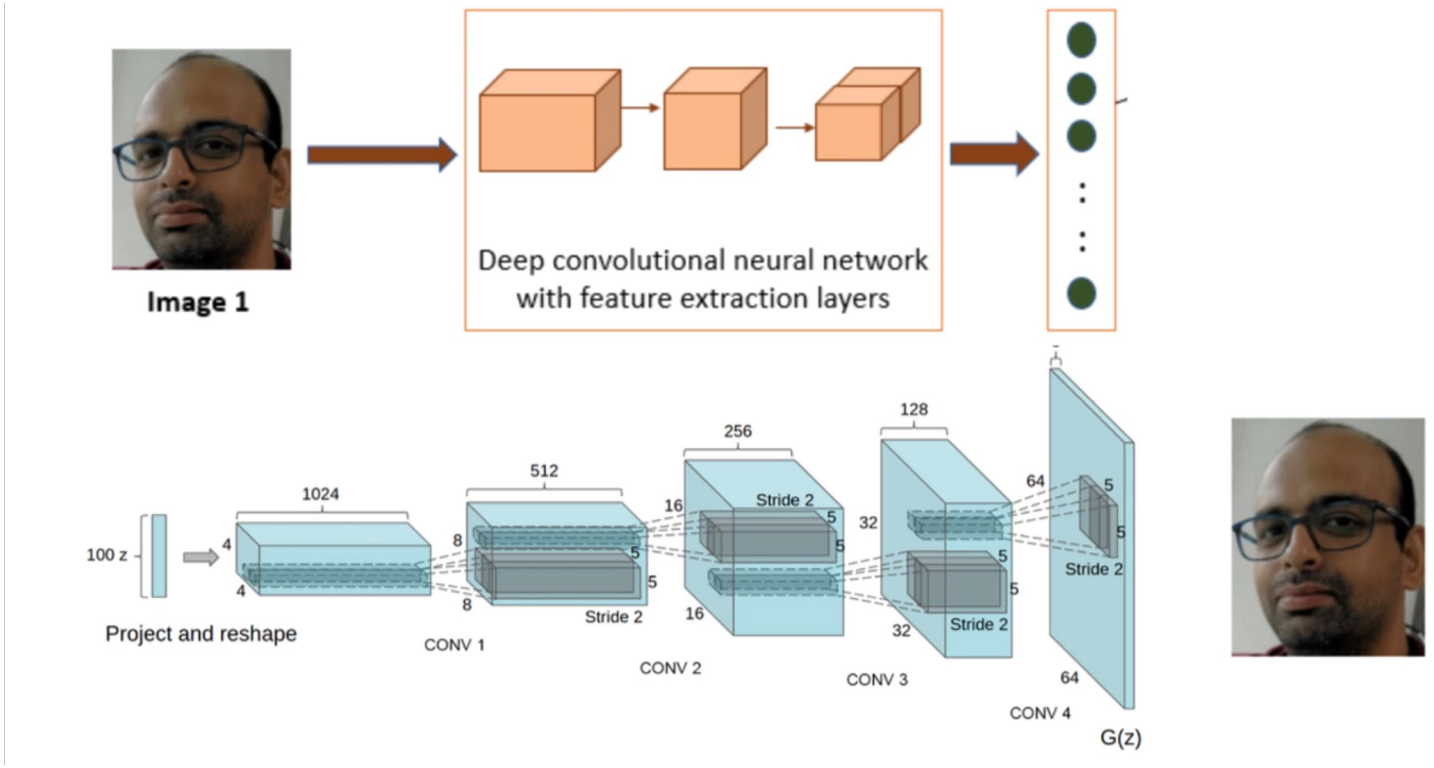
We need to start with *something* in our generation process



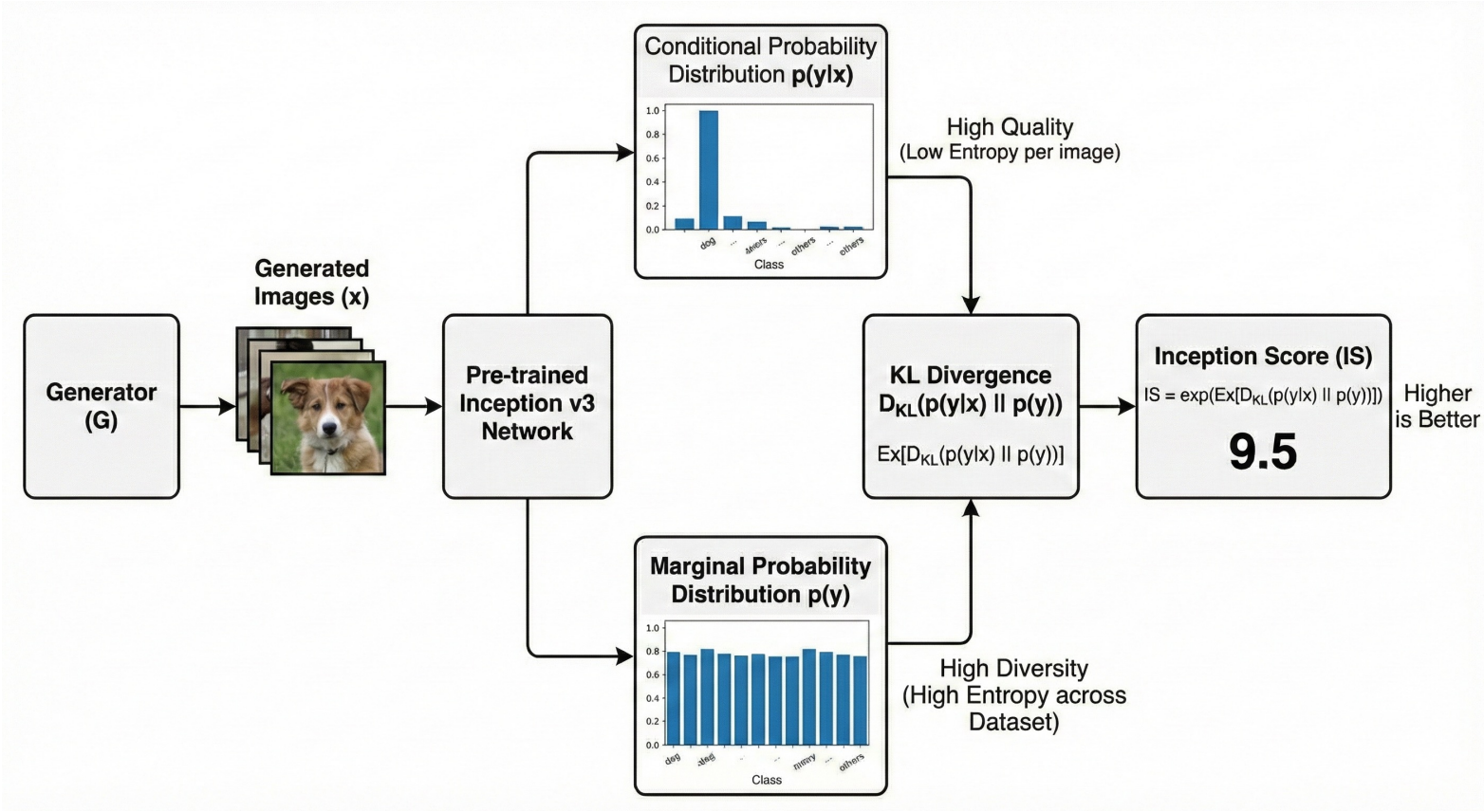
The latent space



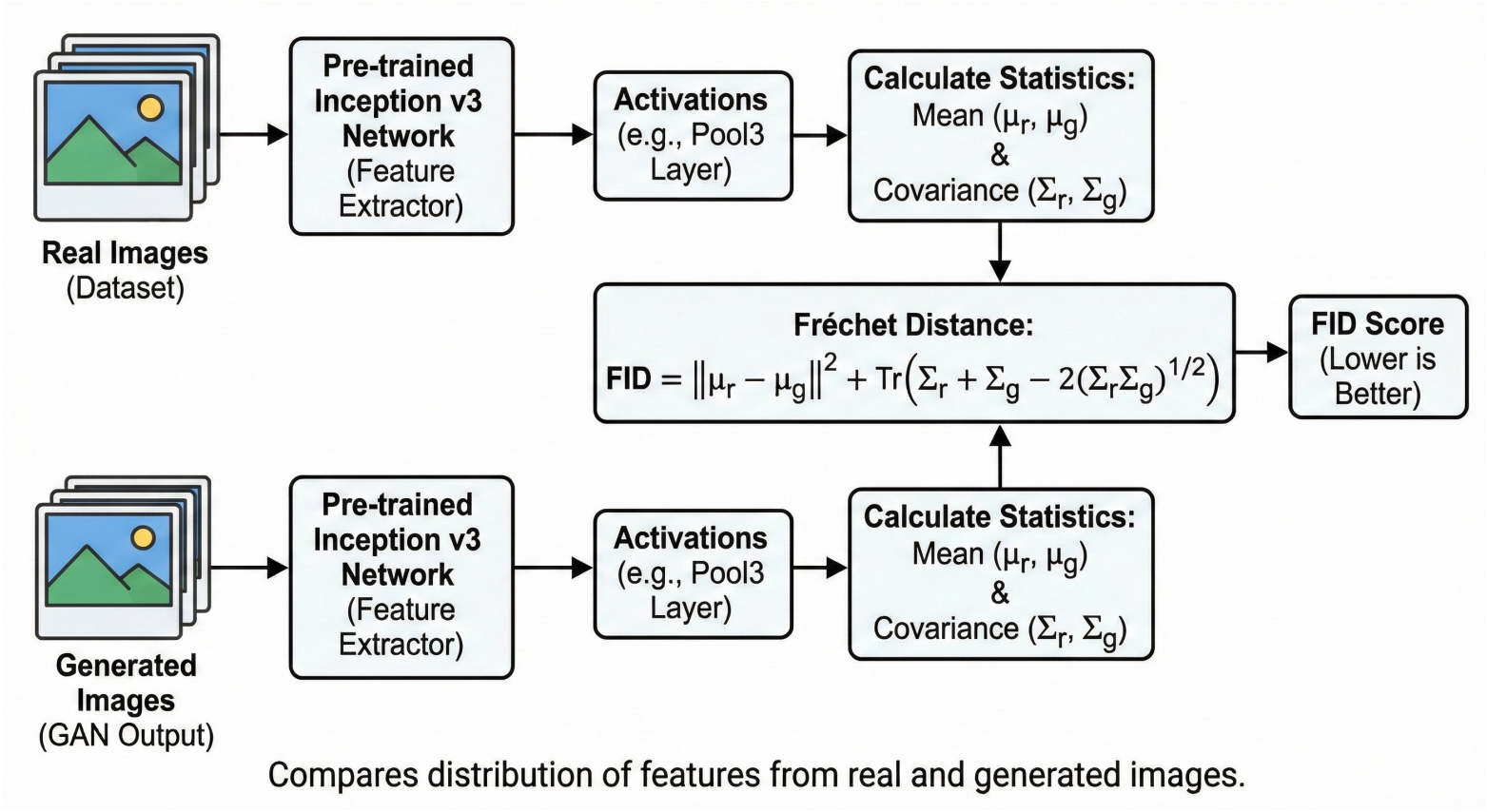
GANs: the big picture



Inception Score



Frechet Inception Distance



You can also use your eyes

Code example (GANs)