ML & Particle Physics

Eilam Gross

Refs:

M. Pierini - private correspondence

Machine Learning in High Energy Physics Community White Paper, https://arxiv.org/abs/1807.02876

Ack Jonathan Shlomi & Sanmay Ganguly

Outline

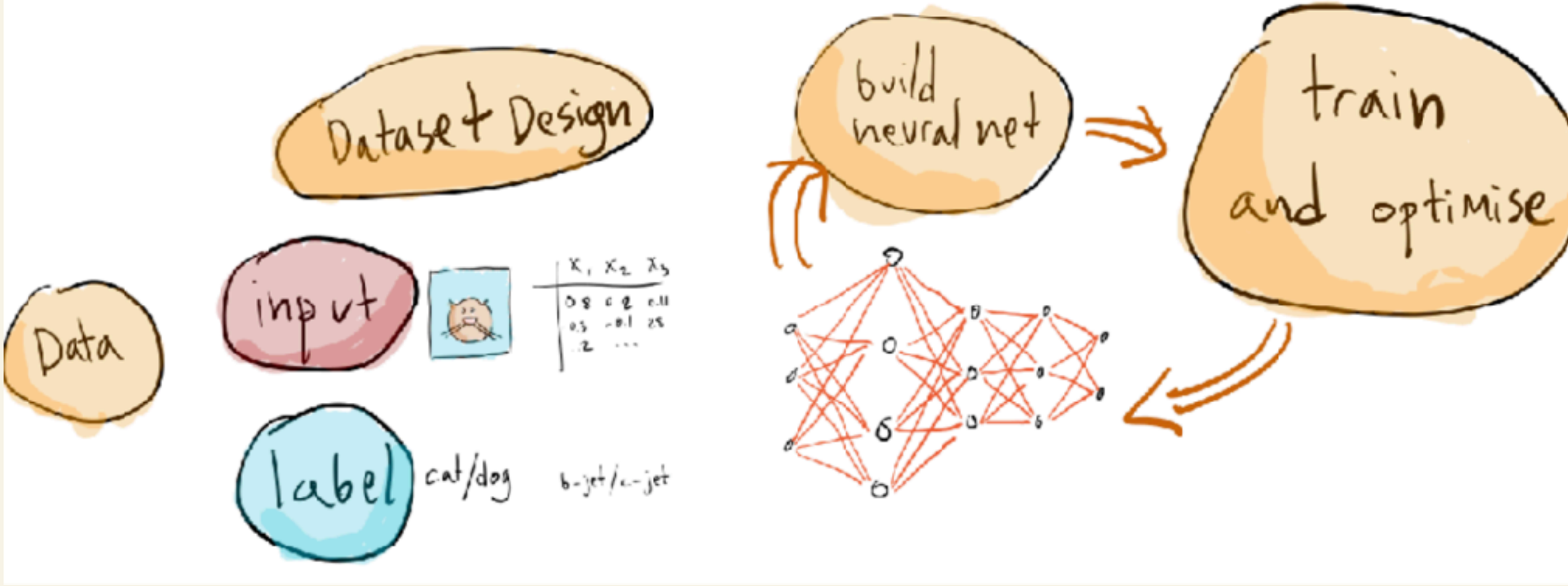
- What is ML and DL (Deep Learning)
- Why ML
- Major Architectures:
  - DNN
  - Convolutional NN
  - Recursive NN
  - Generative Models:
    - GAN
    - Autoencoders
- Note: Graph NN & Reinforcement Learning are not covered

What is ML

- Let X be the observed DATA (tracks, vertices,....)
- Let Y be a lower dimensional target space (jet flavour...)
- Try to find a function f(X—>Y)
- Optimize the function with a metric (loss function) L(y,f(x))

What is ML

- Supervised Learning:
  - Let there be pairs of training DATA $\{x_i, y_i\}$
  - Let there be a family of functions $f_\Phi(x): X \to Y$ parametrised by Φ
  - Minimize the loss $L(f(x), y)$ with respect to Φ. $\Phi = (\omega, b)$



Machine Learning and HEP

- BDT and NN used in HEP for quite some time
- Typically, variables relevant to the physics problem are selected and a machine learning model is trained for classification or regression using signal and background events (or instances).
  - Classification (predicting a label)
    - Particle ID
    - Heavy flavor tagging
    - top/W/H Tag
  - Regression (predicting a continuous quantity)
    - Use cluster shape information to improve calorimeter energy resolution
    - Pileup estimation

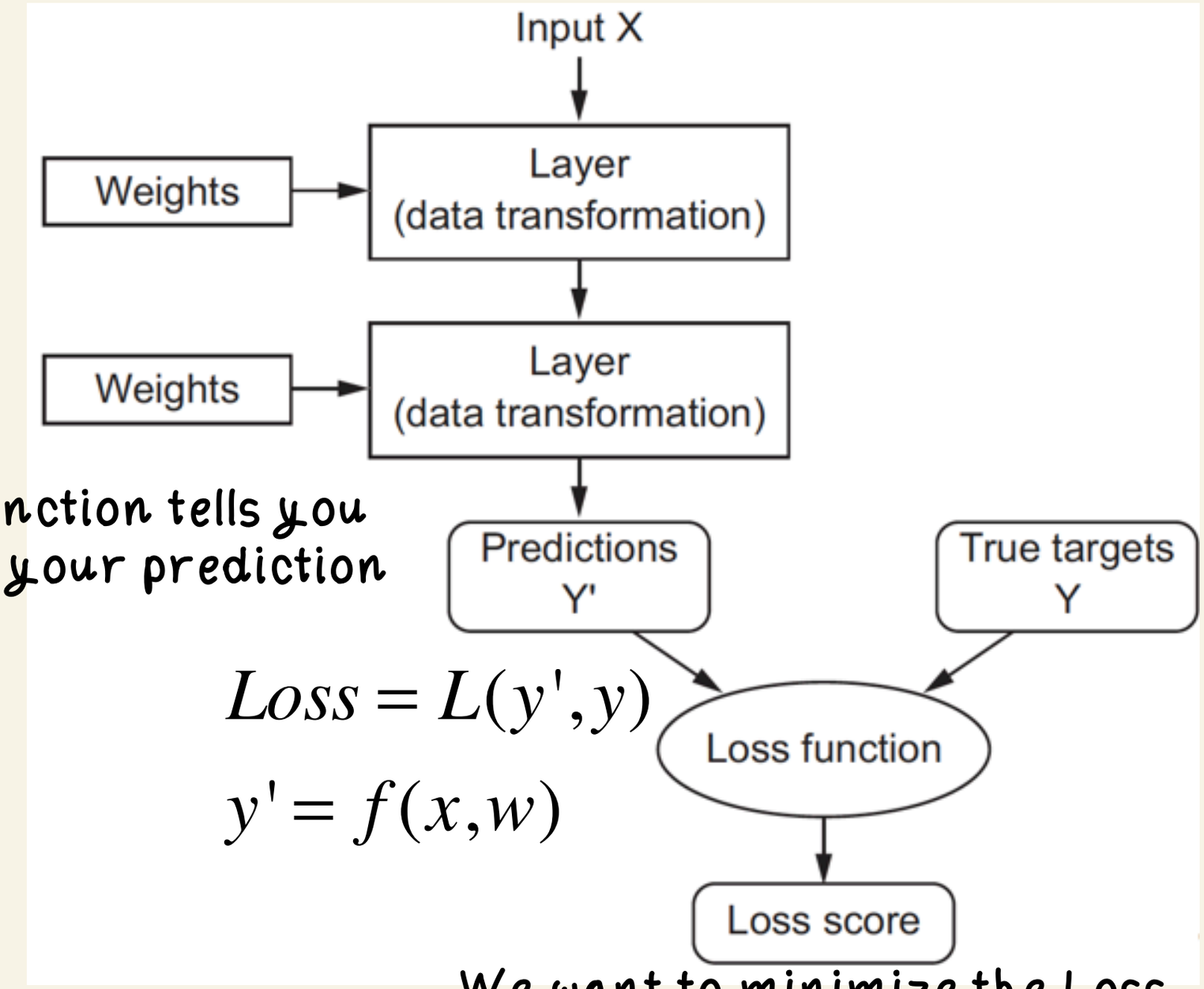
ML and HEP

- ML can do much more than BDT and NN
- HEP is not only about classification or finding new Physics. It is also about Quality of Data and Detectors, Simulations and triggering, where ML is starting to take bigger roles.
- Increase computer power and sophistication in algorithms brought the Deep Learning revolution of the last decade, which is largely affecting HEP

Supervised Learning Neural Nets

Neural Nets

- Layers extract representation of the DATA fed into them, which are supposed to be more meaningful for decoding the DATA



The loss function tells you how good is your prediction

$$Loss = L(y', y)$$

$$y' = f(x, w)$$

We want to minimize the Loss

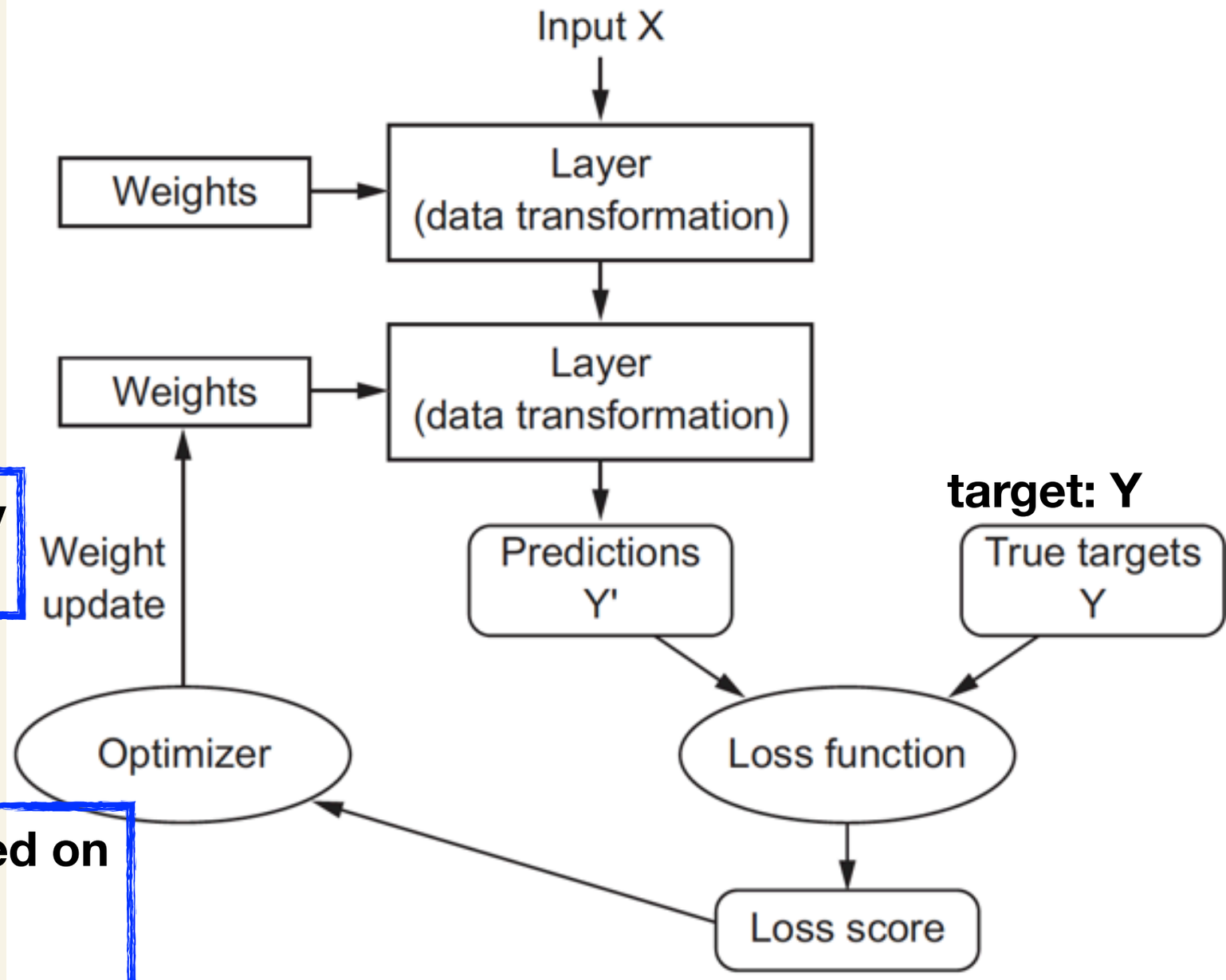
OPTIMIZATION

Loss score gives feedback

Initially weights get random values

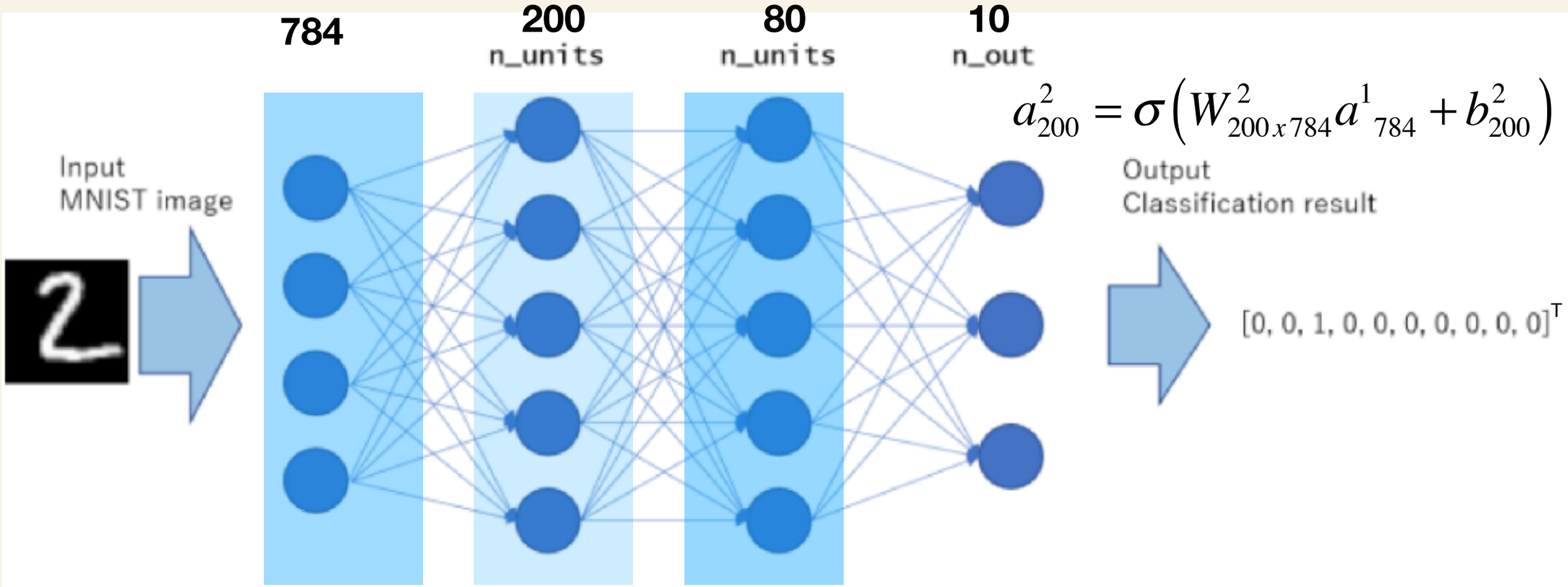
Via training the weights are slowly adjusted so the loss is decreased

The training algorithm is based on
Gradient Descent
Backpropagation



All that Layer Jazz

In MNIST you go, e.g. from $\mathbb{R}^{784} \rightarrow \mathbb{R}^{200} \rightarrow \mathbb{R}^{80} \rightarrow \mathbb{R}^{10}$



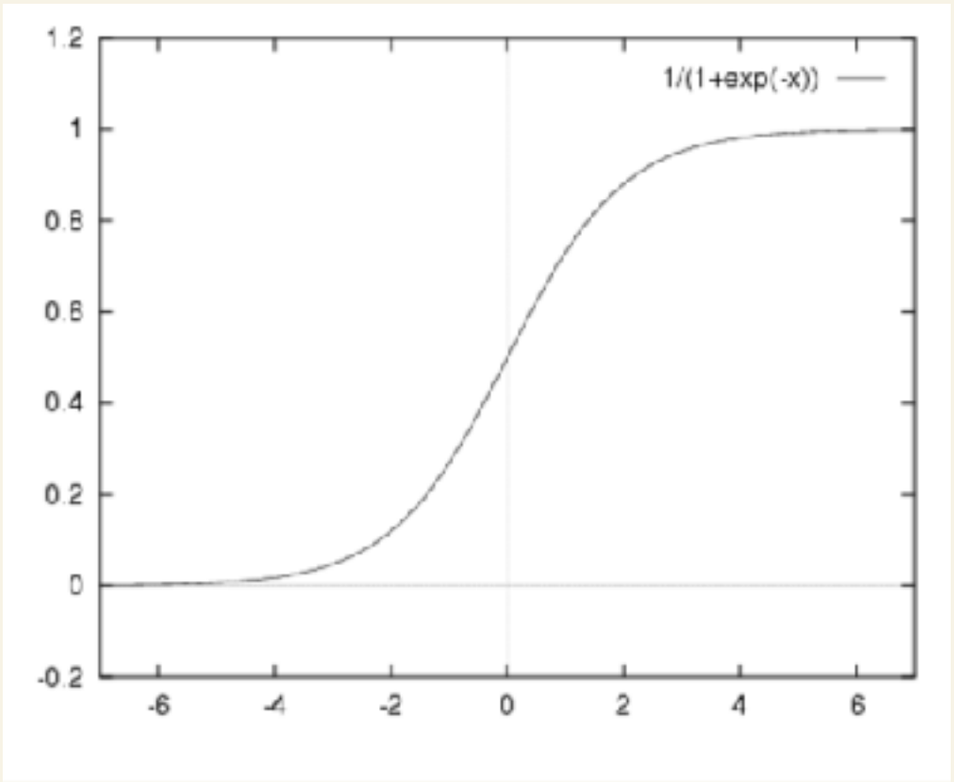
- The input layer X is mapped into internal "hidden" states layer h_i
- $h_{i+1}=g_i(W_i h_i+b_i)$, g is the nonlinear activation function

Activation Functions

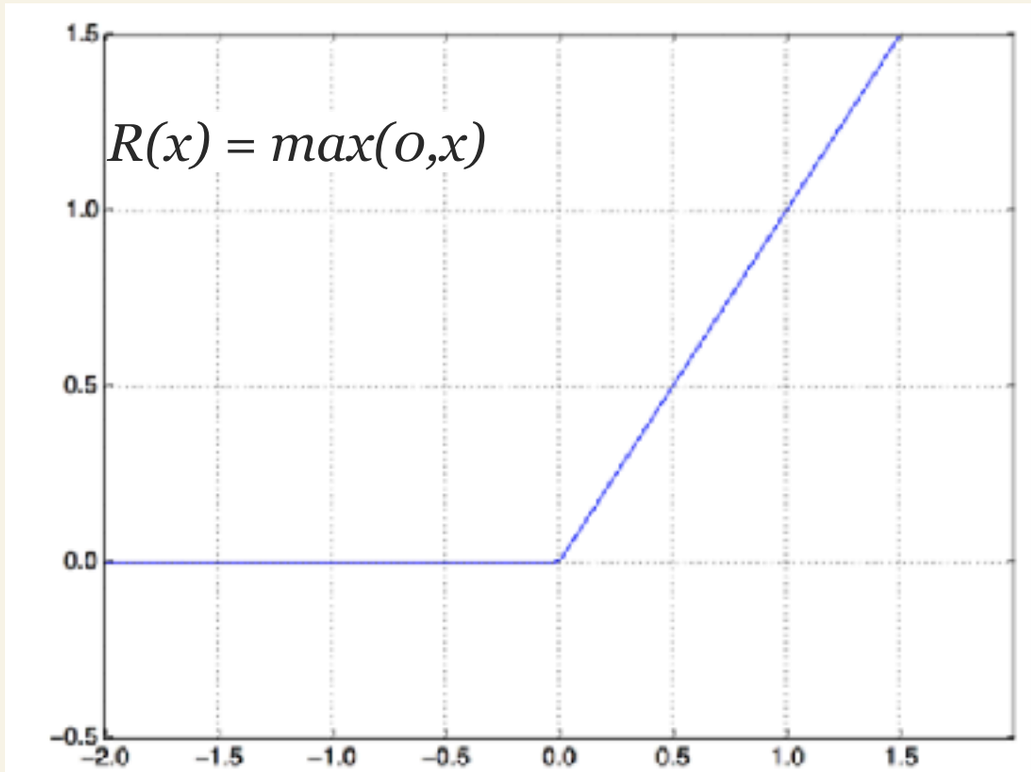
If activation is linear, the output can only be a linear combination of the inputs —>

Our Neural network would not be able to learn

$$\sigma\big(f(w)\big): L_N \rightarrow L_{N+1}$$



Sigmoid Saturates and kill Gradients



ReLu- Rectified Linear units

Many times you use ReLu **for all Hidden Layers and** Sigmoid **in the final layer as to output a probability (a score between 0 and 1)**

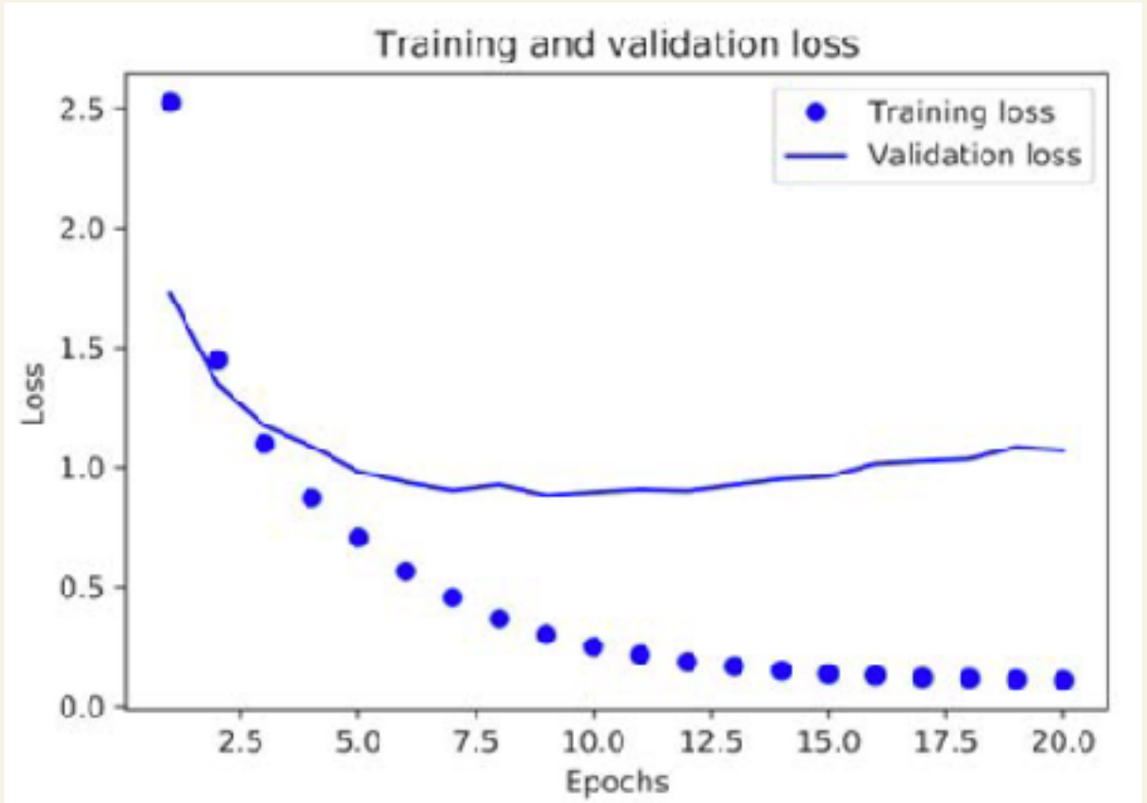
Training

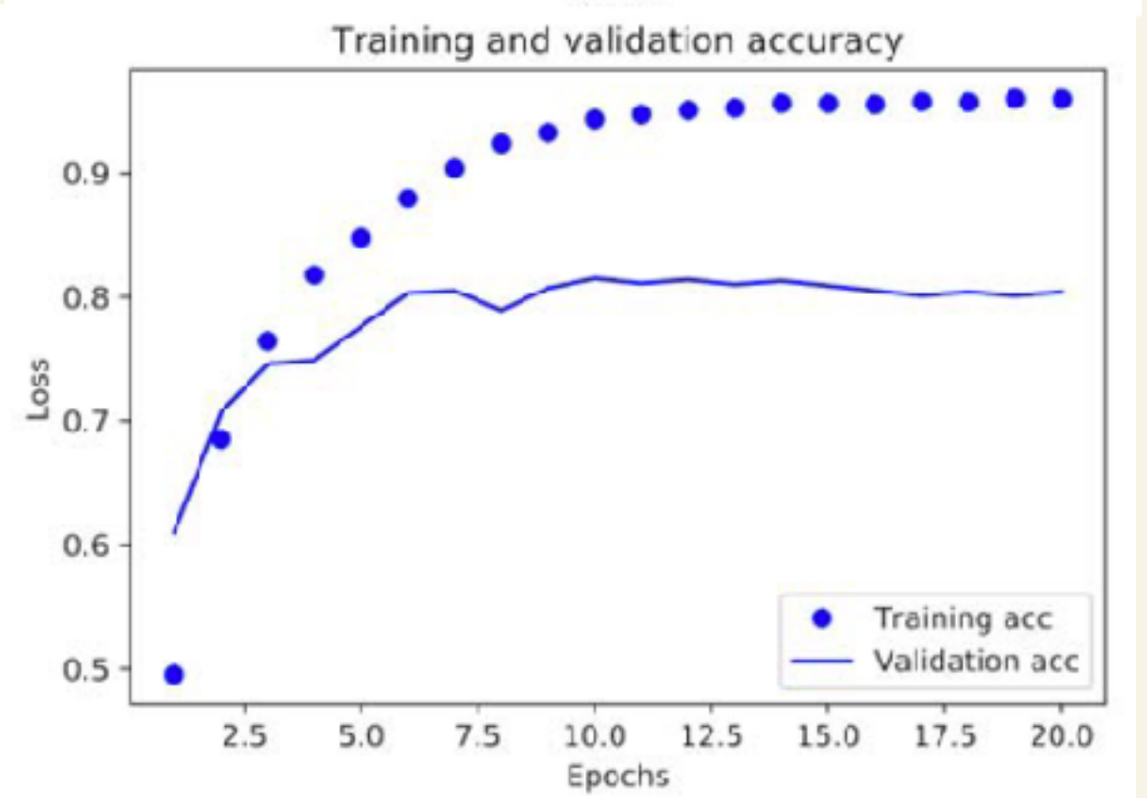
- We divide our labeled DATA into~80% training and 20% validation (sometimes also kept some DATA for test)

- In each epoch we run on a batch (1000s of samples) and adjust the weights

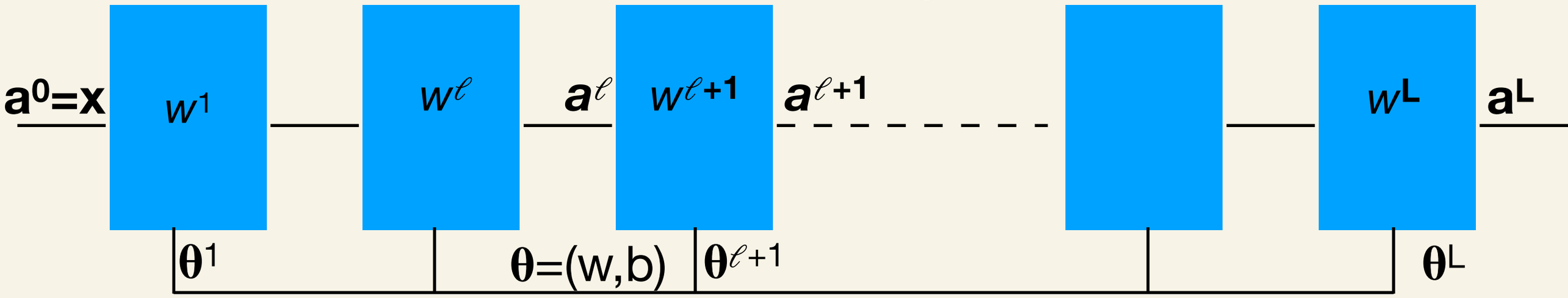
- Our success is measured by the accuracy of our predictions

- This gap between training accuracy and test accuracy is overfitting: Overfitting is a central issue





Backpropagation



Our goal is to find weights and biases that minimize $L(\theta)$

$\theta = (w,b)$ $\quad \Delta L = \sum_{\theta} \frac{\partial L}{\partial \theta_i} \Delta\theta_i \approx \vec{\nabla}_{\theta} L \cdot \Delta\vec{\theta}$

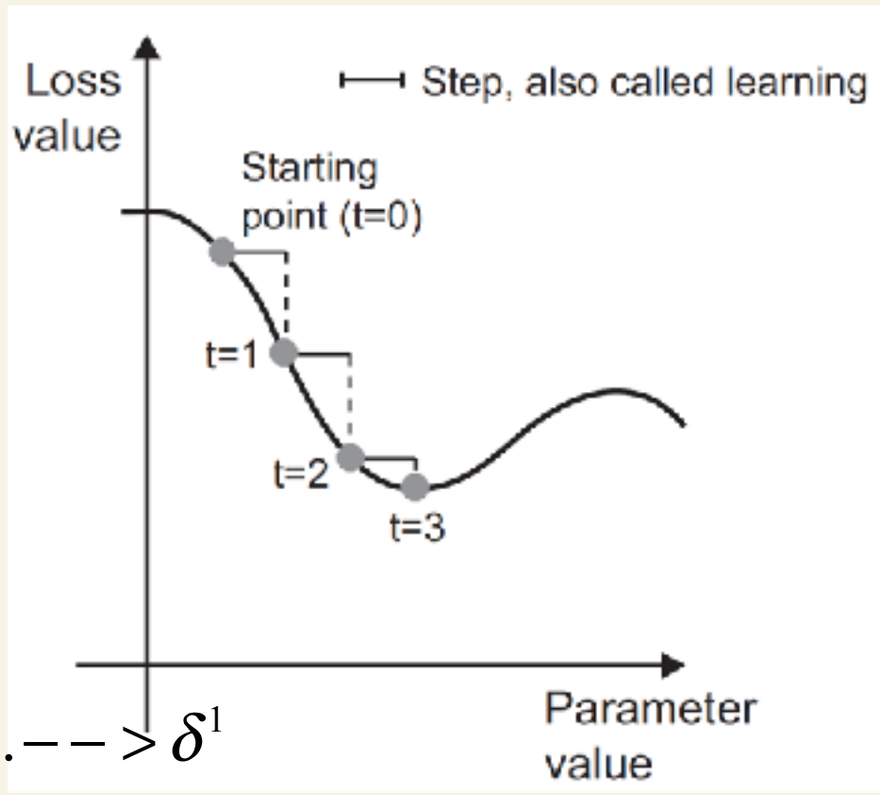
Let η be the learning rate(step size)

We choose $\Delta\vec{\theta}$ so as to make ΔL negative

Gradient Descent

$$\Delta\vec{\theta} = -\eta\vec{\nabla}_{\theta} L \Rightarrow \Delta L \approx -\eta\left\|\vec{\nabla}_{\theta} L\right\|^2 < 0$$

$$Back\,\mathrm{Pr}opagation: \quad \delta^L = \frac{\partial L}{\partial \theta^L} --> \delta^{L-1} --> \delta^{L-2} --> \ldots\ldots --> \delta^1$$



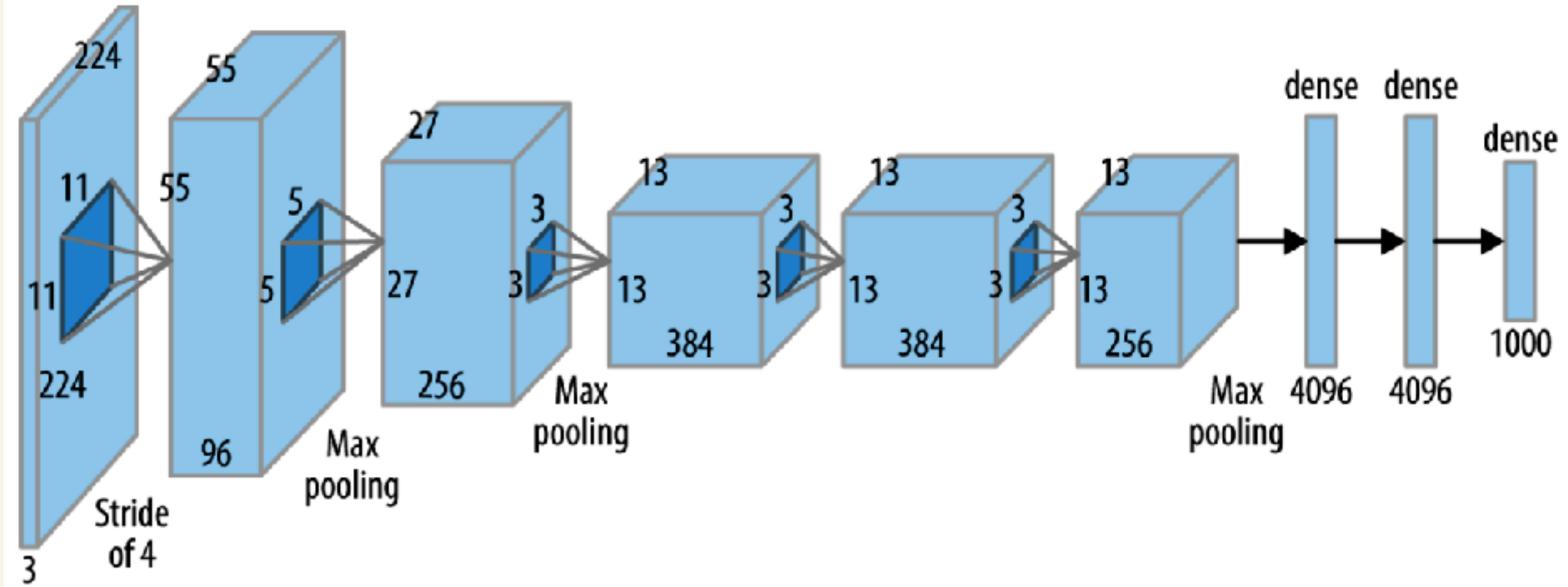
DEEP NN

- A shallow network (a few layers) can do the work but it might require an enormous number of units (neurons) per layer
- Deep Networks seem to suffer from the problem of vanishing gradient as you back propagate deeper and deeper
- Modern computational GPUs, larger training, regularization techniques (dropout units), pre-training..... overcome the vanishing gradient, so Deep Networks are very much back in fashion
- Modern deep learning is characterised by the composition of modular, differentiable components.
- Some argue that the most important innovation in deep learning applied to image processing is the CNN (Convolutional NN)

Convolutional NN

CNN

Convolutional NN (CNN)



- A Fully Connected dense layers NN learn global patterns, CNN learn local patterns

- CNN (Convolutional Neural Network - CONVNET) is made of layers that preserve the spatial characteristics of an image

- CNN is based on SHARED WEIGHTS, which reduces the dimensionality of the problem and introduces translation invariance, highly data efficient on perceptual problems

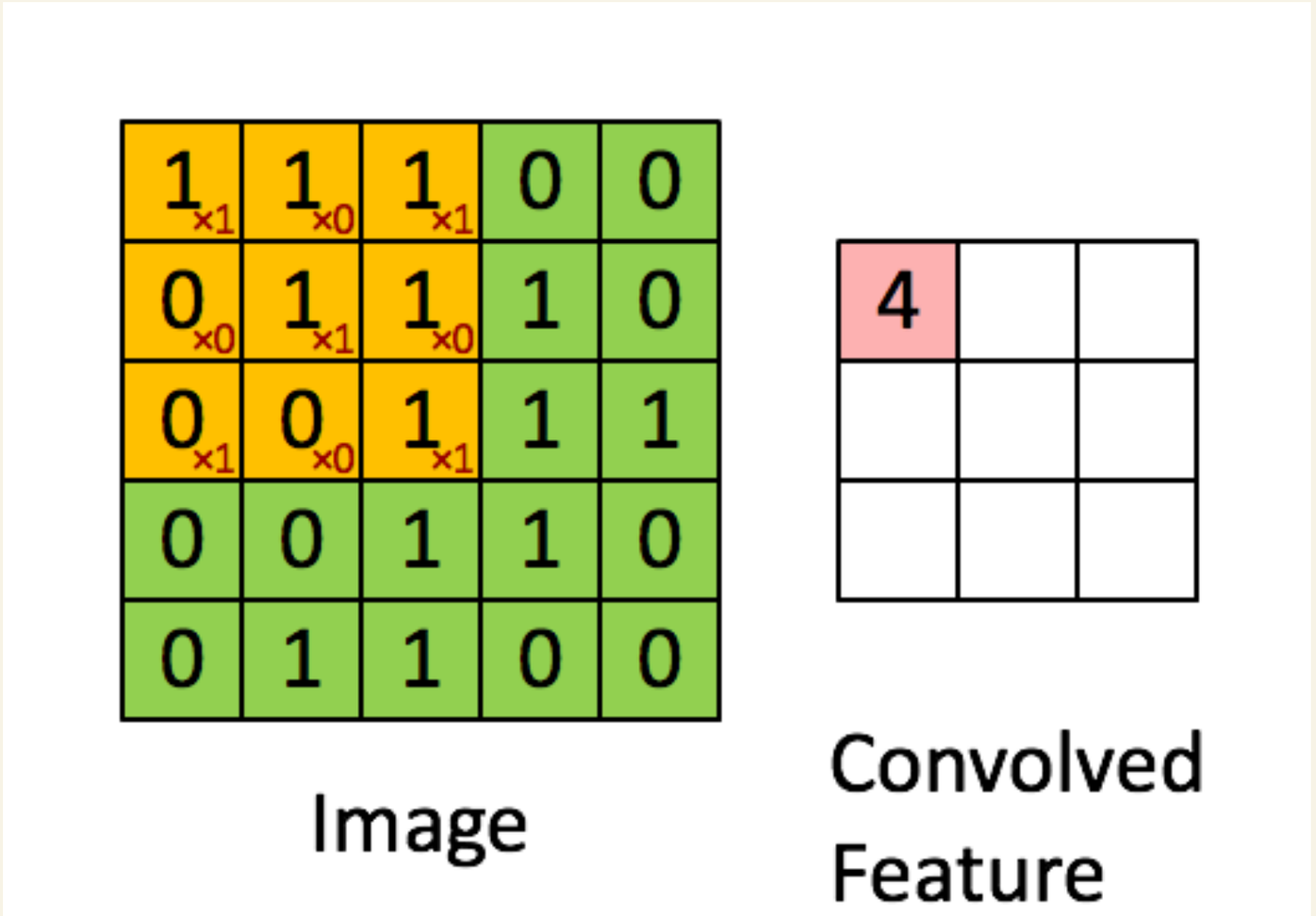
A Convolution

Image

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Kernel

1	0	1
0	1	0
1	0	1



Kernel is a feature detector of the input layer
Kernel is also called a **filter**

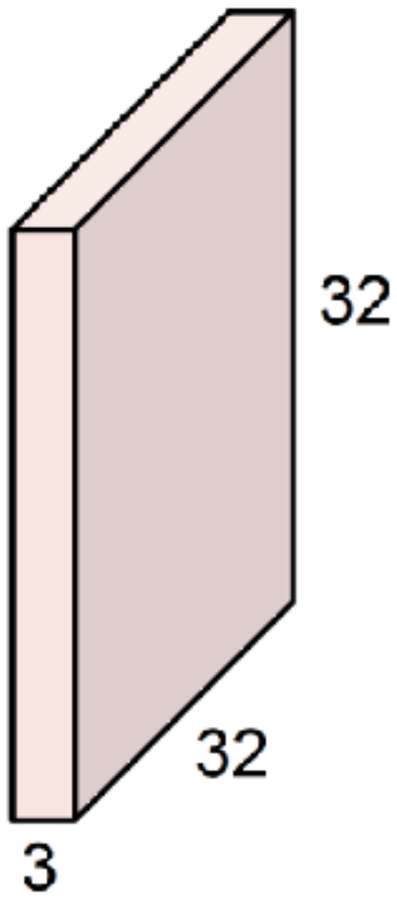
The convolved image is also called a **Feature Map**

CNN

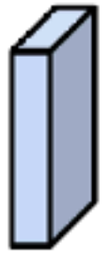
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

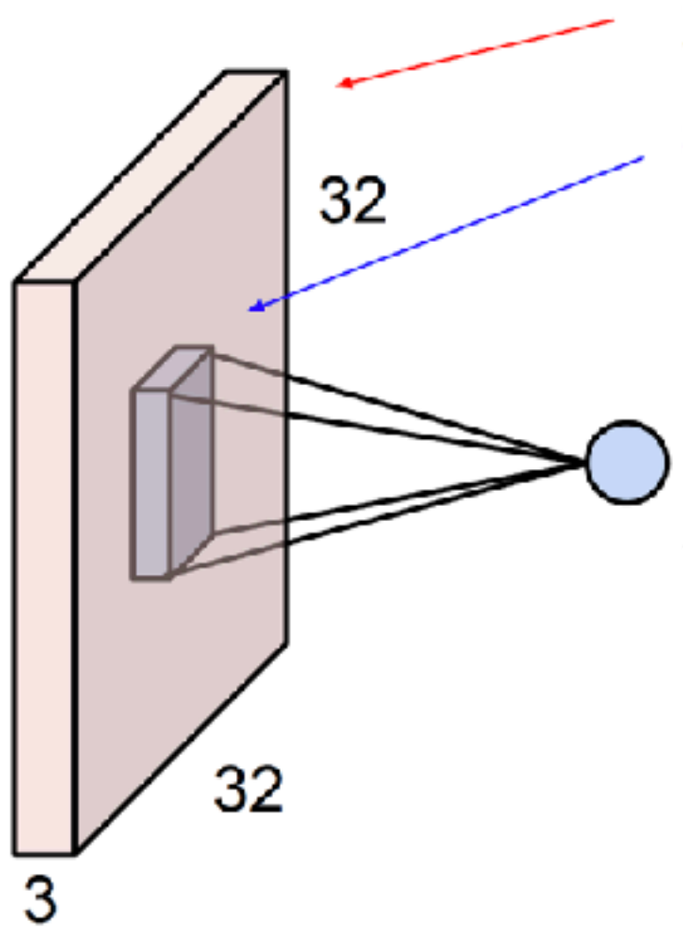
We can put as many filters as we like
The number of filters define the DEPTH of the resulting layer

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

CNN

- The weights are shared, and a feature map is represented by 5x5x3+1=76 parameters

Convolution Layer



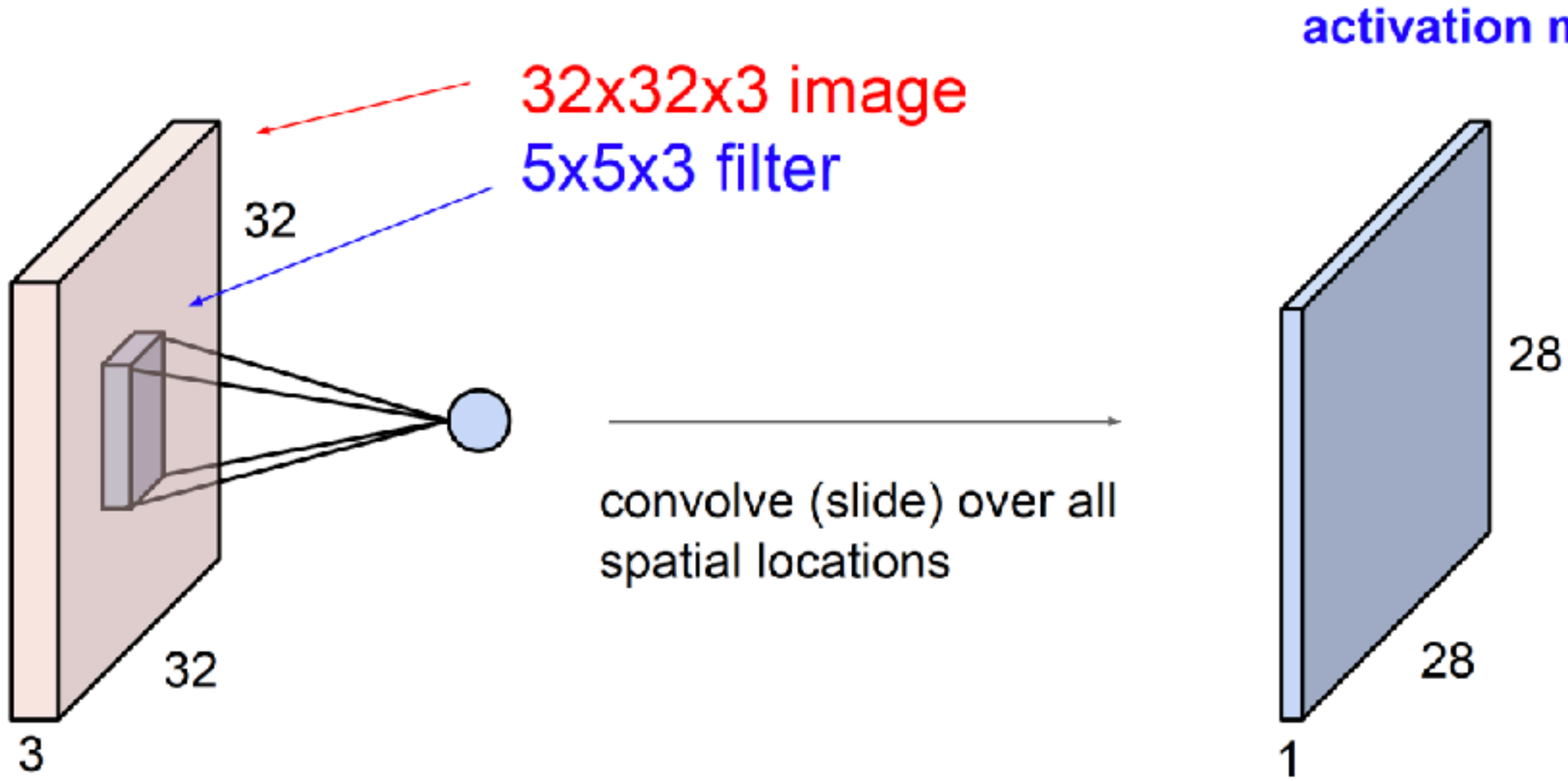
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

CNN

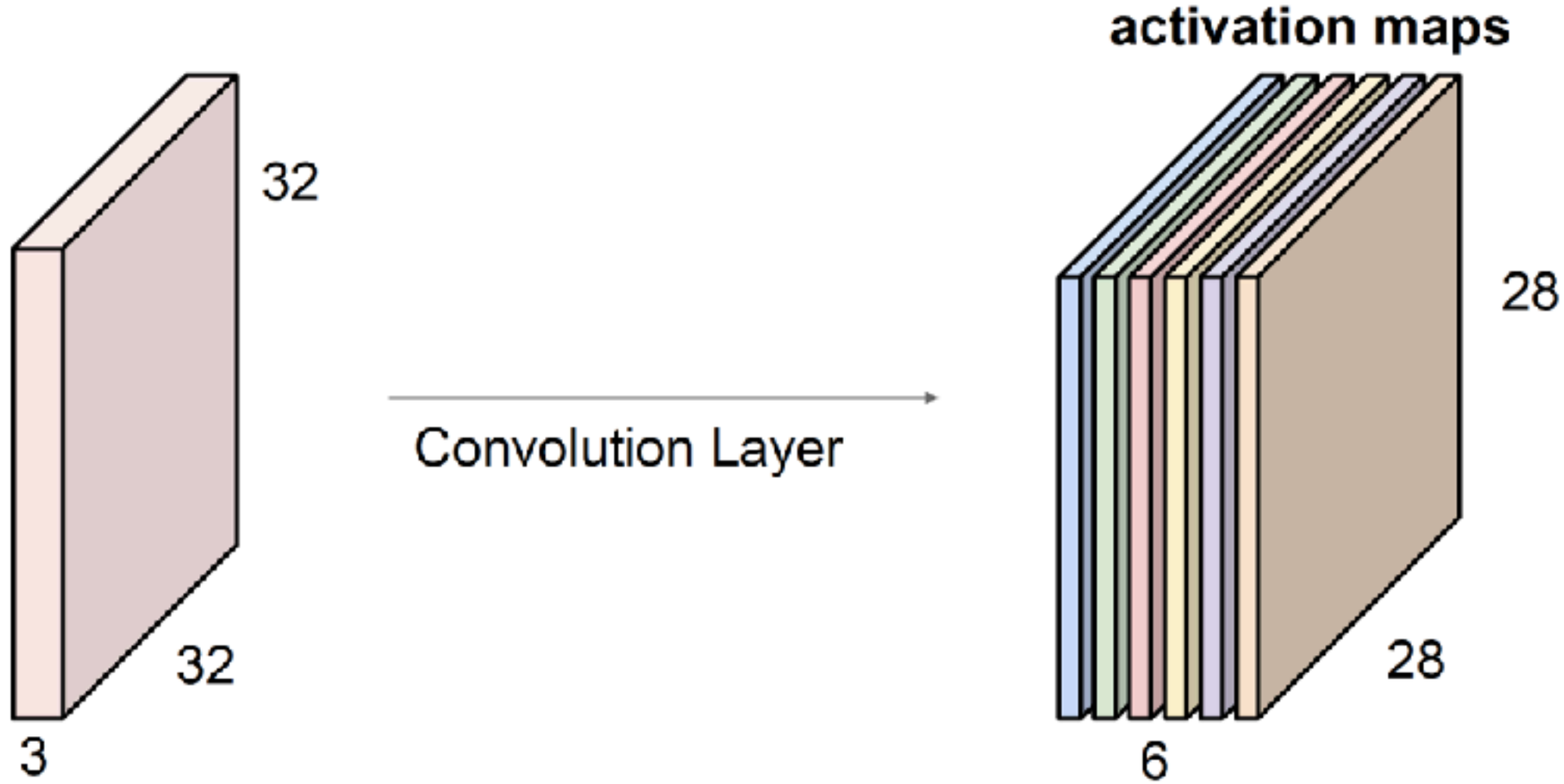
Convolution Layer



http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

CNN

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

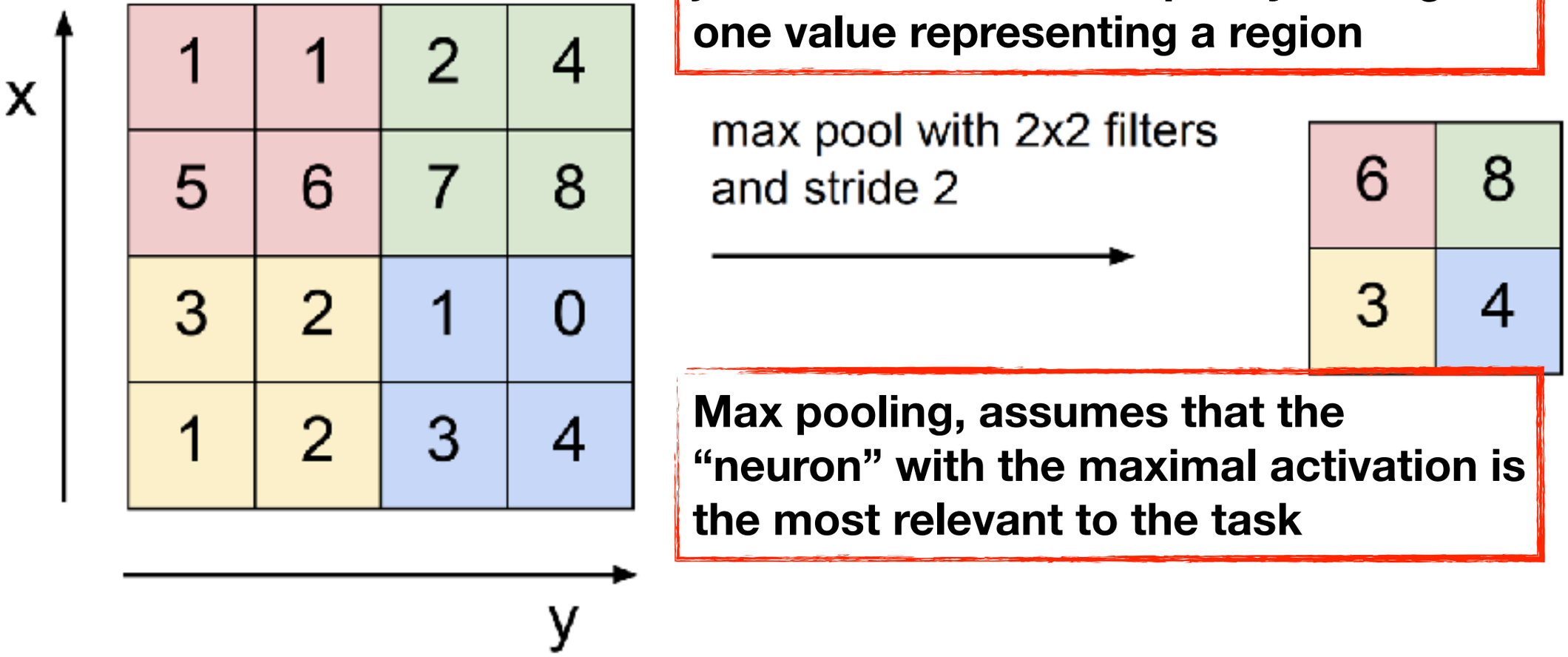


We stack these up to get a "new image" of size 28x28x6!

Pooling Layer

MAX POOLING

For pooling layer, it's typical to set up the stride, so there is no overlap. We just want to downsample by having one value representing a region



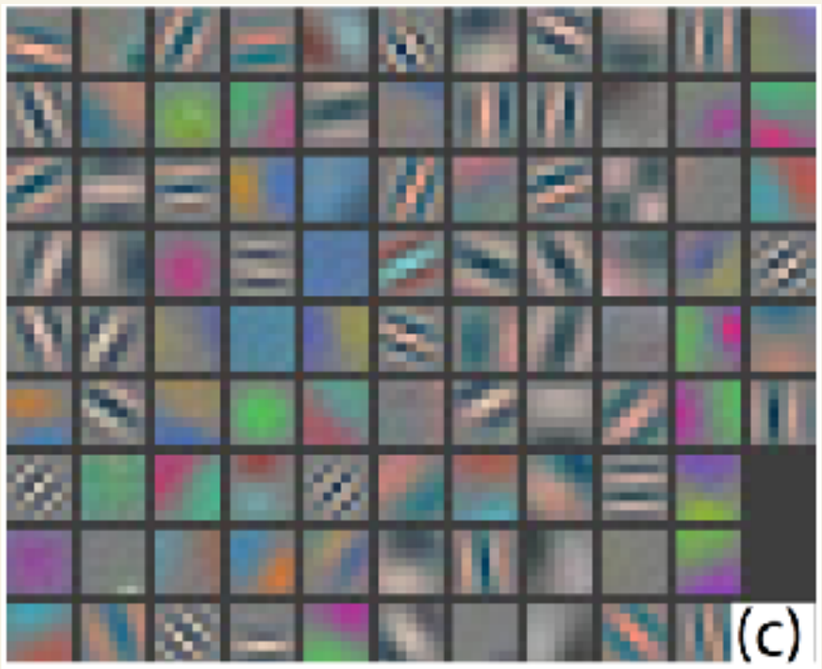
Max pooling, assumes that the "neuron" with the maximal activation is the most relevant to the task

Strides are also used to downsample

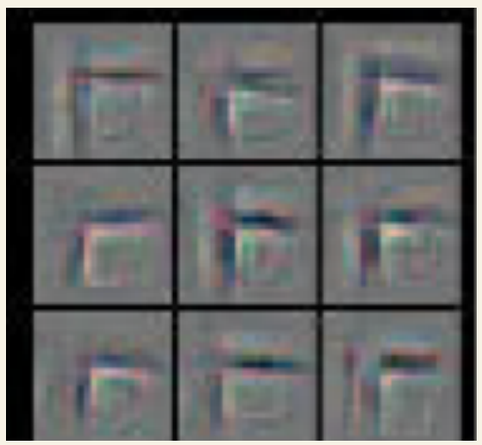
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

Visualizing Filters&Layers

- The layers can be interpreted as building up a hierarchical representation of the data. In natural images, for example, the first layers learn low-level features like edges and corners, the middle layers learn midlevel features like eyes, and the final layers learn high-level features like faces.



Layer 1 96 7x7 learning weights



Layer 1, corresponding Image patches

Deep Layer Dog's face

CNN Architectures

- Over the years Deep Learning evolved through competitions, and novel ideas improved the performance, from LeNet (1998), AlexNet (2012), VGGNet (2014), GoogLeNET (2014),ResNet (2015), ……..

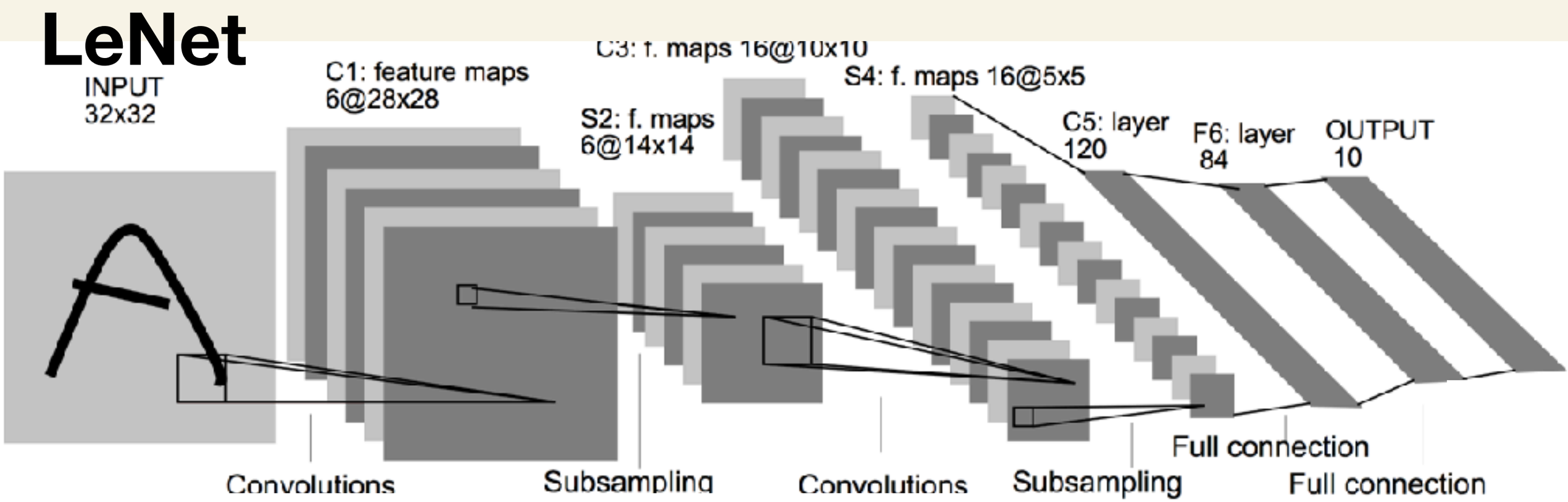
- Recently UNET and Graph NN are the state of the art….

CNN Architectures

- Over the years Deep Learning evolved through competitions, and novel ideas improved the performance, from LeNet (1998), AlexNet (2012), VGGNet (2014), GoogLeNET (2014),ResNet (2015), ……..

- Recently UNET and Graph NN are the state of the art….

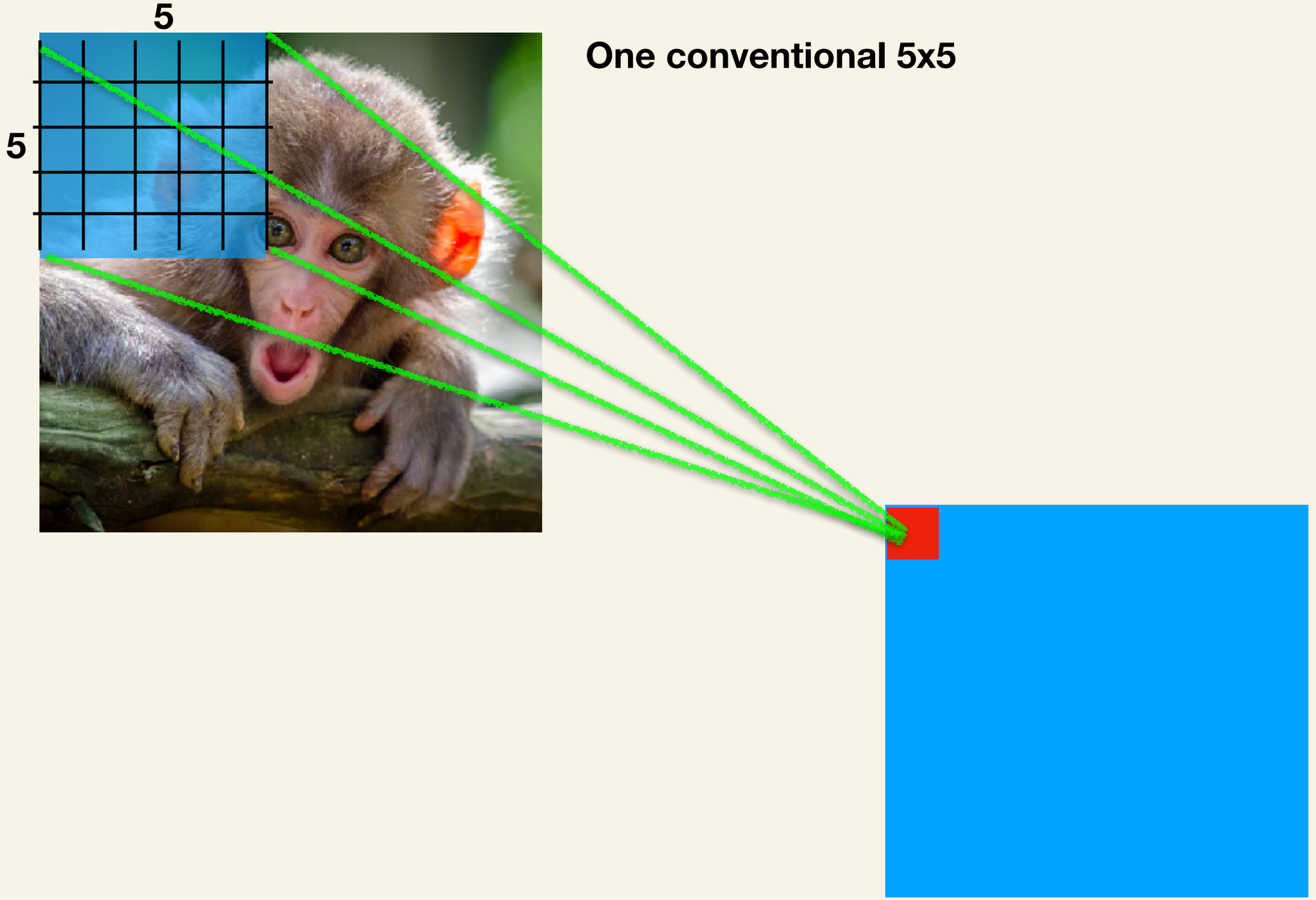
LeNet



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

Stacking Filters -VGGNet Approach

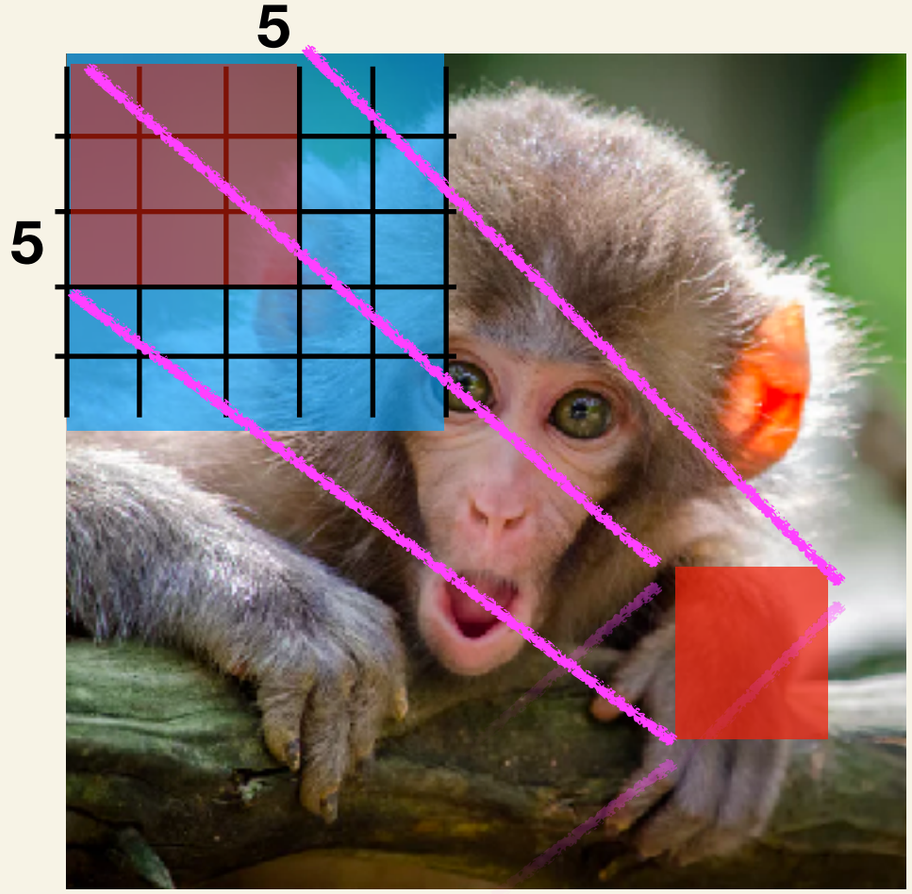
- Stacking 2 3X3 Filters has a receptive field of 1 conventional 5x5



One conventional 5x5

VGGNet Approach

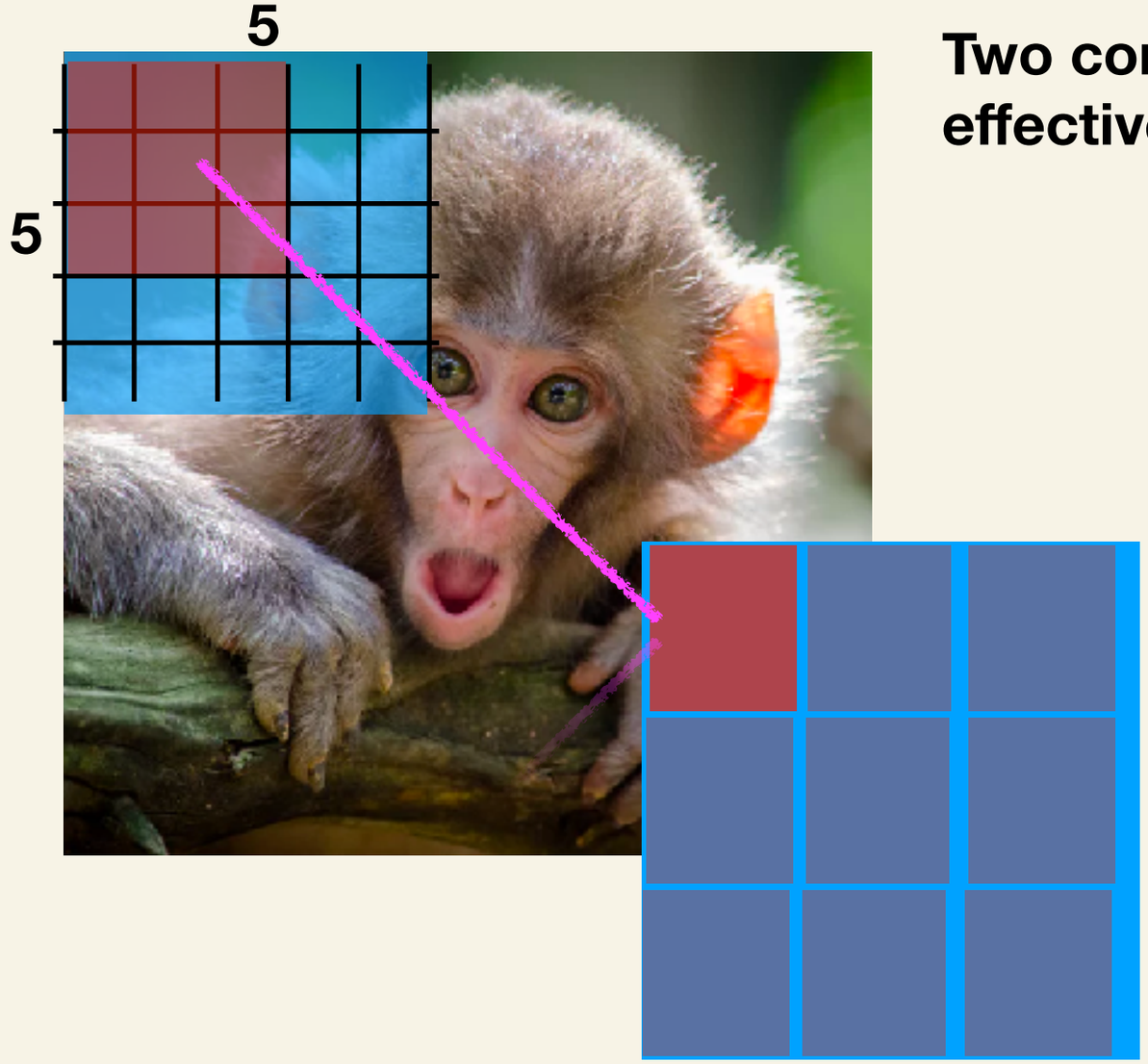
- Stacking 2 3X3 Filters has a receptive field of 1 conventional 5x5



One conventional 3x3 covers an effective receptive field of 3x3

VGGNet Approach

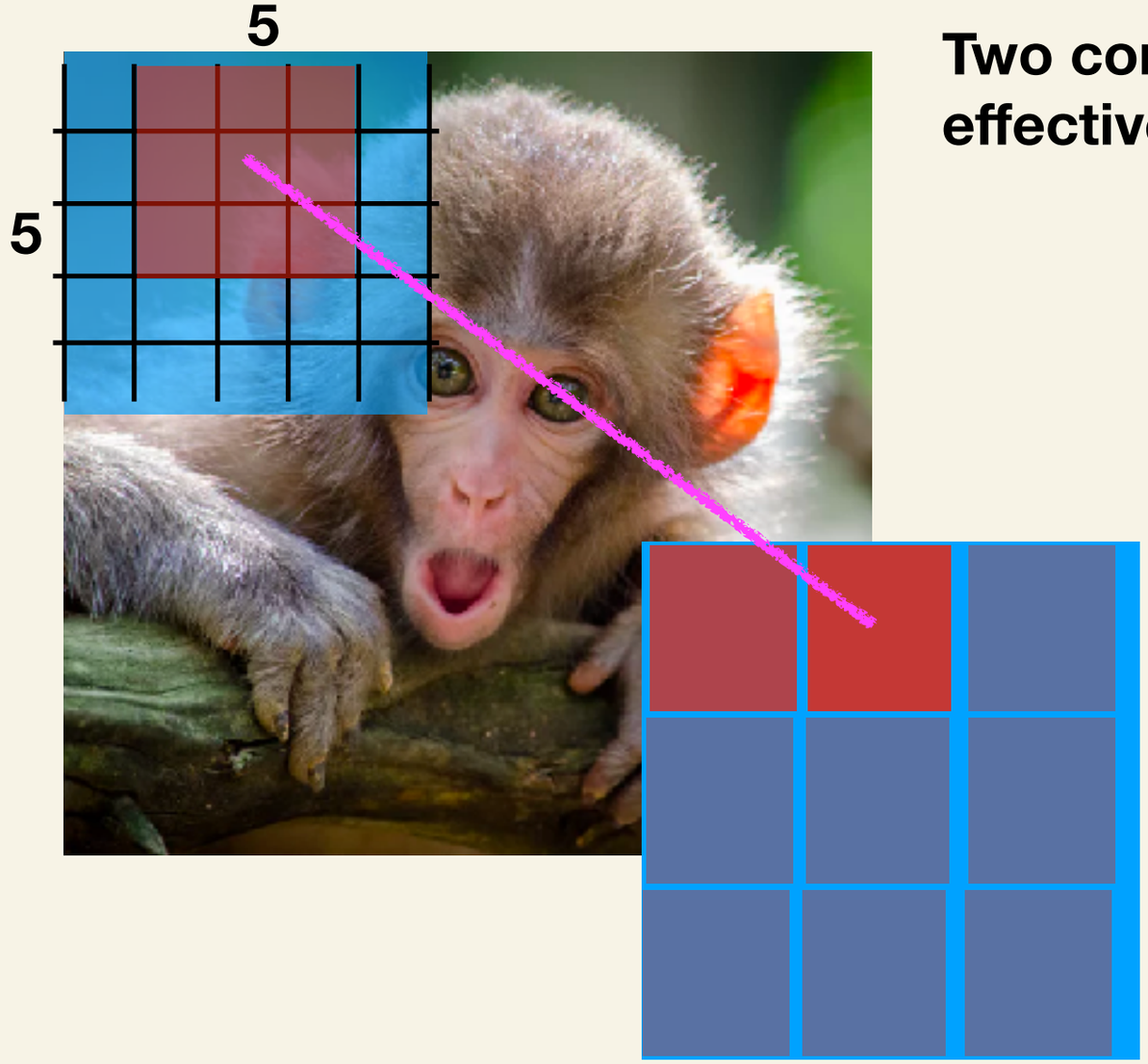
- Stacking 2 3X3 Filters has a receptive field of 1 conventional 5x5



Two conventional 3x3 (stride 1) cover an effective receptive field of 5X5

VGGNet Approach

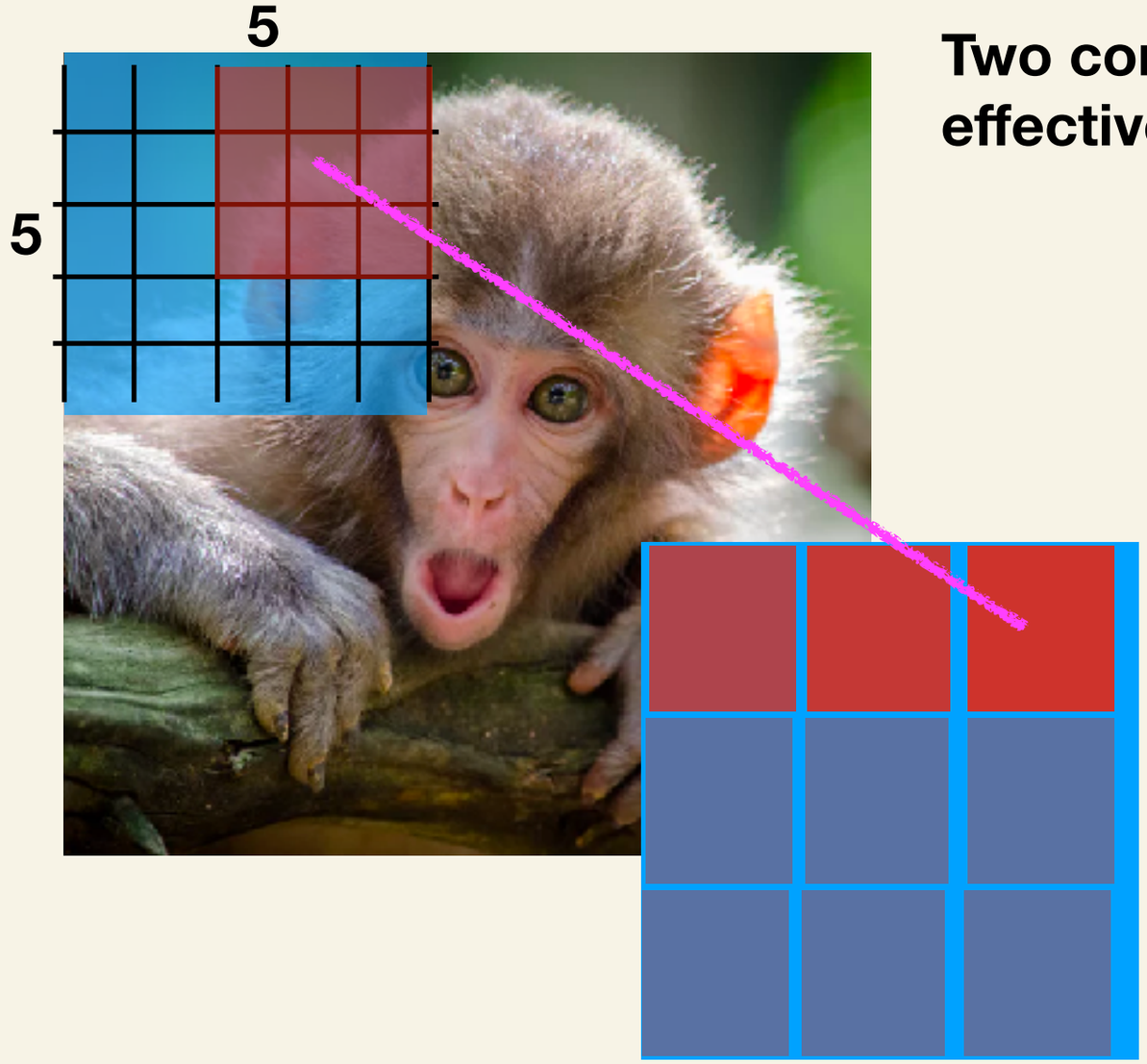
- Stacking 2 3X3 Filters has a receptive field of 1 conventional 5x5



Two conventional 3x3 (stride 1) cover an effective receptive field of 5X5

VGGNet Approach

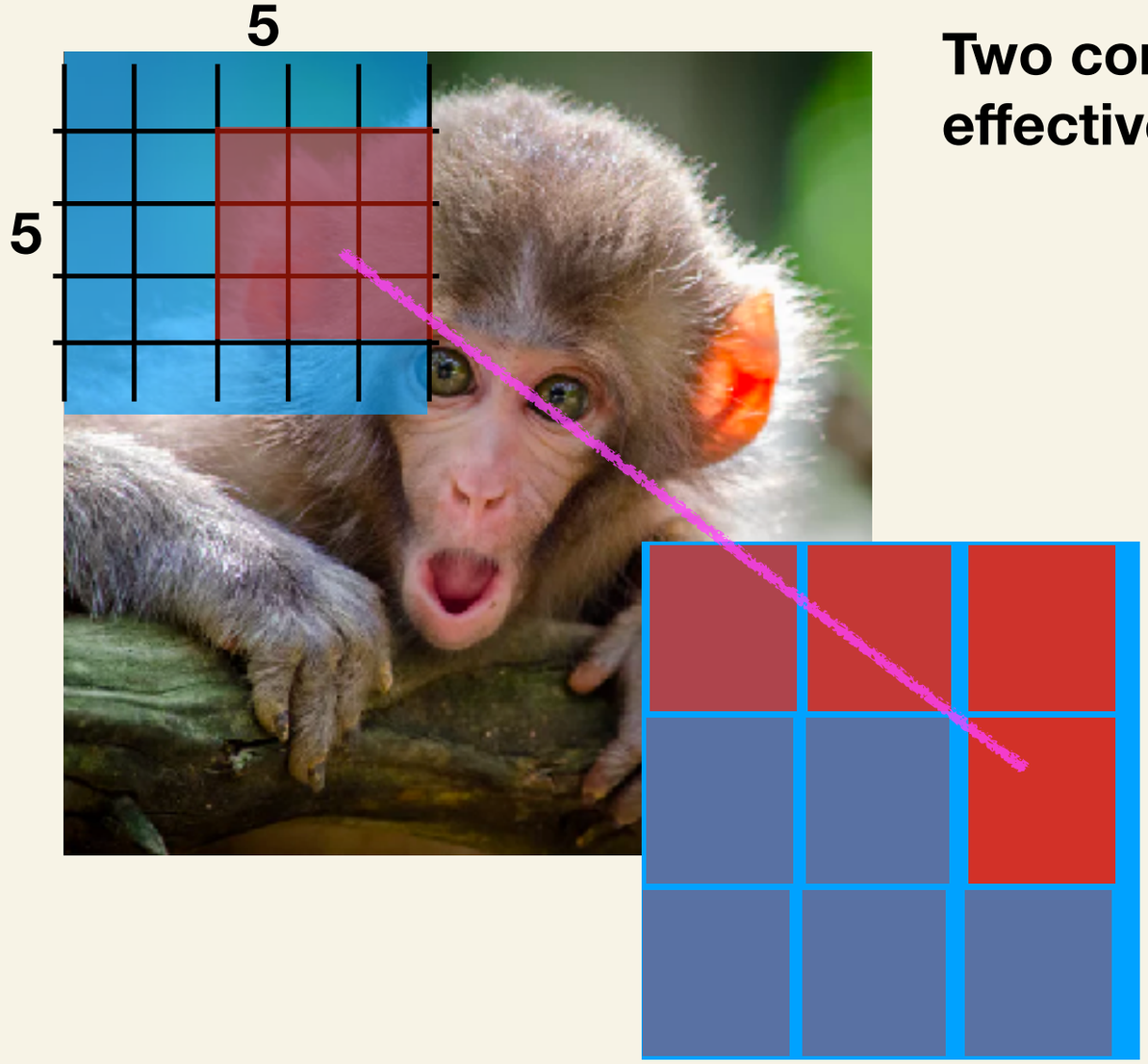
- Stacking 2 3X3 Filters has a receptive field of 1 conventional 5x5



Two conventional 3x3 (stride 1) cover an effective receptive field of 5X5

VGGNet Approach

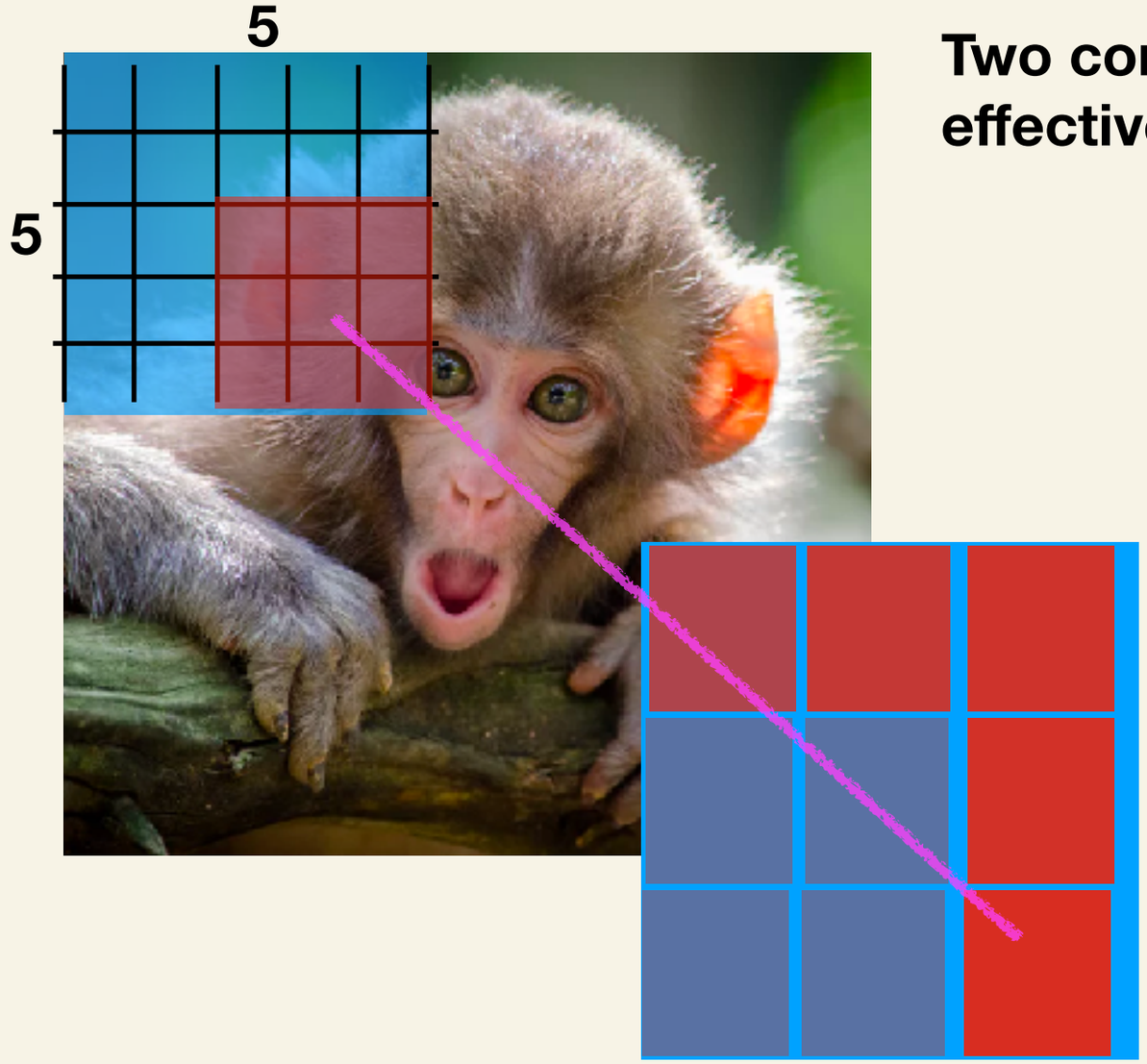
- Stacking 2 3X3 Filters has a receptive field of 1 conventional 5x5



Two conventional 3x3 (stride 1) cover an effective receptive field of 5X5

VGGNet Approach

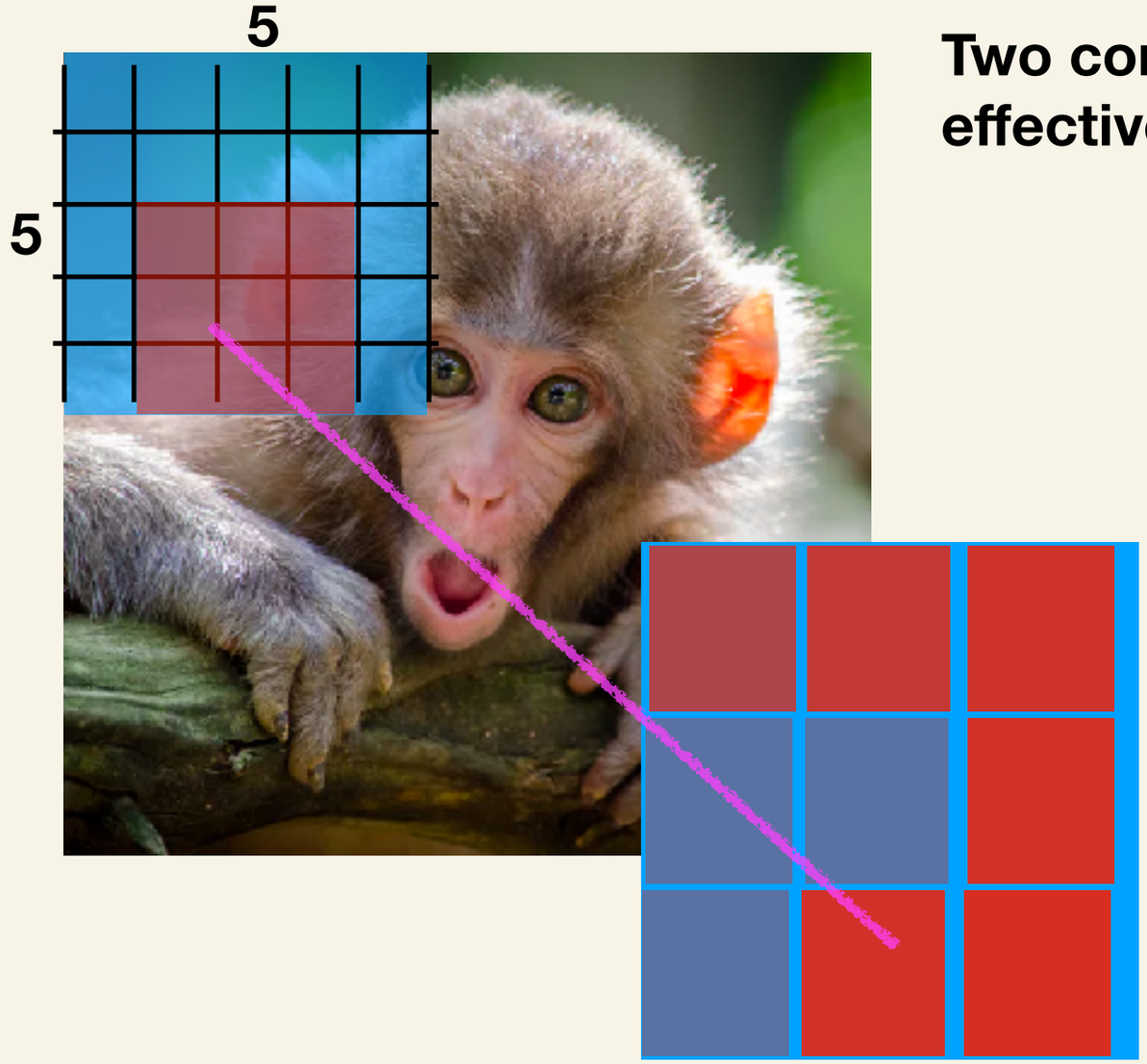
- Stacking 2 3X3 Filters has a receptive field of 1 conventional 5x5



Two conventional 3x3 (stride 1) cover an effective receptive field of 5X5

VGGNet Approach

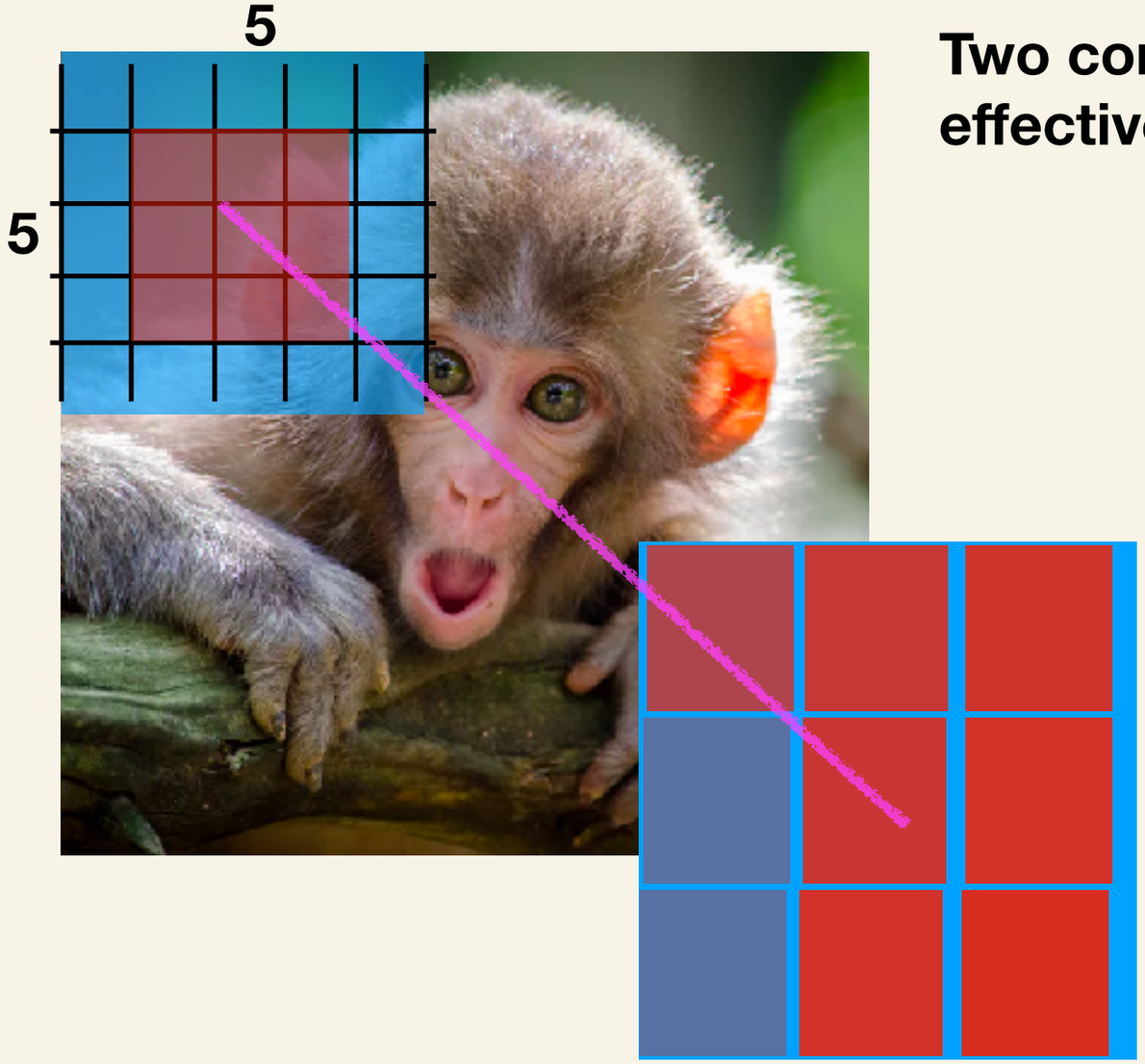
- Stacking 2 3X3 Filters has a receptive field of 1 conventional 5x5



Two conventional 3x3 (stride 1) cover an effective receptive field of 5X5

VGGNet Approach

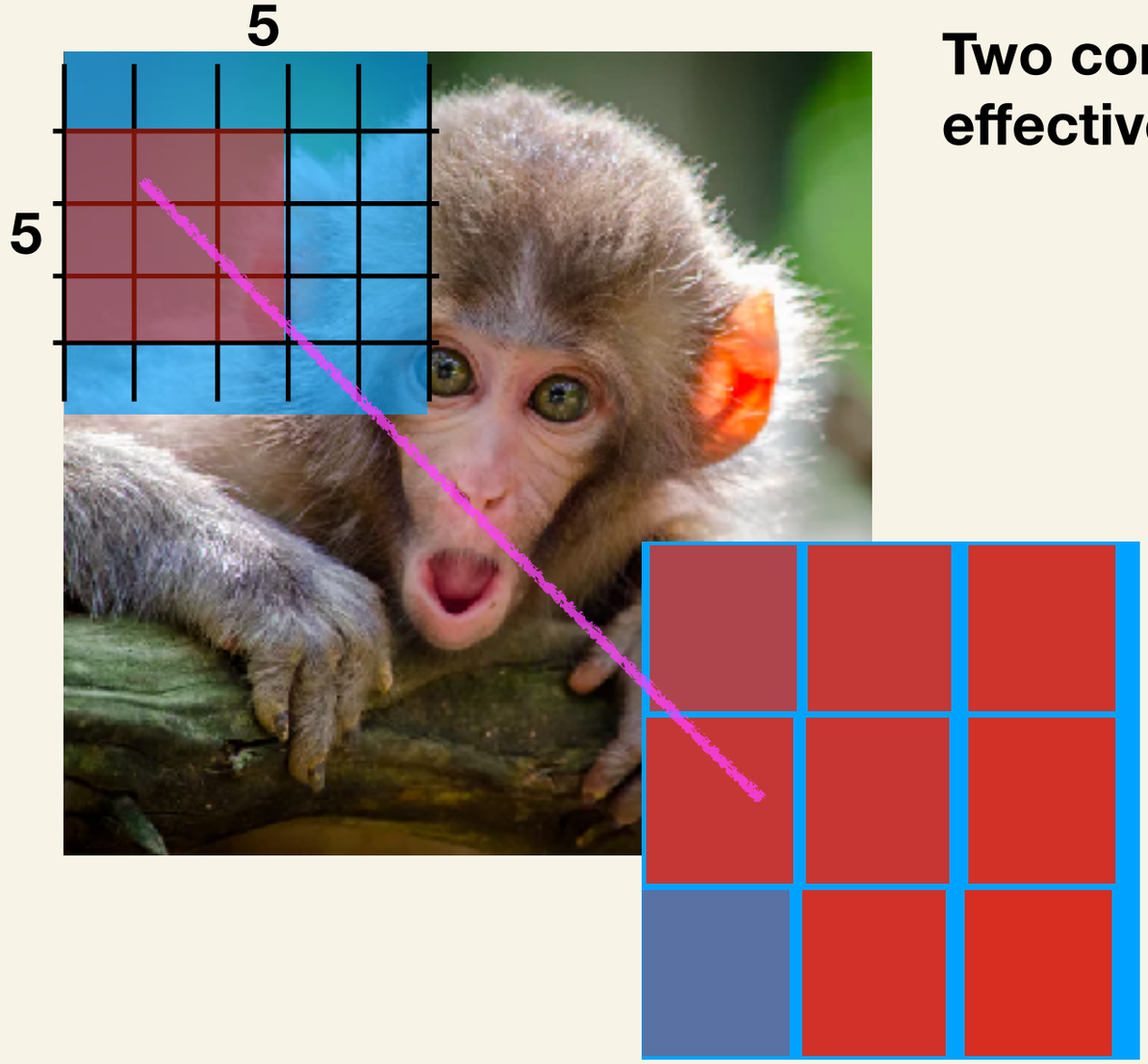
- Stacking 2 3X3 Filters has a receptive field of 1 conventional 5x5



Two conventional 3x3 (stride 1) cover an effective receptive field of 5X5

VGGNet Approach

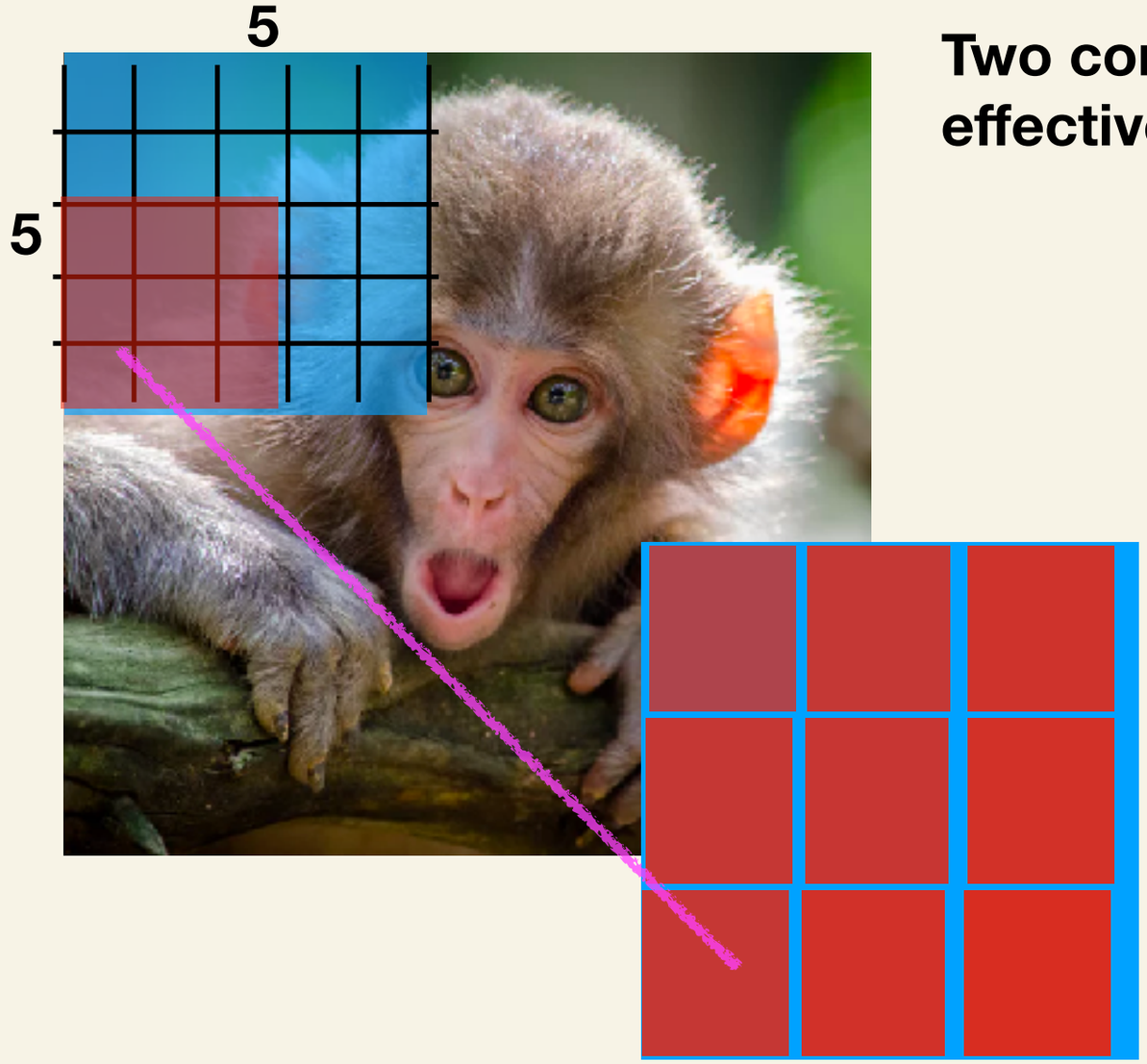
- Stacking 2 3X3 Filters has a receptive field of 1 conventional 5x5



Two conventional 3x3 (stride 1) cover an effective receptive field of 5X5

VGGNet Approach

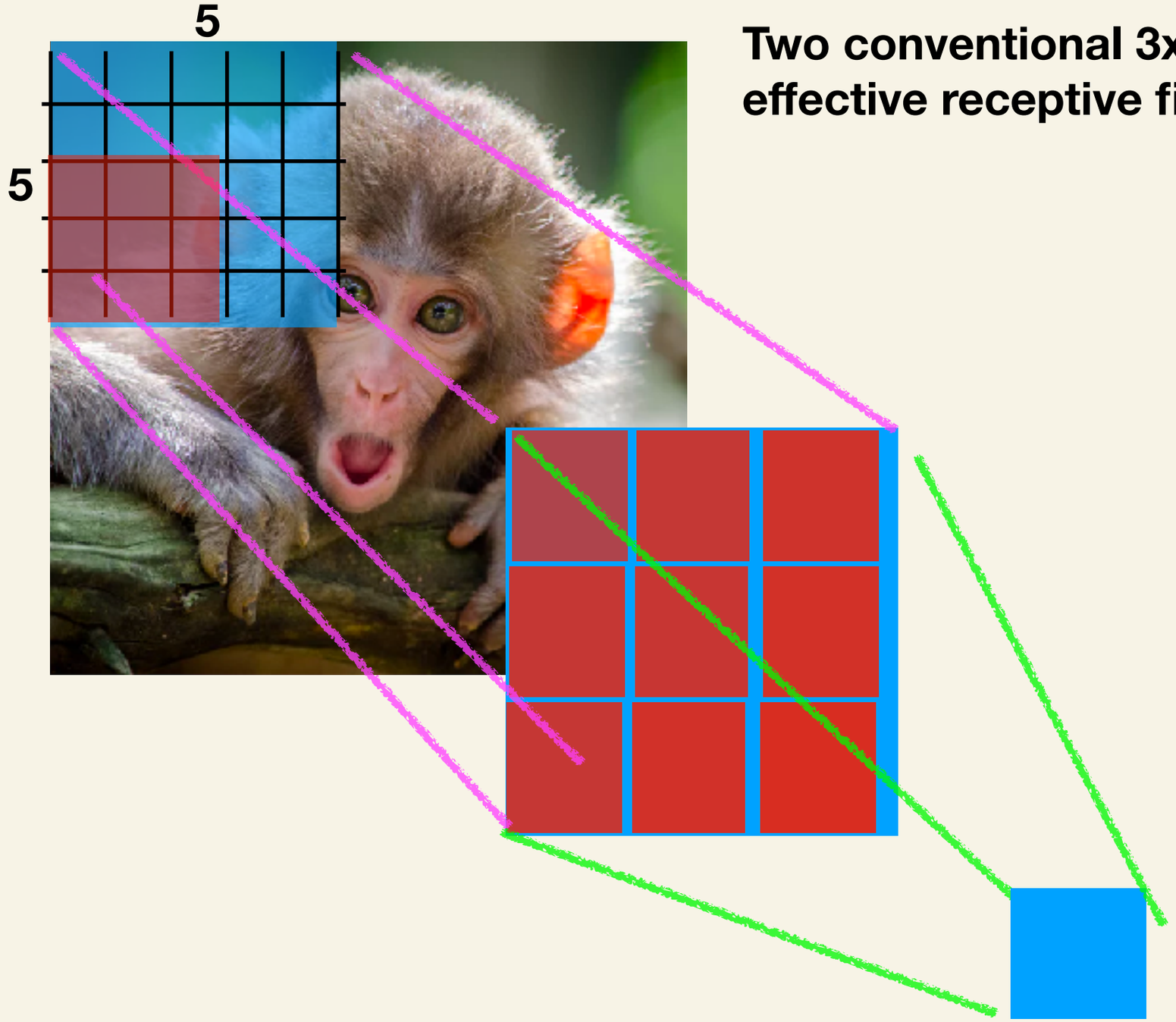
- Stacking 2 3X3 Filters has a receptive field of 1 conventional 5x5



Two conventional 3x3 (stride 1) cover an effective receptive field of 5X5

VGGNet Approach

- Stacking 2 3X3 Filters has a receptive field of 1 conventional 5x5

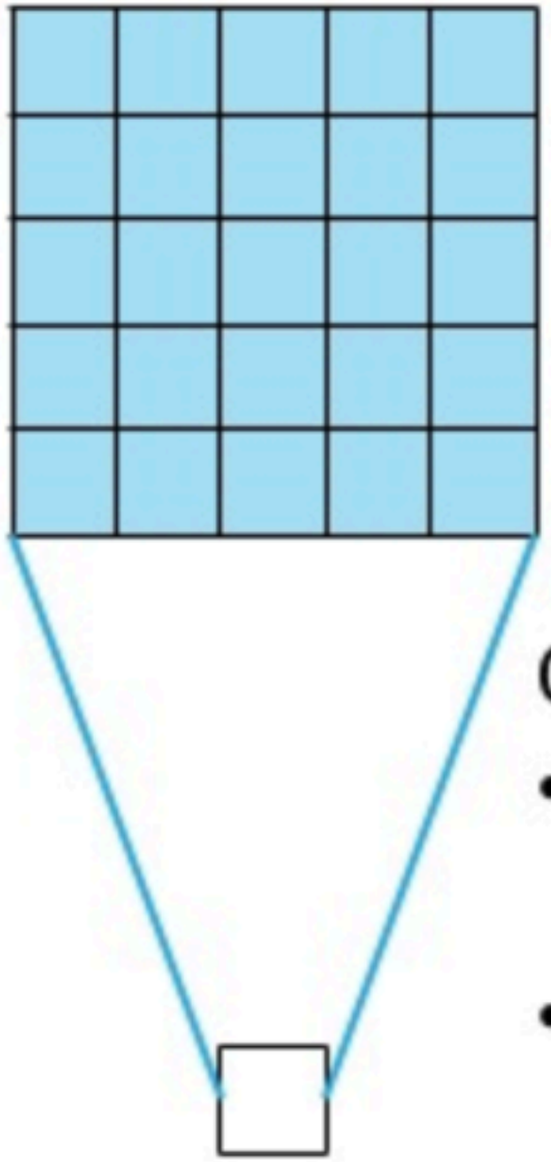


Two conventional 3x3 (stride 1) cover an effective receptive field of 5X5

VGGNet Approach

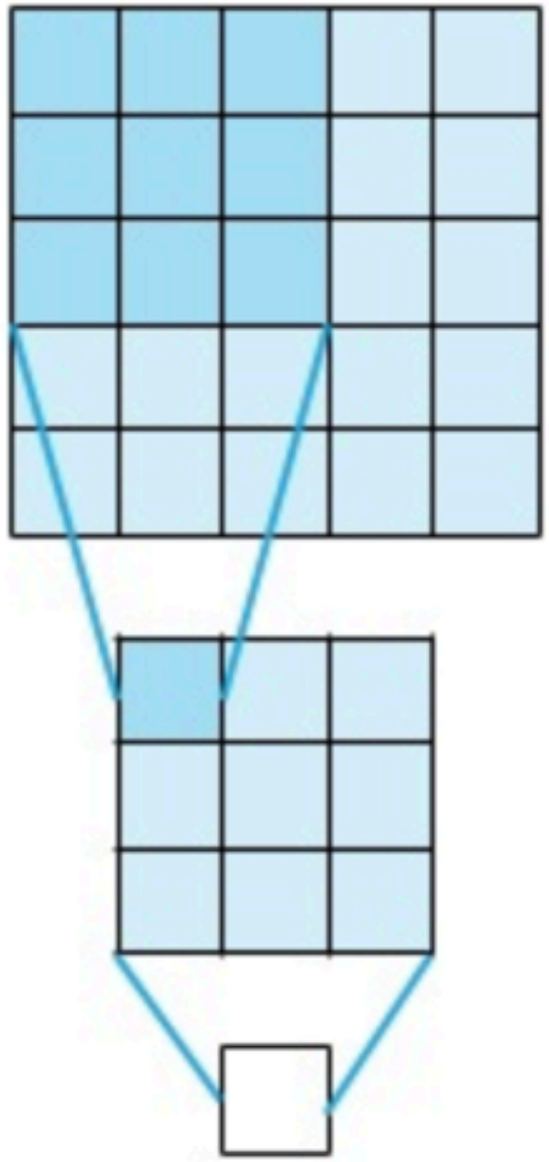
- More layers and smaller filters is better

- More non-linearity, fewer parameters



One 5x5 filter

- Parameters:
  5x5 = 25
- Non-linear:1



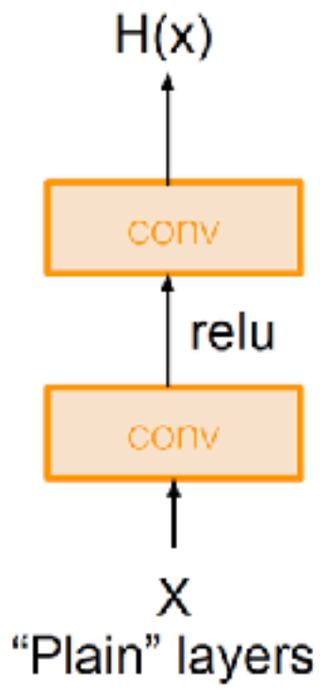
Two 3x3 filters

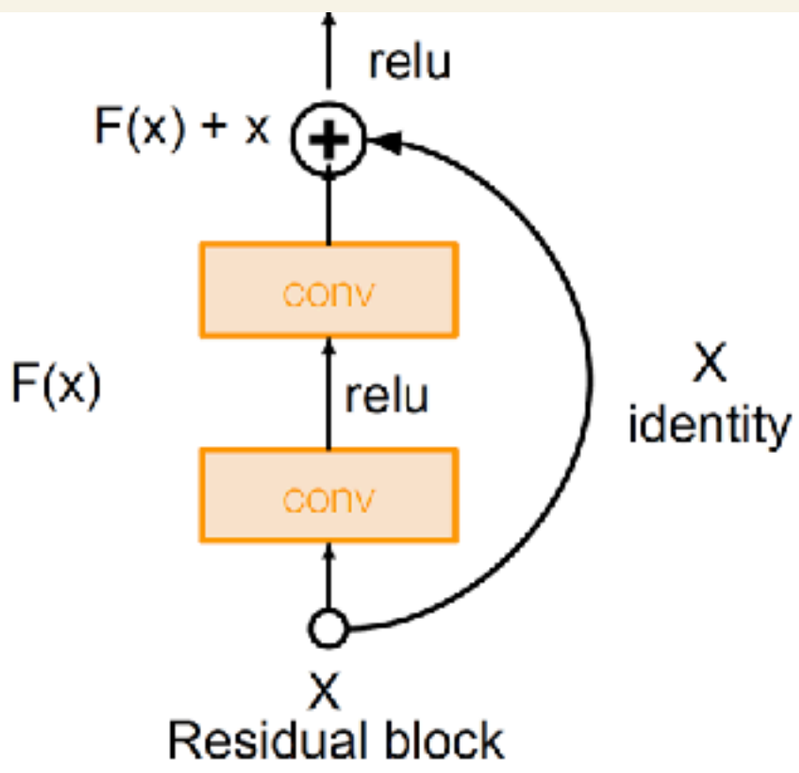
- Parameters:
  3x3x2 = 18
- Non-linear:2

Introducing SKIP connections-ResNet Module

- When stating many layers we might run into optimisation problems

- The idea is to try and help the extended model to learn, so instead of stacking more and more layers, and have them studying some underlying mapping of the desired function, try to fit only the residual mapping!

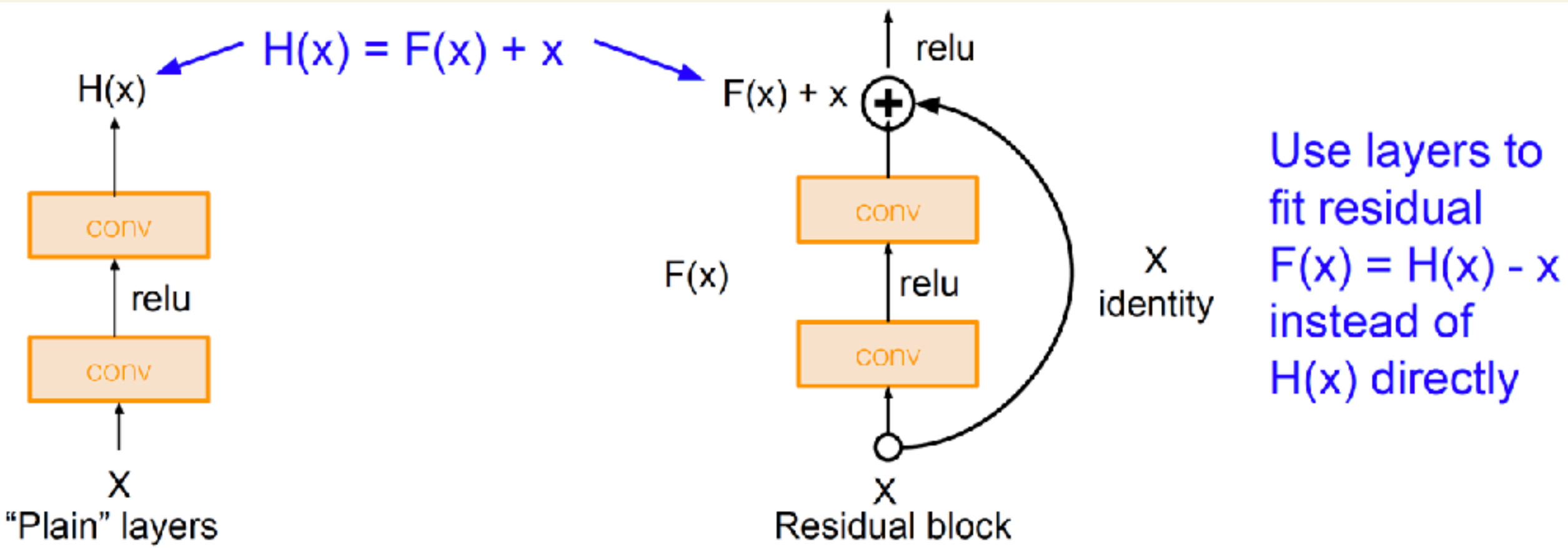




ResNet

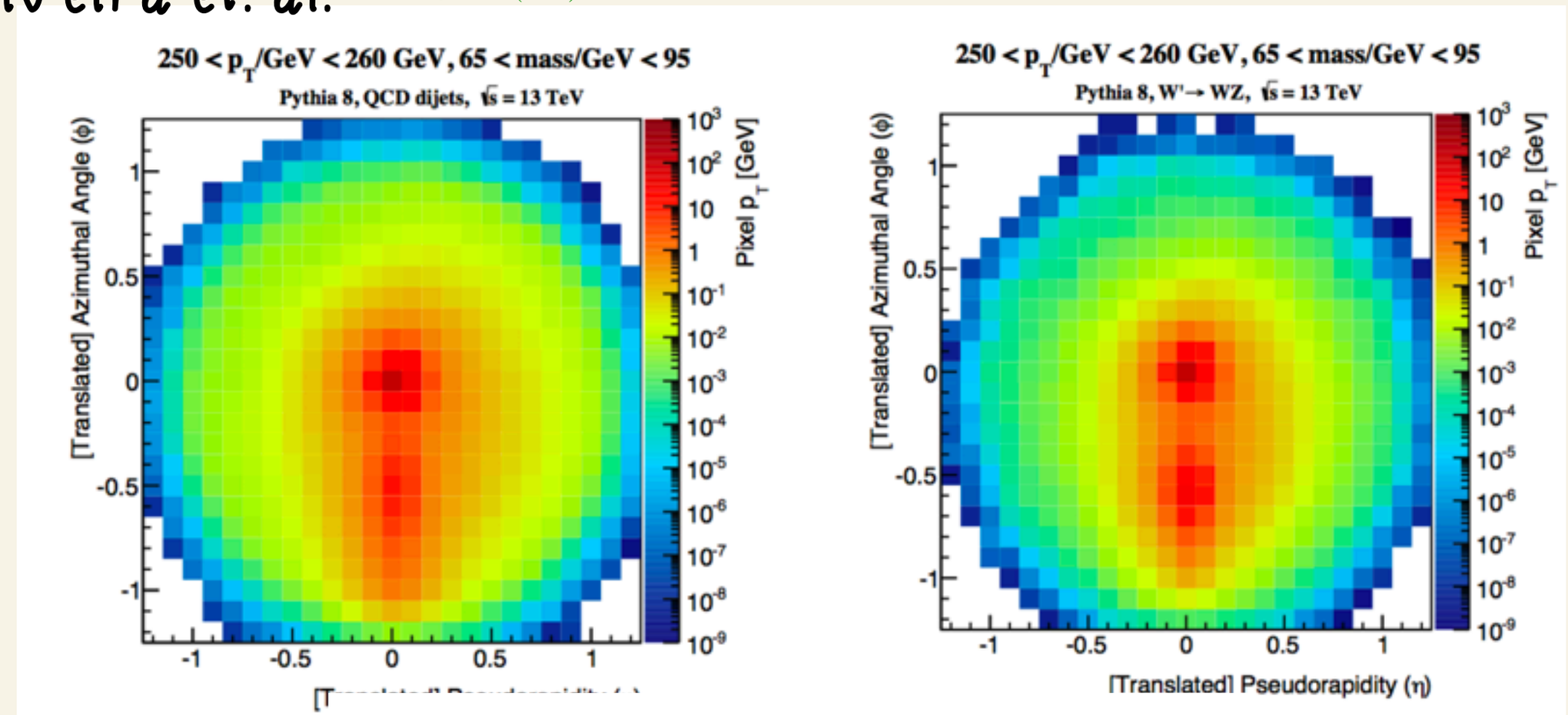
- Instead of learning H(x), help H(x) by defining F(x)=H(x)-x

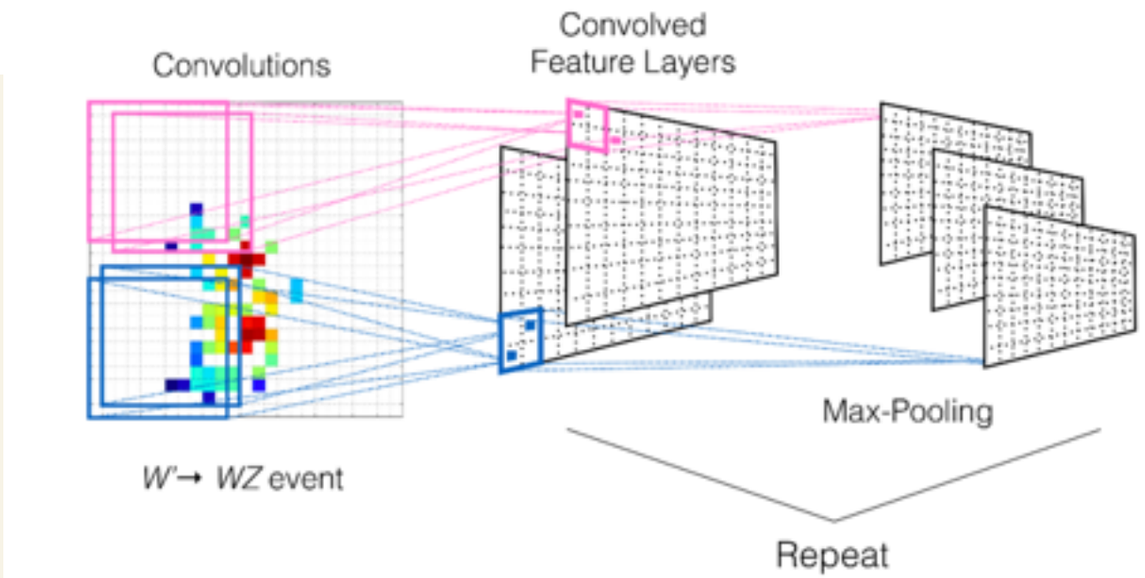
- F(x) is the residual, which is just a residual addition to what we have already learnt



Application to Jet Images

- Oliveira et. al. JHEP07(2016)069





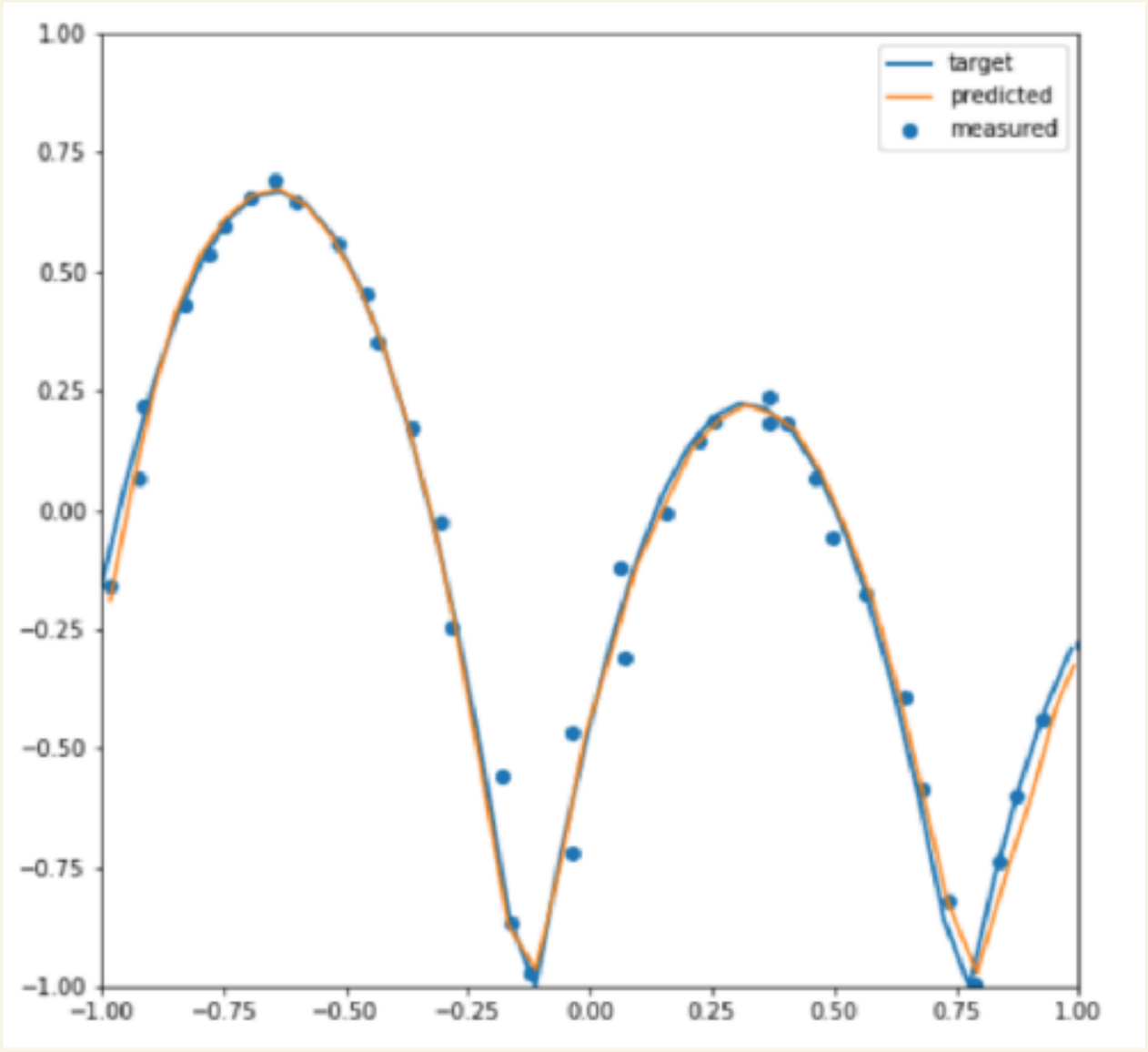
[Dropout→Conv→ReLU→MaxPool] * 3→LRN→[Dropout→FC→ReLU]→Dropout→Sigmoid

Recurrent NN

RNN

Recurrent NN for Sequenced DATA

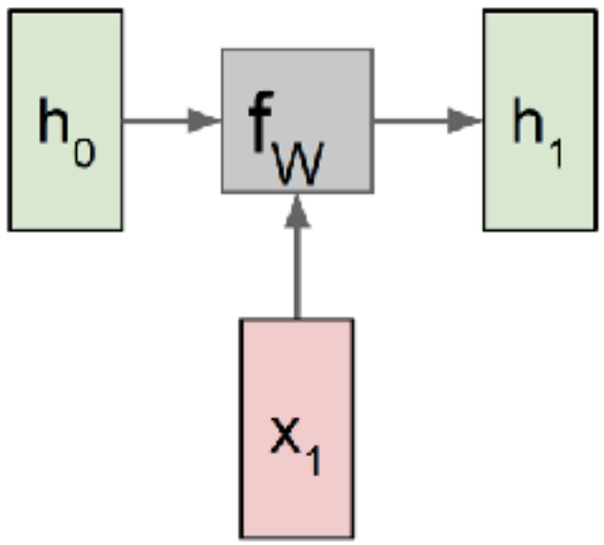
- Predicting the next word in a sentence, or the next coordinates of a Ball



Recurrent Neural Networks

- RNN Computational Graph

- Read input vector x1, update hidden state and then move it forward (at this stage might produce an output or not)

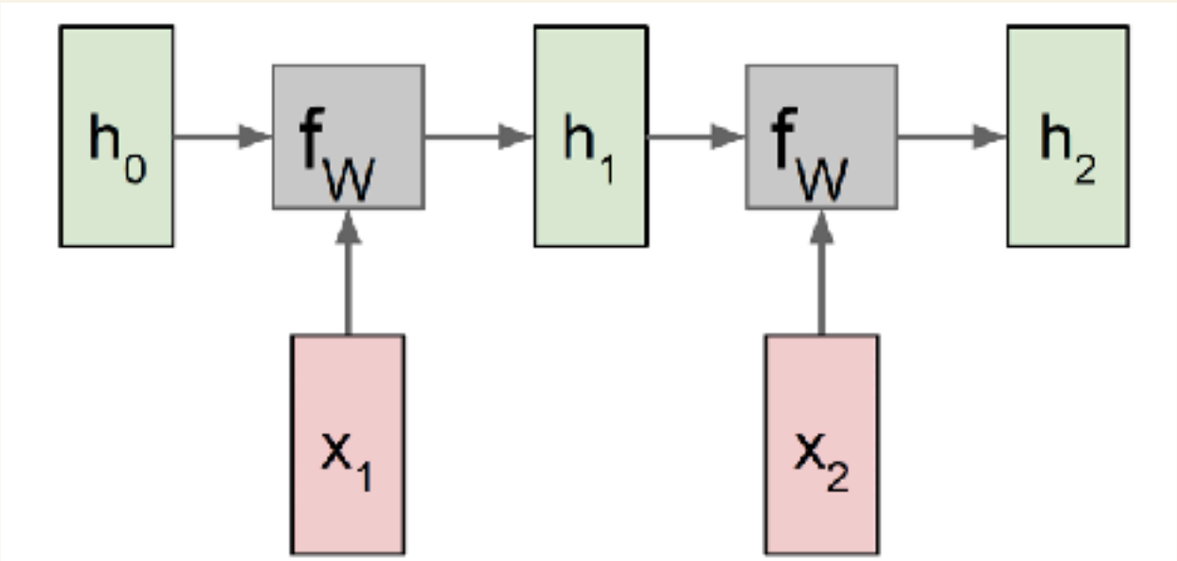


$$h_1 = f_W(h_0, x_1)$$

Recurrent Neural Networks

- RNN Computational Graph

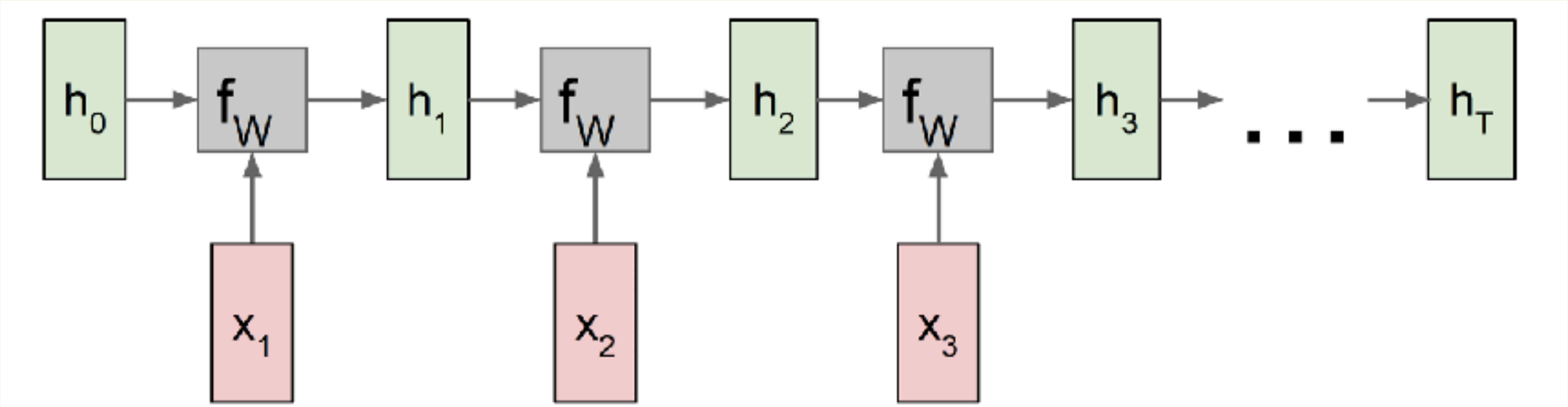
- Read input vector x_2, update hidden state and then move it forward (at this stage might produce an output or not)



$$h_2 = f_W(h_1, x_2) = f_W\left(f_W(h_0, x_1), x_2\right)$$

Recurrent Neural Networks

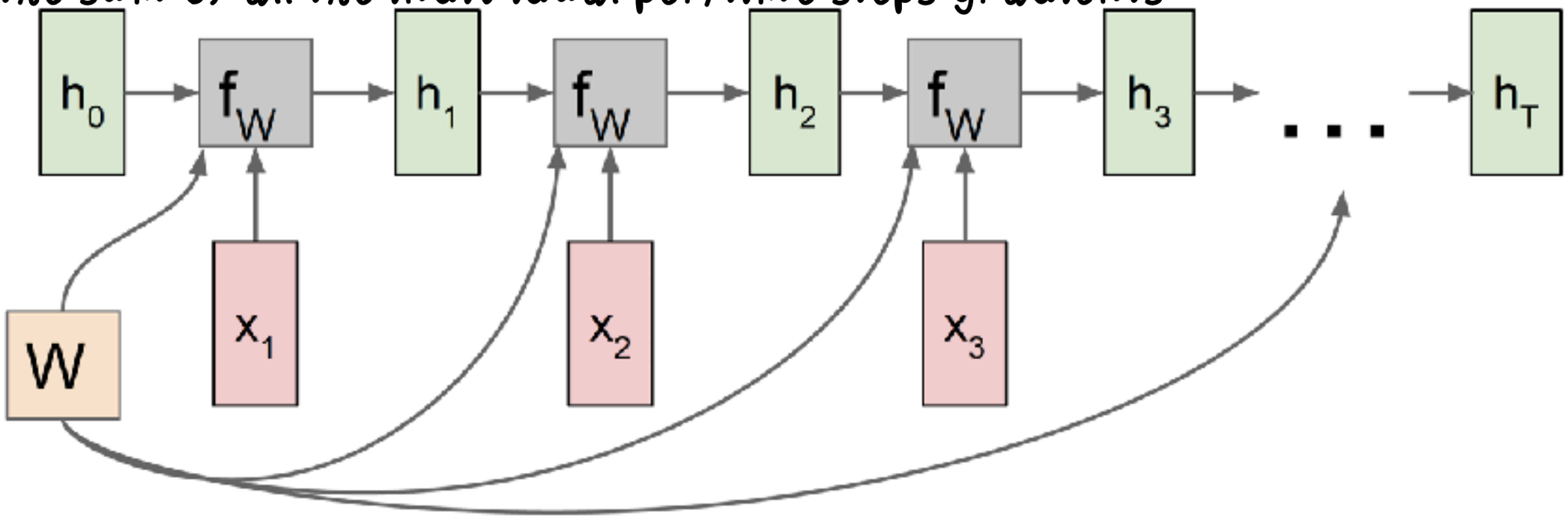
- Read input vector x_t, update hidden state and then move it forward (at this stage might produce an output or not)



$$h_n = f_W(h_{n-1}, x_n) = f_W\left(f_W(h_{n-2}, x_{n-1}), x_n\right) = f_W\left(f_W\left(f_W(h_{n-3,}, x_{n-2}), x_{n-1}\right), x_n\right)$$

Recurrent Neural Networks

- In every step time t there is a unique input x , a unique memory h, but the same W
- We repeat with the same weights matrix till we finish all our input sequence, all of the x_t vectors
- In the back propagation we go back through all the time steps, the final gradient is the sum of all the individual per/time steps gradients

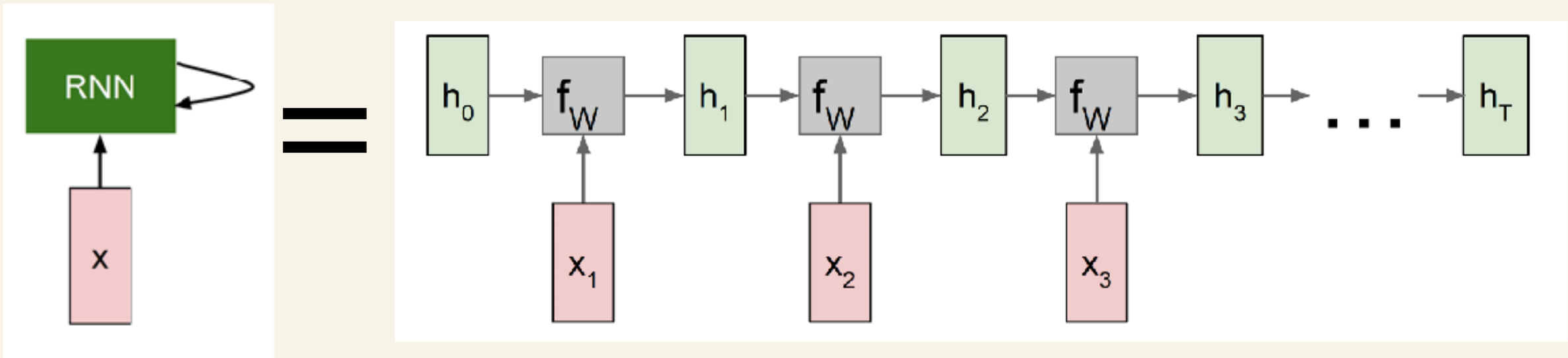


$$h_n = f_W(h_{n-1}, x_n) = f_W\left(f_W(h_{n-2}, x_{n-1}), x_n\right) = f_W\left(f_W\left(f_W(h_{n-3,}, x_{n-2}), x_{n-1}\right), x_n\right)$$

RNN folded

- Take an input x, feed to the RNN which has some internal hidden state

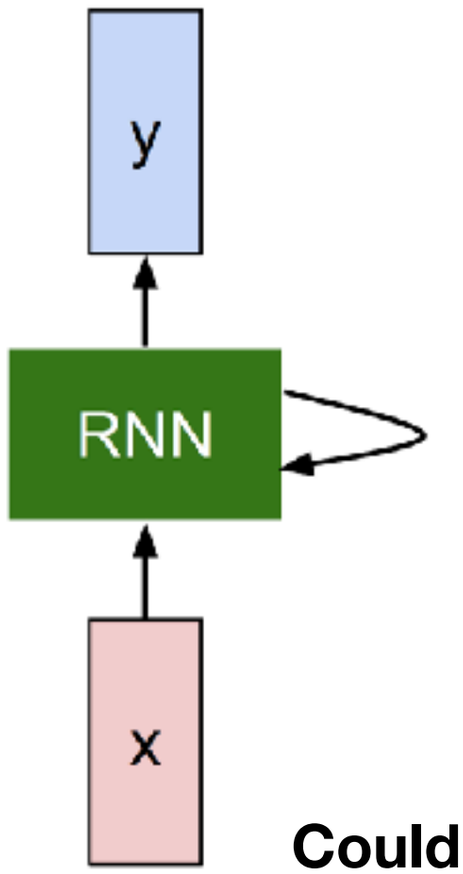
- The hidden state is UPDATED every time the RNN reads a new input and fed back to the net for the next time



A Vanilla RNN

- At some stage we want to predict an output y
  we can attach a FC layer to the hidden state to produce an output

The state consists of a single *"hidden"* vector **h**:



$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Could insert a Softamax to predict probabilities

Application to Jet Physics

Louppe G, Cho K, Becot C, Cranmer K. arXiv:1702.00748 [hep-ph] (2017)

- Louppe et. al. present a novel class of recursive neural networks built upon an analogy between QCD and natural languages. In the analogy,
  four-momenta are like words and
  the clustering history of sequential recombination jet algorithms is like the parsing of a sentence.

- The approach works directly with the four-momenta of a variable-length set of particles, and the jet-based tree structure varies on an event-by-event basis.

- Analogy is extended from individual jets (sentences) to full events (paragraphs), and show for the first time an event-level classifier operating on all the stable particles produced in an LHC event.

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Unsupervised Learning

Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

- No need for annotation

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

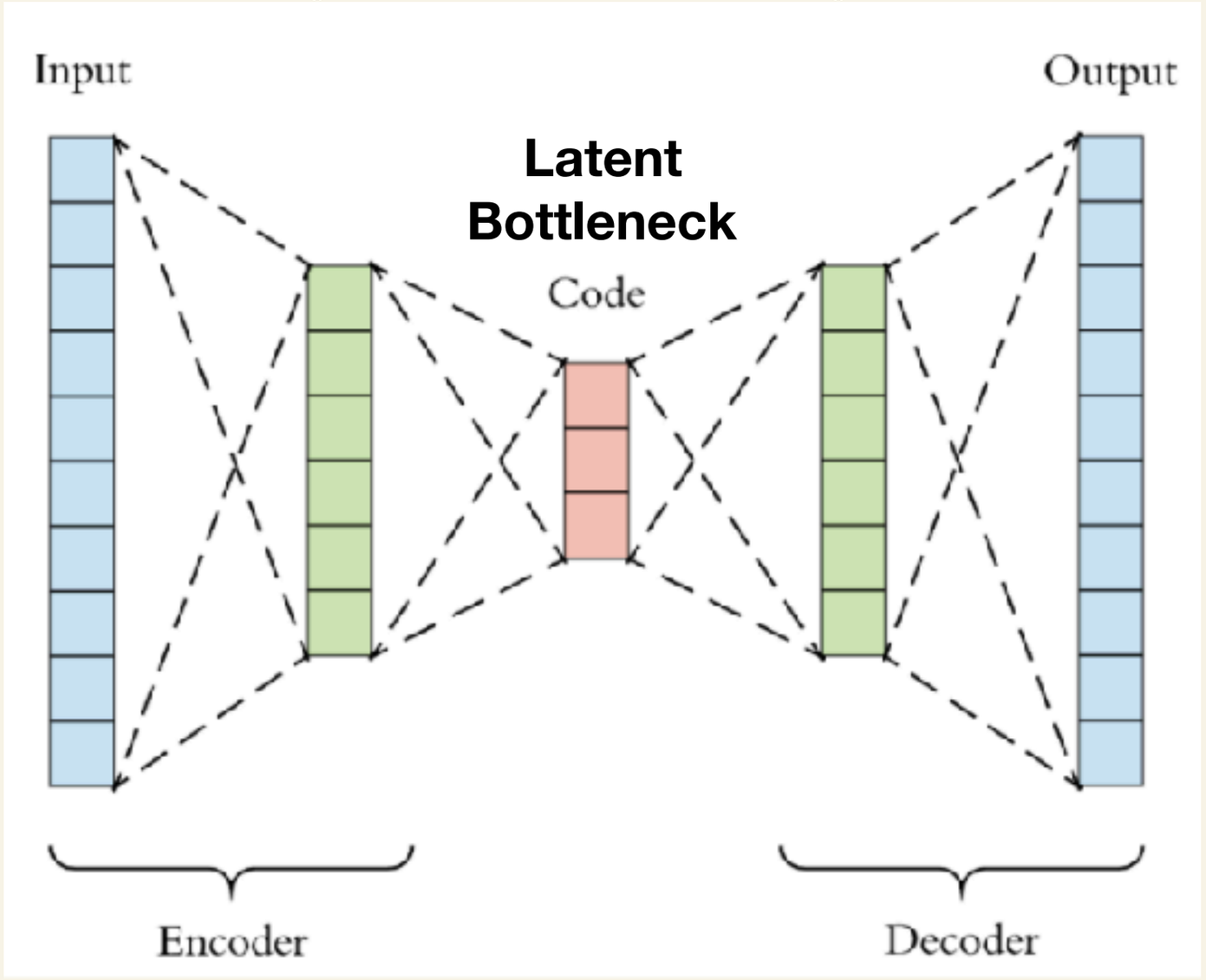
- If we manage to understand the underlying features of our DATA , it's a huge step towards understanding the visual world around us

Unsupervised Learning
Auto Encoders

Traditional Autoencoders

- Autoencoders are basically a form of compression, similar to the way an audio file is compressed using MP3, or an image file is compressd using JPEG.

- The encoder function, denoted by ϕ, maps the original data X, to a latent space F, which is present at the bottleneck.
The decoder function, denoted by ψ, maps the latent space F at the bottleneck to the output.

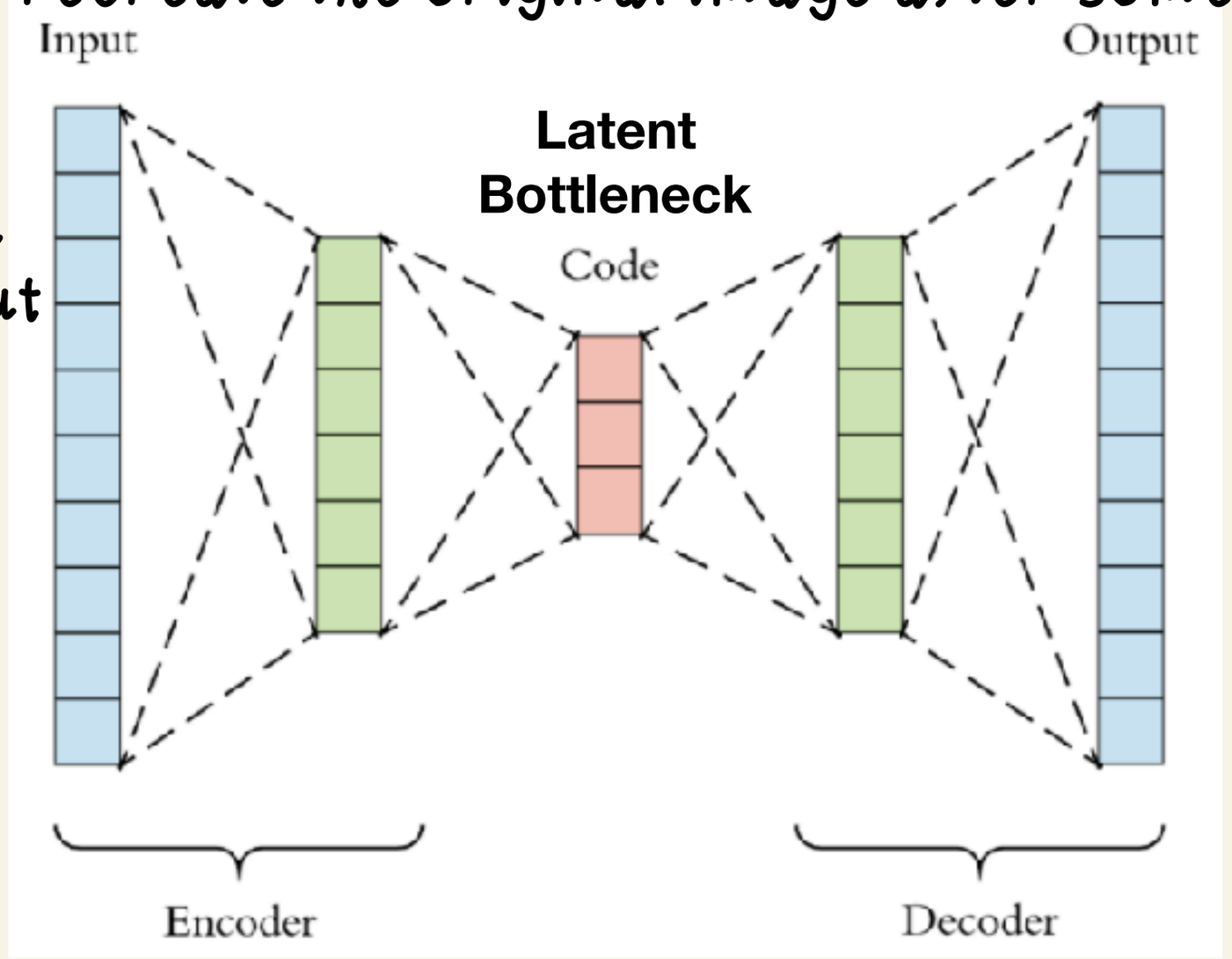


$$\phi : \mathcal{X} \to \mathcal{F}$$
$$\psi : \mathcal{F} \to \mathcal{X}$$
$$\phi, \psi = \underset{\phi,\psi}{\arg\min} \|X - (\psi \circ \phi)X\|^2$$

Traditional Autoencoders

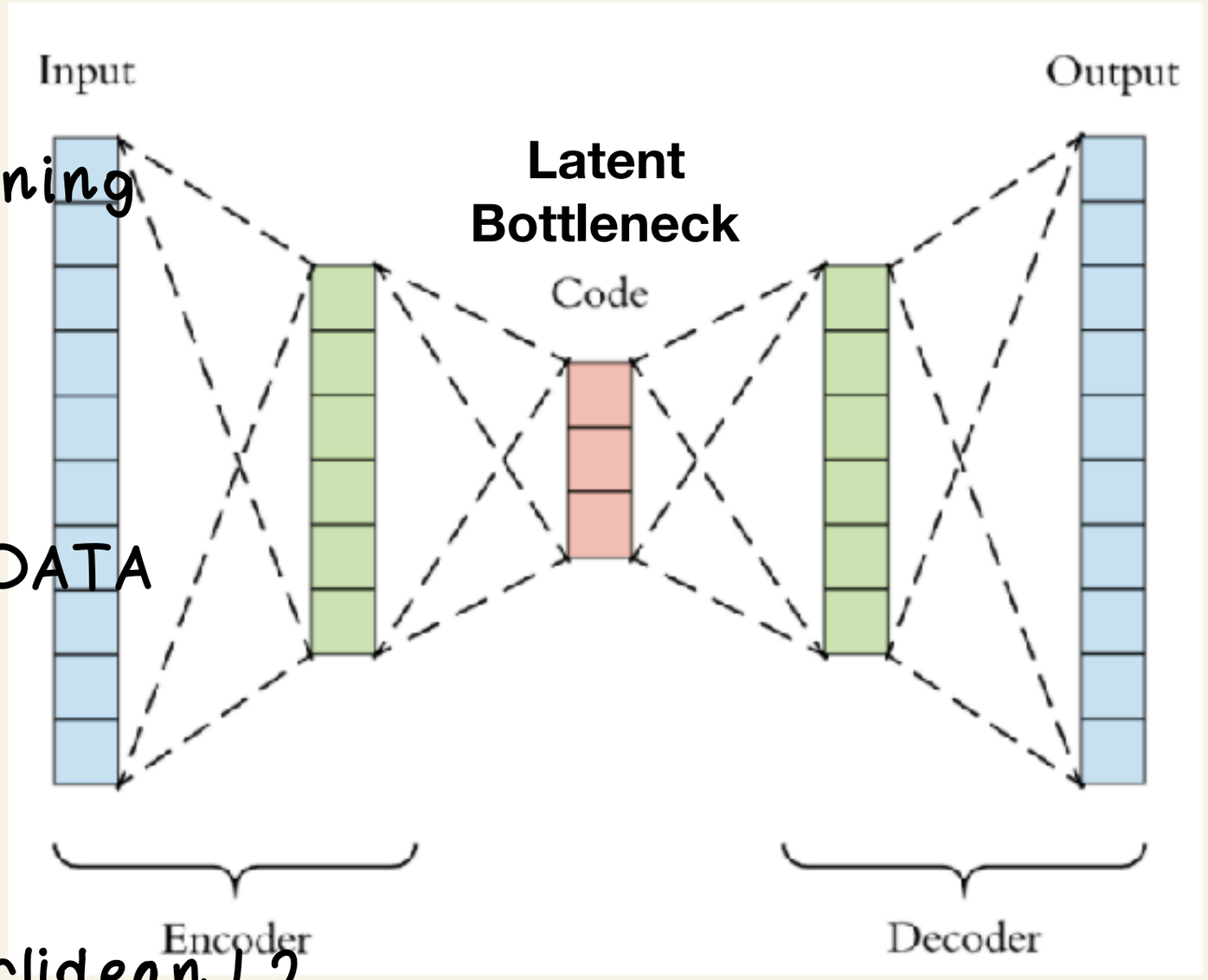
- The output, in this case, is the same as the input function. Thus, we are basically trying to recreate the original image after some non-linear compression.

- z is the latent compressed representation of the input from which we try to decompress the data into its original form



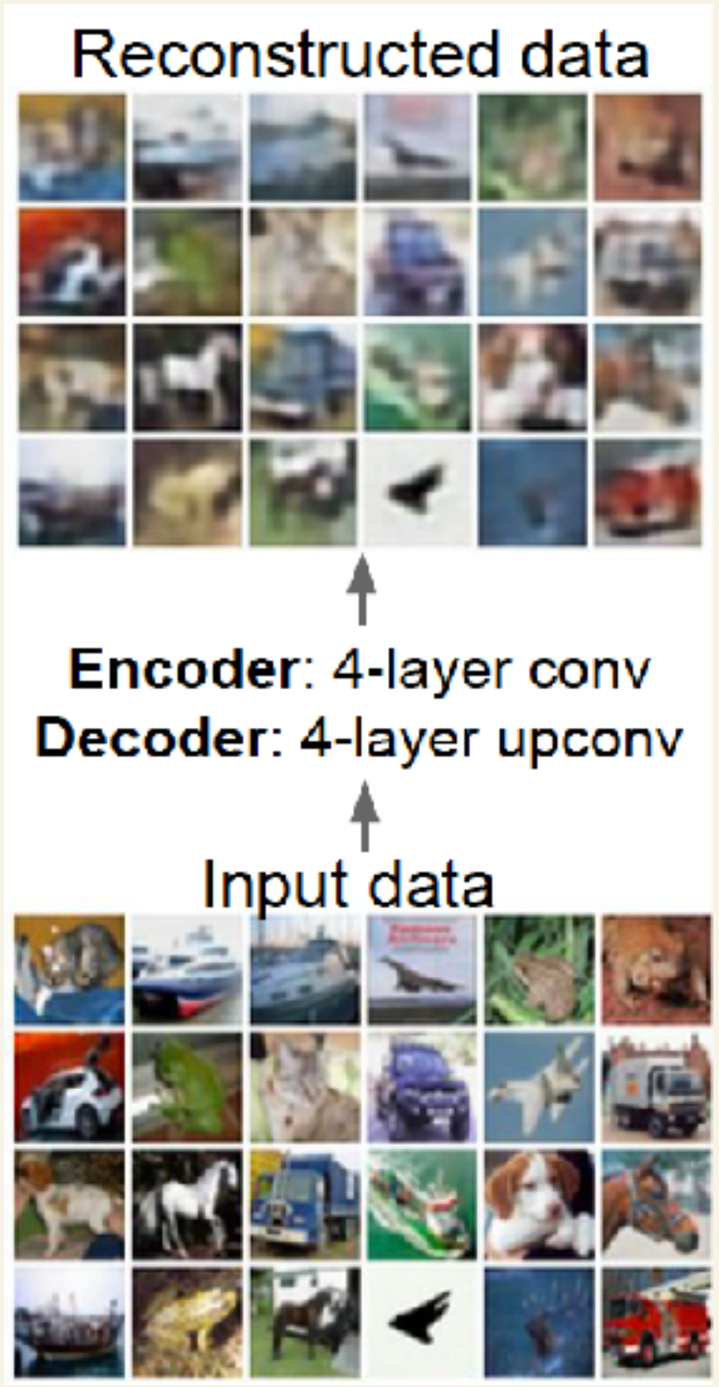
Traditional Autoencoders

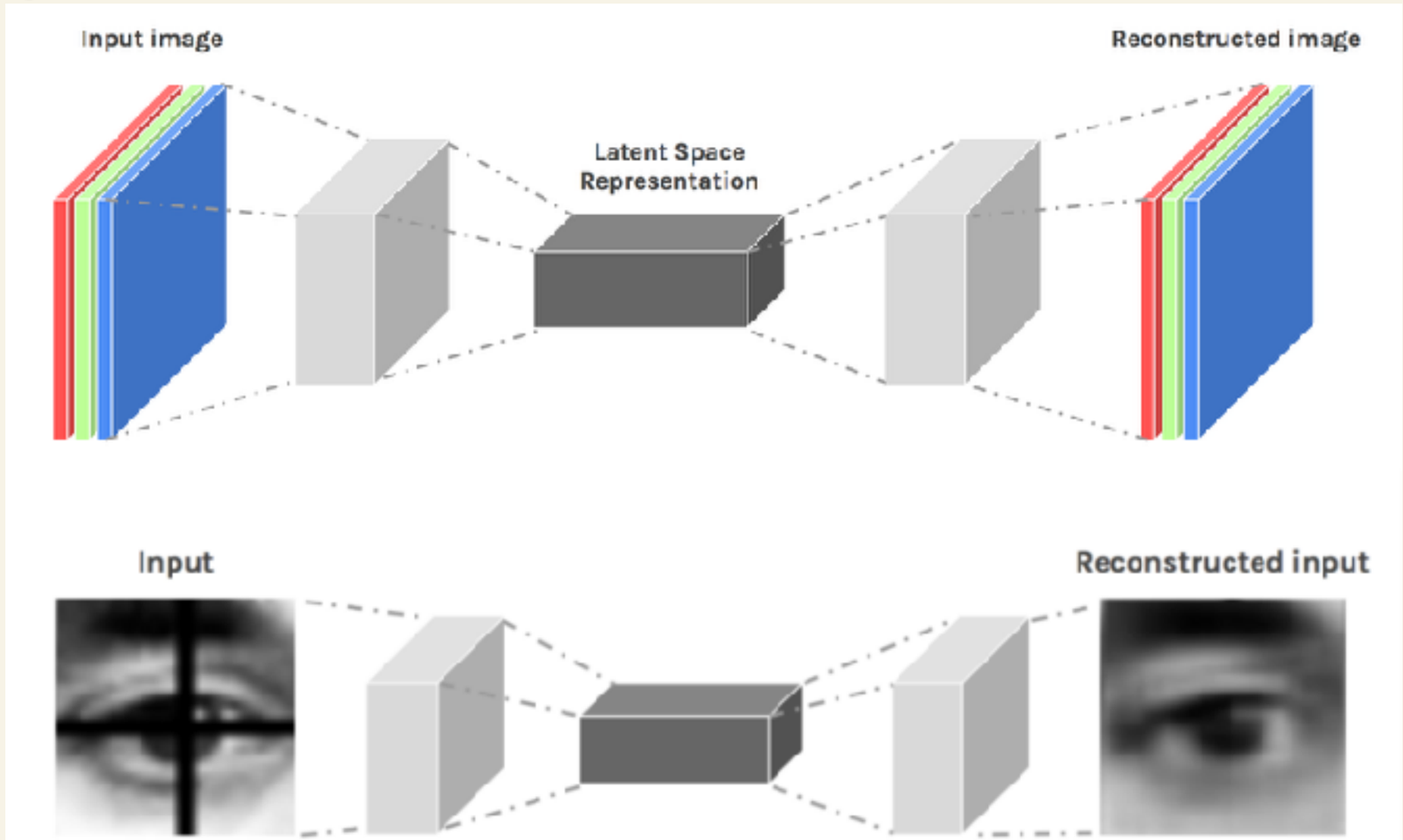
- z is with a smaller dimension than x, but it cannot be too small. to maintain its usefulness
- This is unsupervised learning
  No labels are involved
- The net will be forced to efficiently summarise the features of the input DATA
- Decoder and Encoder sometimes share weights
- Loss function is often Euclidean L2



Traditional Autoencoders

- And it works

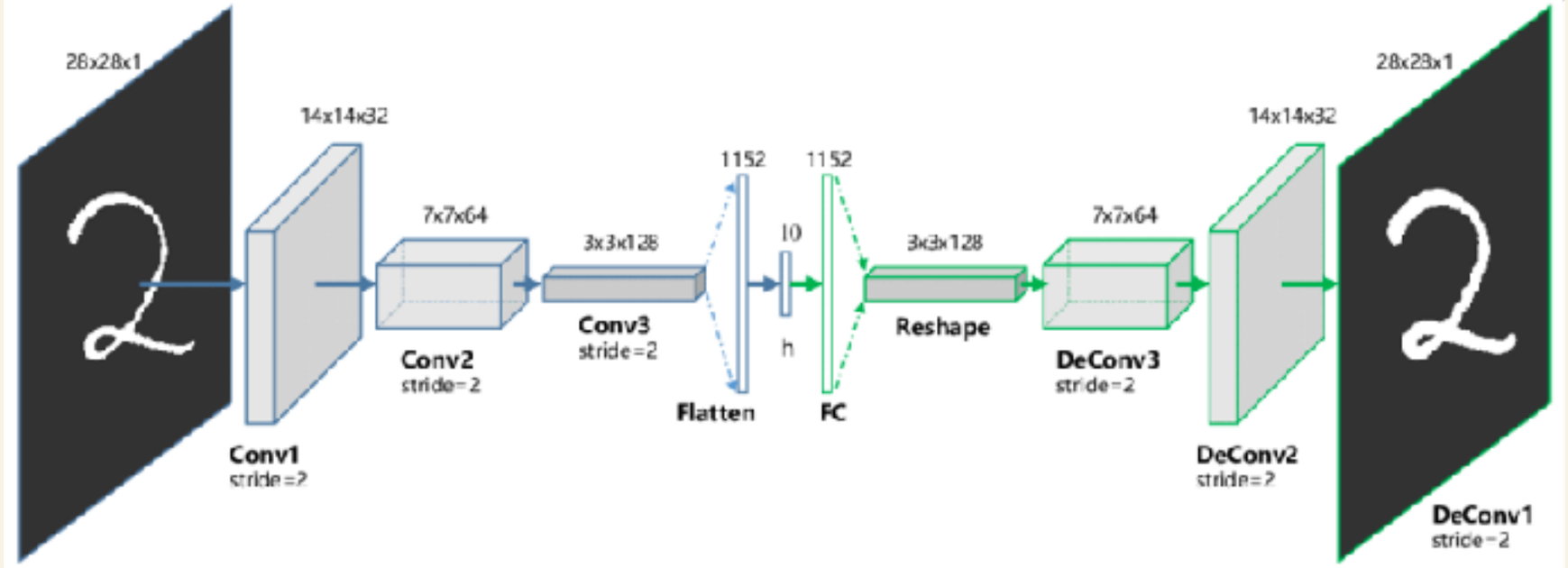


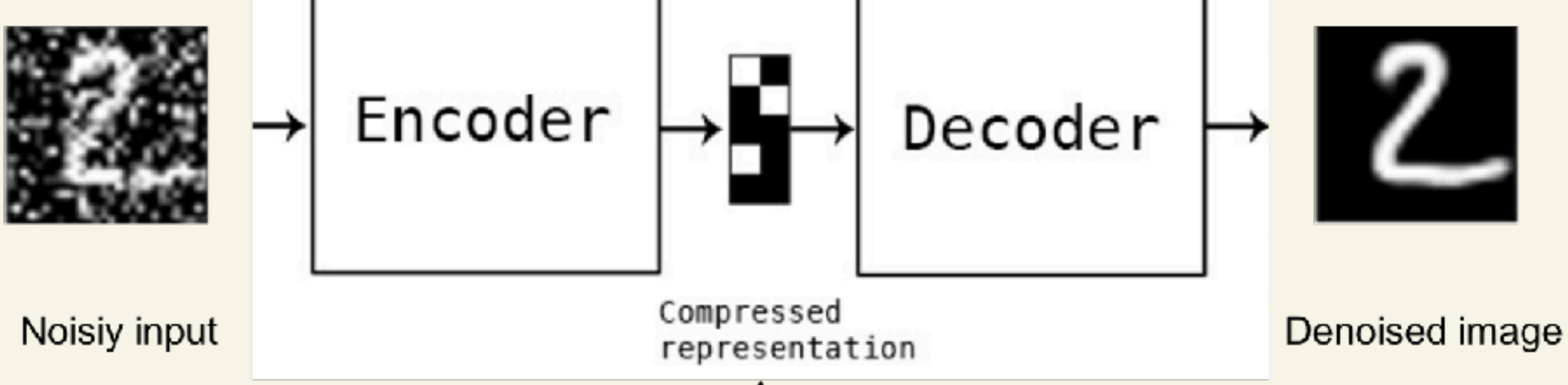


- Usually, you'll see transposed convolution layers used to increase the width and height of the layers. They work almost exactly the same as convolutional layers, but in reverse.

https://towardsdatascience.com/autoencoders-introduction-and-implementation-3f40483b0a85

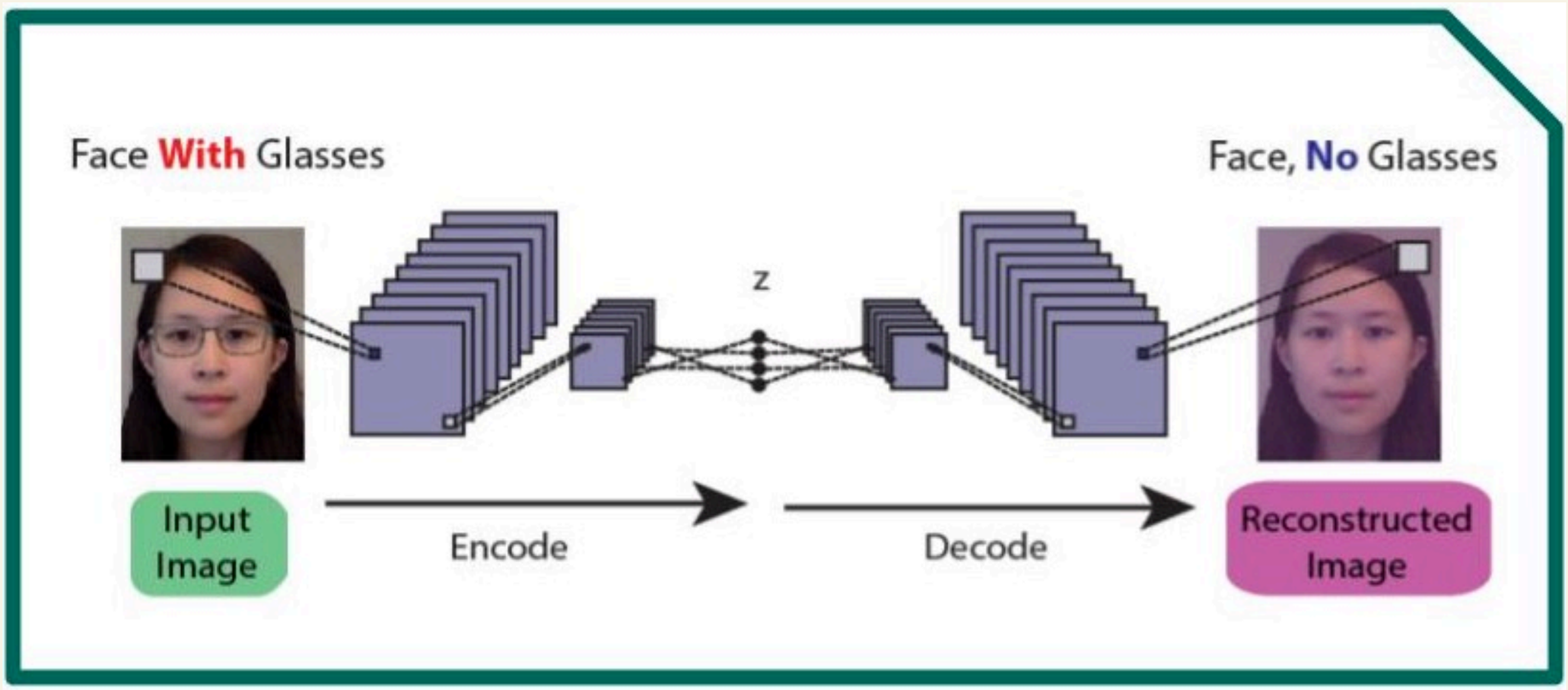
Denoising Autoencoder





Autoencoder Application

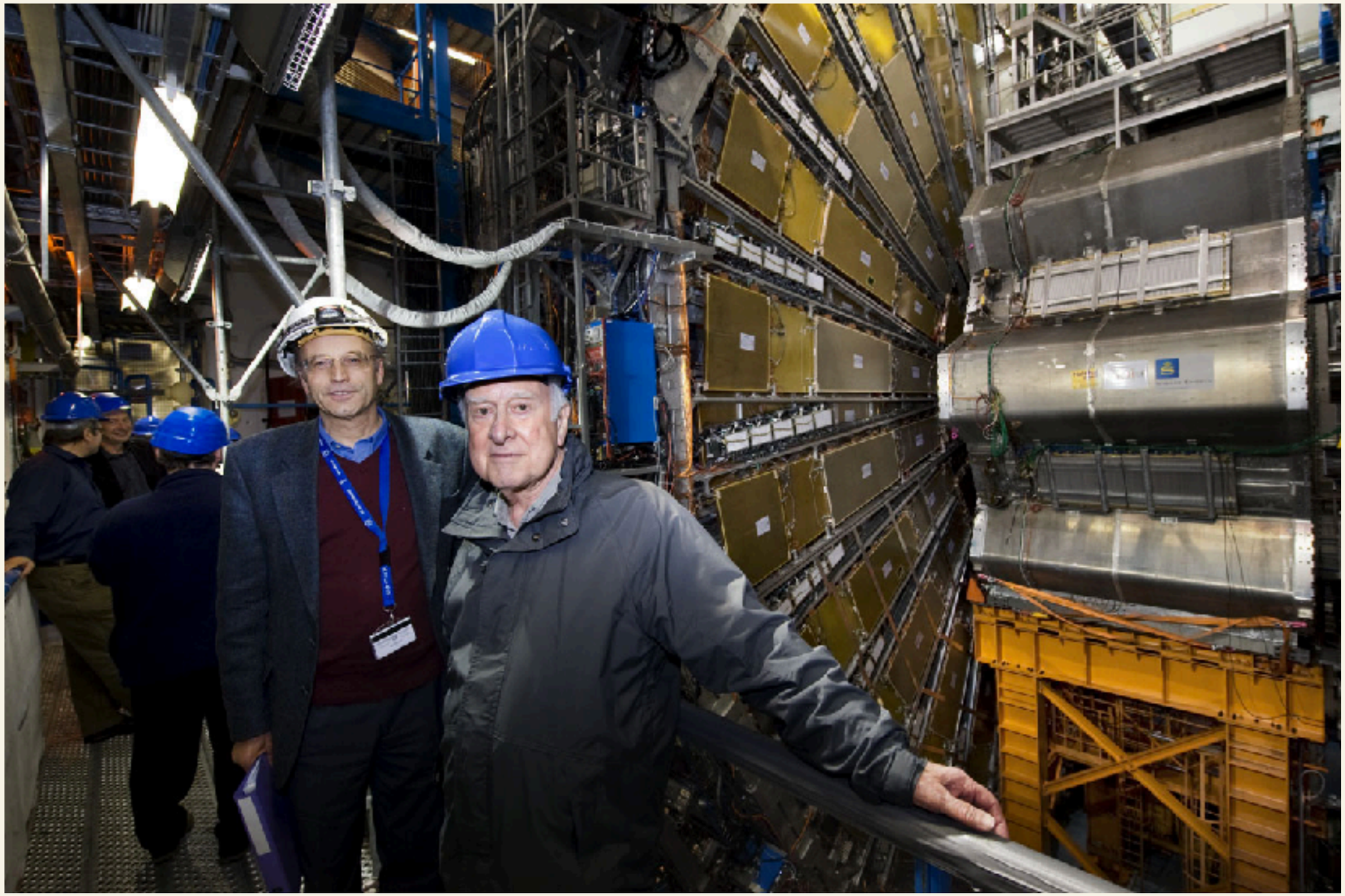
- Remove glasses or put glasses on



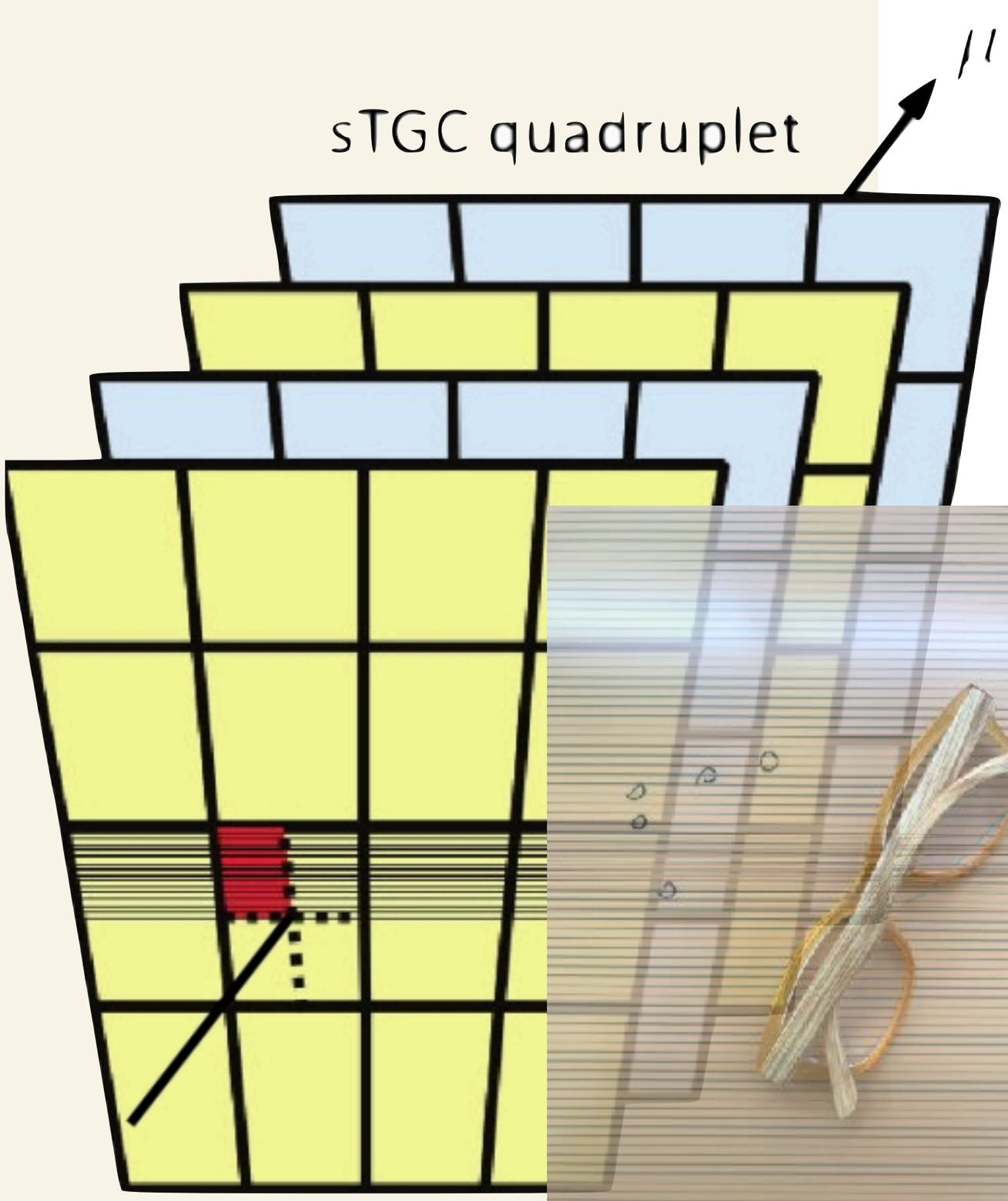
Autoencoders for Anomaly Detection

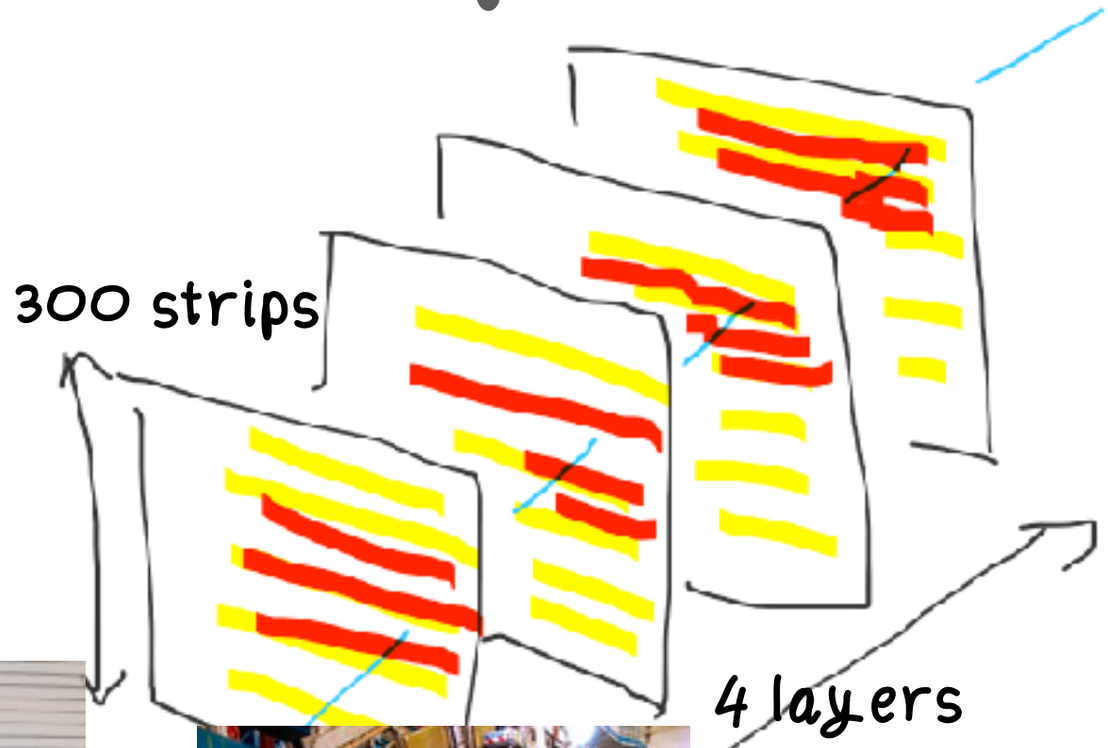
- Use Auto Encoders to teach the network how the SM "looks like" and then identify anomalies in the REAL DATA with respect to the SM.

Real Life Example - Weizmann



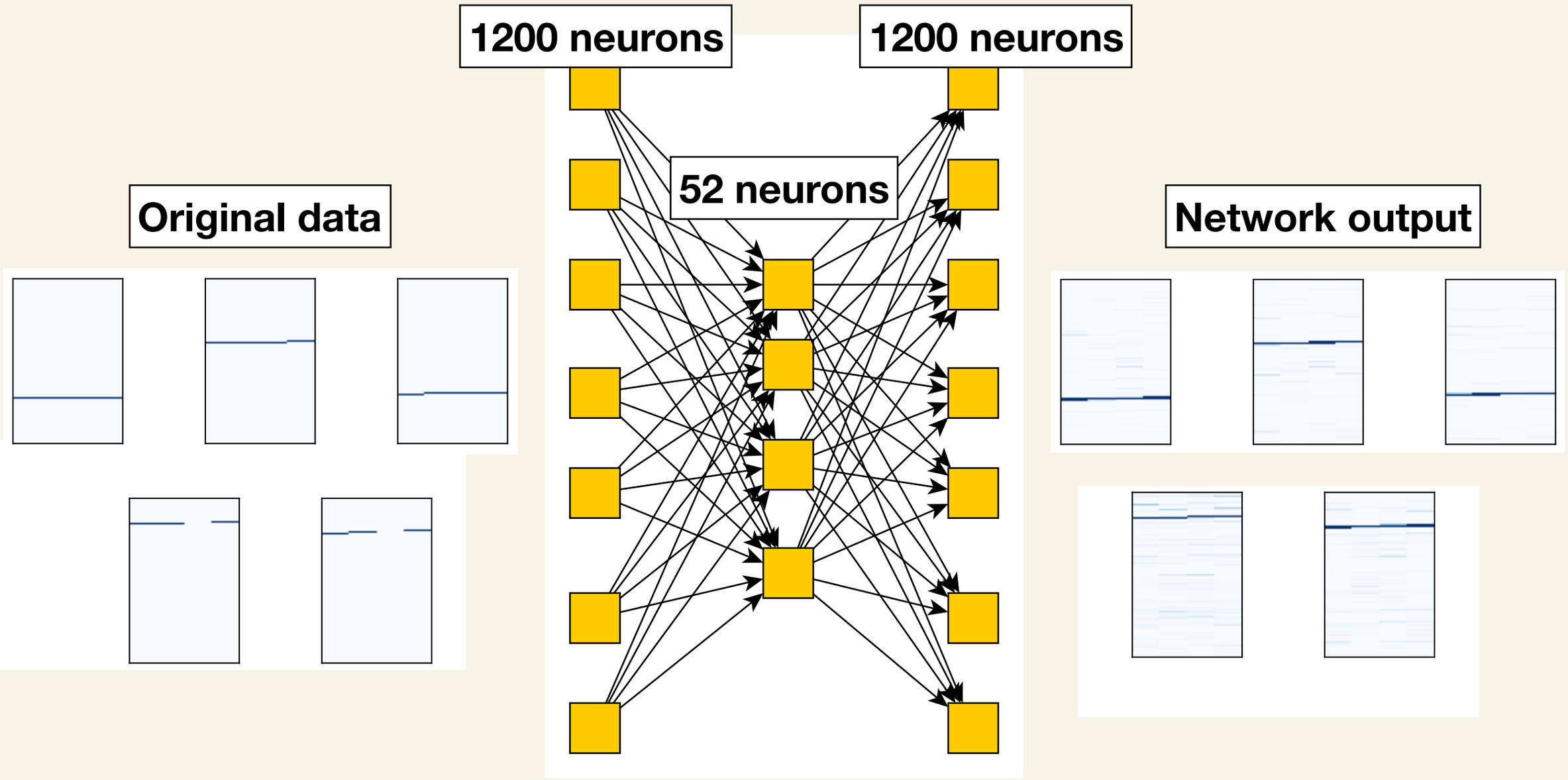
Real Life Example





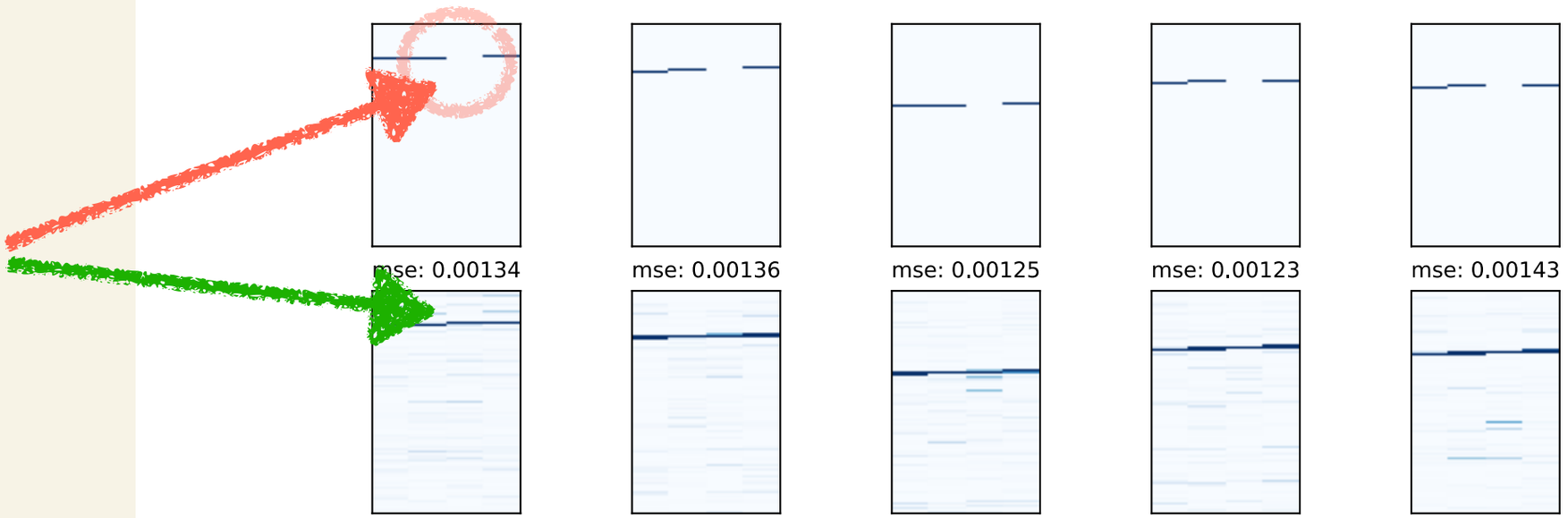
Single particle passing through each event, at random angle

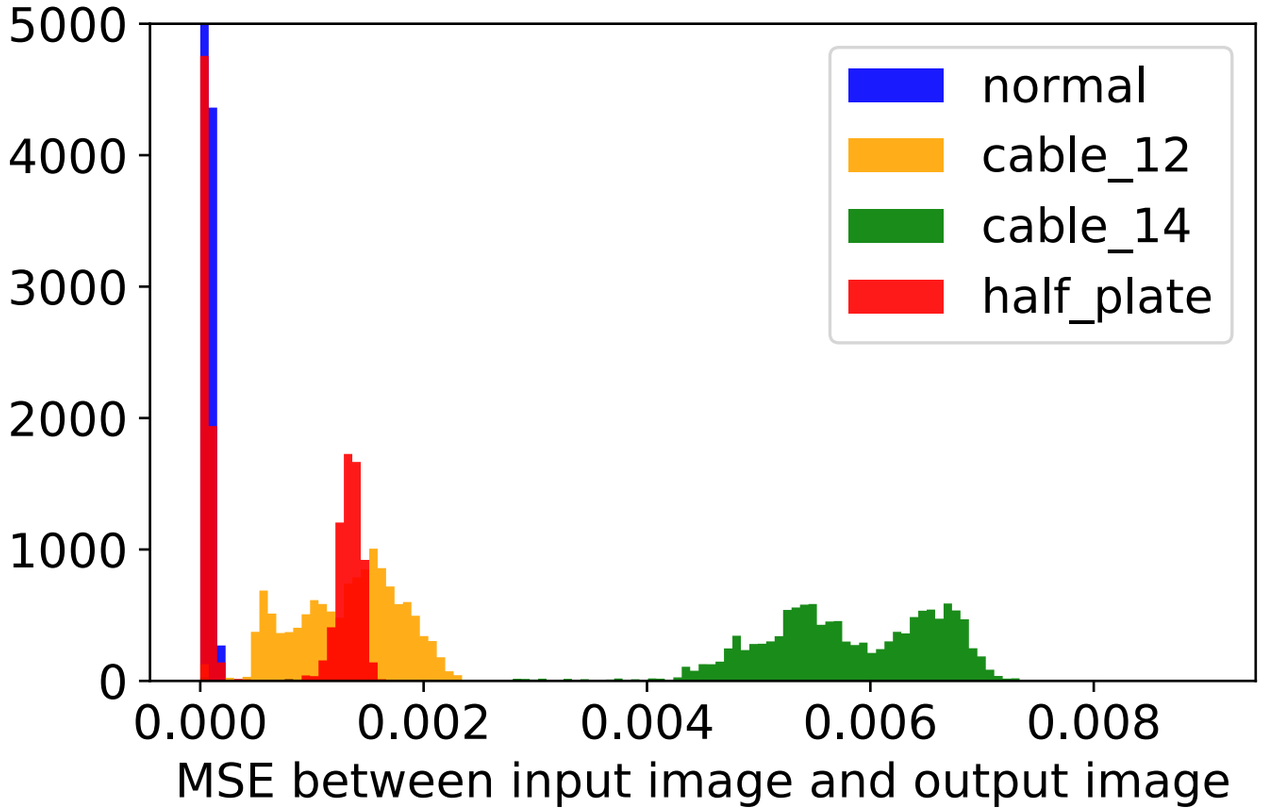
Real Life Example



Real Life Example

"Missing" hit





Generative Models
VAE & GAN

Generative Models

- Deep Learning provides some Generative solutions which can aid in complicated and time consuming events generation

- Variational AutoEncoders (VAE) and Generative Adversarial Networks (GAN) are used to generate DATA like images

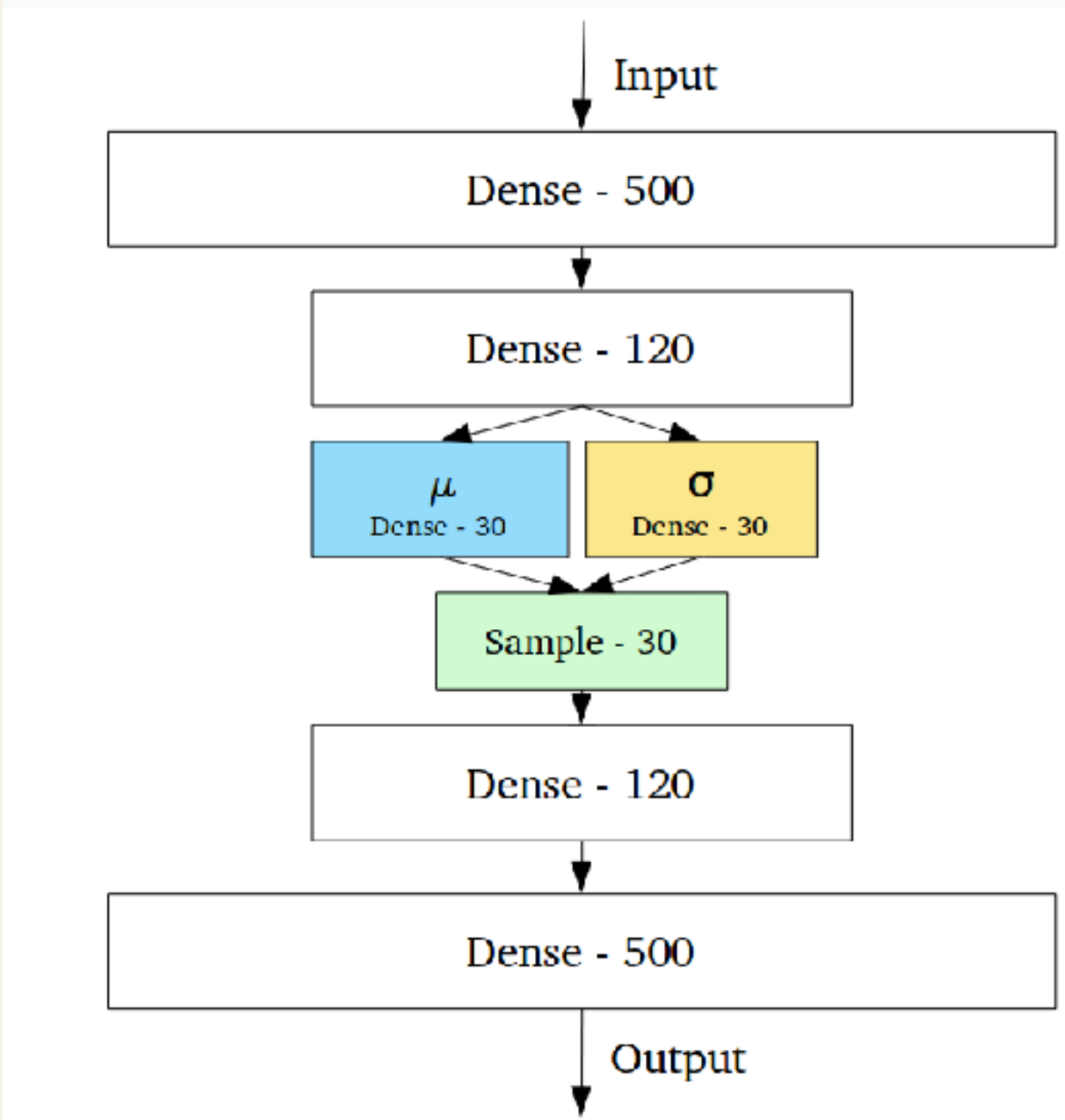
- VAE is trying to learn the distribution of the DATA from which one can sample DATA like

Variational Autoencoders - Simple VAE - Intuition

- VAE are constructed so their latent spaces are, by design, continuous, allowing easy random sampling and interpolation.

- The encoder does not output an encoding vector of size n, rather, outputting two vectors of size n: a vector of means, μ, and another vector of standard deviations, σ.

- From these vectors we sample a new fake DATA like object



VAE

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

Maximize Likelihood of original input X

Putting it all together: maximizing the likelihood lower bound

$$P(z) = \mathcal{N}(0,1)$$

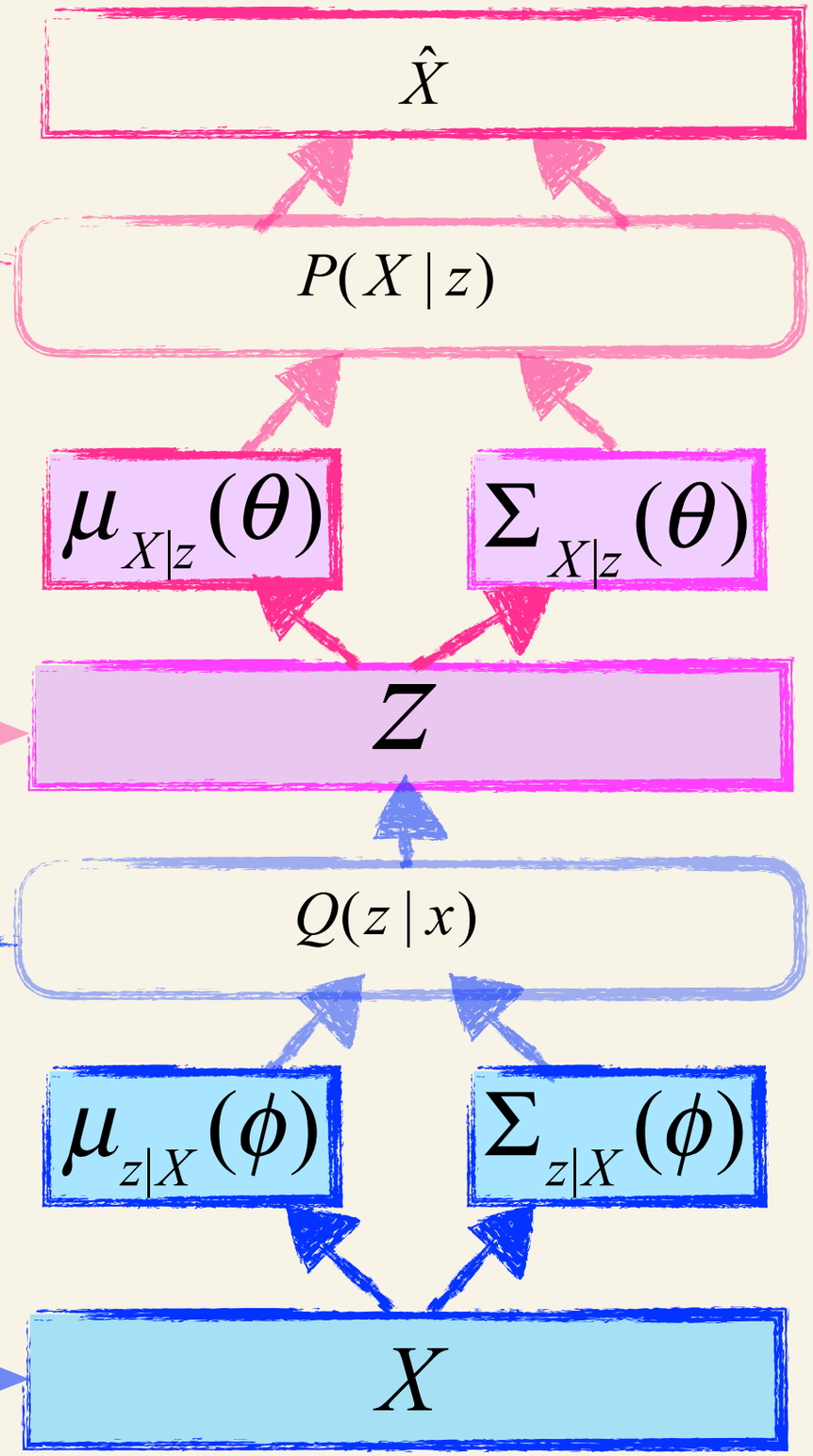
$$\log P_\theta(X) \le E_{z\sim Q_\phi(z|X)}[\log(P_\theta(X\,|\,z)] - D_{KL}\left[Q_\phi(z\,|\,X) \,\|\, P_\theta(z)\right]$$

θ

Make posterior distribution of z close to prior

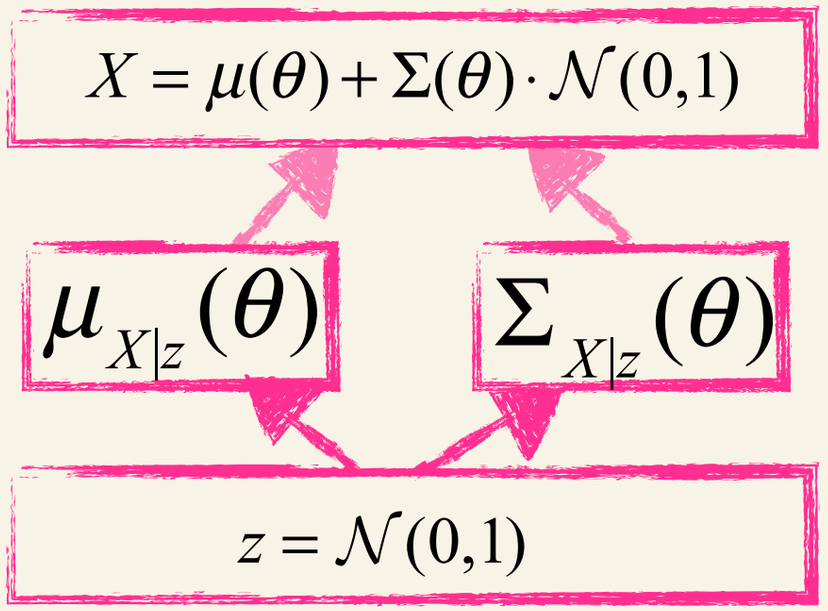
For every minibatch of input data: compute this forward pass, and then backprop!

ϕ



We update our model by continuously update the decoder and encoder parameters, Φ and ϑ

VAE

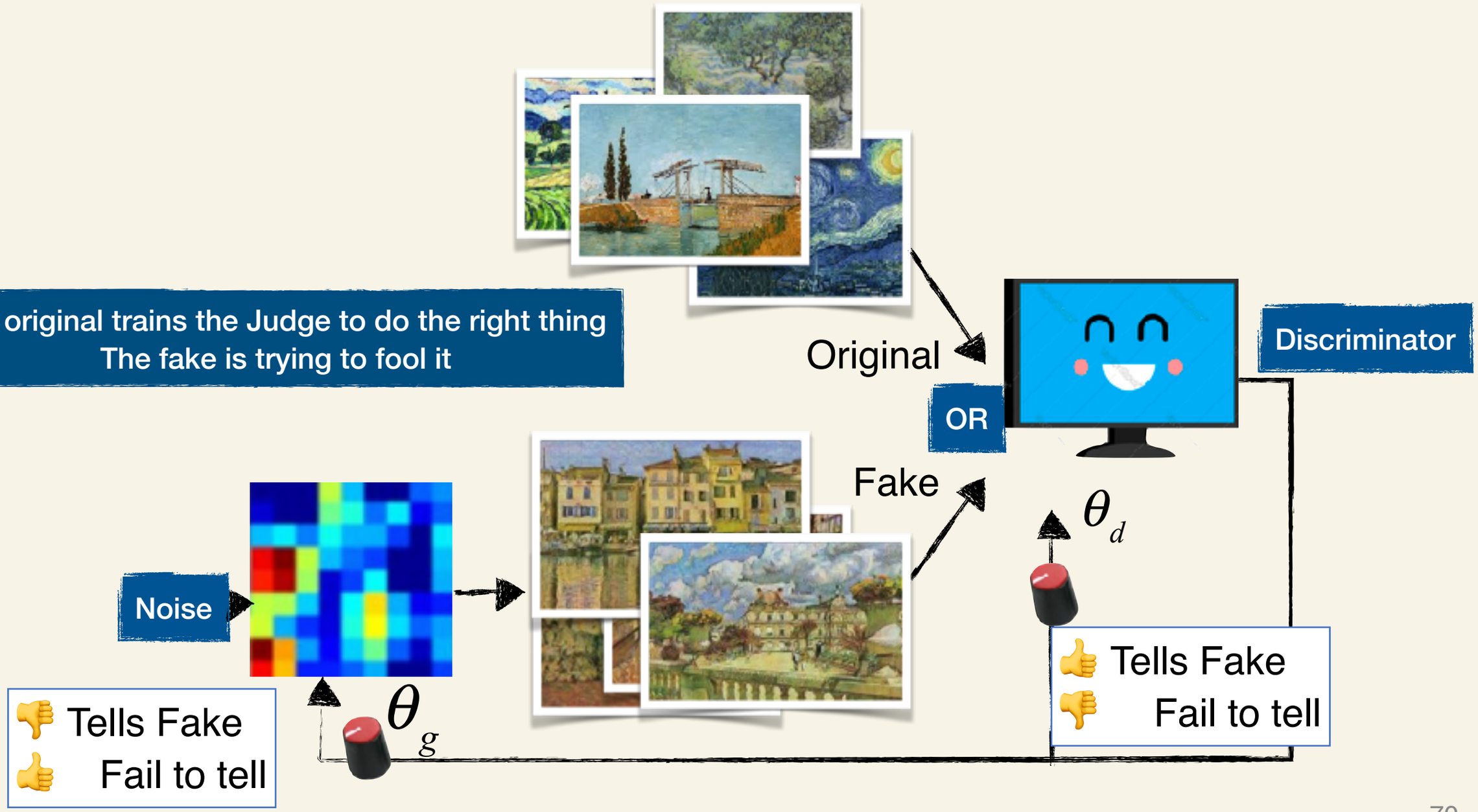

$X = \mu(\theta) + \Sigma(\theta) \cdot \mathcal{N}(0,1)$
$\mu_{X|z}(\theta)$
$\Sigma_{X|z}(\theta)$
$z = \mathcal{N}(0,1)$

Happiness is a Warm GAN

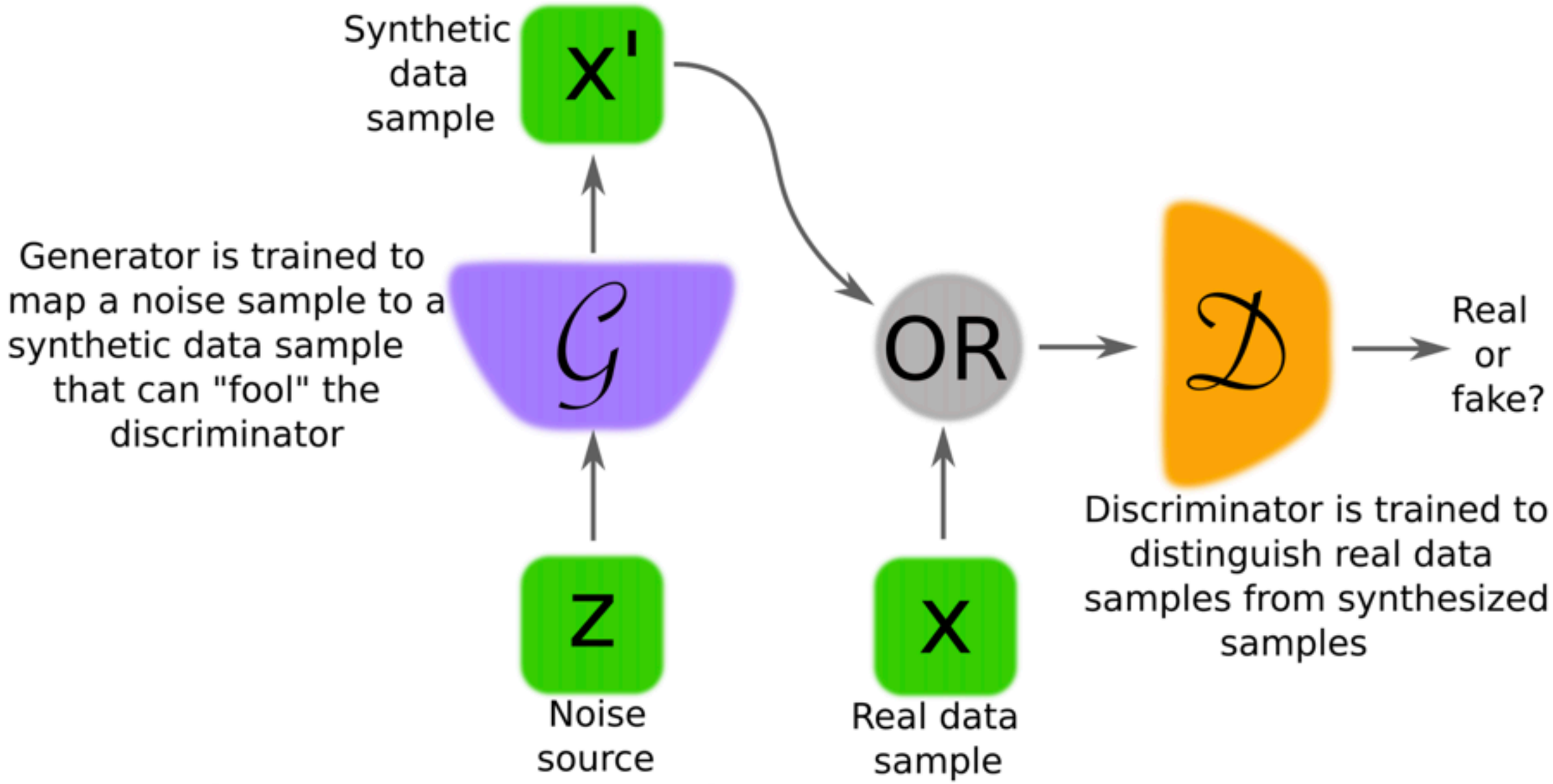
"Generative Adversarial Networks is the most interesting idea in the last 10 years in Machine Learning."
—Yann LeCun, Director of AI Research at Facebook AI

- Celebrities who have never existed

GAN-GOCH


The original trains the Judge to do the right thing
The fake is trying to fool it
Original
Discriminator
OR
Fake
Noise
θ_d
θ_g
Tells Fake
Fail to tell
Tells Fake
Fail to tell

GAN Philosophy Recap



- The goal of the generator is to fool the discriminator whose goal is to be able to distinguish between true and generated data. So, when training the generator, we want to maximise the “loss” while we try to minimise it for the discriminator.

https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

GAN MATH

- The expected absolute error of the discriminator can be expressed as

$$E(\theta_g,\theta_d)=\frac{1}{2}\left(E_{x\sim p_{data}}\left[1-\mathcal{D}_{\theta_d}(x)\right]+E_{x\sim p_g}\left[\mathcal{D}_{\theta_d}\left(\mathcal{G}_{\theta_g}(x)\right)\right]\right)$$

$$real\ DATA\rightarrow \mathcal{D}_{\theta_d}(x)\sim 1 \quad fake\ DATA\rightarrow \mathcal{D}_{\theta_d}\left(\mathcal{G}_{\theta_g}(x)\right)\sim 0$$

- Discriminator (θd) wants to minimize objective such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake) -
  Generator (θg) wants to maximise objective such that D(G(z)) is close to 1
  (discriminator is fooled into thinking generated G(z) is real)

- The goal of the generator is to fool the discriminator whose goal is to be able to distinguish between true and generated data. So, when training the generator, we want to maximise this error while we try to minimise it for the discriminator.

- minmax Game $\max_{\theta_g}\left(\min_{\theta_d} E(\theta_g,\theta_d)\right)$

Fast Event Generation (GAN)

- HL-LHC necessitates the need to simulate trillions of events. With O(10 min) to simulate a multi-jets event in GEANT, this becomes a true bottle neck

- Fast simulation exists but at the price of oversimplification

- What we need is to be able to generate the global events in an accurate way without the knowledge of internal details

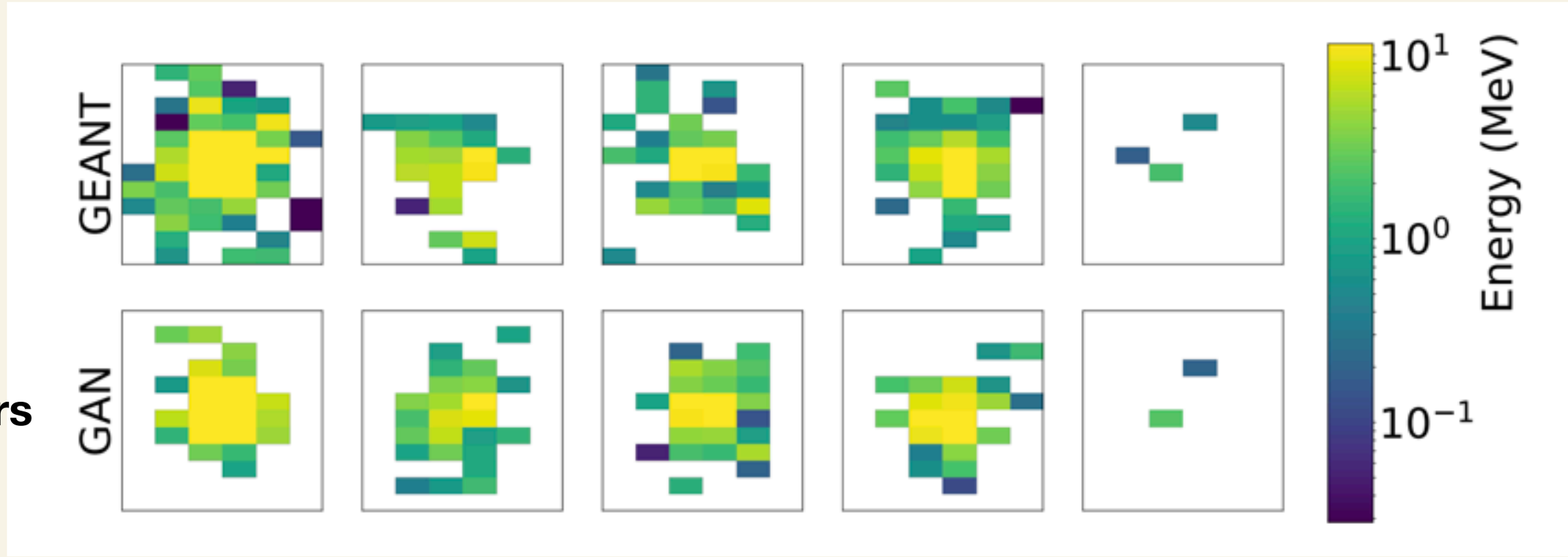
What can we (hope to) do with GAN

- Use neural networks to generate events similar to those used to train a network .
- This will allow faster detector simulations while preserving the accuracy of a full simulation.
- GANs offer an alternative to simulation, with an order of magnitude improvement in the simulation speed, but yet not accurate enough. There is still a long way to go till GANs are effectively used and are able to reproduce the interactions of particles with material and "simulate" a real detector.
- GEANT is so slow that its hard to imagine one can keep on using it for HL.
- R&D is required to improve the GANs, but its a highly promising avenue to pursue.

First Bird: CaloGAN

- CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks
  Paganini et. al. arXiv:1712.1032

nearest neighbours from Training



Reducing systematics (GAN)

- One can use the adversarial component of the GAN to train the network to be insensitive to a specific systematics at the price of reduced (classifier) performance

Semantic Segmentation UNET

Pixel level Classification

- So far we classified images as a whole…
  We answered questions like: What is the content if the entire image as a whole

This image is CC0 public domain

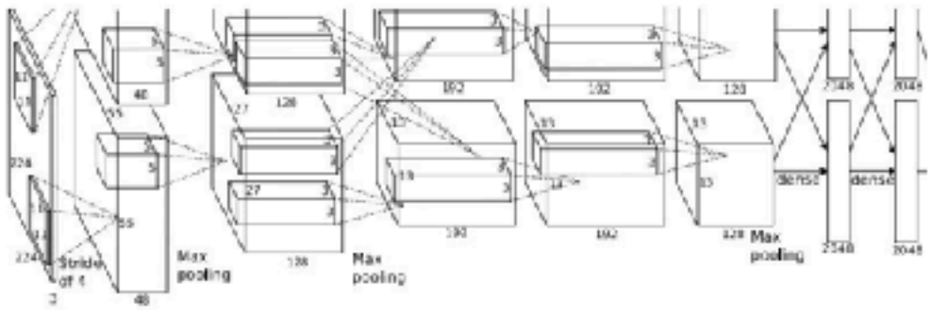
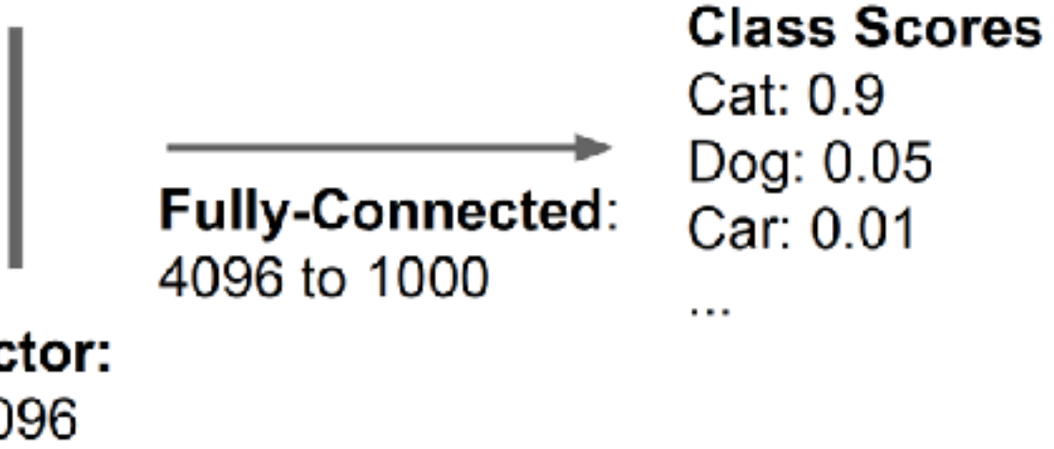
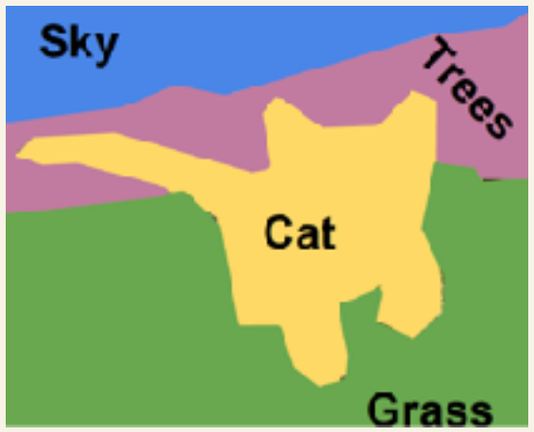


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

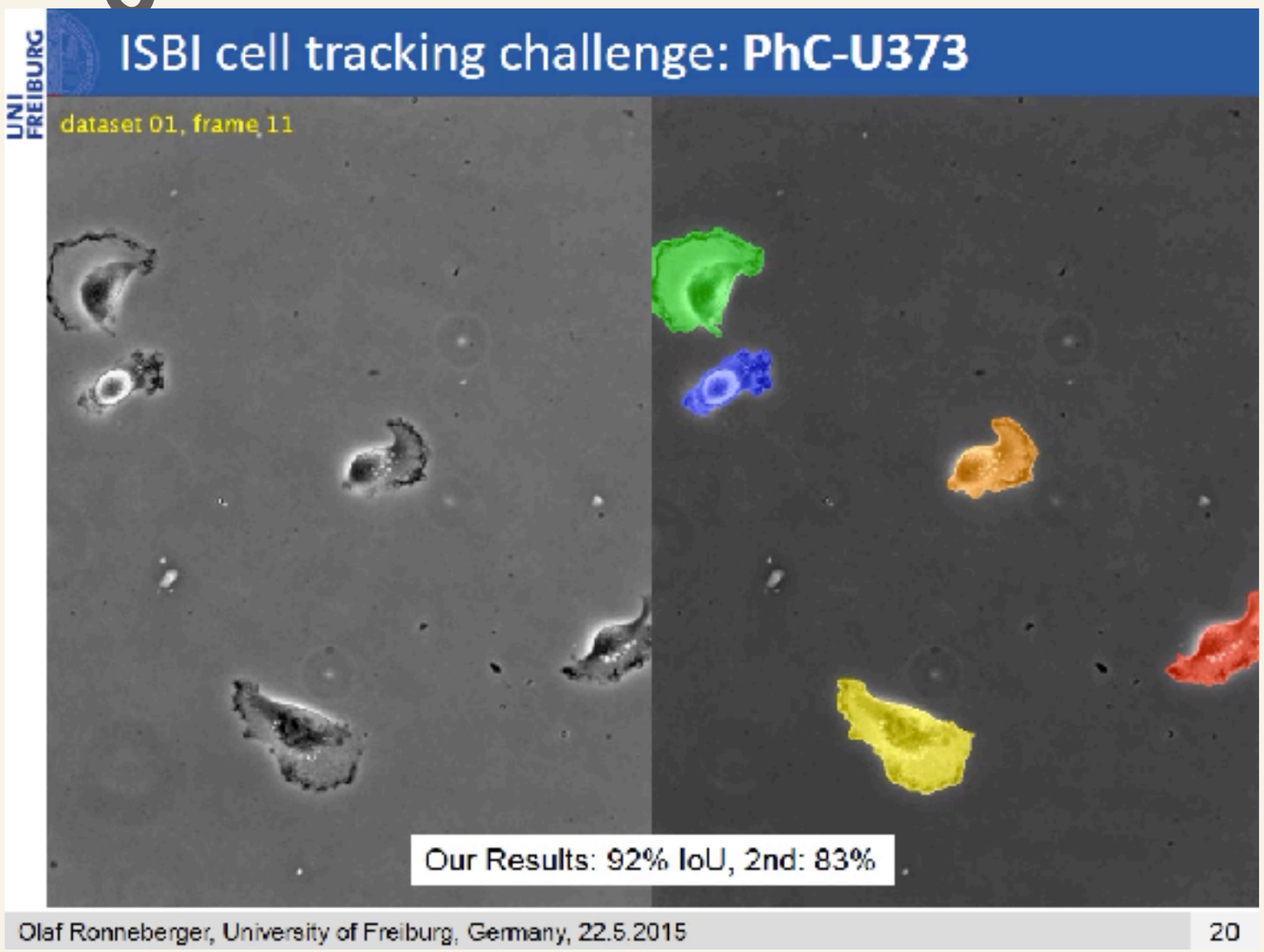


- Now we want to try and categorise pixels in an image.. To what object they belong



Semantic Segmentation: U-NET

- U-NET is an example of a modern pixel by pixel classification net that is aiming at addressing the localisation vs classification tradeoff



https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

93% IoU

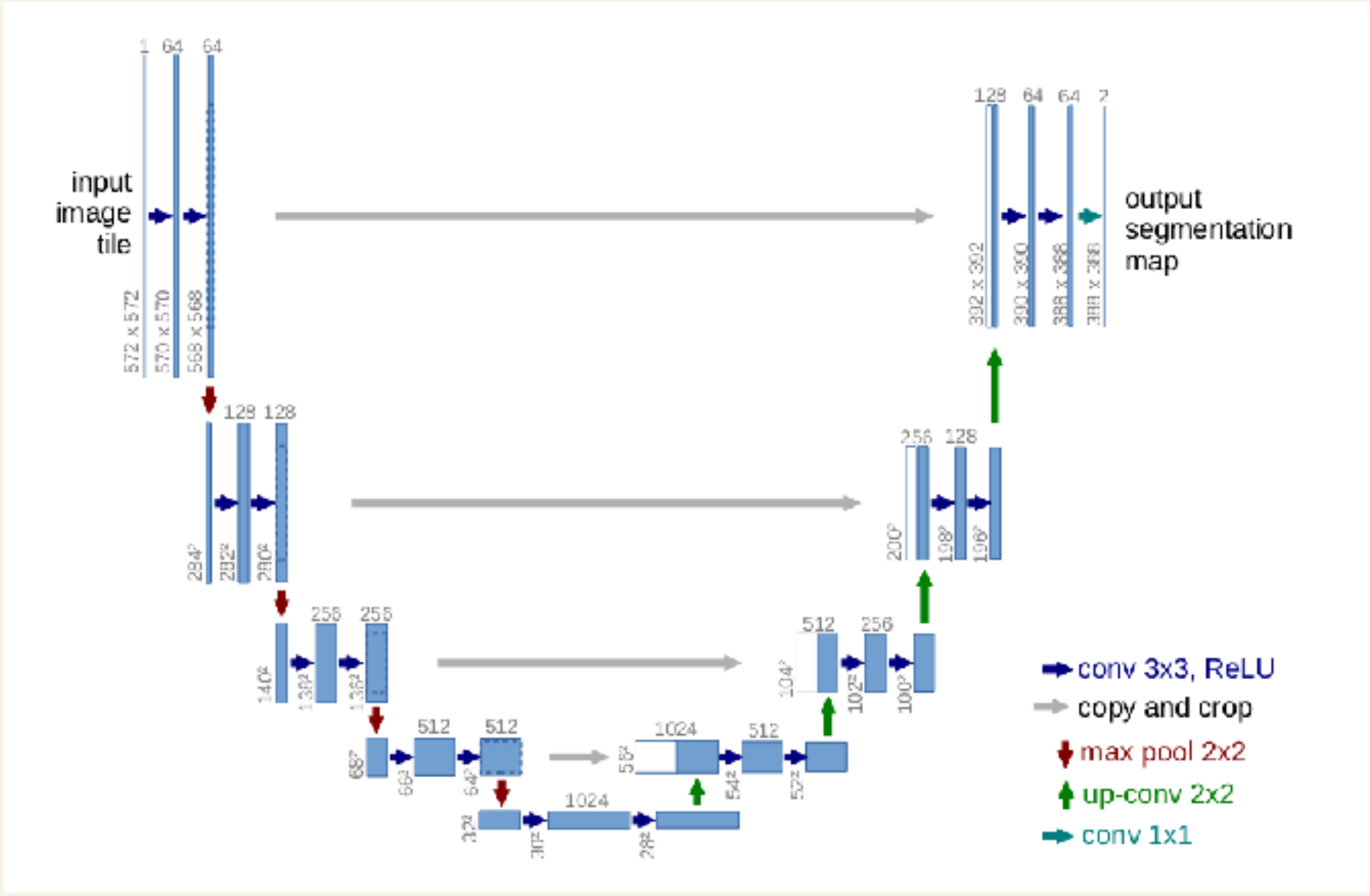
The IoU (Intersection Over Union) is the ratio between the area of overlap and the area of union between the ground truth and the predicted areas.

Ronneberger et. al. https://arxiv.org/pdf/1505.04597.pdf

Semantic Segmentation: U-NET

- U-NET is an example of a modern pixel by pixel classification net that is aiming at addressing the localisation vs classification tradeoff

- Pixel level classification is ideal for Energy Flow algorithms



Ronneberger et. al. https://arxiv.org/pdf/1505.04597.pdf

Wrap Up

NP mining at the trigger level

- There are whole classes of events, for example beauty and charm hadrons or low-mass dark matter signatures, which are so abundant that it is not affordable to store all of the events for later analysis.

- In order to fully exploit the physics reach of the LHC, it will increasingly be necessary to perform more of the data analysis in real-time.

- ML may be the only hope of performing real-time reconstruction that enables real-time analysis.

- The hope is that ML will be able to fully do tracking, vertexing. and jets analysis at early stages allowing to lower for example, jet trigger thresholds. And all that at the RAW level

What can we (hope to) do with ML

The MECCA of ML in HEP

- can ML eventually run the experiment using Reinforced Learning

- Reinforced algorithms based on Deep Learning DAQ Monitoring, Hardware Failure detection auto encoders can make recommendation for best operation of the detector (or even take the decisions….).

- In short WE CAN GO HOME

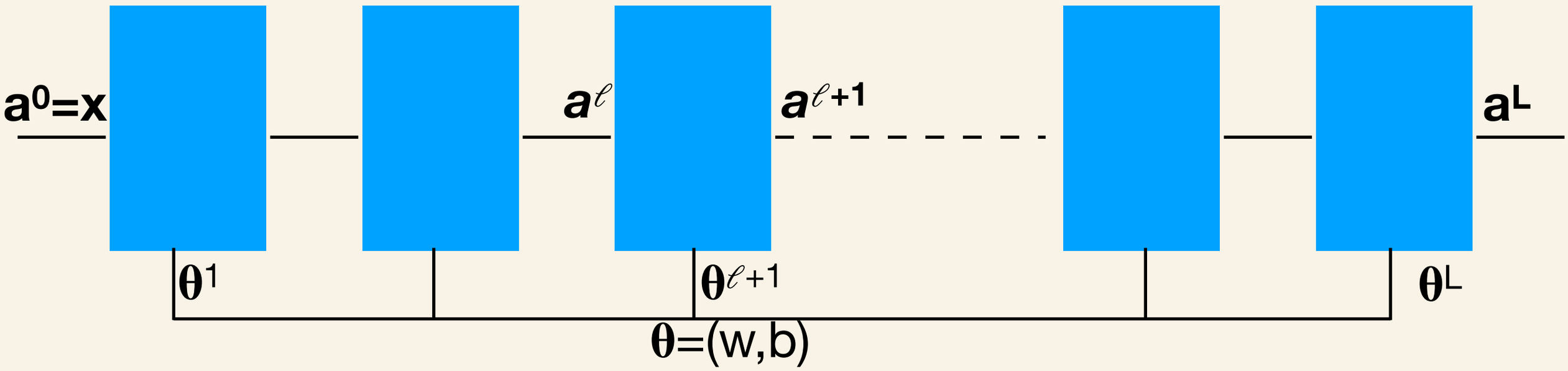
BACKUP
Backpropagation

Backpropagation (of errors)

Backpropagation

$$Gradient\ Descent\ Optimization \Rightarrow \theta_t \rightarrow \theta_{t+1}\ with\ \nabla_\theta C(\theta)$$

$$But\ how\ do\ we\ get\ \nabla_\theta C(\theta)?\quad \left(\frac{\partial C}{\partial w}, \frac{\partial C}{\partial b}\right) with\ O\left(10-100K\ parameters\right)$$

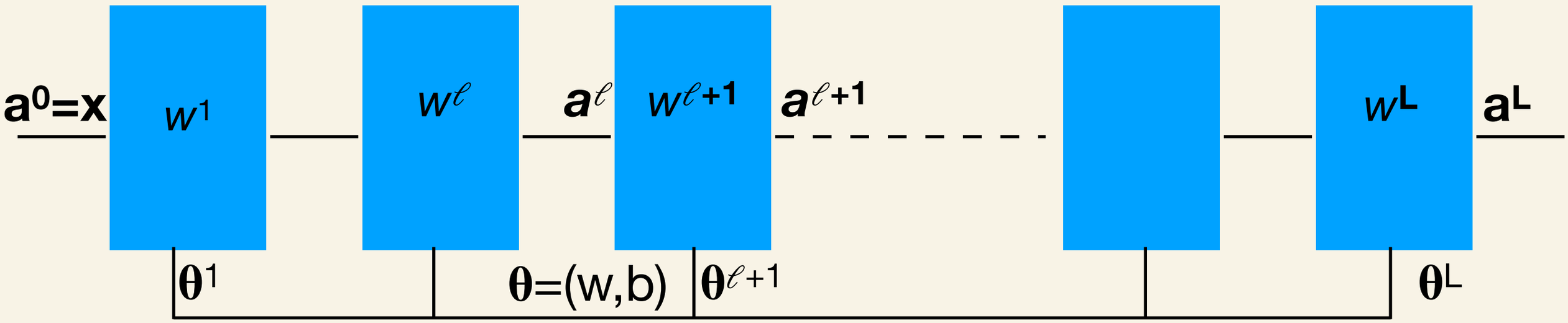


Backpropagation is backward propagation of errors – an error is computed at the output and distributed backwards throughout the network's layers

$$given\ w^\ell, b^\ell, a^\ell\ and\ the\ Cost\ function\ C(\theta)$$

$$Start\ from\ layer\ L\ and\ calculate\ \frac{\partial C}{\partial b^\ell}\ and\ \frac{\partial C}{\partial w^\ell}$$

Backpropagation



The weights and biases are given,we want to update them

$$C = C\left(y'(x), y\right) = C\left(a^L(x), y\right); \quad a^L = \sigma(z^L); \quad z^L(\theta) = w^L a^{L-1} + b^L$$

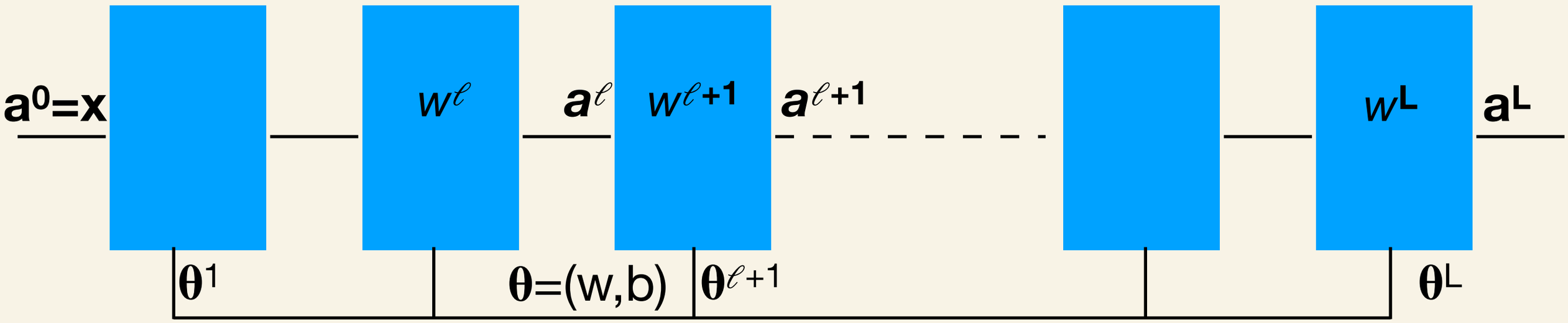
error at output Layer: $\delta^L = \dfrac{\partial C}{\partial z^L};$

Chain Rule: $\delta^\ell = \left(w^{\ell+1}\right) \cdot \delta^{\ell+1} \sigma'(z^\ell)$

BackPropagation: $\delta^L --> \delta^{L-1} --> \delta^{L-2} --> \ldots\ldots --> \delta^1$

We will show that: $\dfrac{\partial C}{\partial b^\ell} = \delta^\ell; \quad \dfrac{\partial C}{\partial w^\ell} = a^{\ell-1}\delta^\ell$

Backpropagation



The weights and biases are given,we want to update them

$$a^L=\sigma(z^L);\quad z^L=w^L a^{L-1}+b^L$$

error at output Layer: $\delta^L=\dfrac{\partial C}{\partial z^L}=\dfrac{\partial C}{\partial a^L}\dfrac{\partial a^L}{\partial z^L}=\dfrac{\partial C}{\partial a^L}\sigma'(z^L);$

Chain Rule: $\delta^\ell=\dfrac{\partial C}{\partial z^\ell}=\dfrac{\partial C}{\partial z^{\ell+1}}\dfrac{\partial z^{\ell+1}}{\partial z^\ell}=\left(w^{\ell+1}\right)\cdot\delta^{\ell+1}\sigma'(z^\ell)$

Back Propagation: $\delta^L-->\delta^{L-1}-->\delta^{L-2}-->.....-->\delta^1$

We will show that: $\dfrac{\partial C}{\partial b^\ell}=\delta^\ell;\quad \dfrac{\partial C}{\partial w^\ell}=a^{\ell-1}\delta^\ell$

Backpropagation and SGD Manual

$$w^k \to w'^k = w^k - \eta \frac{\partial C}{\partial w^k} \qquad b'^\ell = b^\ell - \eta \frac{\partial C}{\partial b^\ell} \qquad C = \frac{1}{m}\Sigma_x C_x \qquad \vec{\nabla} c = \frac{1}{m}\Sigma_x \vec{\nabla} c_x$$

m *is size of mini batch*

Once you have a way to back propagate the errors, we can apply Gradient Descent.

Given a set of weights and activations, we use Feedforward to calculate:

1. $z^{x,\ell} = w^\ell a^{x,\ell-1} + b^\ell, \qquad a^{x,\ell} = \sigma(z^{x,\ell})$
2. *Calculate error at last layert*;*i.e. output error* :

$$\delta^{x,L} = \nabla_{a^L} C_x \odot \sigma'(z^{x,L})$$

$$\left(s \odot t\right)_j = s_j t_j$$

Backpropagation:

3. *For each* $\ell = L-1, L-2, \ldots\ldots, 1$ *compute* :

$$\delta^{x,\ell} = \left(w^{\ell+1}\right)^T \delta^{x,\ell+1} \odot \sigma'(z^{x,\ell})$$

4. *Gradient Descent*

For each $\ell = L, L-1, \ldots\ldots, 2, 1$ *update weights*

$$w^\ell \to w^\ell - \frac{\eta}{m}\Sigma_x \delta^{x,\ell}$$

Backpropagation and SGD

$$w^k \to w'^k = w^k - \eta \frac{\partial C}{\partial w^k} \qquad b'^\ell = b^\ell - \eta \frac{\partial C}{\partial b^\ell} \qquad C = \frac{1}{m}\Sigma_x C_x \qquad \vec{\nabla} c = \frac{1}{m}\Sigma_x \vec{\nabla} c_x$$

m *is size of mini batch*

Once you have a way to back propagate the errors, we can apply Gradient Descent.

Given a set of weights and activations, we use Feedforward to calculate:

1. $z^{x,\ell} = w^\ell a^{x,\ell-1} + b^\ell, \qquad a^{x,\ell} = \sigma(z^{x,\ell})$
2. *Calculate error at last layert*; *i.e. output error* :

$$\delta^{x,L} = \nabla_{a^L} C_x \odot \sigma'(z^{x,L})$$

$$\left(s \odot t\right)_j = s_j t_j$$

Backpropagation:

3. *For each* $\ell = L-1, L-2, \ldots\ldots, 1$ *compute* :

$$\delta^{x,\ell} = \left(w^{\ell+1}\right)^T \delta^{x,\ell+1} \odot \sigma'(z^{x,\ell})$$

4. *Gradient Descent*

For each $\ell = L, L-1, \ldots\ldots, 2, 1$ *update weights*

$$w^\ell \to w^\ell - \frac{\eta}{m}\Sigma_x \delta^{x,\ell}$$