Moderne Methoden der Datenanalyse:

- Confidence in the NN decision -

- Advanced NN structures: CNNs -

21.7.2023
Jan Kieseler

This is a very rich topic, with enough content for whole courses.
Please consider the following teasers

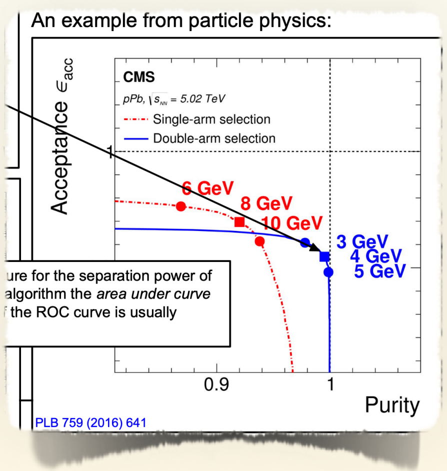
Recap

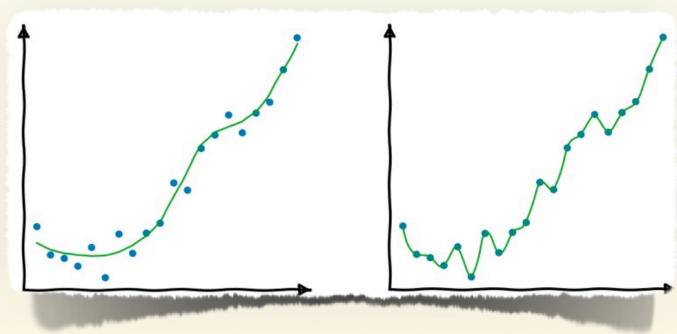
- Feed-forward neural networks can be trained to be powerful classifiers

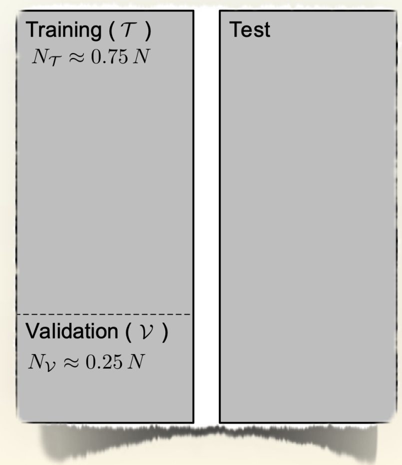
- The training of a NN is subject to many parameter choices
  - Learning rates, regularisation, stopping time

- It is crucial to have well-defined training and test datasets

- It is crucial to define success metrics







- We can determine **that** a NN works well and investigate the output

$$\sigma_{t\bar{t}} = 803 \text{ pb}$$

$$\sigma_{t\bar{t}} = 803 \ \pm 2 \text{ (stat.)} \pm 25 \text{ (syst.)} \pm 20 \text{ (lumi.) pb}$$

What about uncertainties on NN?

- Some terminology from Machine Learning

	Aleatoric uncertainties	**Epistemic** uncertainties
Machine learning	Inherent due to statistical variance	Due to insufficient knowledge of laboratory conditions
	$\approx$	$\approx$
Physics	**Statistical** uncertainties	**Systematic** uncertainties

- This is a hot topic in machine learning

Aleatoric uncertainties

- Reminder: a DNN training consists of dataset + architecture + loss function + minimisation

Where are statistical processes in the MLP training?

Aleatoric uncertainties

- Reminder: a DNN training consists of
  dataset + architecture + loss function + minimisation

Where are statistical processes
in the MLP training?

- Random initialisation of weights and biases

- Random choice of mini batches

- Stochastic minimisation procedures

- Random distinction of training, (test), and validation sample

- The whole sample is sampled from the ground truth

Estimation of aleatoric uncertainties: some teasers

Estimation of aleatoric uncertainties: some teasers

Deep Ensembles

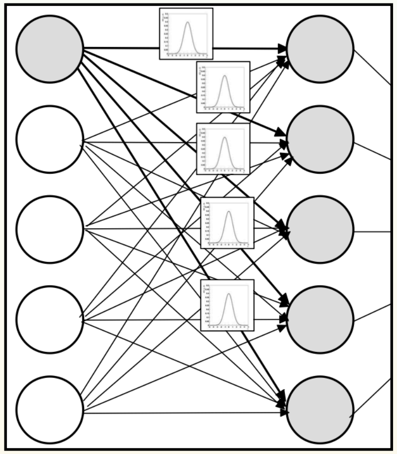
- Initialise identical NNs with varying random seeds and check the distribution of outcomes
- Obvious frequentist approach

Estimation of aleatoric uncertainties: some teasers

Deep Ensembles

- Initialise identical NNs with varying random seeds and check the distribution of outcomes
- Obvious frequentist approach

Bayesian methods



$$\omega \rightarrow p(\omega \,|\, \hat{y}(x))$$

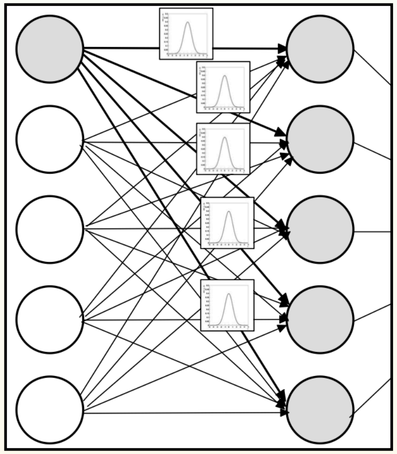
- Learns probability distribution over *possible* neural networks
- Won't be covered here
- Resources and tutorial e.g. [arxiv:2007.06823]

Estimation of aleatoric uncertainties: some teasers

Deep Ensembles

- Initialise identical NNs with varying random seeds and check the distribution of outcomes
- Obvious frequentist approach

Bayesian methods



$$\omega \rightarrow p(\omega \,|\, \hat{y}(x))$$

- Learns probability distribution over *possible* neural networks
- Won't be covered here
- Resources and tutorial e.g. [arxiv:2007.06823]

Dropout

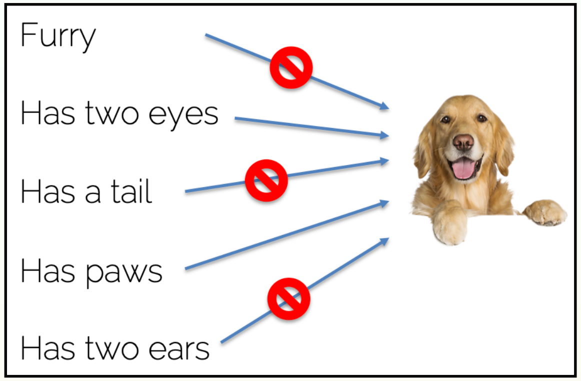
- Next slide

arXiv:1506.02142, >6k citations

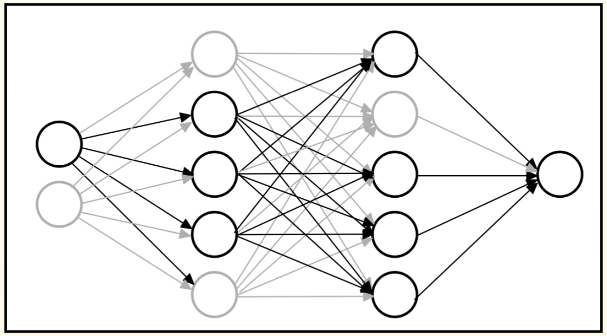
Dropout to estimate uncertainty

- Full proof too much for this lecture

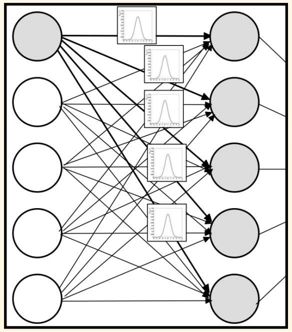
- Dropout during training time forces the network to create redundant representations



Sample

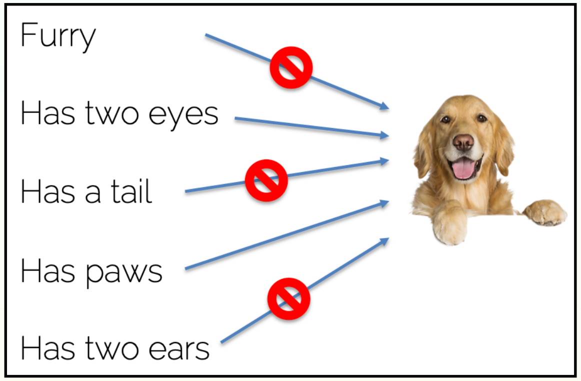


≈ arXiv:1506.02142

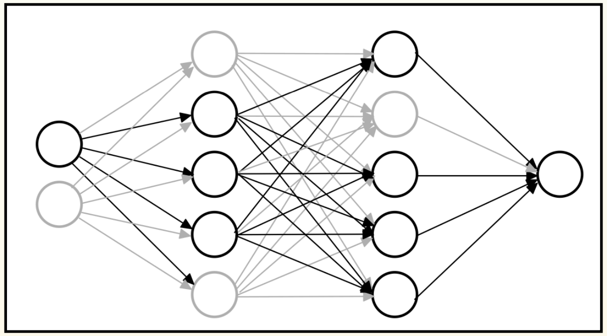


Dropout to estimate uncertainty

- Full proof too much for this lecture
- Dropout during training time forces the network to create redundant representations
- Dropout during inference/test time (MC) samples from these redundant (but all different!) representations

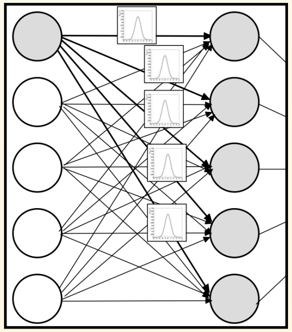


Sample



≈

arXiv:1506.02142



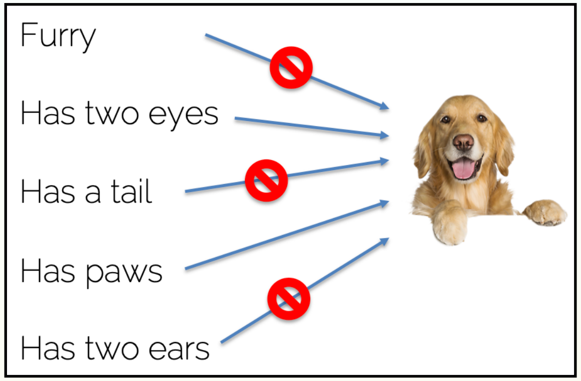
Dropout to estimate uncertainty

- Full proof too much for this lecture

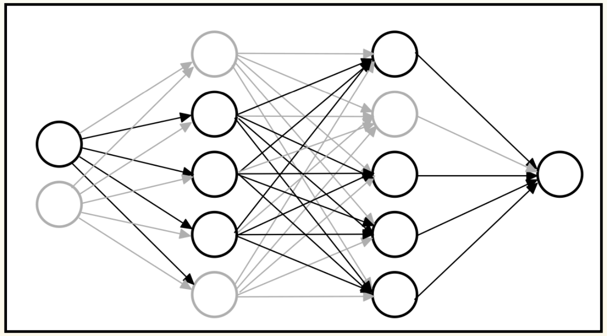
- Dropout during training time forces the network to create redundant representations

- Dropout during inference/test time (MC) samples from these redundant (but all different!) representations

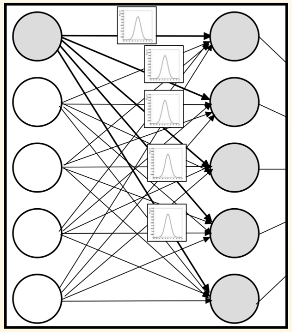
- If dropout is placed before **every** MLP layer in the DNN, this sampling approximates a Bayesian FF NN → uncertainties can be estimated



Sample

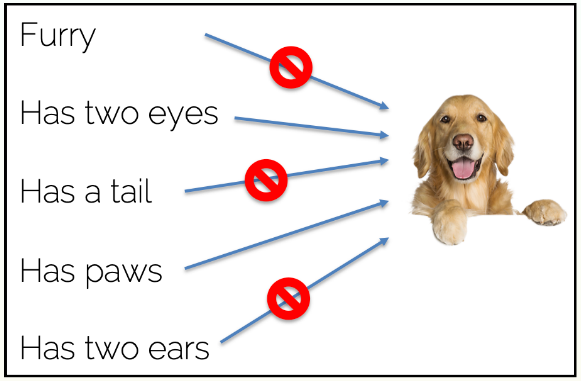


≈ arXiv:1506.02142

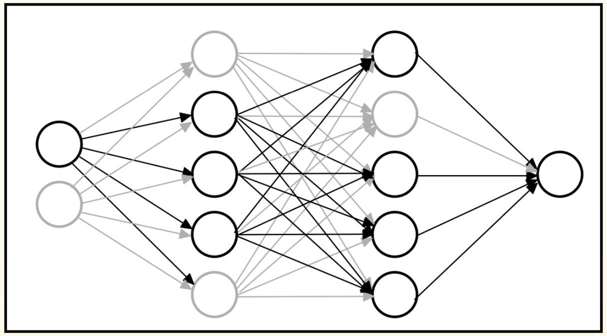


Dropout to estimate uncertainty

- Full proof too much for this lecture
- Dropout during training time forces the network to create redundant representations
- Dropout during inference/test time (MC) samples from these redundant (but all different!) representations
- If dropout is placed before **every** MLP layer in the DNN, this sampling approximates a Bayesian FF NN → uncertainties can be estimated
- Powerful and **easy to use** tool

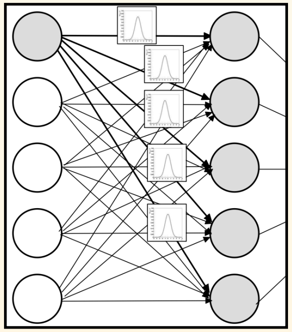


Sample



≈

arXiv:1506.02142



Dropout to estimate uncertainty

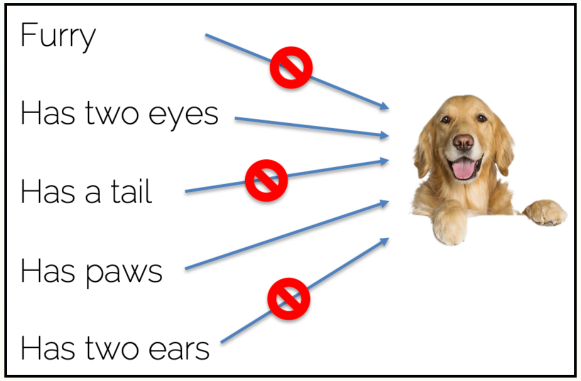
- Full proof too much for this lecture

- Dropout during training time forces the network to create redundant representations

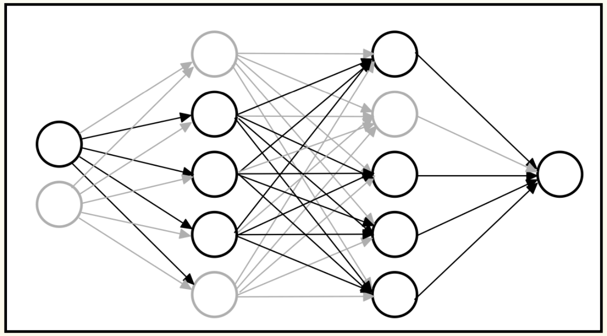
- Dropout during inference/test time (MC) samples from these redundant (but all different!) representations

- If dropout is placed before **every** MLP layer in the DNN, this sampling approximates a Bayesian FF NN → uncertainties can be estimated

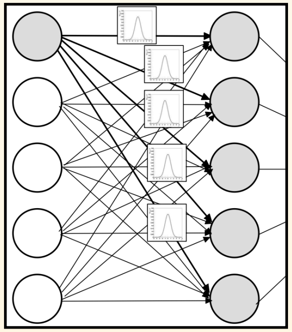
- Powerful and **easy to use** tool
- Can also cover epistemic uncertainties



Sample



≈ arXiv:1506.02142



Epistemic uncertainties

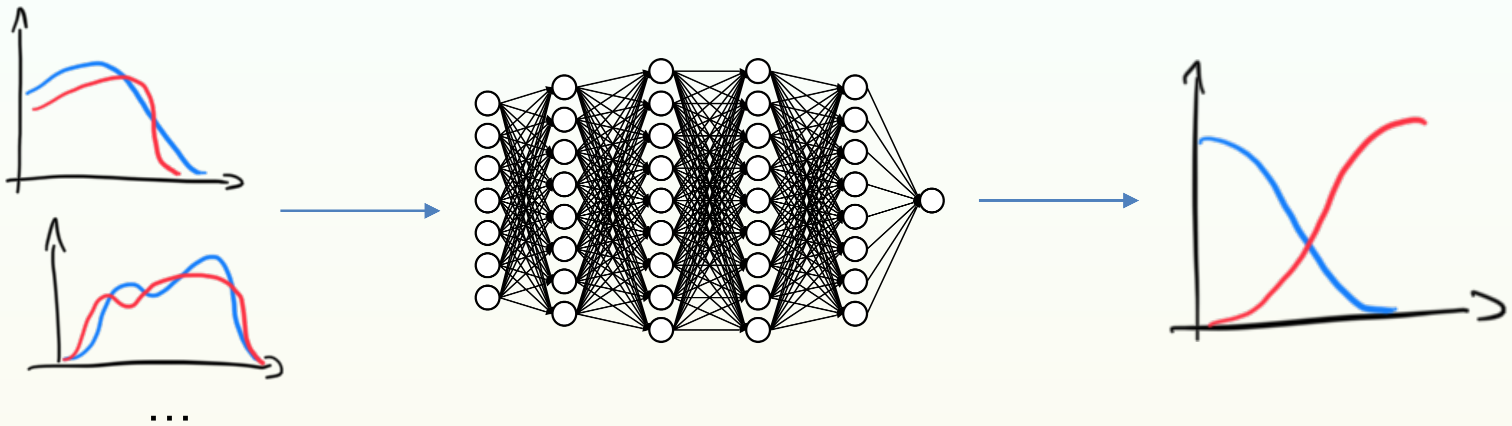
- The model does not have enough degrees of freedom to map the ground truth
  $\rightarrow$ underfitting

- The model systematically maps specific, non-general properties of the training sample
  $\rightarrow$ overfitting

- Differences between training and test sample
  $\rightarrow$ bias

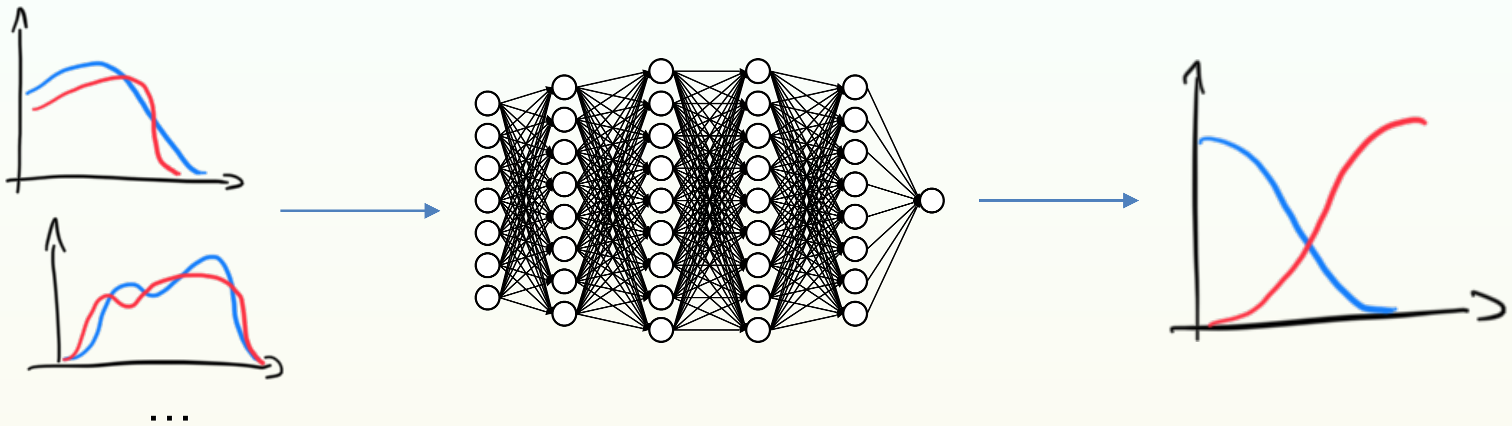
- Much as systematic uncertainties, epistemic uncertainties can be reduced on the basis of additional information

A DNN in an analysis



What additional uncertainties have to be taken into account due to the presence of the NN, to trust our measurement?

A DNN in an analysis

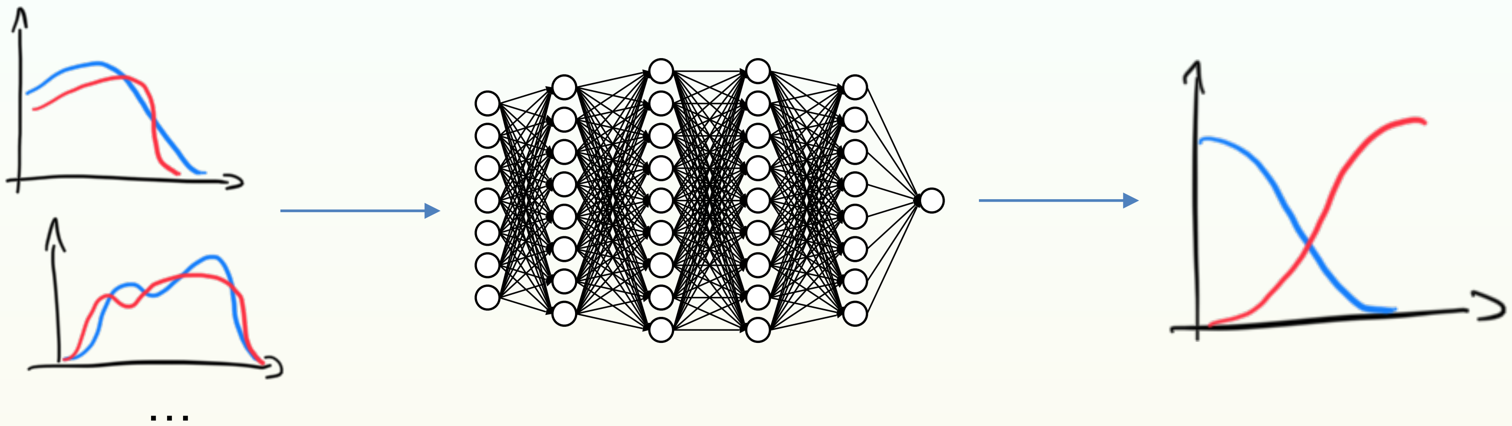


What additional uncertainties have to be taken into account due to the presence of the NN, to trust our measurement?

✓

After training NN acts like a deterministic high-level variable → no additional uncert. required.

A DNN in an analysis

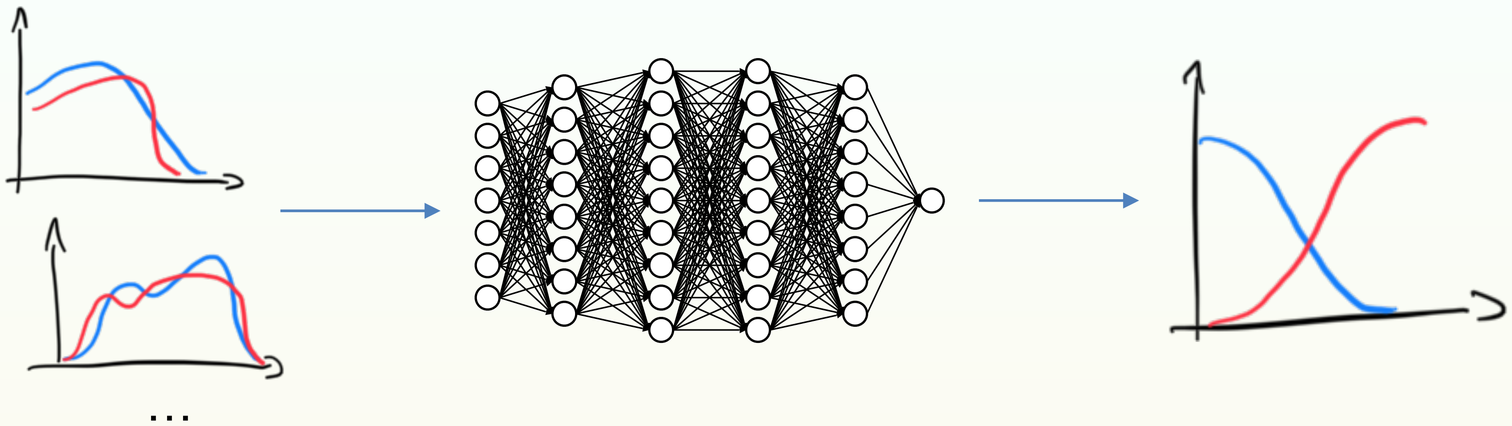


What additional uncertainties have to be taken into account due to the presence of the NN, to trust our measurement?

After training NN acts like a deterministic high-level variable → no additional uncert. required.

Intrinsic (stat.) uncertainties of NN training only of importance for reproducibility of the training.

A DNN in an analysis



What additional uncertainties have to be taken into account due to the presence of the NN, to trust our measurement?

✓

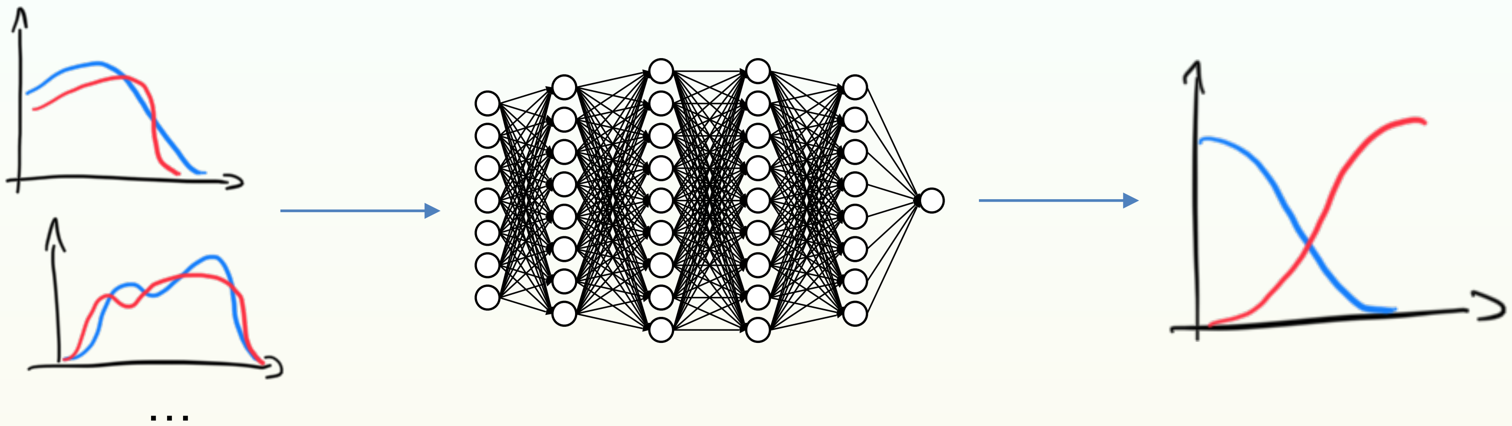
After training NN acts like a deterministic high-level variable → no additional uncert. required.

✓

Intrinsic (stat.) uncertainties of NN training only of importance for reproducibility of the training.

The NN is 'just' a function $y = \Phi_\omega(x)$

A DNN in an analysis



What additional uncertainties have to be taken into account due to the presence of the NN, to trust our measurement?

✓

After training NN acts like a deterministic high-level variable
→ no additional uncert. required.

!!!!

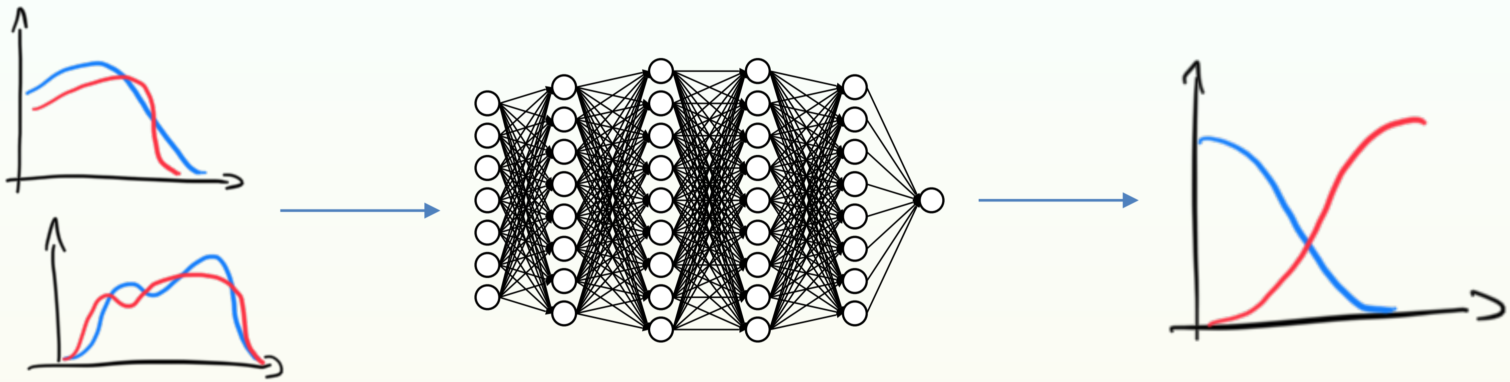
NN exploits input variable space much deeper than e.g. cut-based selection requirements
→ thorough control of input space required.

✓

Intrinsic (stat.) uncertainties of NN training only of importance for reproducibility of the training.

The NN is 'just' a function $y = \Phi_{\omega}(x)$

A DNN in an analysis



…

What additional uncertainties have to be taken into account due to the presence of the NN, to trust our measurement?

✓

After training NN acts like a deterministic high-level variable
→ no additional uncert. required.

!!!!

NN exploits input variable space much deeper than e.g. cut-based selection requirements
→ thorough control of input space required.

✓

Intrinsic (stat.) uncertainties of NN training only of importance for reproducibility of the training.

The NN is ‘just’ a function $y = \Phi_{\omega}(x)$

Is the function acting *in the right way* on the **well understood** inputs?

Reminder:

- most measurements compare data to a simulation (hypothesis test)
- The DNN is trained on simulation

$y = \Phi_\omega(x)$,

x not understood, possibly mismodelled, no idea what the DNN does

$y = \Phi_\omega(x)$,

x well understood,

correlations between x well understood, the DNN captures the 'right' features

Unboxing the NN

- Exploit Taylor expansion of $\hat{y}_j$ in x with fixed ω and b after training to identify the x_i with with largest influence on $\hat{y}_j$:

$$< t_\alpha > = \frac{1}{N}\sum_{k=1}^{N}\left|t_\alpha(\{x^{(k)}\})\right|$$

N: sample size
t_α: Taylor coefficient labeled by α

- Introduce *generalised features* of the input feature space:
$\alpha = x_1,\ x_2,\ \ldots$ 1st order feature
$\alpha = x_1x_1,\ x_1x_2,\ x_2x_1,\ \ldots$ 2nd order feature
…

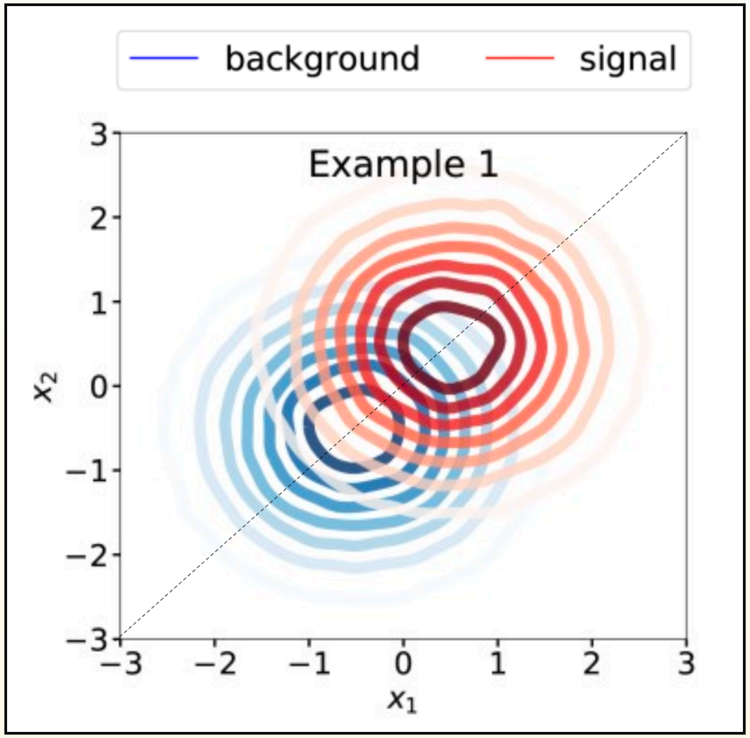
arXiv:1803.08782

Meaning of generalised features

- 1st oder feature:
  - Physical location of of feature/marginal distributions, e.g. signal at small x_1, signal at large x_1

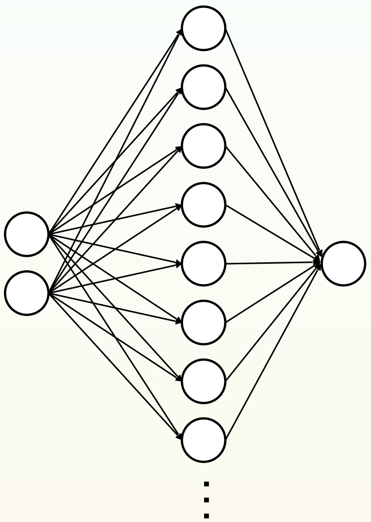
- 2nd order feature:
  - $x_i x_j$: Linear correlations across two features
  - $x_i x_i$: “Self-correlations”, i.e. curvature of $\hat{y}_j$ w.r.t. x_i (since it is the 2nd derivative)

Simple example



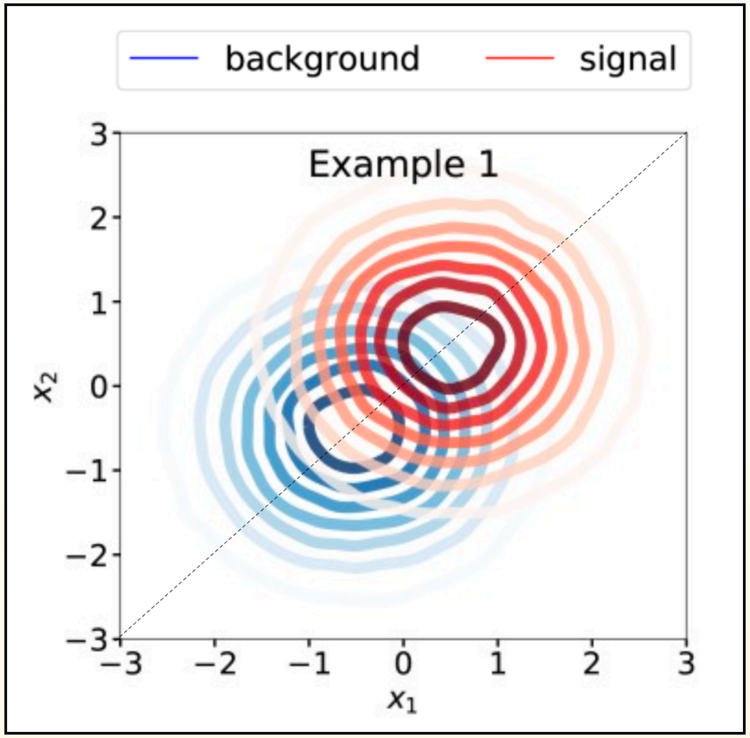
- Two input features x_1 and x_2
- Binary classification
- Signal and background samples from normal distributions: $\mu_S = (0.5, 0.5)$ and $\mu_{BG} = (-0.5, -0.5)$
- The task is symmetric

DNN and training



- One hidden layer with 100 nodes
  - tanh activation
  - sigmoid on output
- Loss function: binary cross entropy
- Minimiser: Adam, lr=1e-4
- Mini-batch training with early stopping after 30 epochs (more later)

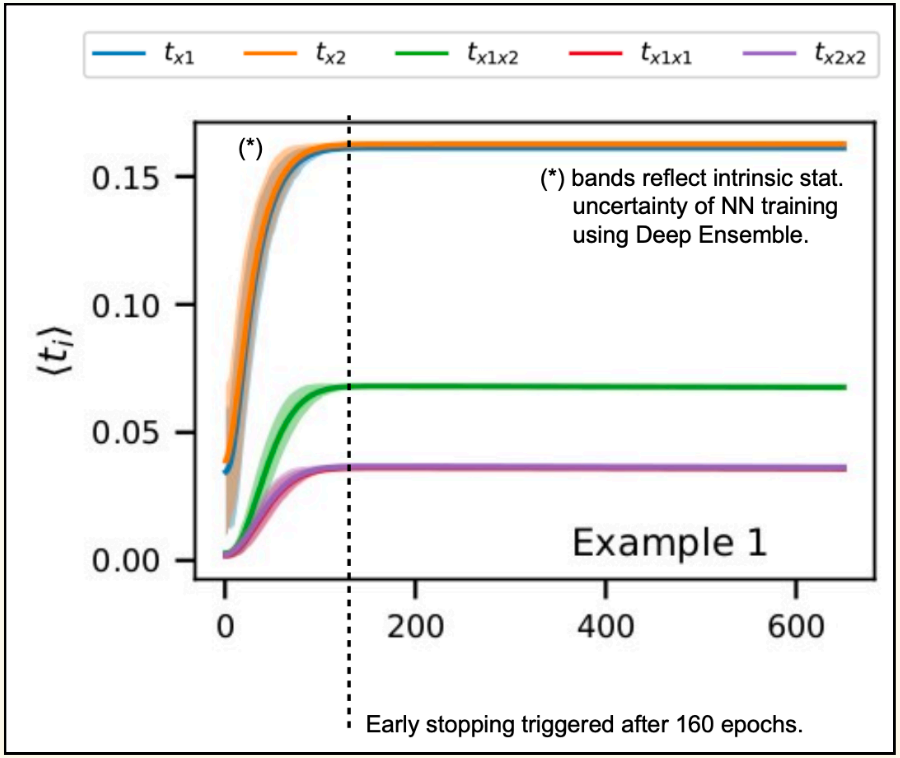
Taylor coefficients



α	Value after training
x_1	0.16
x_2	0.16
$x_1 x_2$	0.07
$x_1 x_1$	0.04
$x_2 x_2$	0.04

- x_1 and x_2 found to be most influential (distinction of S and BG by location)
- $x_i x_j$ indicate that correlations play a role (for S and BG x_1 and x_2 are linearly correlated)

Taylor coefficients as a function of time



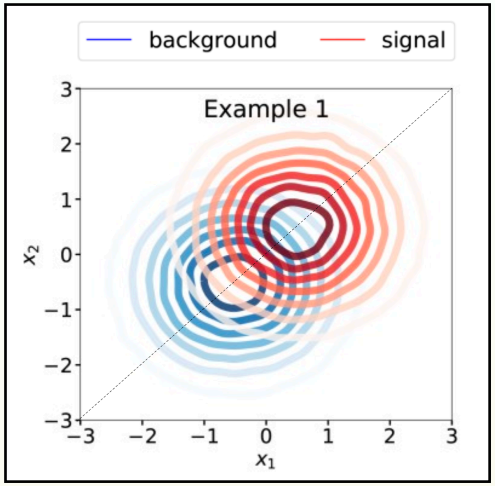
α	Value after training
x_1	0.16
x_2	0.16
$x_1 x_2$	0.07
$x_1 x_1$	0.04
$x_2 x_2$	0.04

- x_1 and x_2 found to be most influential (distinction of S and BG by location)
- $x_i x_j$ indicate that correlations play a role (for S and BG x_1 and x_2 are linearly correlated)
- After convergence $< t_\alpha >$ are stable and **reproducible** even though the NNs themselves are different

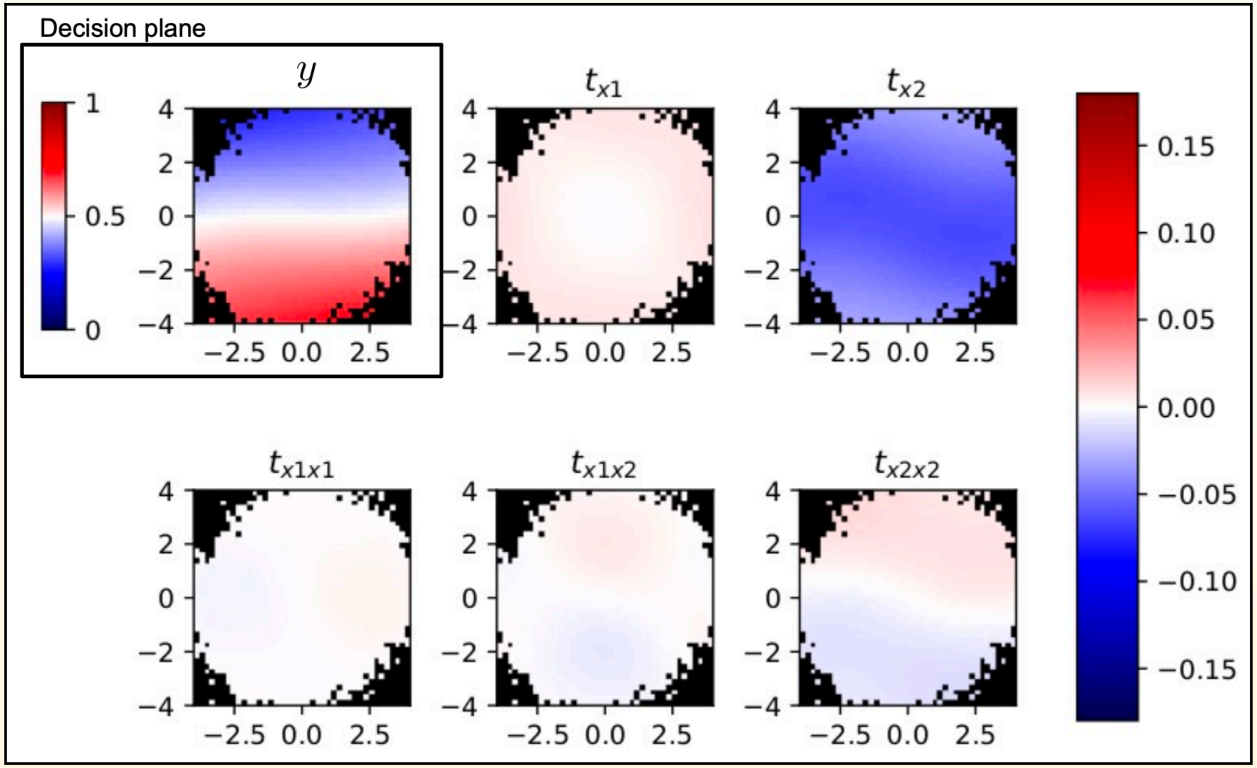
- Allows monitoring of the training process

Considering sample space

- Monitor what phase space regions the NN identifies as important and at what point in the training it starts to investigate them:



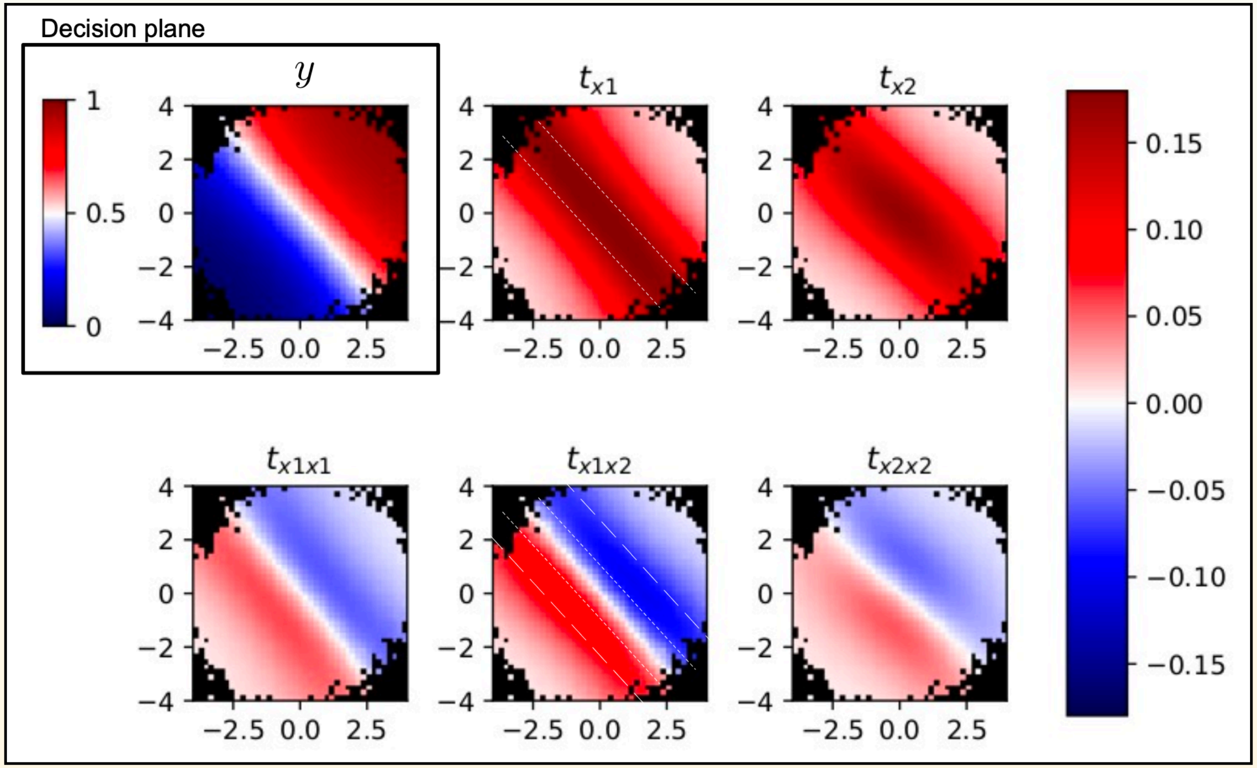
- After epoch 1:

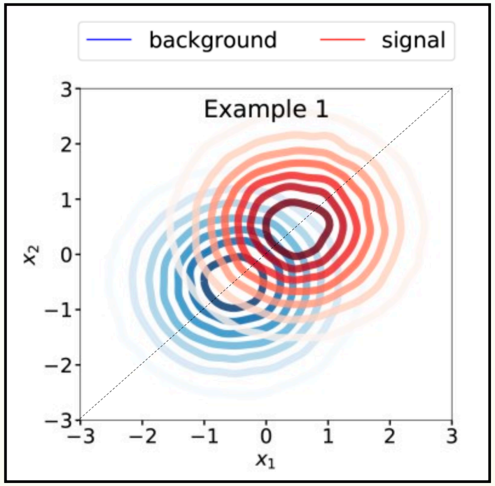


Considering sample space

- Monitor what phase space regions the NN identifies as important and at what point in the training it starts to investigate them:

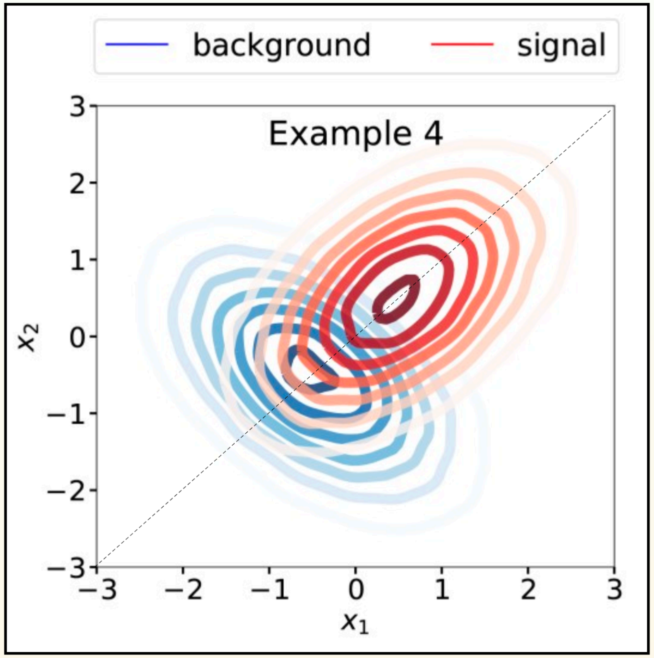
- After epoch 50:

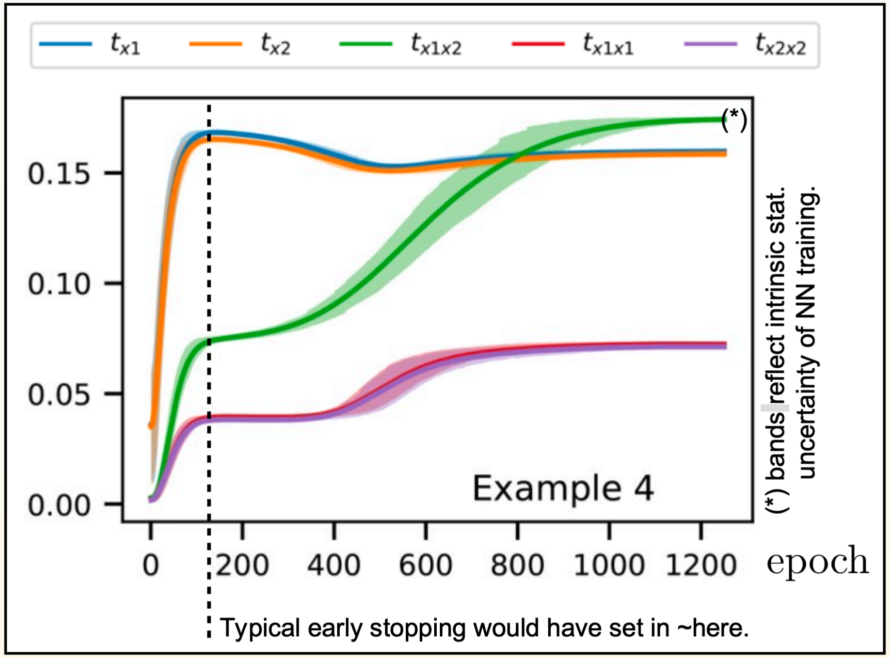




A slightly more complex task

- Adding different linear correlations to S and BG



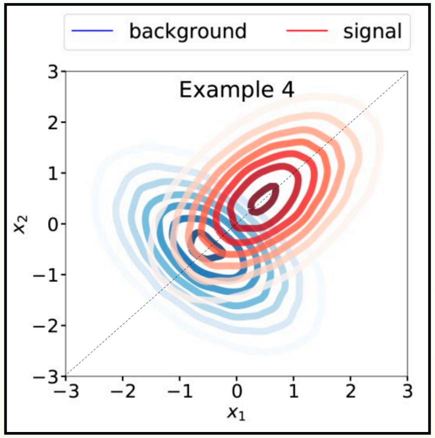


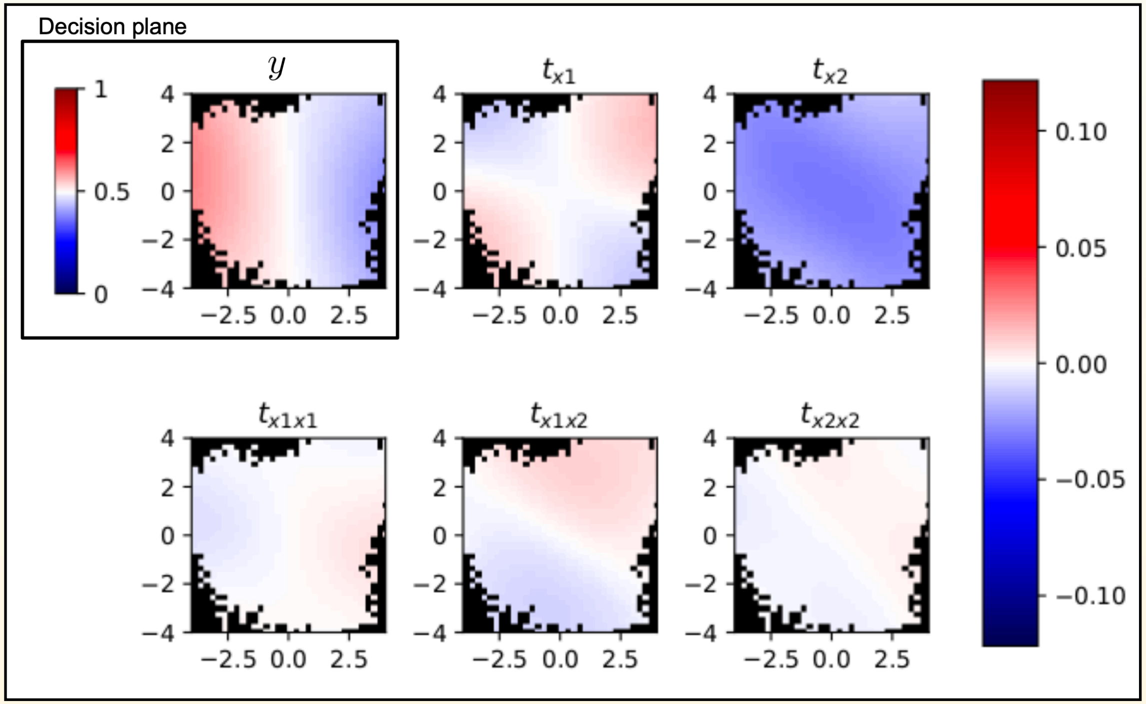
- Steep initial learning curve

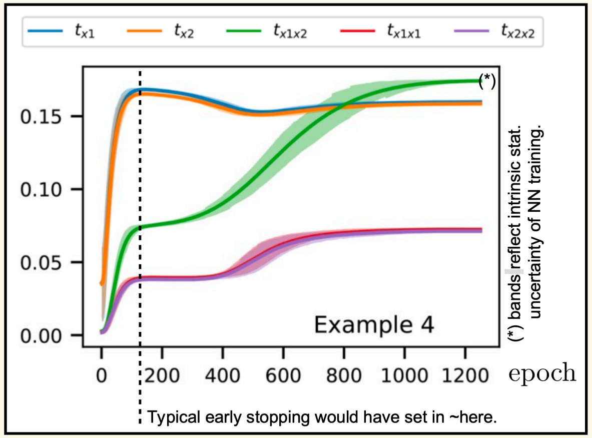
- Only later, the additional importance of the correlation $x_1 x_2$ is identified as being important (improving area-under-ROC by 10%)

Watch the NN learn

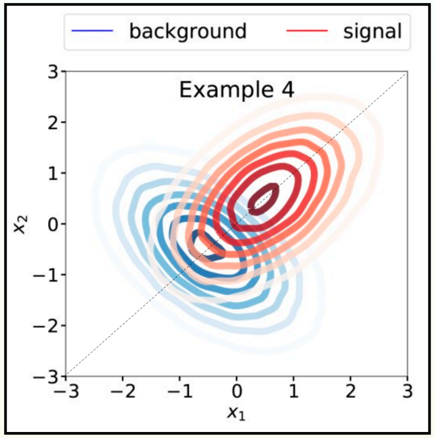
- After epoch 1



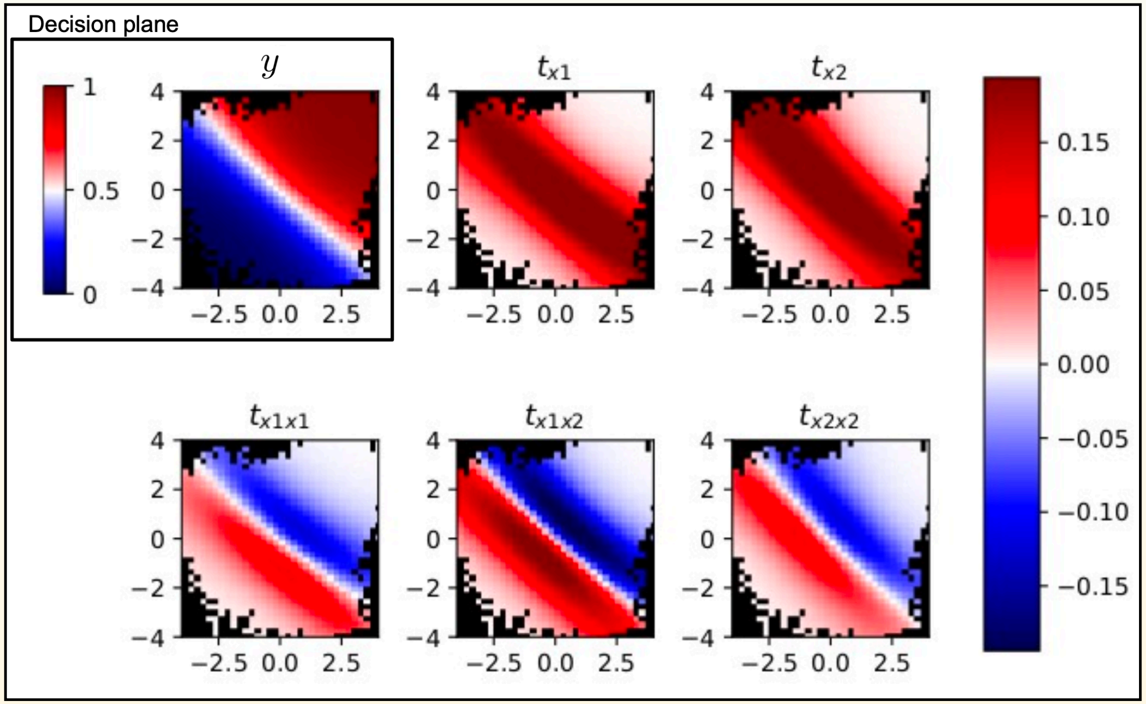


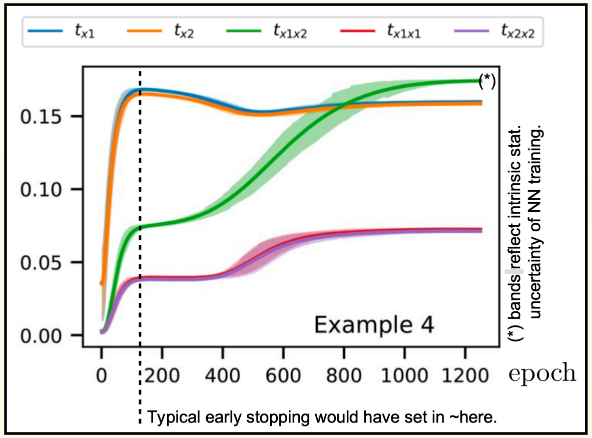


Watch the NN learn



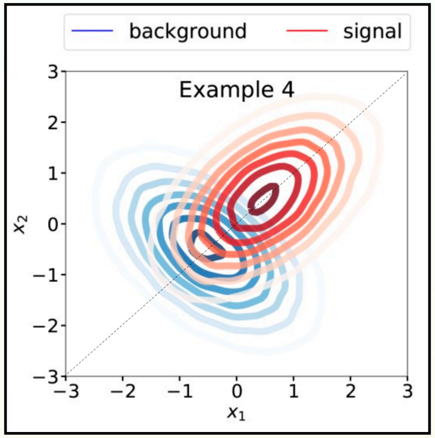
- After epoch 100

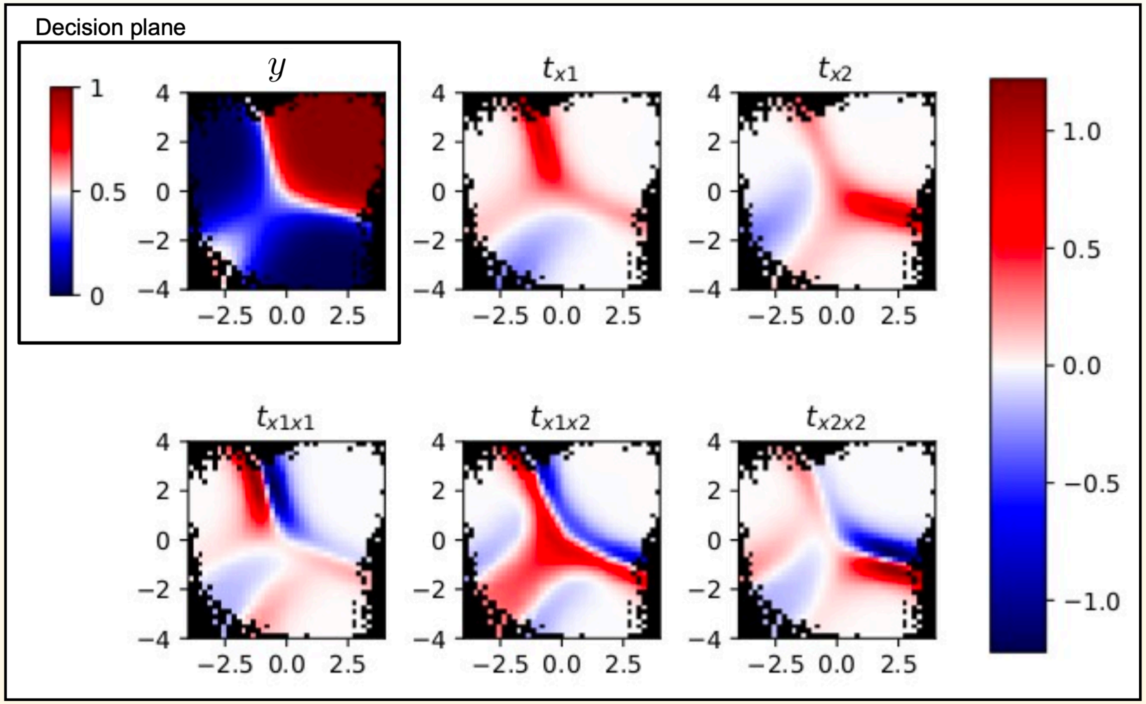


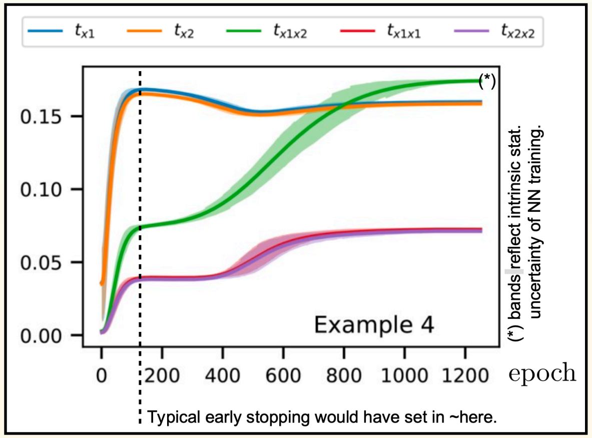


Watch the NN learn

- After epoch 2000
- Learned that S and BG are separated in feature space. Difference in correlations missed until epoch ~1000



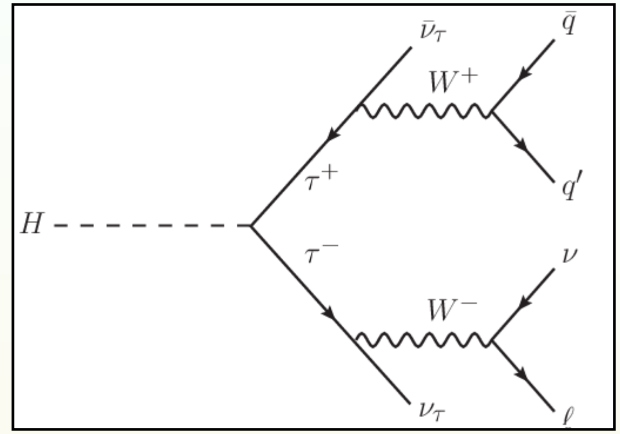


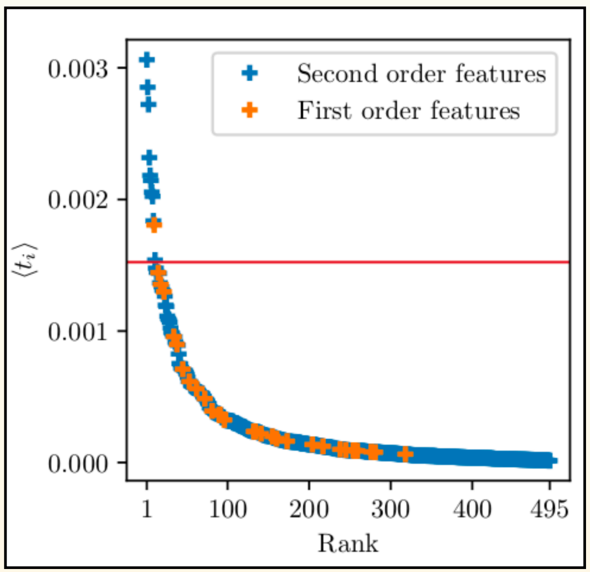


- Powerful tool to check convergence; in general be careful with early stopping

A real-life example

- ATLAS data from the ML challenge of 2014 [1]
- $< t_\alpha >$ used for an unambiguous importance ranking



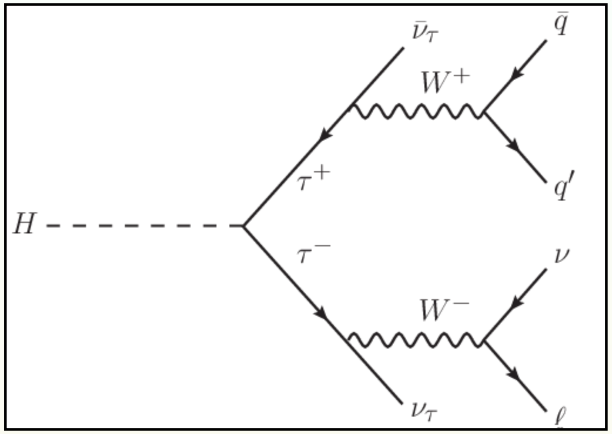


Rank	Variables		$\langle t_i \rangle \times 10^{-3}$
1	DER_mass_vis	DER_pt_ratio_lep_tau	3.061
2	DER_deltar_tau_lep	DER_mass_vis	2.852
3	DER_mass_vis	PRI_lep_pt	2.722
4	DER_deltar_tau_lep	DER_pt_ratio_lep_tau	2.318
5	DER_pt_ratio_lep_tau	PRI_lep_pt	2.182
6	DER_mass_vis	DER_mass_vis	2.144
7	DER_mass_MMC	DER_mass_vis	2.056
8	DER_deltar_tau_lep	PRI_lep_pt	2.023
9	DER_mass_jet_jet	DER_mass_vis	1.837
10	DER_mass_vis		1.806
11	DER_mass_MMC	DER_pt_ratio_lep_tau	1.539
12	DER_mass_transverse_met_lep	DER_mass_vis	1.478
13	DER_mass_jet_jet	DER_pt_ratio_lep_tau	1.447
14	DER_deltar_tau_lep	DER_mass_MMC	1.446
15	DER_pt_ratio_lep_tau		1.443
16	DER_mass_MMC	PRI_lep_pt	1.438
17	DER_deltar_tau_lep	DER_mass_jet_jet	1.366
18	DER_deltar_tau_lep		1.355
19	DER_mass_jet_jet	PRI_lep_pt	1.337
20	DER_mass_MMC	DER_mass_MMC	1.312
	...		

PhD Stefan Wunsch ETP
https://cds.cern.ch/record/2751100?ln=de

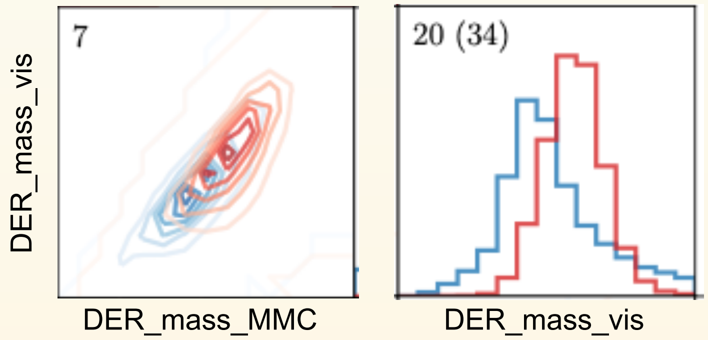
[1] https://higgsml.lal.in2p3.fr/documentation/

Conclusions of the Higgs boson ML challenge



Rank	Variables		$\langle t_i \rangle \times 10^{-3}$
1	DER_mass_vis	DER_pt_ratio_lep_tau	3.061
2	DER_deltar_tau_lep	DER_mass_vis	2.852
3	DER_mass_vis	PRI_lep_pt	2.722
4	DER_deltar_tau_lep	DER_pt_ratio_lep_tau	2.318
5	DER_pt_ratio_lep_tau	PRI_lep_pt	2.182
6	DER_mass_vis	DER_mass_vis	2.144
7	DER_mass_MMC	DER_mass_vis	2.056
8	DER_deltar_tau_lep	PRI_lep_pt	2.023
9	DER_mass_jet_jet	DER_mass_vis	1.837
10	DER_mass_vis		1.806
11	DER_mass_MMC	DER_pt_ratio_lep_tau	1.539
12	DER_mass_transverse_met_lep	DER_mass_vis	1.478
13	DER_mass_jet_jet	DER_pt_ratio_lep_tau	1.447
14	DER_deltar_tau_lep	DER_mass_MMC	1.446
15	DER_pt_ratio_lep_tau		1.443
16	DER_mass_MMC	PRI_lep_pt	1.438
17	DER_deltar_tau_lep	DER_mass_jet_jet	1.366
18	DER_deltar_tau_lep		1.355
19	DER_mass_jet_jet	PRI_lep_pt	1.337
20	DER_mass_MMC	DER_mass_MMC	1.312

- Without knowing the physics, the NN has:
  - Identified mvis and MMC mass as important
  - Identified that both peak in S and BG; peaks gets rated high
  - Identified correlations to be more important than 1st order features



- The DNN has indeed identified the relevant physics features, not spurious outliers

Intermediate summary

- In machine learning, we distinguish between aleatoric and epistemic uncertainties ($\approx$ statistical and systematic uncertainties)
- There are methods to estimate both, and a lot of research refining them
- For most bread-and-butter applications in physics, the DNN can be taken as a deterministic function
- What matters often most is to understand and model the inputs well and guarantee a physically meaningful DNN output

Reproducibility

Epistemic

Aleatoric

Correlations

Time for questions

Ranking

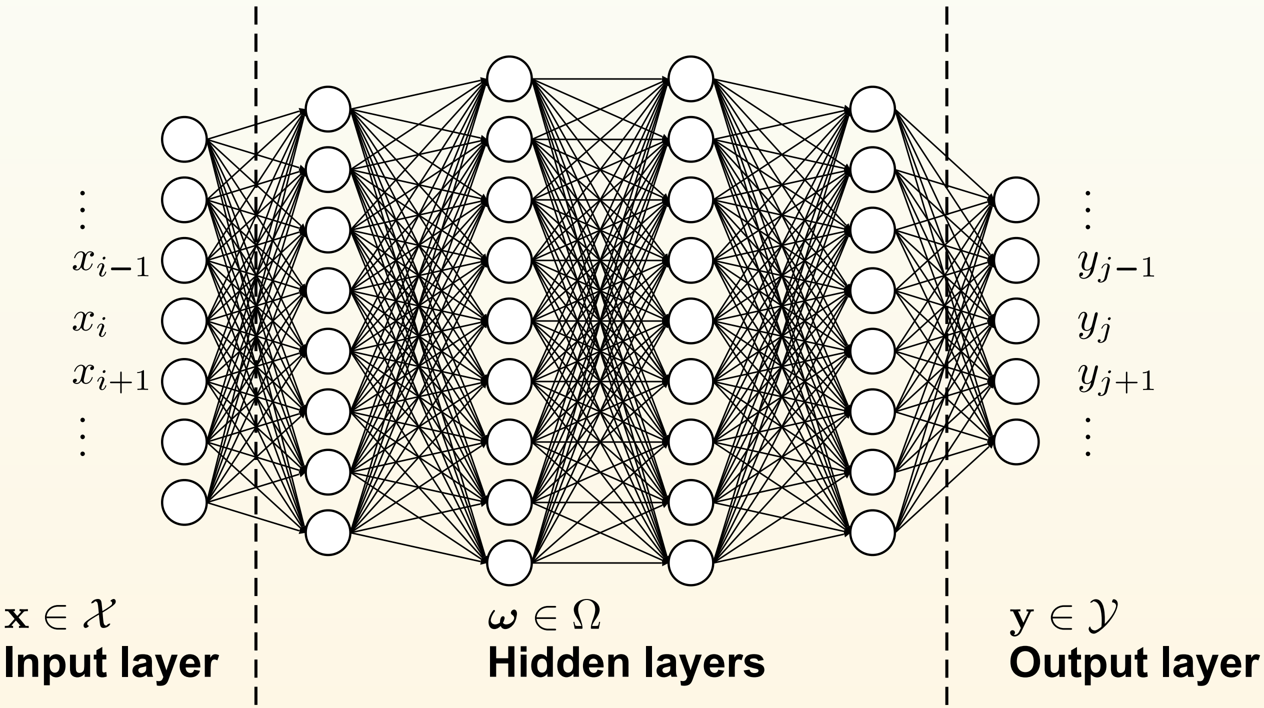
Deterministic

Moderne Methoden der Datenanalyse:

Advanced Neural Network Structures: CNNs

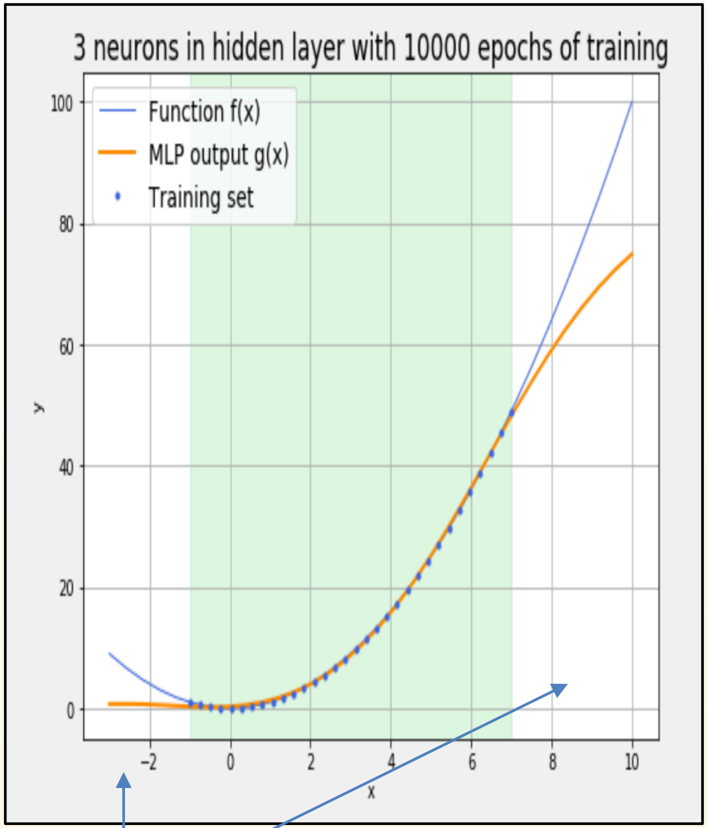
Recap: DNNs and their parameters

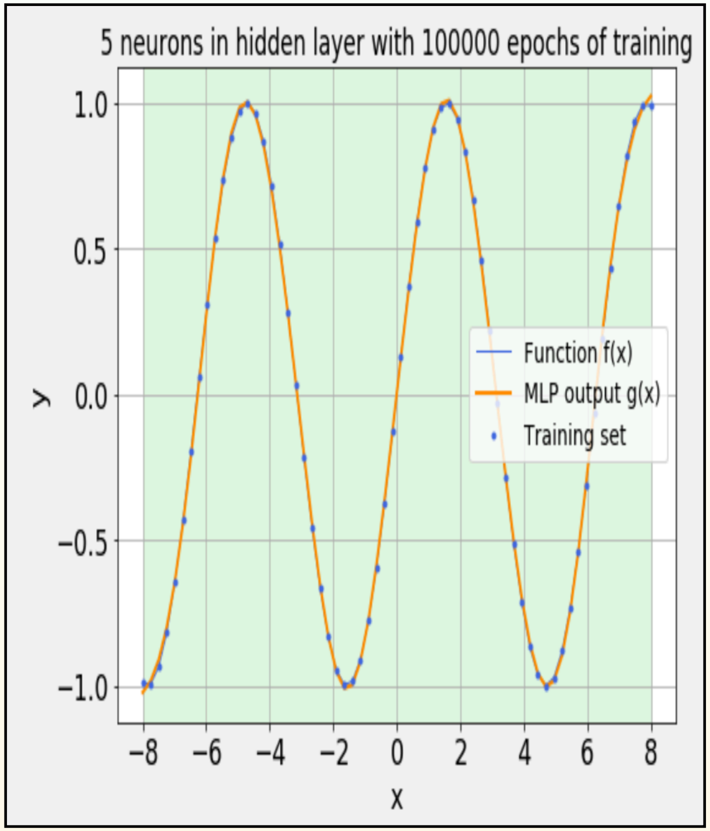
- All nodes of consecutive layers are connected with each other
- Typically an ANN is called “deep” if it has >4 hidden layers
- Referred to as Multi-Layer Perceptron, Feed-Forward NN



DNNs as universal function approximators

- Very simple NN: one hidden layer, one input, one output, tanh activation





https://notebook.community/kit-cel/lecture-examples/mloc/ch3_Deep_Learning/pytorch/function_approximation_with_MLP

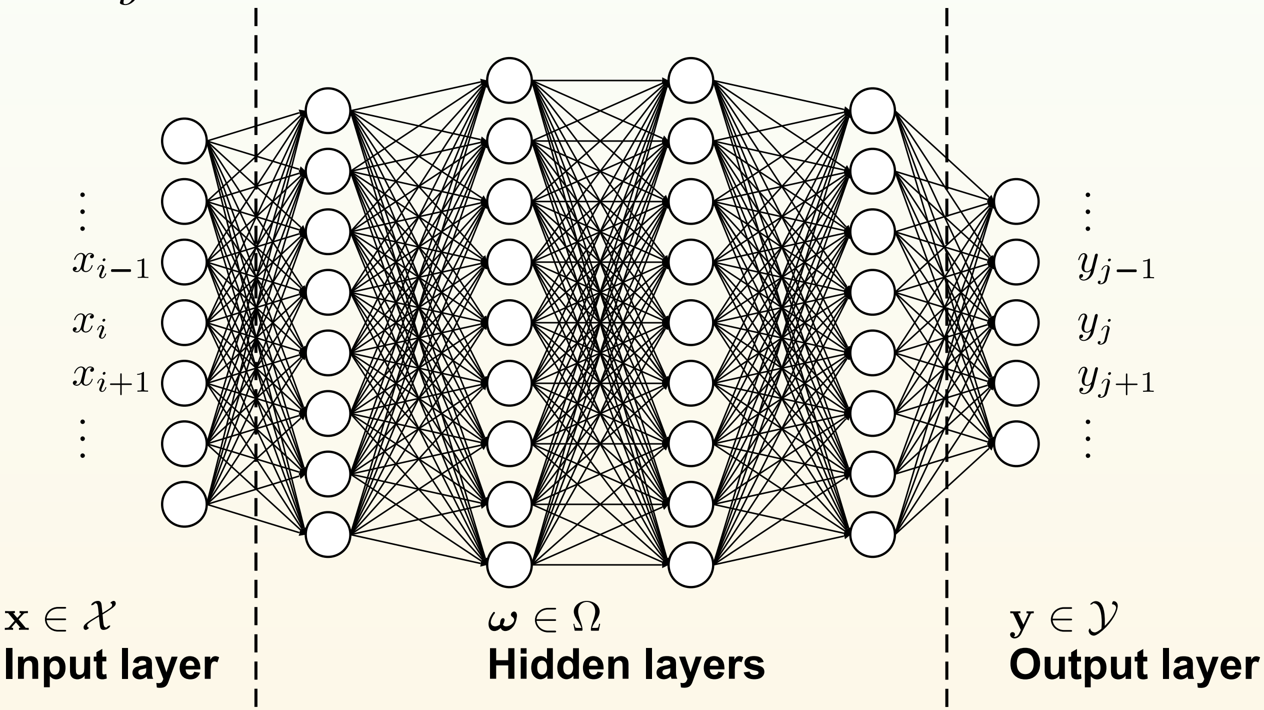
"Out-of-distribution"

- DNNs are universal function approximators, already with very few parameters
  - But beware of extrapolation / out-of-distribution effects
- How many parameters are needed?

Counting parameters

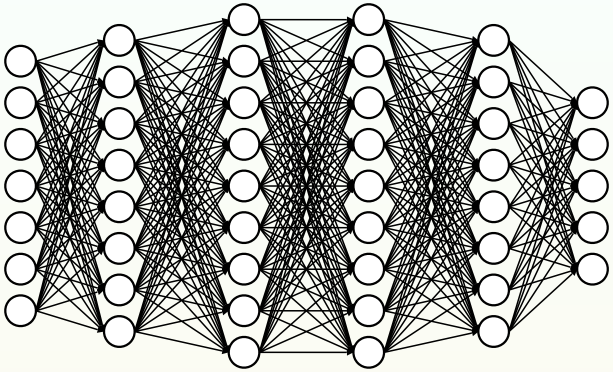
$$N_\omega = \ 7 \times 8 + 8 \times 9 + 9 \times 9 + 9 \times 8 + 8 \times 5 = 321$$

$$N_b \geq \ 8 + 9 + 9 + 8 + 0 = 34$$



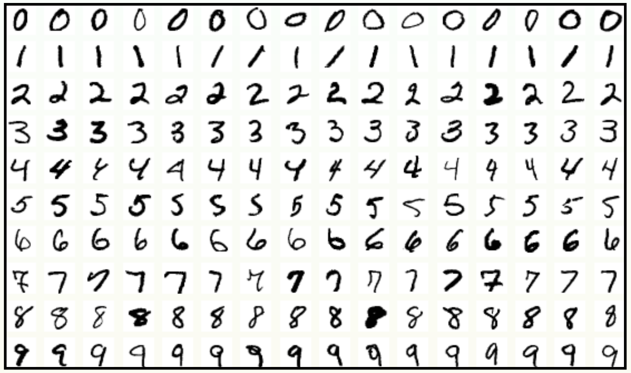
- Typical small MLPs: about 10k - 100k
- ChatGPT: 1.5 Billion
- More free parameters → more expressivity

More parameters → more resources



~10k-100k parameters
Trains in minutes on your laptop
Uses ~10 Wh of electricity

↔



MNIST [L. Deng, IEEE 2012]
60k images

Not an MLP!

OpenAI ChatGPT**4.0**

~1.5Trillion parameters
Trained 6 month
$100M for compute, roughly 10 000 MWh

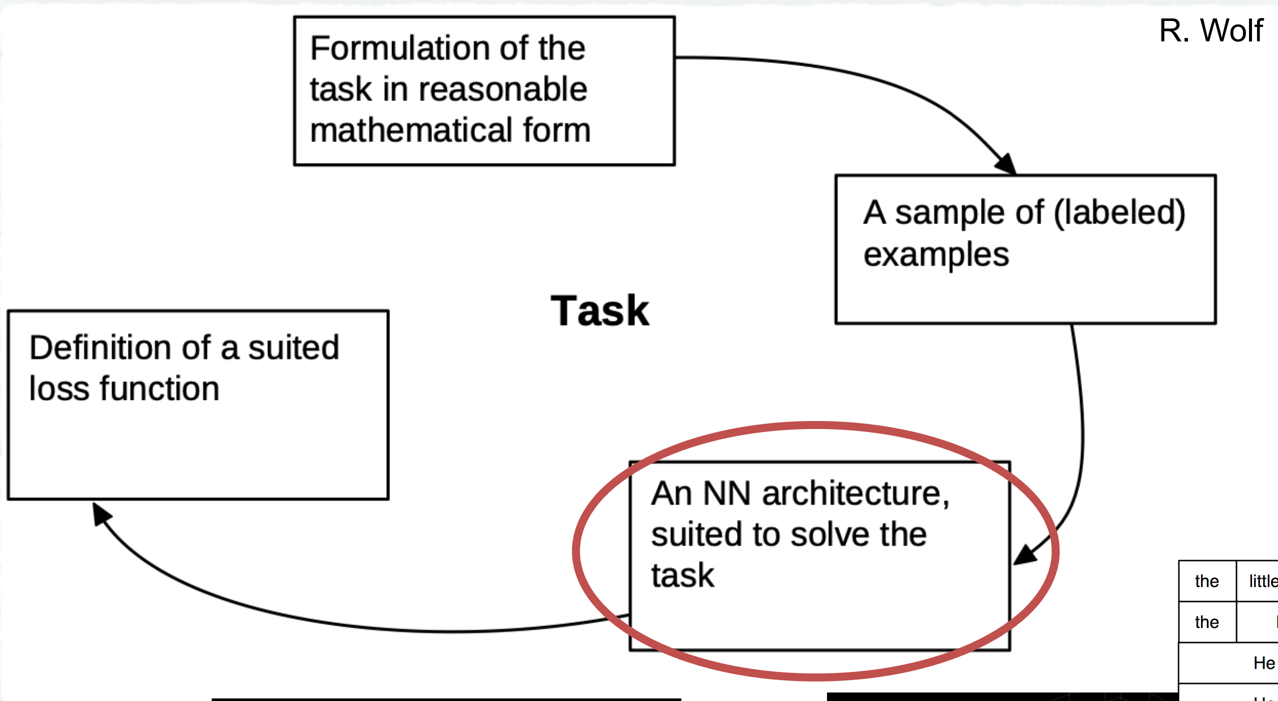
Only estimates, no official numbers

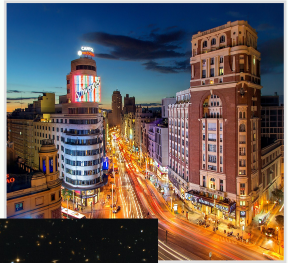
↔

Common Crawl

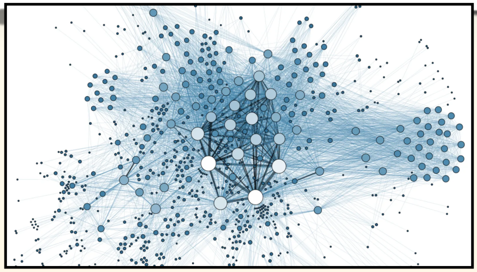
- More parameters:
  - More training data
  - More resources to evaluate
  - Even more resources to train

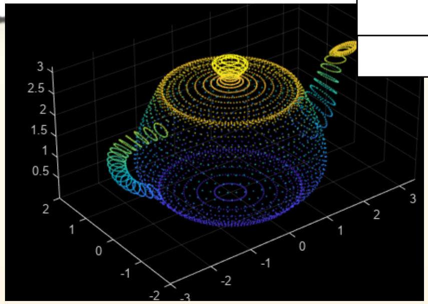
Recap

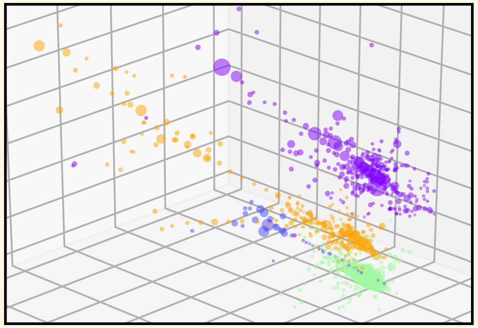




<table>
<tr><td>the</td><td>little</td><td>bear</td><td>saw</td><td>the</td><td>fine</td><td>fat</td><td>trout</td><td>in</td><td>the</td><td>brook</td></tr>
<tr><td>the</td><td colspan="2">bear</td><td>saw</td><td>the</td><td colspan="3">trout</td><td>in</td><td colspan="2">it</td></tr>
<tr><td colspan="3">He</td><td>saw</td><td colspan="4">it</td><td colspan="3">there</td></tr>
<tr><td colspan="3">He</td><td colspan="5">ran</td><td colspan="3">there</td></tr>
<tr><td colspan="3">He</td><td colspan="8">ran</td></tr>
</table>



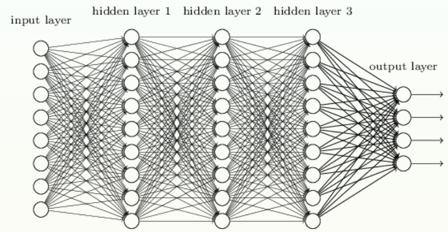


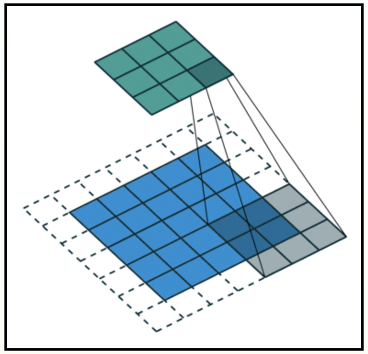


- Architecture needs to fit the desired output ✓
- Architecture needs to fit the **input data**

Main building blocks of architectures

- MLP / Feed forward ✓

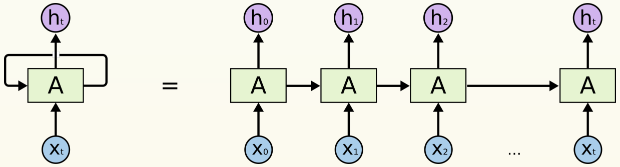




- CNNs

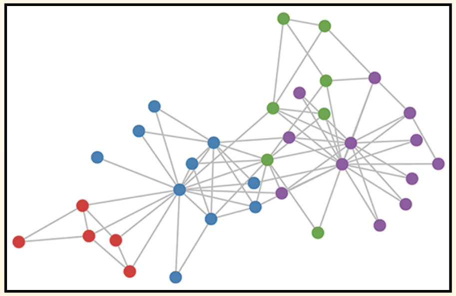
- RNNs

Next time



- Attention

- GNNs



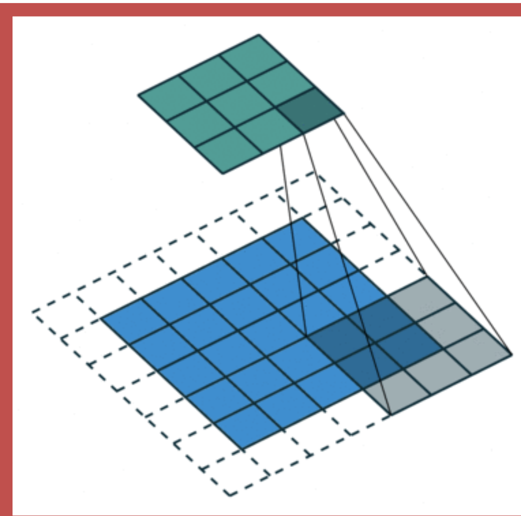
Recap

Formulation of the task in reasonable mathematical form

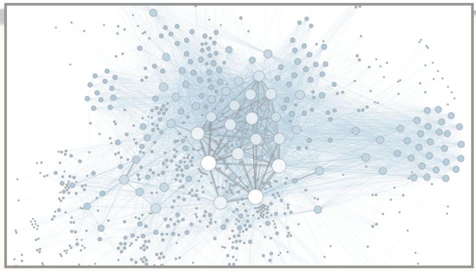
Task

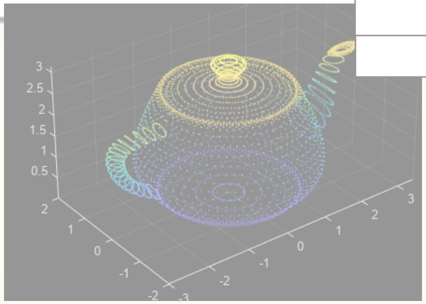
Definition of a suited loss function

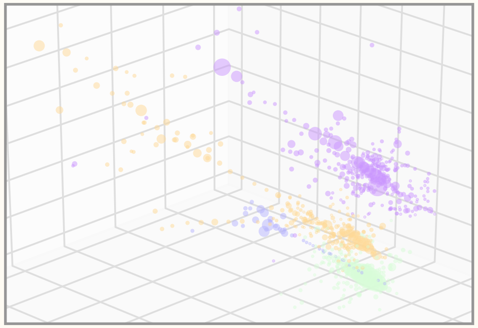
An NN architecture, suited to solve the task



the	little	bear	saw	the	fine	fat	trout	in	the	brook
the	bear		saw	the	trout			in	it	
He			saw	it				there		
He			ran					there		
He			ran							





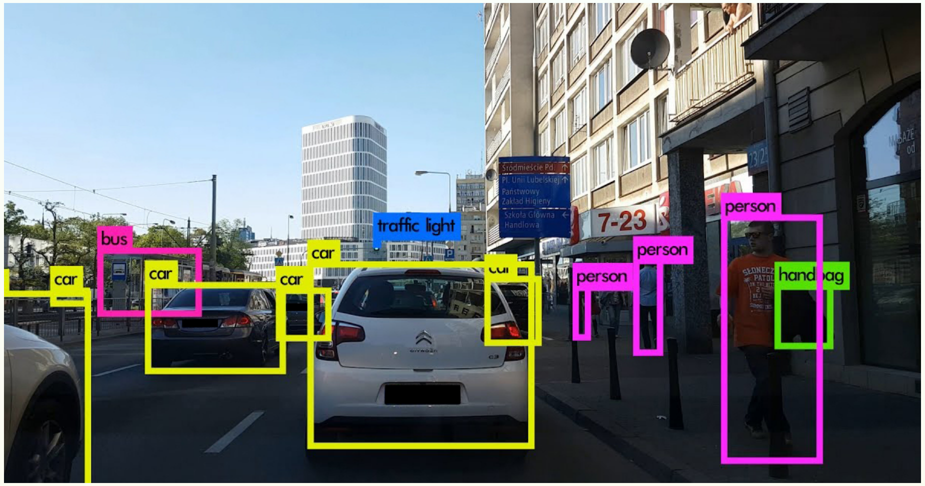


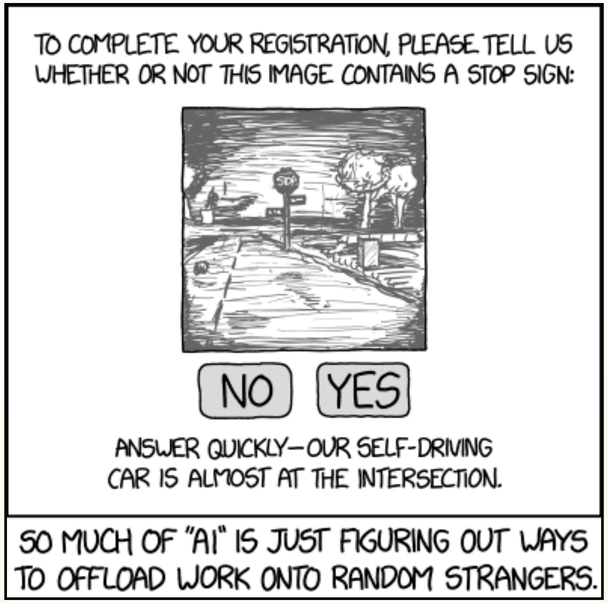
- Architecture needs to fit the desired output ✓
- Architecture needs to fit the **input data**

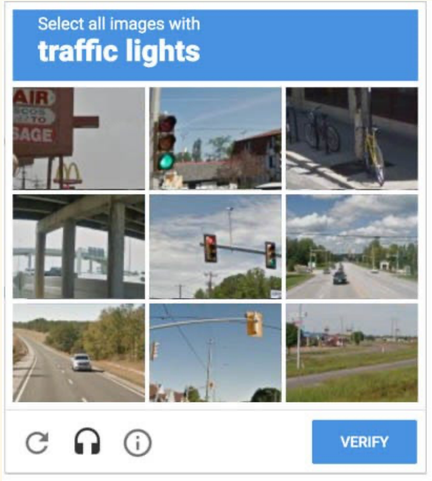
Convolutional Neural Networks

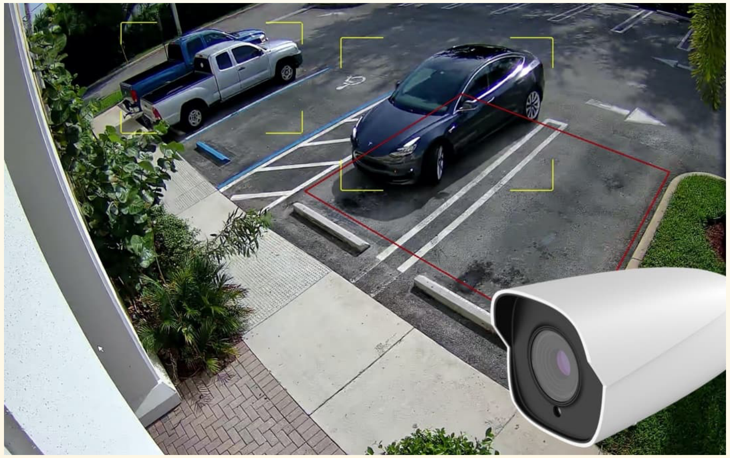
Image-like data

CNNs are everywhere and at the core of computer vision





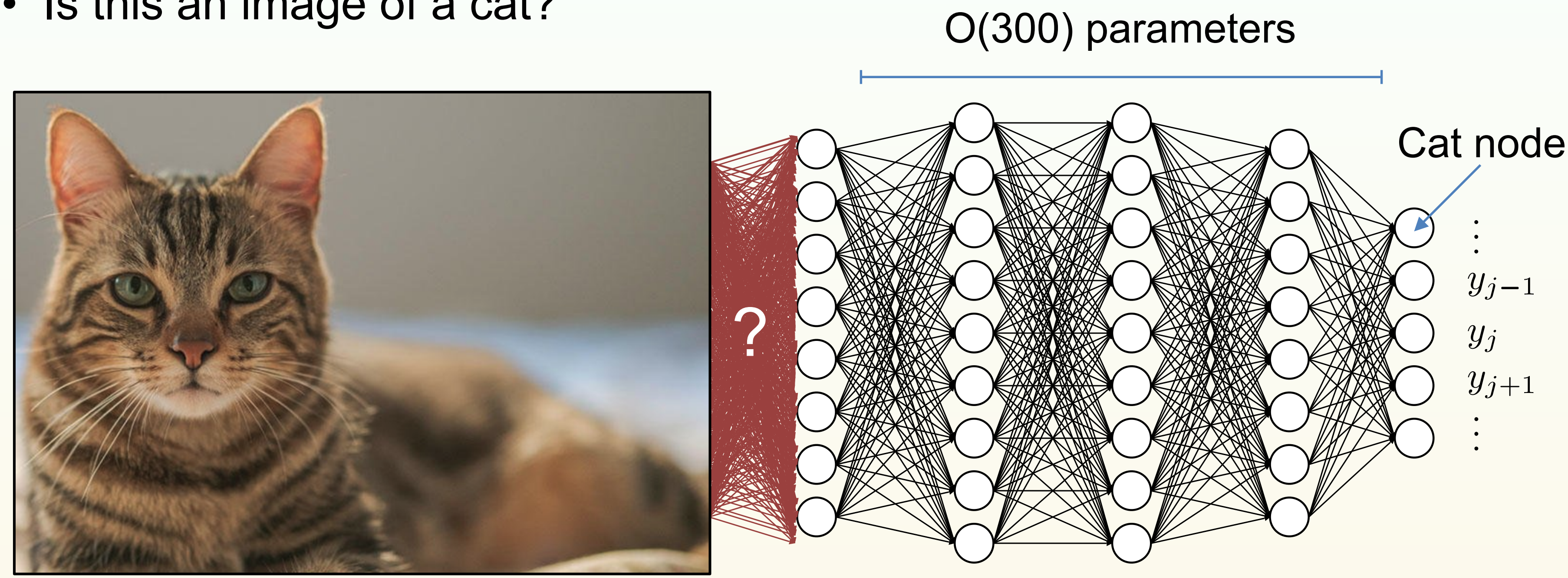




- Self-driving cars
- Surveillance
- Skin cancer detection
- …
- Particle physics

Structure counts

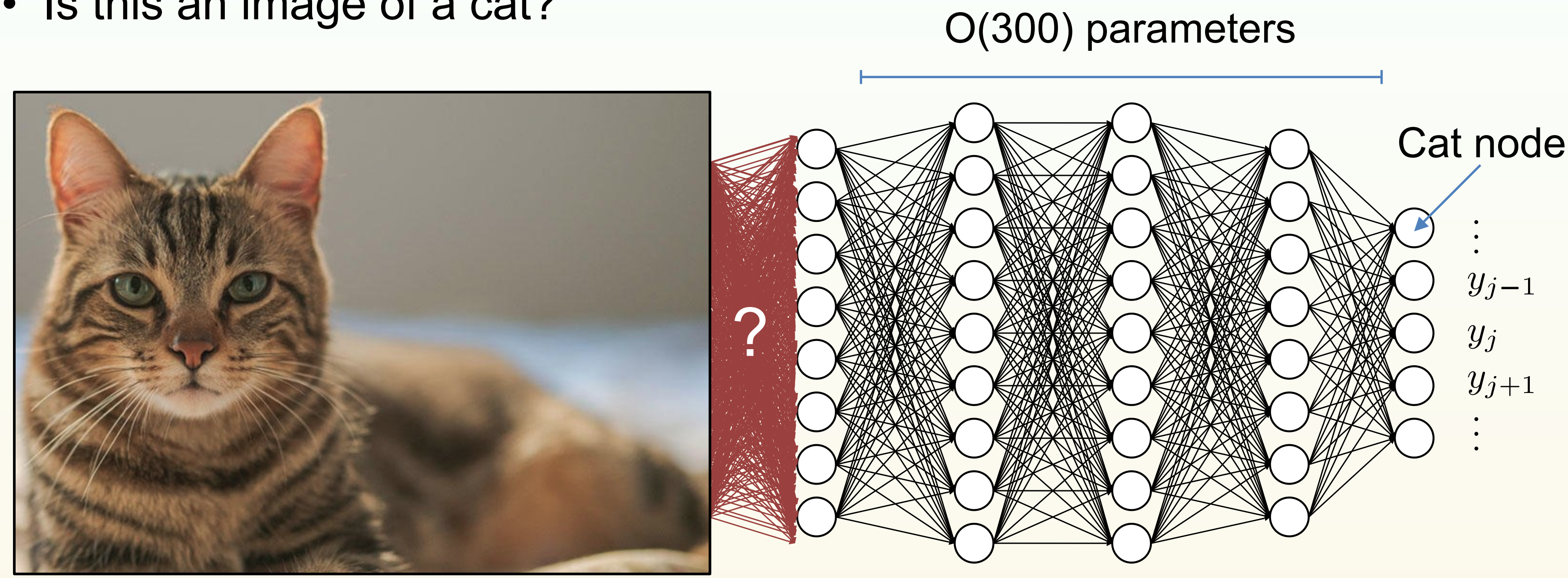
- Is this an image of a cat?



- Typical (phone) cameras 10-50 MP
- How many parameters does the first layer have?

Structure counts

- Is this an image of a cat?



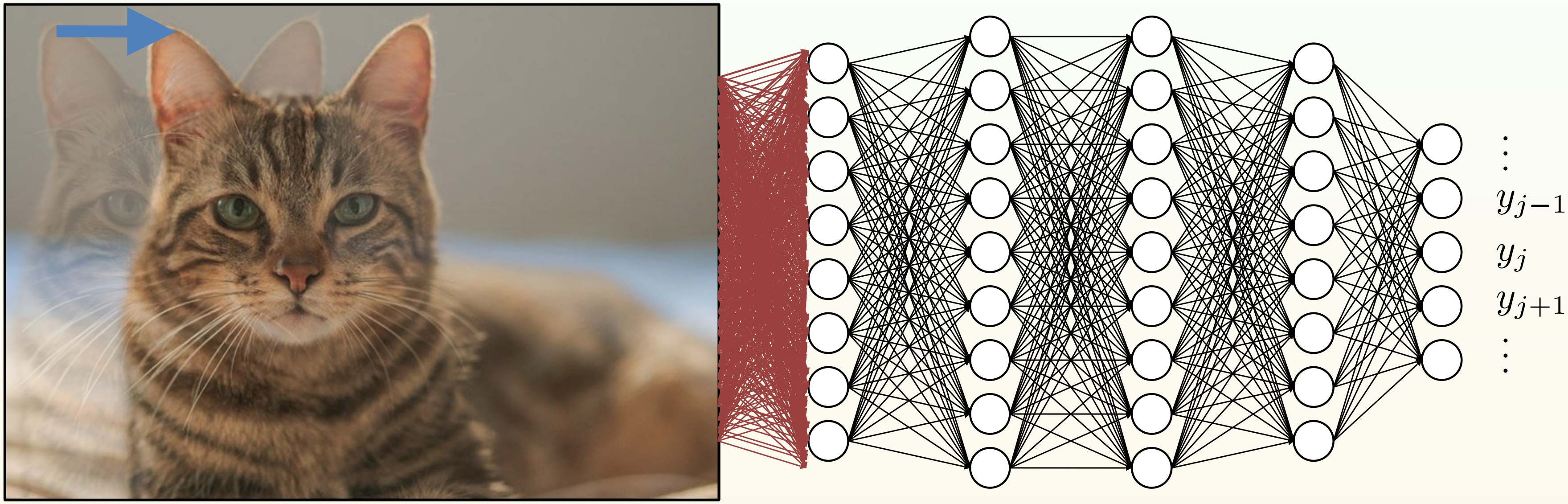
- Typical (phone) cameras 10-50 MP
- How many parameters does the first layer have?

- In this example: **80 - 400 million parameters** in first layer

- Also, this architecture will not perform well

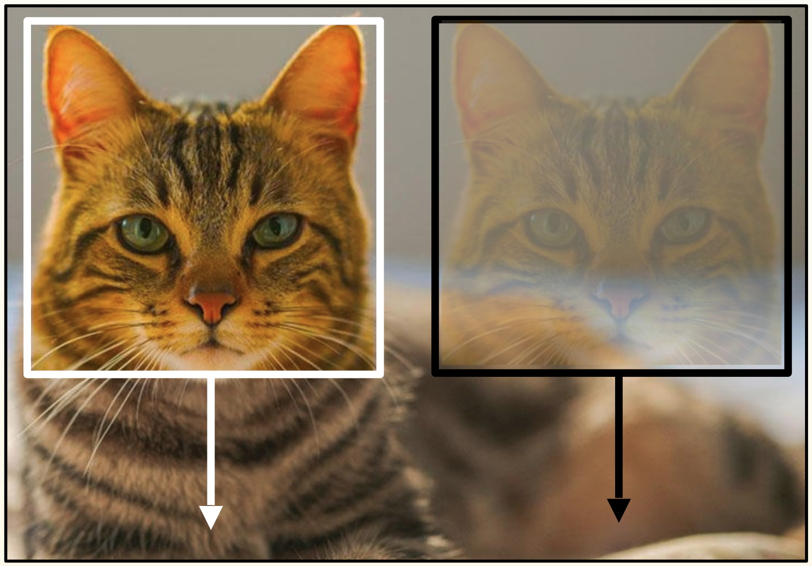
Structure counts

- What if the cat moved?



- Present **entirely different** input to the DNN
- This complexity cannot be captured by as little as 8 nodes
  - Lack of expressivity
- Solution: exploit the **structure** of the data

Introducing filters



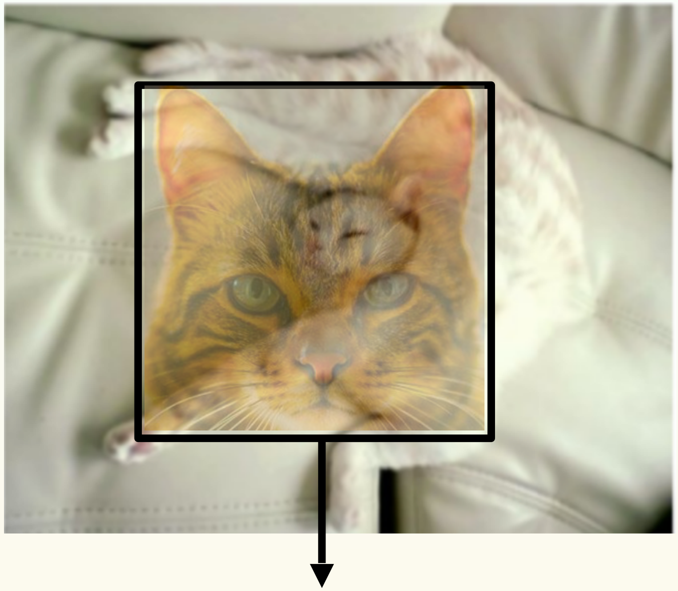
Very cat-like:
Score = 1

Not at all cat-like
Score = 0

- Create a cat-face filter (no ML here)

- Slide it over the image
- Take maximum of all cat scores: image cat score
- We found the cat

Cats come in different shapes



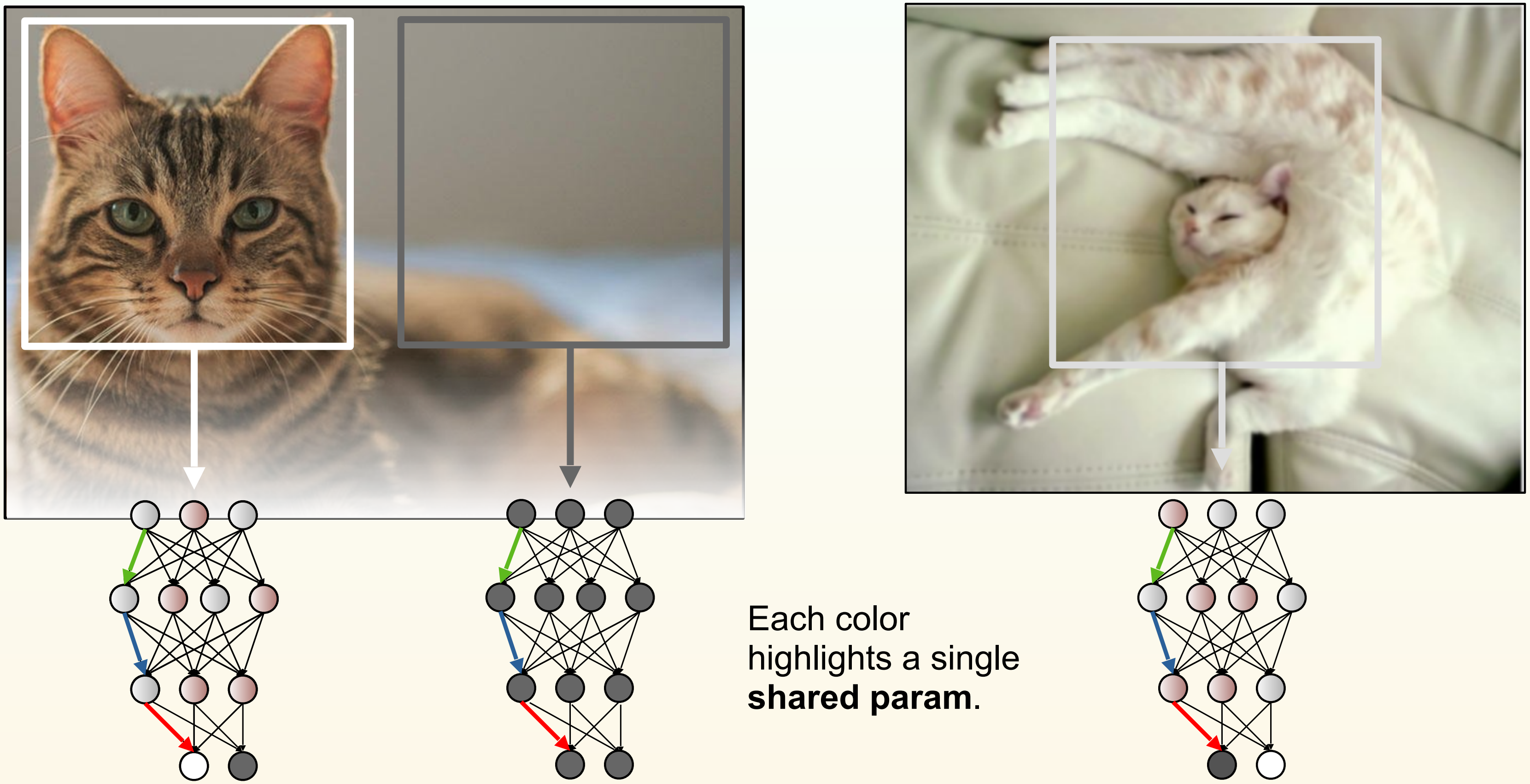
Not a cat

Not a cat

- Many different very complex filters are needed

- Can be solved by
  - **Learning filters from examples**
  - Abstraction

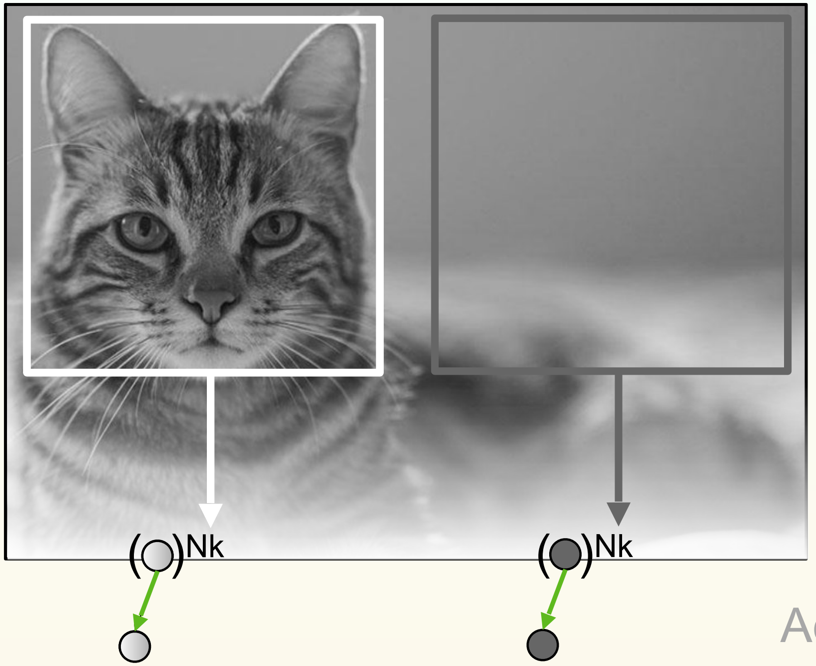
Learning the filters



Each color
highlights a single
shared param.

- Learn (approximations of) different shapes
- Represent them by (combinations of) output nodes

A CNN kernel: step by step



- Inputs x
- For one *channel*:

Learnable bias

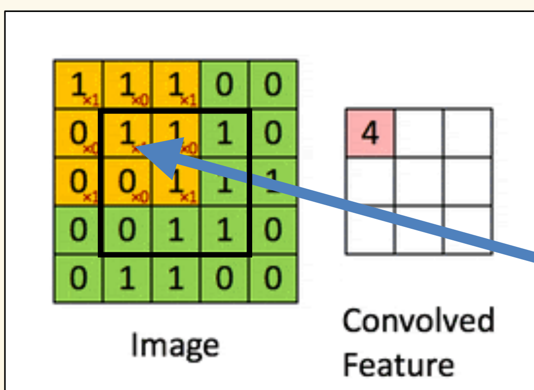
Kernel size

$$y_j = \theta\left(\sum_{i}^{N_k} \omega_i \; x_{I(j,i)} - T\right)$$

Activation function

Learnable weights:
Relative position to j

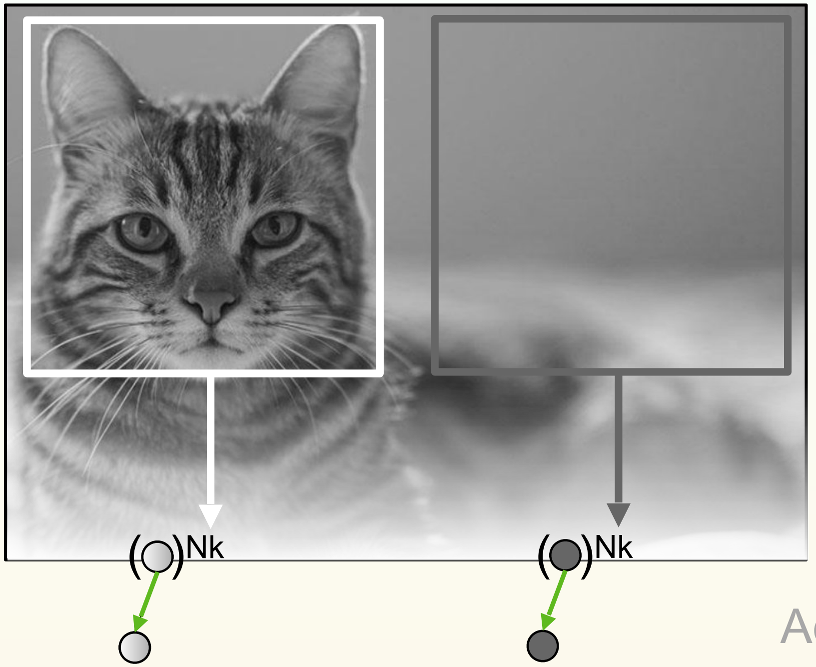
Index m of the pixel on the i-th place in the neighbourhood of j



$$I(7,i) = \left\{ \begin{array}{l} \\ \text{+2 for full row} \\ \text{+2 for full row} \end{array} \boxed{\begin{array}{ccc} 1 & 2 & 3 \\ 6 & 7 & 8 \\ 11 & 12 & 13 \end{array}} \right\}$$

conditions at the edges → wait a few slides

A CNN kernel: step by step



- Inputs x

- For one *channel*:

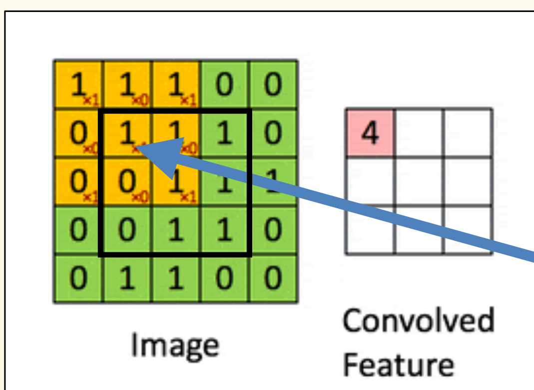
Learnable bias

Kernel size

$$y_j = \theta\left(\sum_{i}^{N_k} \omega_i \; x_{I(j,i)} - T\right)$$

Activation function

Learnable weights:
Relative position to j

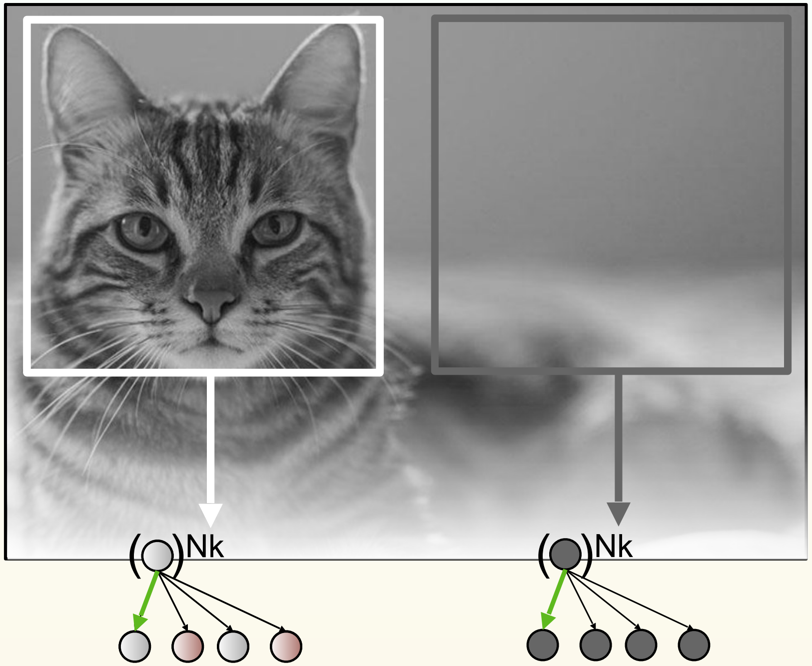


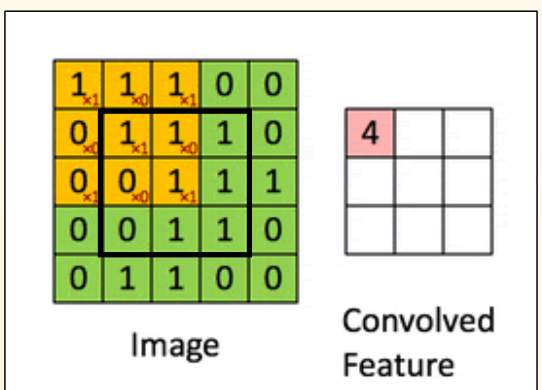
Index m of the pixel on the i-th place in the neighbourhood of j

$$I(7,i) = \left\{ \begin{array}{l} \\ \text{+2 for full row} \\ \text{+2 for full row} \end{array} \boxed{\begin{array}{ccc} 1 & 2 & 3 \\ 6 & 7 & 8 \\ 11 & 12 & 13 \end{array}} \right\}$$

conditions at the edges → wait a few slides

Multiple output channels



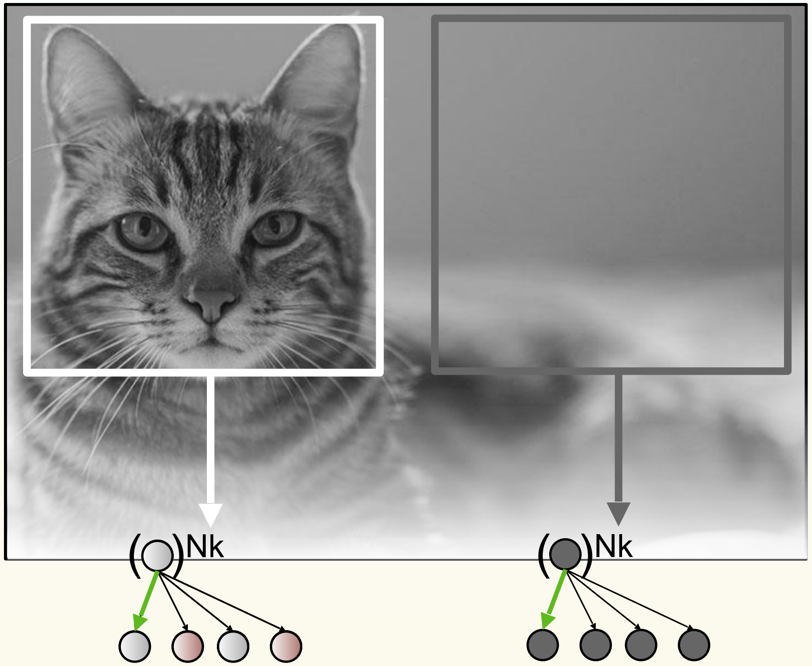


- Inputs x
- For N_c output channels (α)

$$y_{j\alpha} = \theta\left(\sum_{i}^{N_k} \omega_{i\alpha}\, x_{I(j,i)} - T_{\alpha}\right)$$

The weights are still shared and depend only on **relative position** w.r.t. pixel j (and α)

Multiple output channels

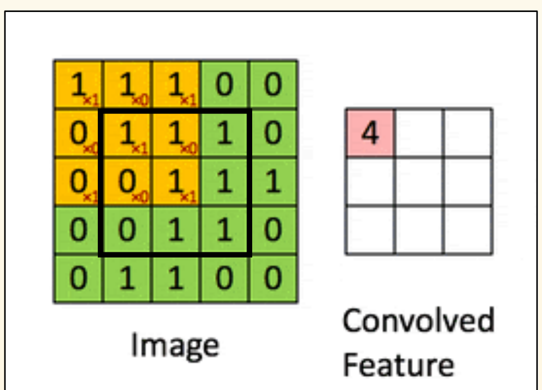


- Inputs x

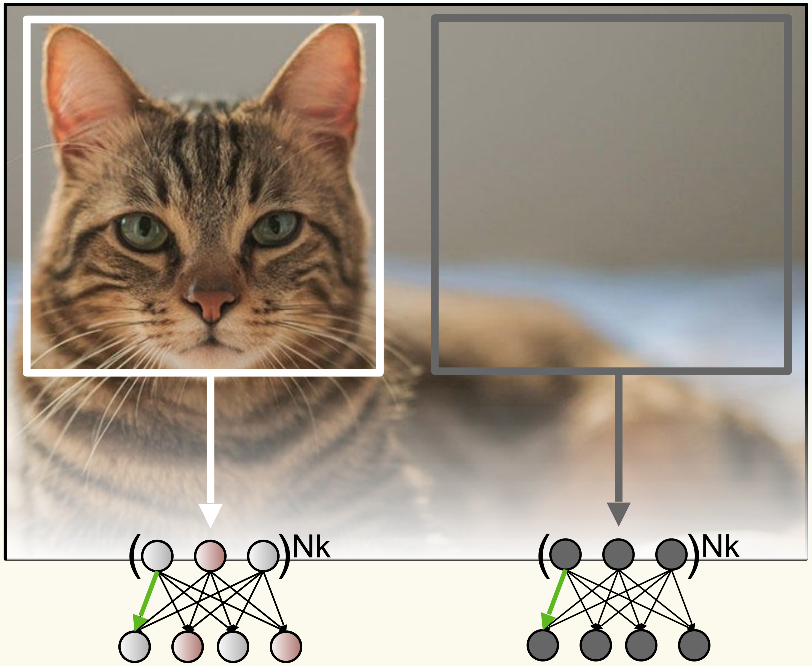
- For N_c output channels (α)

$$y_{j\alpha} = \theta\left(\sum_{i}^{N_k} \omega_{i\alpha}\, x_{I(j,i)} - T_{\alpha} \right)$$

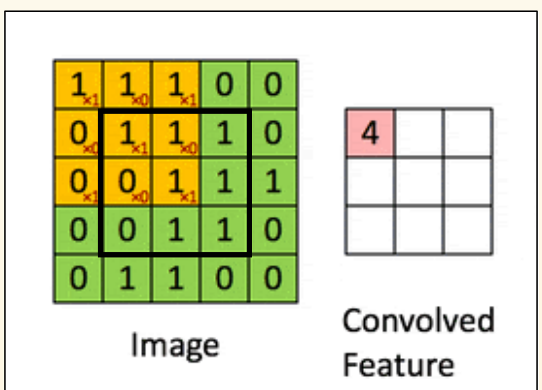
The weights are still shared and depend only on **relative position** w.r.t. pixel j (and α)



Multiple input channels



One *kernel* ≙ one dense MLP layer



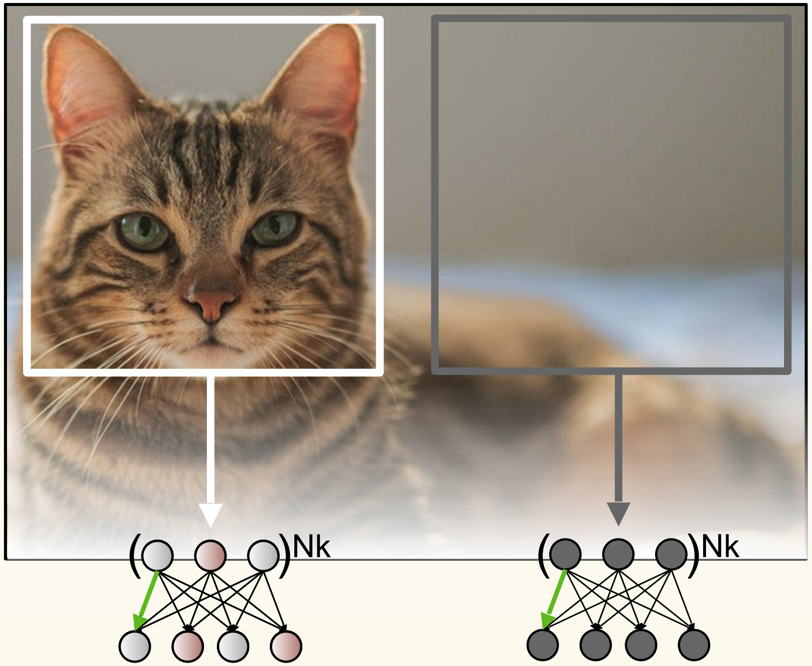
- Inputs x
- For N_F input channels/features

$$y_{j\alpha} = \theta\left(\sum_{\beta}^{N_F} \sum_{i}^{N_k} \omega_{i\alpha\beta} \; x_{I(j,i)\beta} - T_\alpha \right)$$

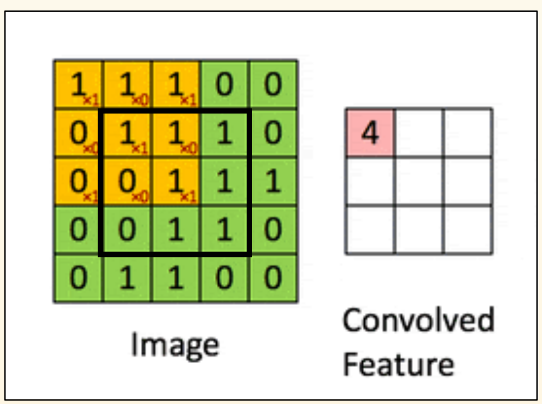
Still strictly relative

- This is a complete convolutional layer

Multiple input channels



One *kernel* ≙ one dense MLP layer



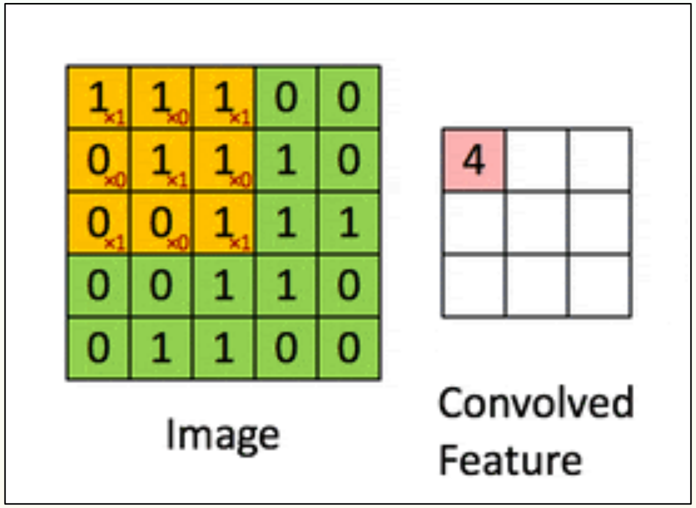
- Inputs x
- For N_F input channels/features

$$y_{j\alpha} = \theta\left(\sum_{\beta}^{N_F} \sum_{i}^{N_k} \omega_{i\alpha\beta}\, x_{I(j,i)\beta} - T_\alpha \right)$$

Still strictly relative

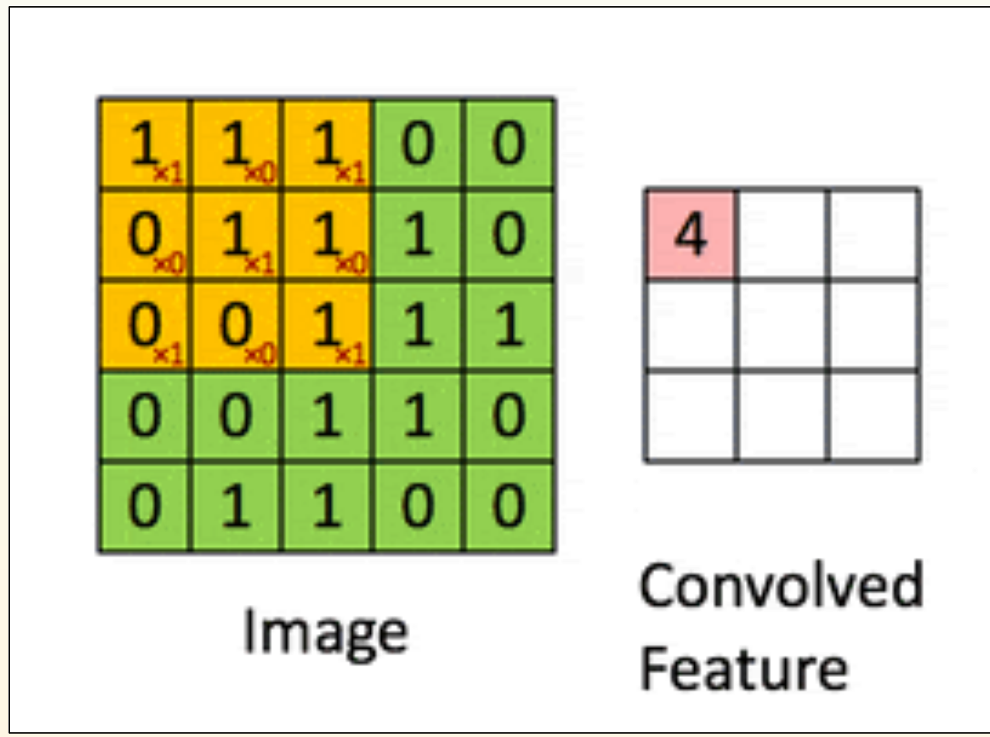
- This is a complete convolutional layer

Example



Kernel

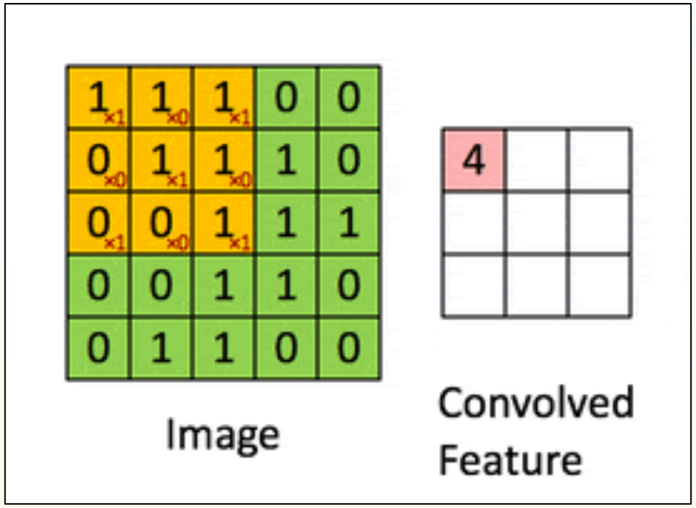
$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$



- No activation
- No bias
- One input
- One output

$$y_j = \sum_{i}^{N_k} \omega_i \; x_{I(j,i)}$$

Example

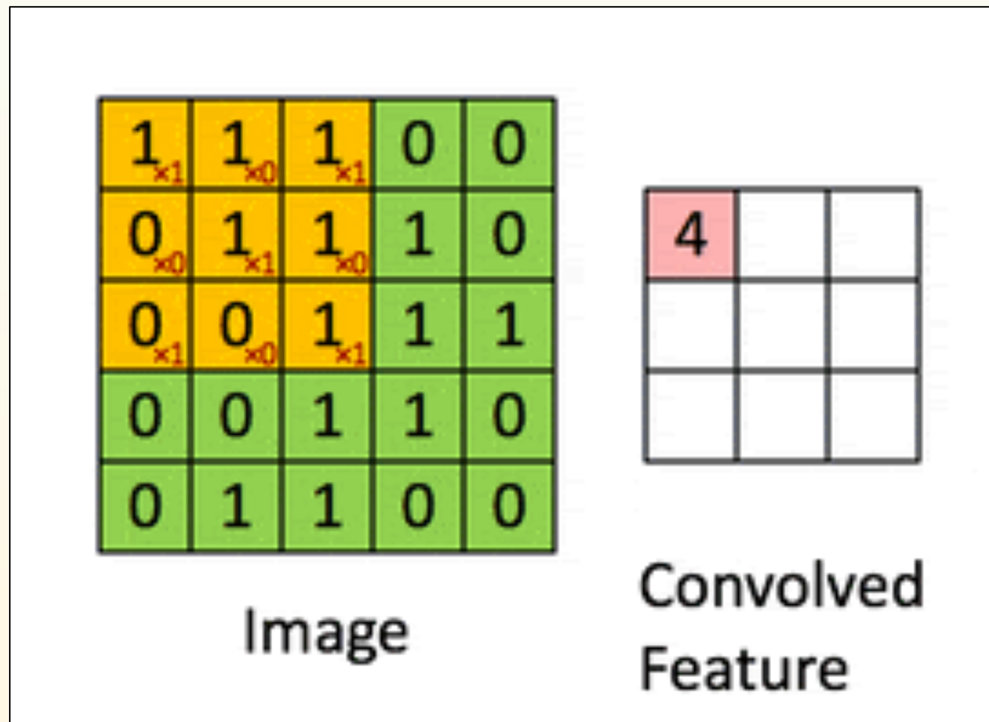


Kernel

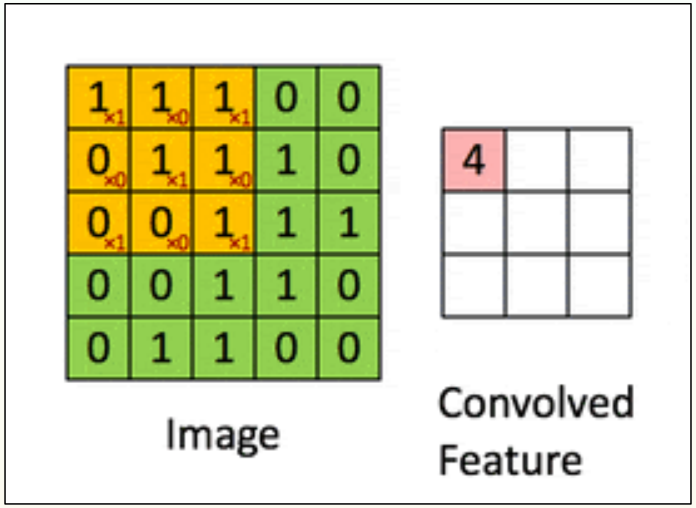
$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

- No activation
- No bias
- One input
- One output

$$y_j = \sum_{i}^{N_k} \omega_i \; x_{I(j,i)}$$

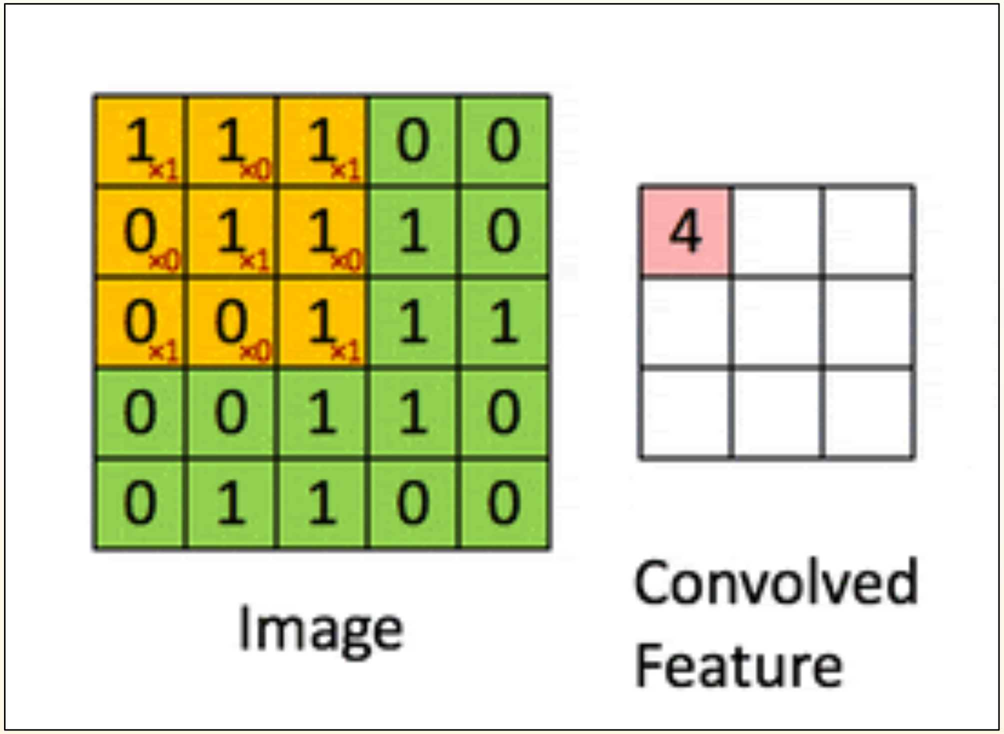


Example



Kernel

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$



- No activation
- No bias
- One input
- One output

$$y_j = \sum_i^{N_k} \omega_i \; x_{I(j,i)}$$

$$y_{j\alpha} = \theta\left(\sum_{\beta}^{N_F}\sum_{i}^{N_k}\omega_{i\alpha\beta}\; x_{I(j,i)\beta} - T_\alpha\right)$$

Parameters

Filter

Time for some (more) questions

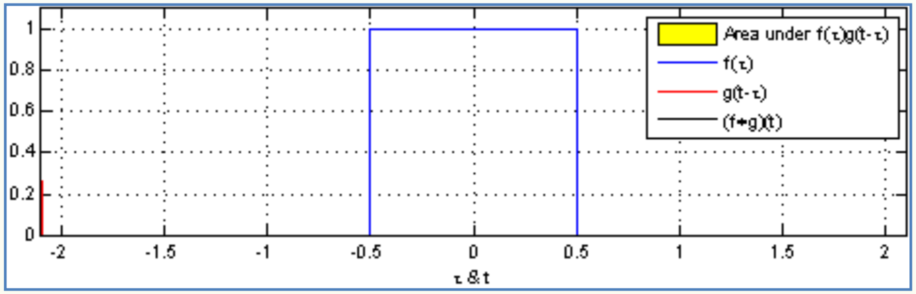
Channels

Neighbourhood

Kernel

Bias

Longer side note: where is the convolution?



https://en.wikipedia.org/wiki/Convolution [accessed 13.7.23]

- CNN:

$$y_j = \sum_{i}^{N_k} \omega_i \; x_{I(j,i)}$$

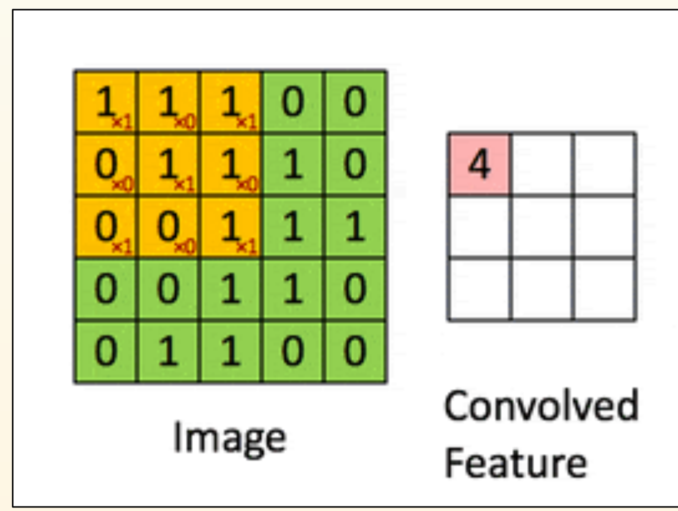
'n-m' hidden here

- Convolution:

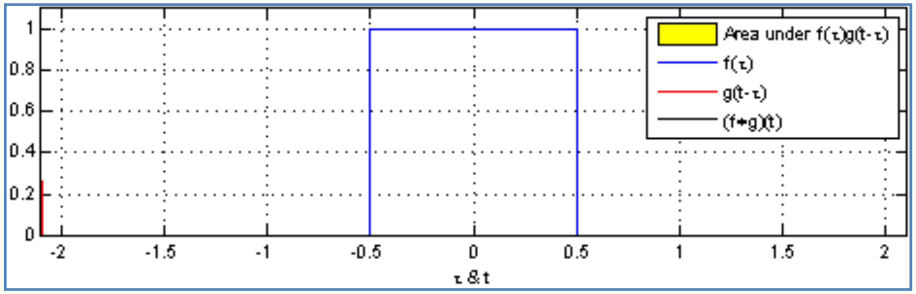
$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau) g(t-\tau) d\tau$$

- Discrete:

$$(f * g)[n] = \sum_{m=-\infty}^{+\infty} f[m] g[n-m]$$



Longer side note: where is the convolution?



https://en.wikipedia.org/wiki/Convolution [accessed 13.7.23]

- CNN:

$$y_j = \sum_{i}^{N_k} \omega_i \; x_{I(j,i)}$$

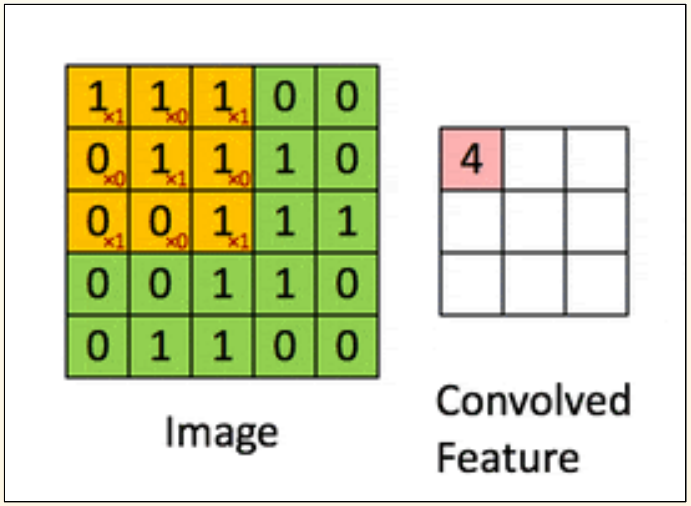
'n-m' hidden here

- Convolution:

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau) g(t-\tau) d\tau$$

- Discrete:

$$(f * g)[n] = \sum_{m=-\infty}^{+\infty} f[m] g[n-m]$$



Re-shuffle symbols

$$(f * g)[n] = \sum_{m=-\infty}^{+\infty} f[m]g[n-m]$$

$$y_j = \sum_{i}^{N_k} \omega_i \; x_{m=I(j,i)}$$

Index m of the pixel on the i-th place
in the neighbourhood of j

Re-shuffle symbols

$$(f * g)[n] = \sum_{m=-\infty}^{+\infty} f[m]g[n-m]$$

$$y_j = \sum_{i}^{N_k} \omega_i \, x_{m=I(j,i)}$$

Index m of the pixel on the i-th place
in the neighbourhood of j

Pixels
in image

Switch perspective

$$= \sum_{m=1}^{N_p} \omega_{i=I^{-1}(j,m)} x_m$$

✓

The i-th place for a pixel with index m
in the neighbourhood of j

If not in neighbourhood: extend kernel
such that $\omega = 0$

$$\begin{array}{ccccccc} & & & \vdots & & & \\ & 0 & 0 & 0 & 0 & 0 & \\ & 0 & 1 & 0 & 1 & 0 & \\ \cdots & 0 & 0 & 1 & 0 & 0 & \cdots \\ & 0 & 1 & 0 & 1 & 0 & \\ & 0 & 0 & 0 & 0 & 0 & \\ & & & \vdots & & & \end{array}$$

Re-shuffle symbols

$$(f * g)[n] = \sum_{m=-\infty}^{+\infty} f[m]g[n-m]$$

$$y_j = \sum_{i}^{N_k} \omega_i \; x_{m=I(j,i)}$$

Index m of the pixel on the i-th place
in the neighbourhood of j

Switch perspective

Pixels
in image

$$= \sum_{m=1}^{N_p} \omega_{i=I^{-1}(j,m)} x_m$$

The i-th place for a pixel with index m
in the neighbourhood of j

If not in neighbourhood: extend kernel
such that $\omega = 0$

$$\begin{array}{ccccccc} & & & \vdots & & & \\ & 0 & 0 & 0 & 0 & 0 & \\ & 0 & 1 & 0 & 1 & 0 & \\ \cdots & 0 & 0 & 1 & 0 & 0 & \cdots \\ & 0 & 1 & 0 & 1 & 0 & \\ & 0 & 0 & 0 & 0 & 0 & \\ & & & \vdots & & & \end{array}$$

$$y_j = \sum_{m=1}^{N_p} x[m] \; \omega_{i=I^{-1}(j,m)}$$

Simple replacement as $x[m] = x_m$

It is a convolution

$$(f * g)[n] = \sum_{m=-\infty}^{+\infty} f[m]g[n-m]$$

$$y_j = \sum_{m=1}^{N_p} x[m]\ \omega_{I^{-1}(j,m)}$$

$I^{-1}(j,m)$ can be rephrased as a distance index

$$\begin{array}{ccccccc} & & & \vdots & & & \\ & 0 & 0 & 0 & 0 & 0 & \\ & 0 & 1 & 0 & 1 & 0 & \\ \cdots & 0 & 0 & 1 & 0 & 0 & \cdots \\ & 0 & 1 & 0 & 1 & 0 & \\ & 0 & 0 & 0 & 0 & 0 & \\ & & & \vdots & & & \end{array}$$

This is index j!

It is a convolution

$$(f * g)[n] = \sum_{m=-\infty}^{+\infty} f[m]g[n-m]$$

$$y_j = \sum_{m=1}^{N_p} x[m]\ \omega_{I^{-1}(j,m)}$$

$$\begin{array}{ccccccc} & & & \vdots & & & \\ & 0 & 0 & 0 & 0 & 0 & \\ & 0 & 1 & 0 & 1 & 0 & \\ \cdots & 0 & 0 & 1 & 0 & 0 & \cdots \\ & 0 & 1 & 0 & 1 & 0 & \\ & 0 & 0 & 0 & 0 & 0 & \\ & & & \vdots & & & \end{array}$$

This is index j!

$I^{-1}(j,m)$ can be rephrased as a distance index

Define $\tilde{\omega}[j-m] = \omega_{I^{-1}(j,m)}$ *

$$y_j = \sum_{m=1}^{N_p} x[m]\ \tilde{\omega}[j-m] \quad \leftrightarrow \quad (f * g)[n] = \sum_{m=-\infty}^{+\infty} f[m]g[n-m]$$

- **A convolutional neural network layer is indeed equivalent to a convolution**

It is a convolution

$$(f * g)[n] = \sum_{m=-\infty}^{+\infty} f[m]g[n-m]$$

$$y_j = \sum_{m=1}^{N_p} x[m]\ \omega_{I^{-1}(j,m)}$$

$$\begin{array}{ccccccc} & & & \vdots & & & \\ & 0 & 0 & 0 & 0 & 0 & \\ & 0 & 1 & 0 & 1 & 0 & \\ \cdots & 0 & 0 & 1 & 0 & 0 & \cdots \\ & 0 & 1 & 0 & 1 & 0 & \\ & 0 & 0 & 0 & 0 & 0 & \\ & & & \vdots & & & \end{array}$$

This is index j!

$I^{-1}(j,m)$ can be rephrased as a distance index

Define $\tilde{\omega}[j-m] = \omega_{I^{-1}(j,m)}$ *

$$y_j = \sum_{m=1}^{N_p} x[m]\ \tilde{\omega}[j-m] \quad \leftrightarrow \quad (f * g)[n] = \sum_{m=-\infty}^{+\infty} f[m]g[n-m]$$

✓ ✓ ✓

- **A convolutional neural network layer is indeed equivalent to a convolution**

* technically, depending on the definition, this could implement a convolution or cross correlation, possibly implementing a sign flip w.r.t. convolution. In practice this does not matter since ω_i are learnable and can re-absorb the flip. A detailed explanation can be found here: https://ai.stackexchange.com/questions/21999/do-convolutional-neural-networks-perform-convolution-or-cross-correlation

Translational equivariance as direct consequence

The convolution commutes with translations, meaning that

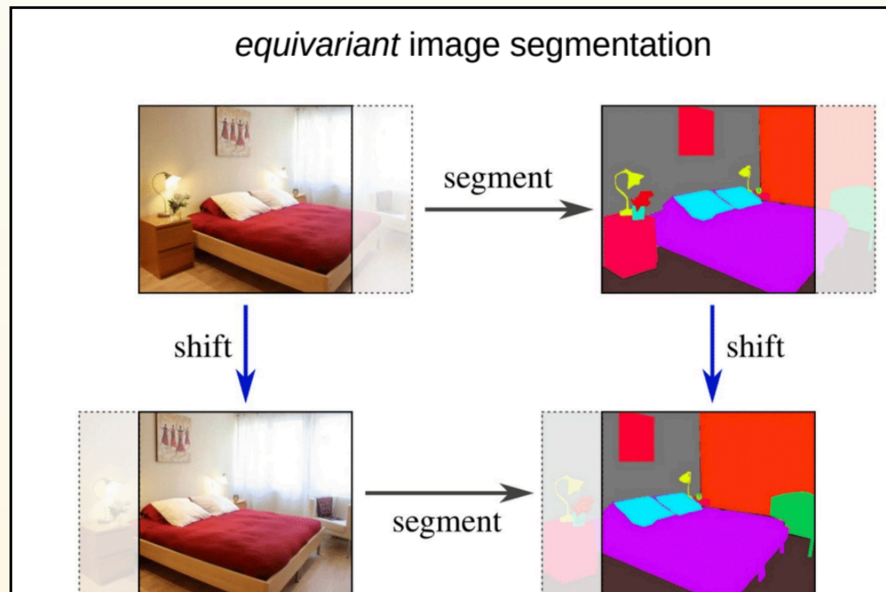
$$\tau_x(f * g) = (\tau_x f) * g = f * (\tau_x g)$$

where $\tau_x f$ is the translation of the function *f* by *x* defined by

$$(\tau_x f)(y) = f(y - x).$$

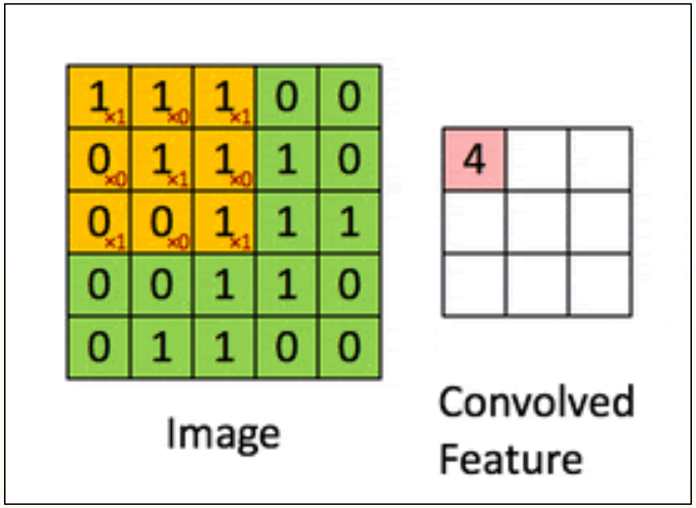
https://en.wikipedia.org/wiki/Convolution

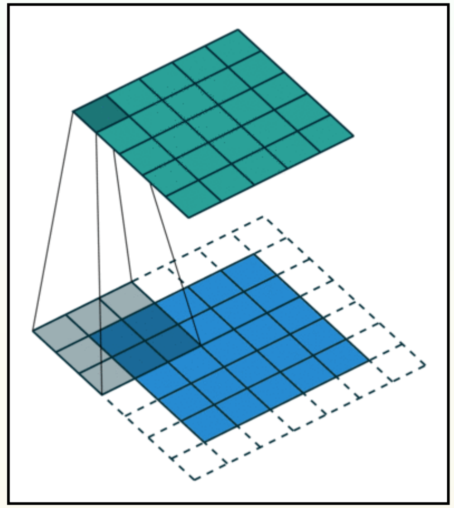
- Convolutions and translation commute
- Shift + convolution is the same as convolution + shift
- This is referred to translation **equivariance** (not invariance)



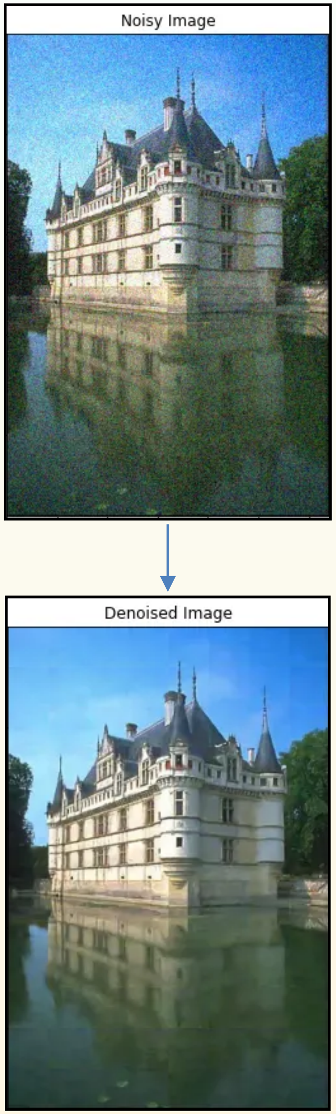
Credit: Maurice Weiler

Conditions at the edges





arxiv:1603.07285

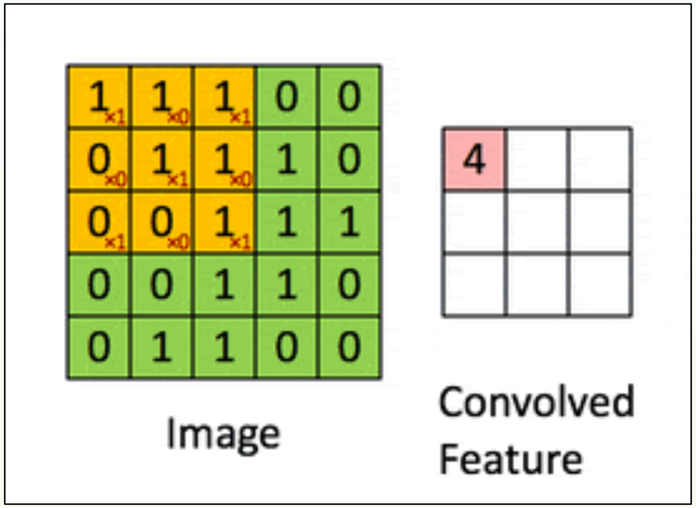


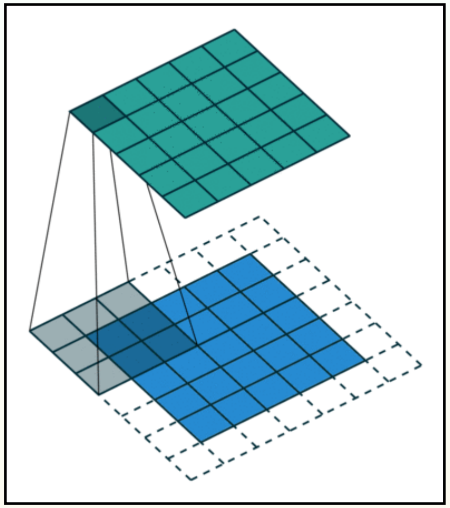
- For a 3 x 3 kernel, the image size will be reduced by 2 pixels on top and bottom
  - For a 5 x 5 kernel?

- If this is not desired (zero) padding the image can help

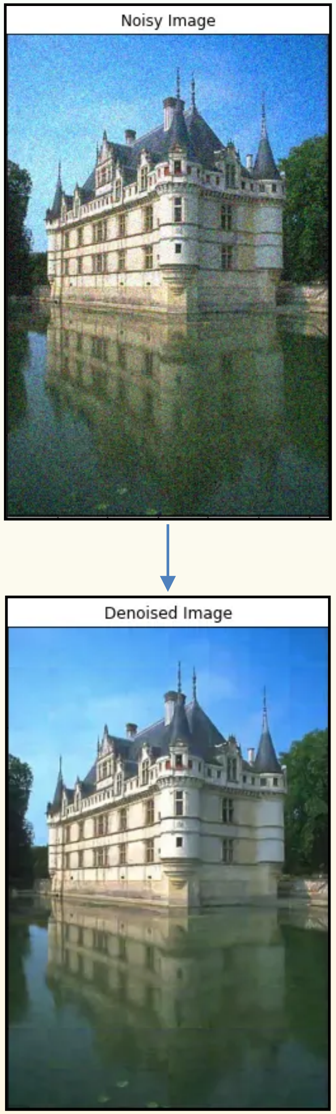
https://medium.com/analytics-vidhya/noise-removal-in-images-using-deep-learning-models-3972544372d2

Conditions at the edges





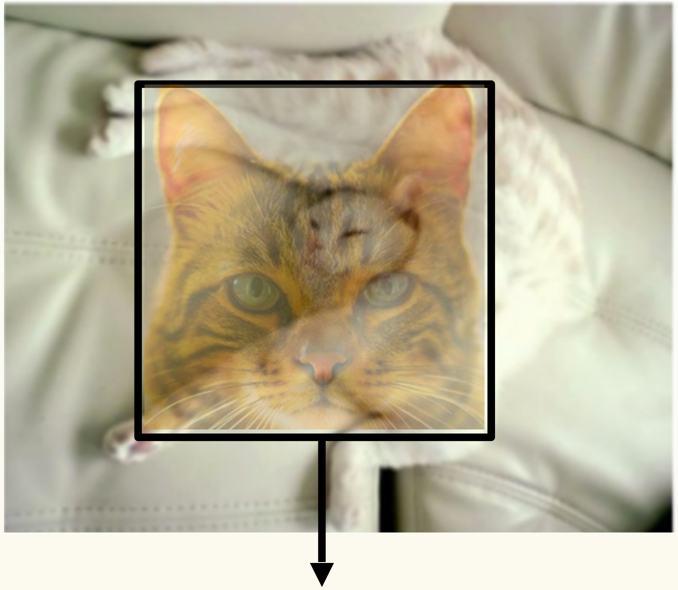
arxiv:1603.07285



- For a 3 x 3 kernel, the image size will be reduced by 2 pixels on top and bottom
  - For a 5 x 5 kernel?
- If this is not desired (zero) padding the image can help

https://medium.com/analytics-vidhya/noise-removal-in-images-using-deep-learning-models-3972544372d2

Cats (still) come in different shapes



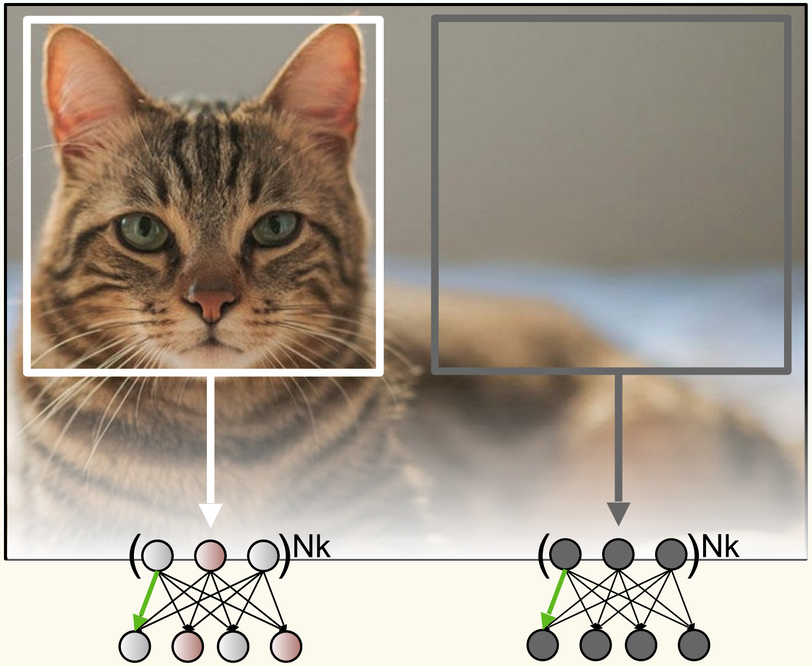
Not a cat

Not a cat

- Many different very complex filters are needed

- Can be solved by
  - Learning filters from examples ✓
  - **Abstraction**

Breaking up the problem into smaller parts



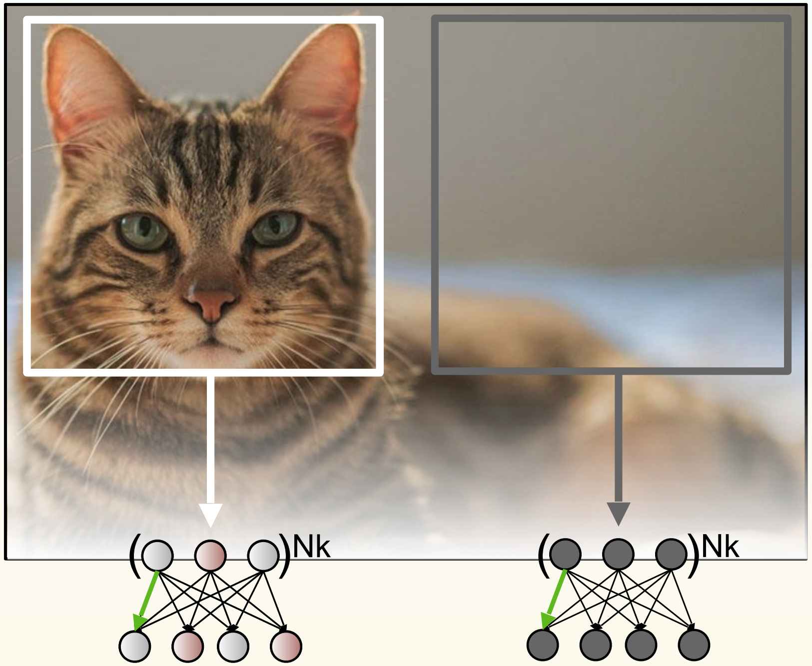
One *kernel* ≙ one dense MLP layer

$$y_{j\alpha} = \theta\left(\sum_{\beta}^{N_F} \sum_{i}^{N_k} \omega_{i\alpha\beta}\, x_{I(j,i)\beta} - T_\alpha \right)$$

- This is one complete convolutional layer with $\alpha \in \{1, \ldots, N_C\}$

- Counting weights:
  how many do we have?

Breaking up the problem into smaller parts



One *kernel* ≙ one dense MLP layer

$$y_{j\alpha} = \theta\left(\sum_{\beta}^{N_F}\sum_{i}^{N_k} \omega_{i\alpha\beta}\, x_{I(j,i)\beta} - T_\alpha\right)$$

- This is one complete convolutional layer with $\alpha \in \{1,\ldots,N_C\}$

- Counting weights: how many do we have?

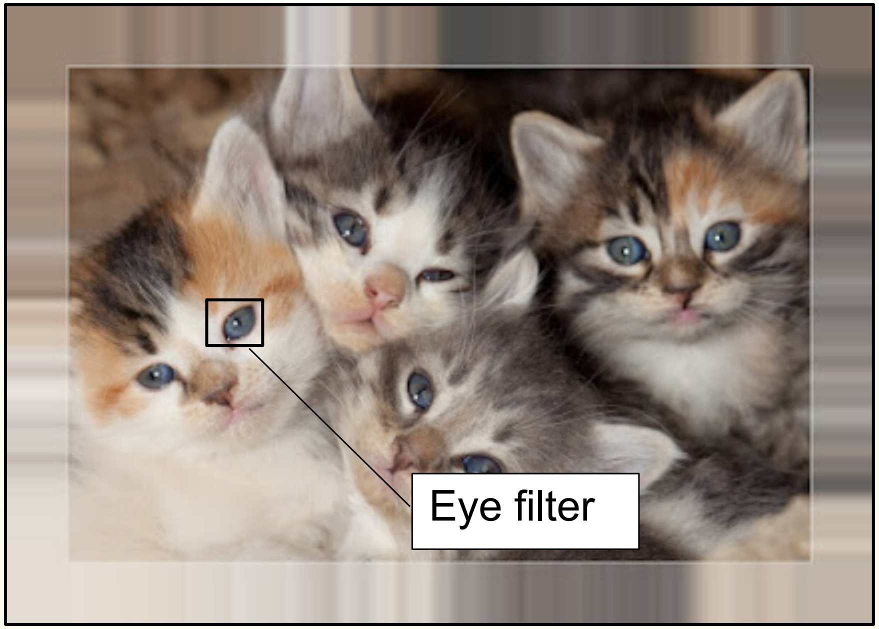
$N_C \cdot N_F \cdot N_k$

- With $N_k \approx H \otimes W$, kernels must not be too big
- Smaller kernels cannot capture a whole cat

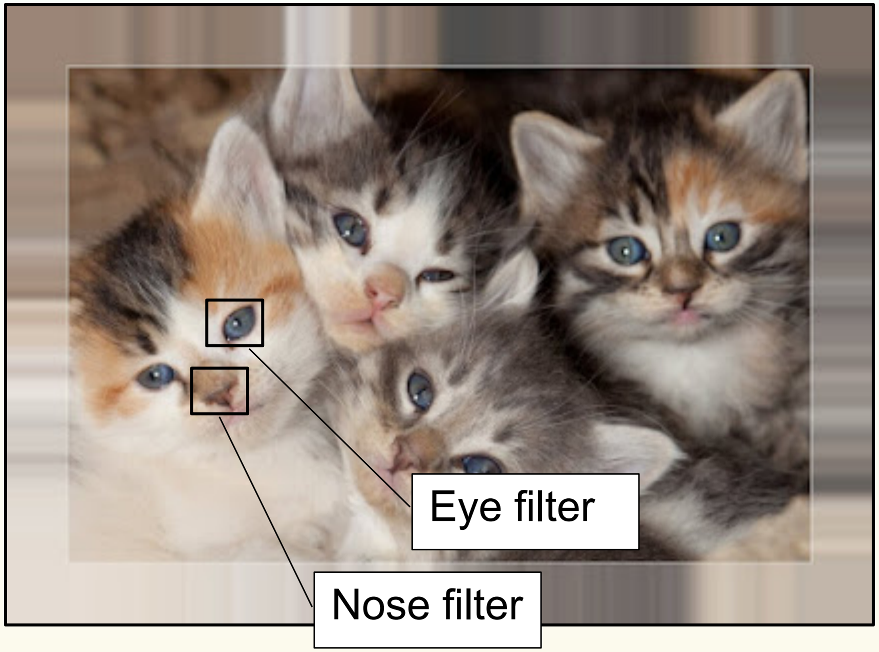
- Break down problem: abstraction and pooling

Abstraction and pooling

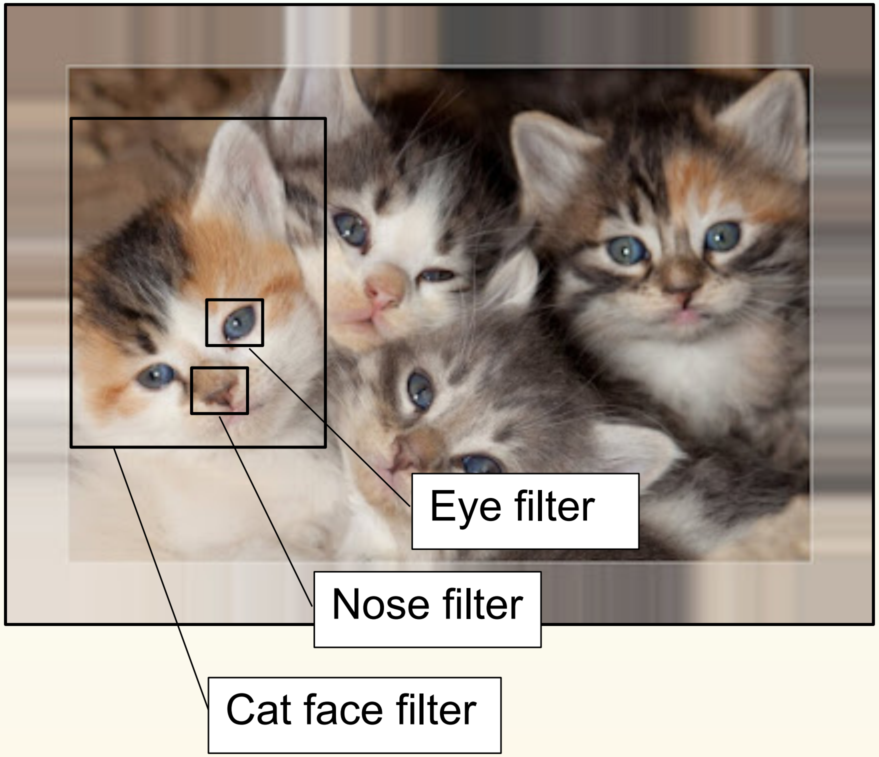
Abstraction and pooling



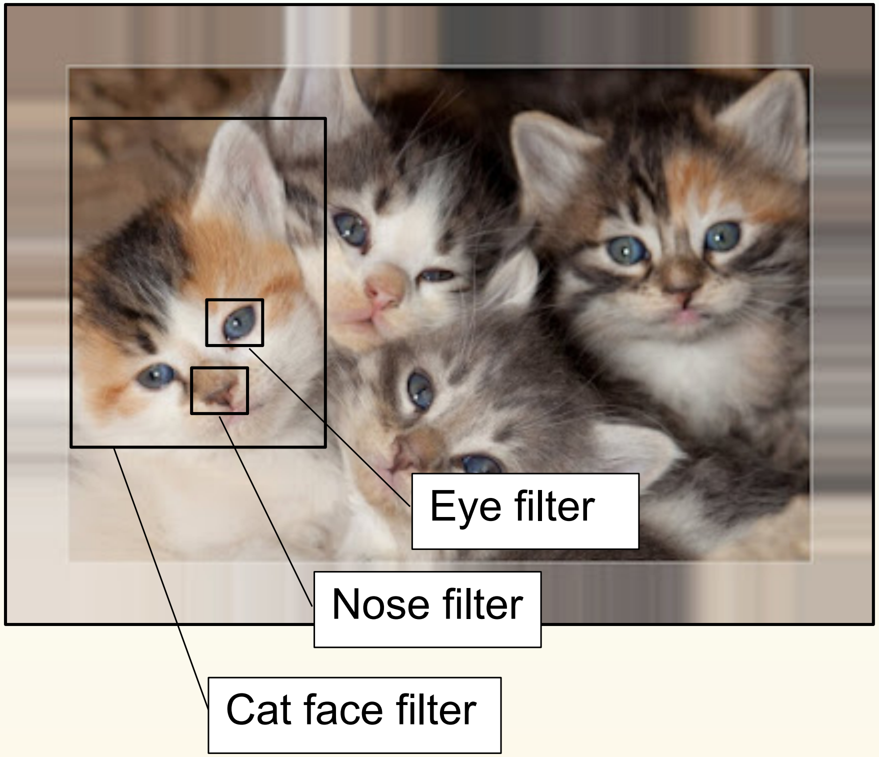
Abstraction and pooling

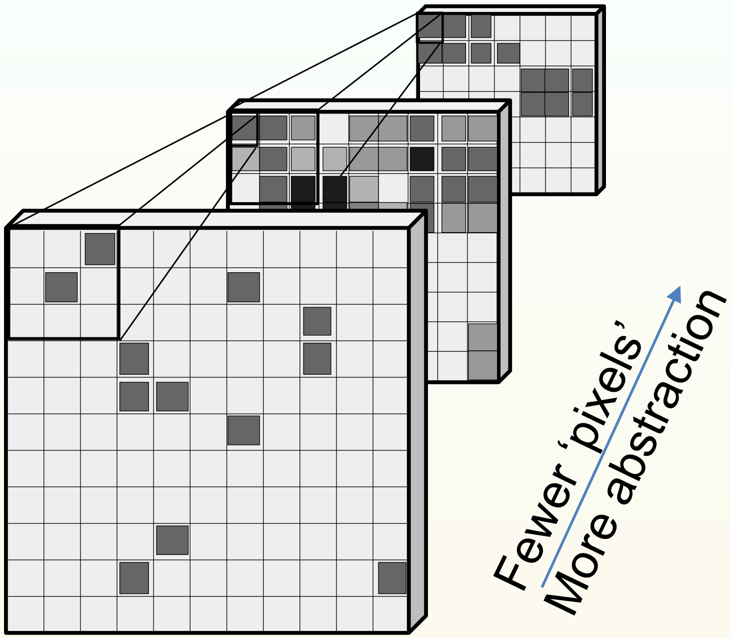


Abstraction and pooling

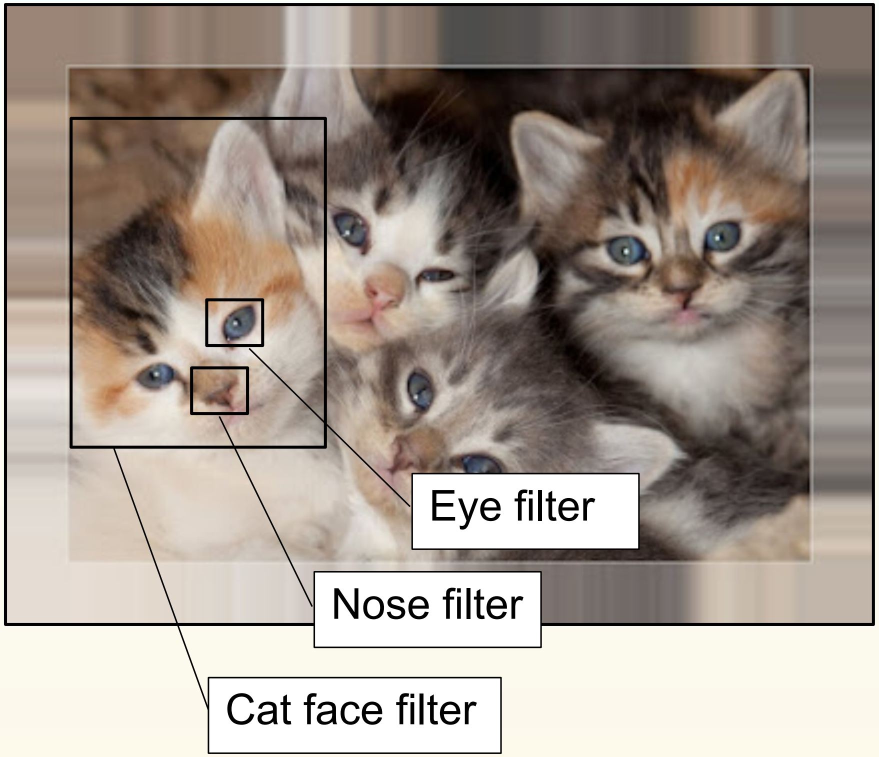


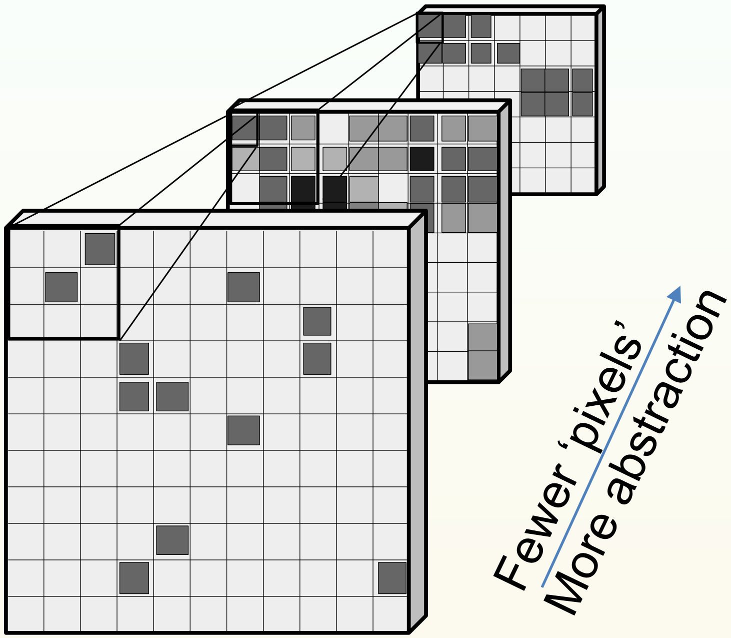
Abstraction and pooling





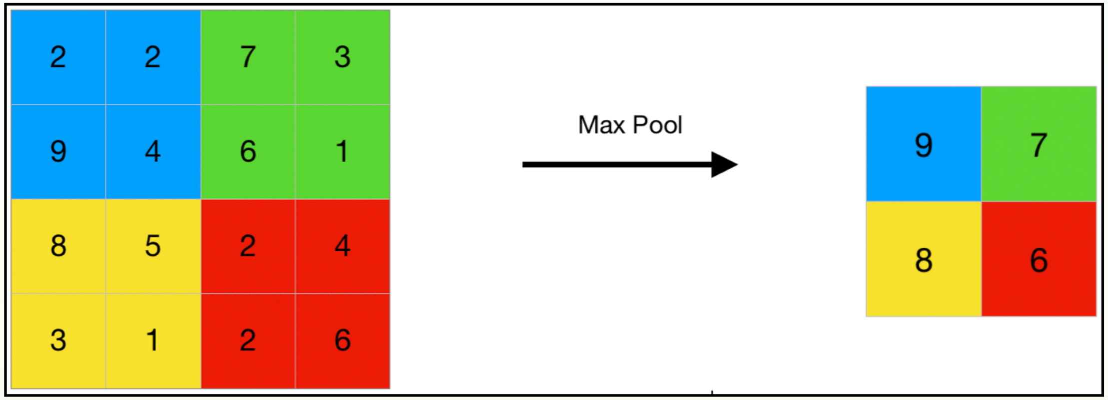
Abstraction and pooling



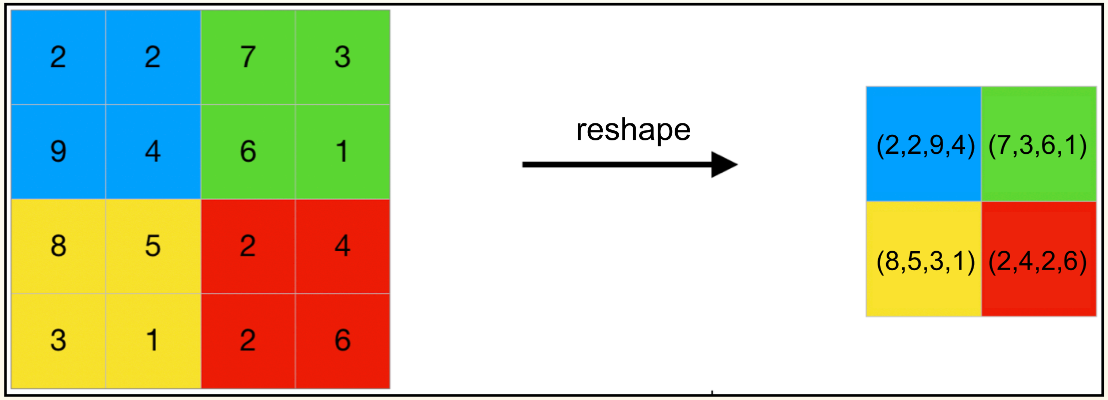


- Use smaller kernels to capture individual features
- Summarise (pool) the filter outputs of several neighbouring pixels
    - Take maximum (max pooling)
    - Take average/sum (average pooling)
    - Reshape tensor
- Go in bigger steps 'skipping' pixels: strides

Pooling

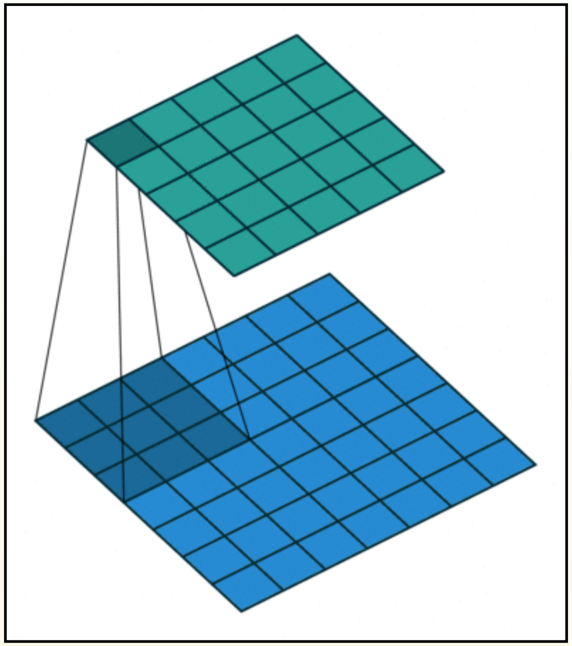


https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

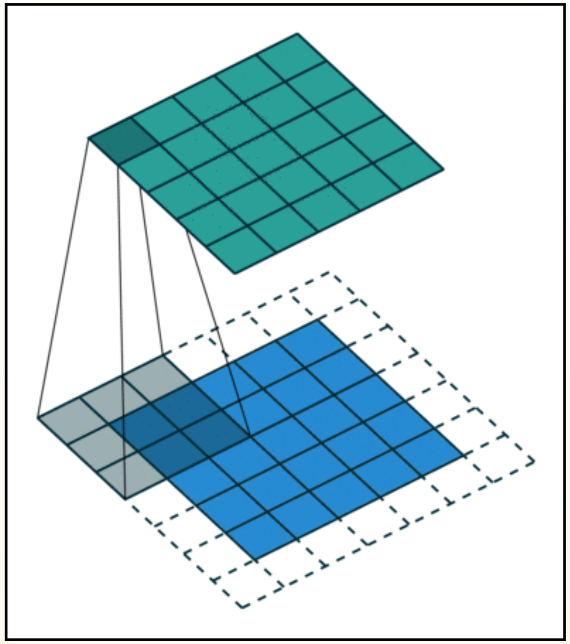


- Max pooling: which filter has triggered the largest output?
  - Is this more of an eye or a nose in that patch
- Reshaping: re-organise the information without removal of information
  - Not used so much, in particular for classification Why?

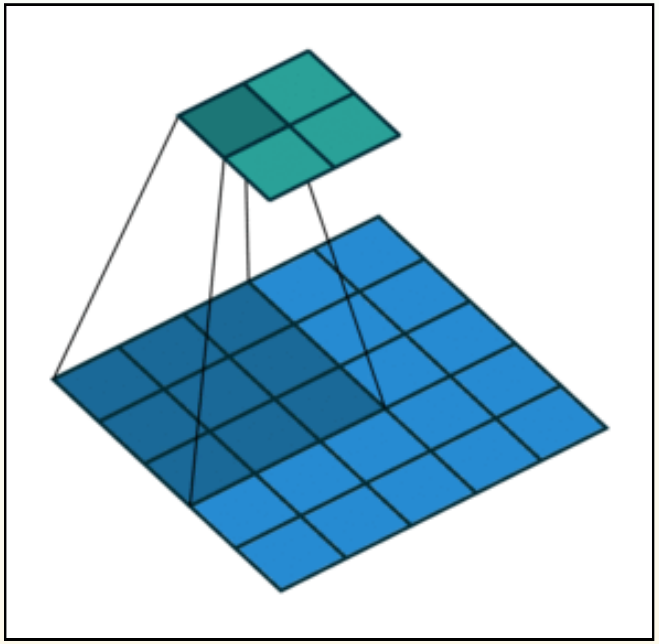
Strides



Stride 1, no padding



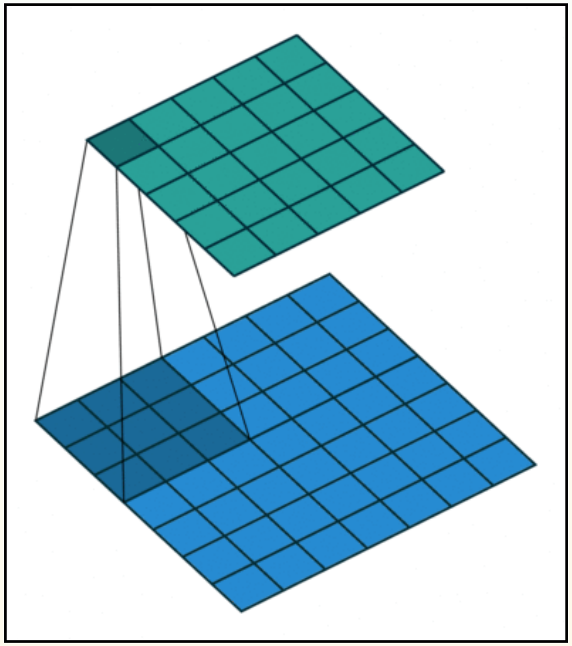
Stride 1, padding



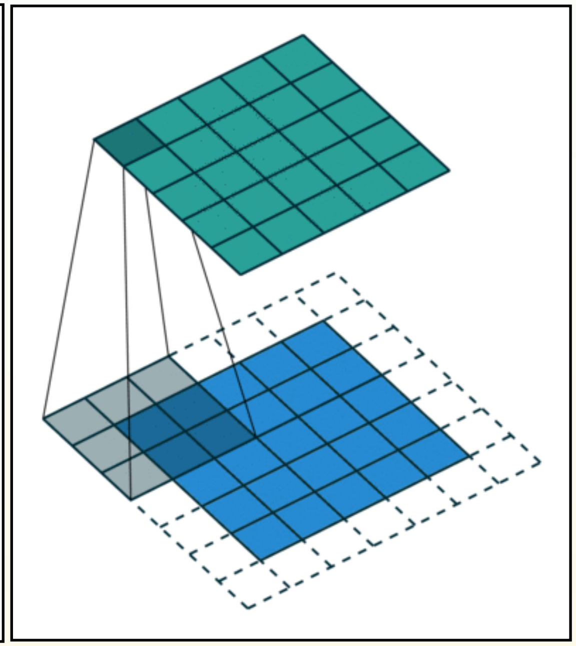
Stride 2

arxiv:1603.07285

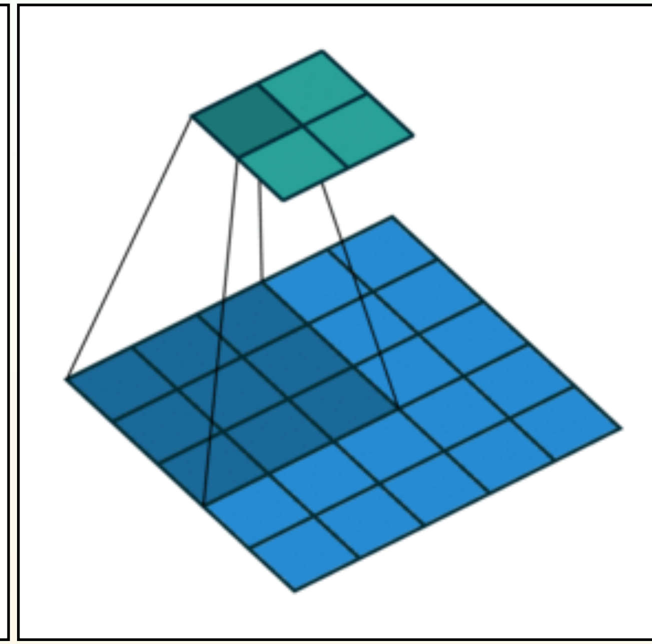
Strides



Stride 1, no padding



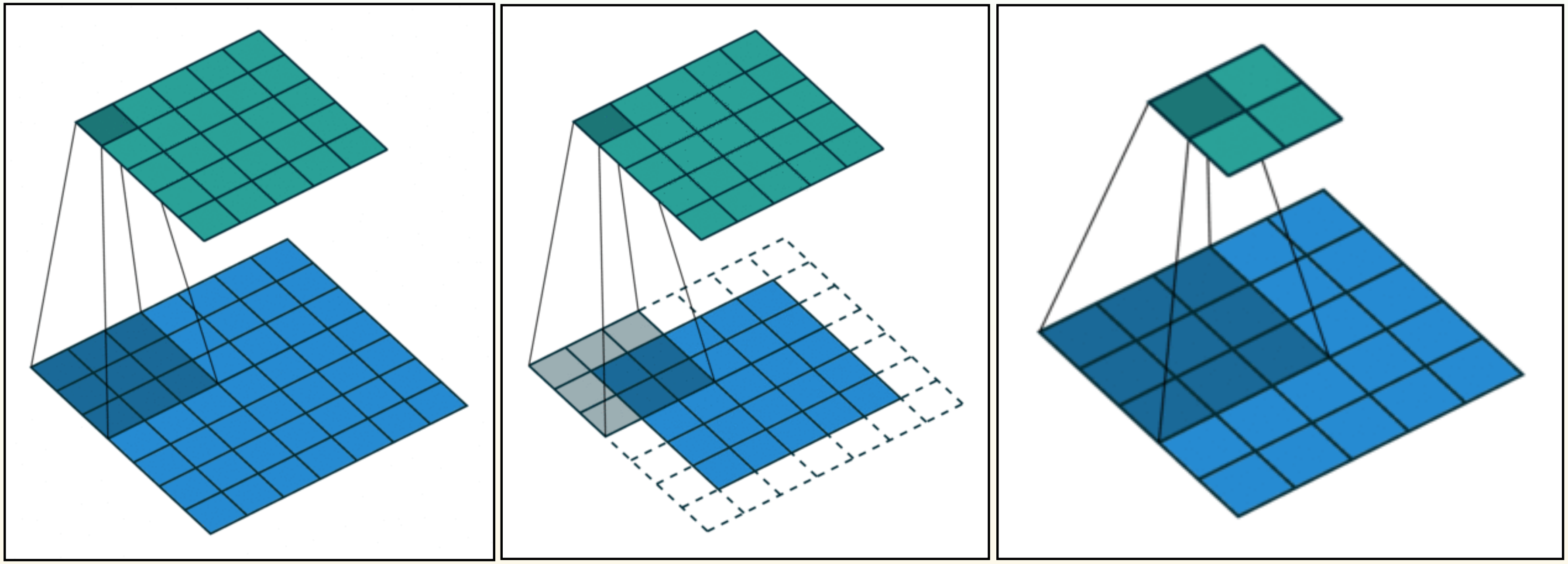
Stride 1, padding



Stride 2

arxiv:1603.07285

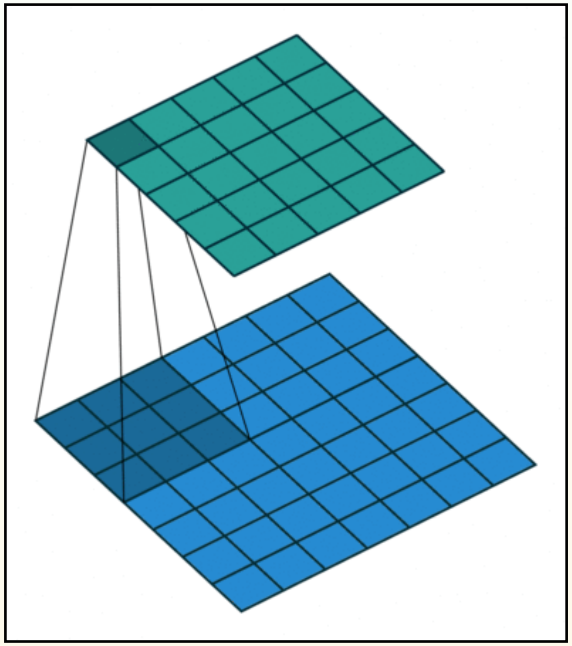
Strides



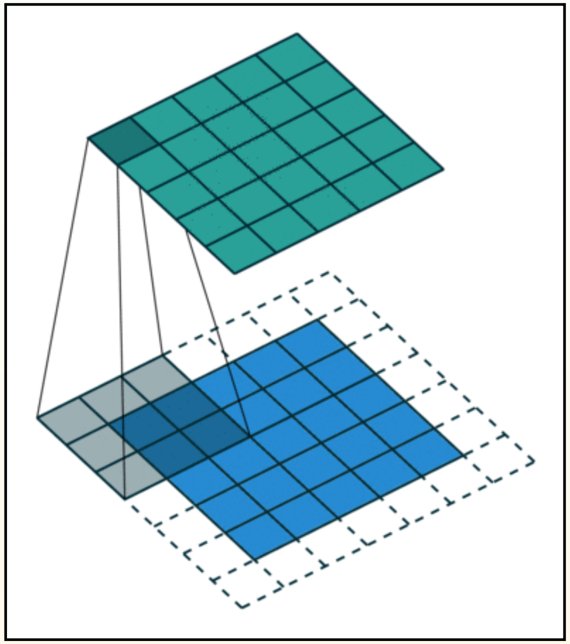
Stride 1, no padding | Stride 1, padding | Stride 2

arxiv:1603.07285

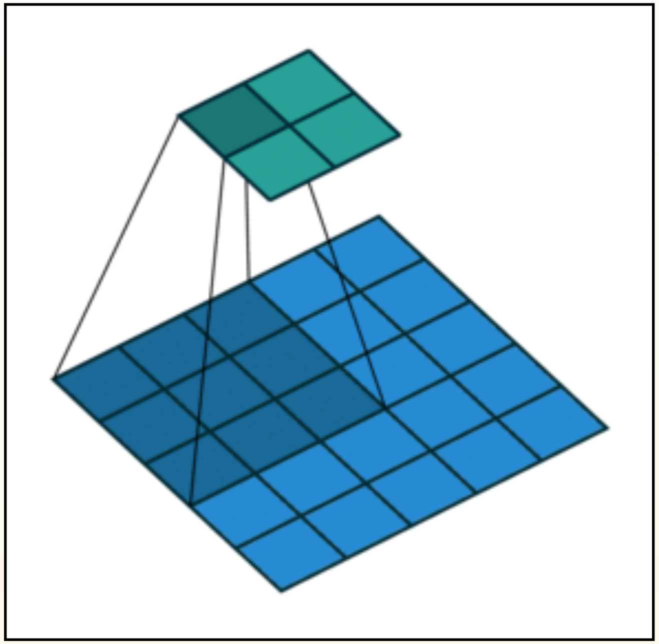
Strides



Stride 1, no padding



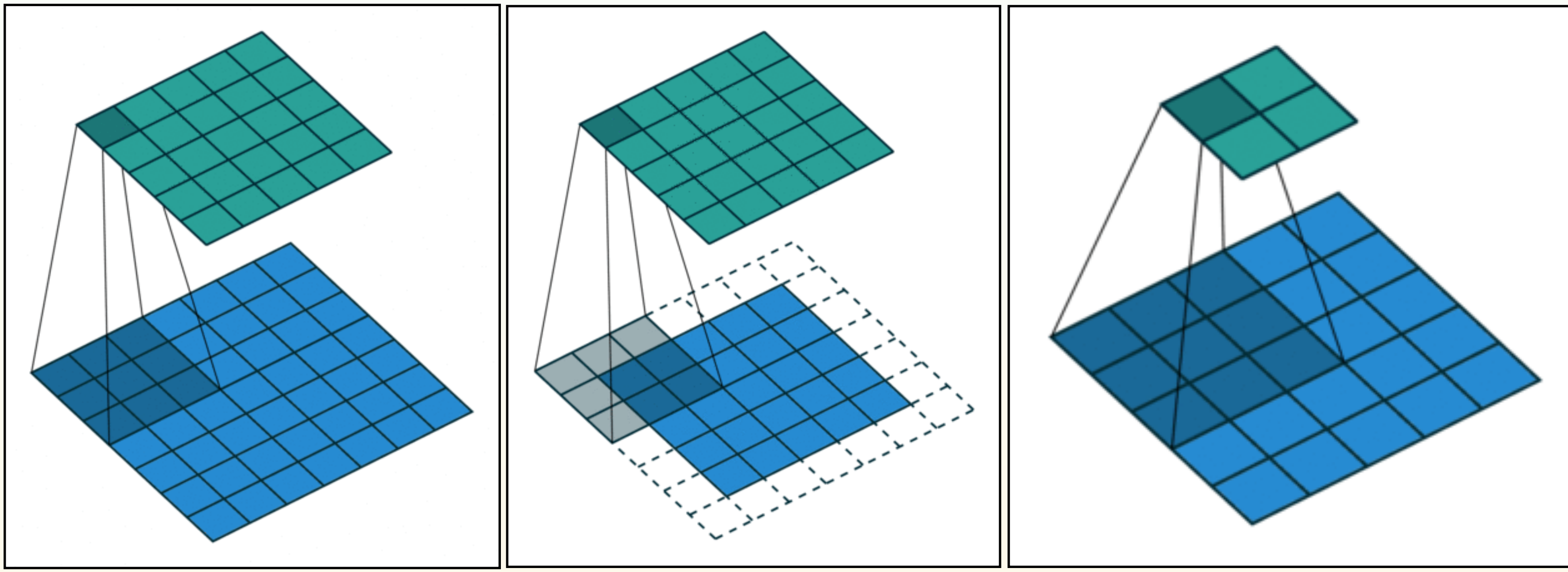
Stride 1, padding



Stride 2

arxiv:1603.07285

Strides

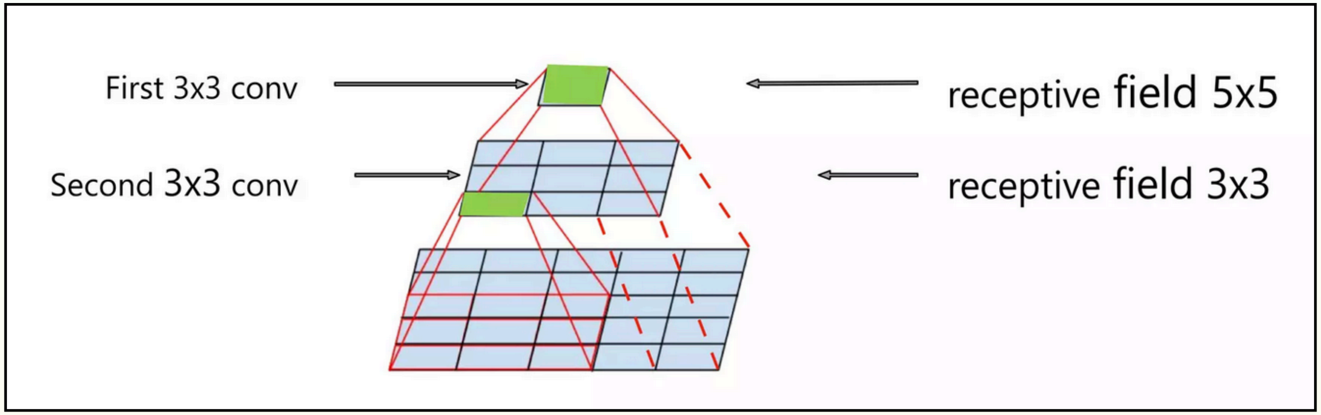


Stride 1, no padding | Stride 1, padding | Stride 2

arxiv:1603.07285

- The stride is the amount the filter ‘moves’ at each step

The notion of the receptive field

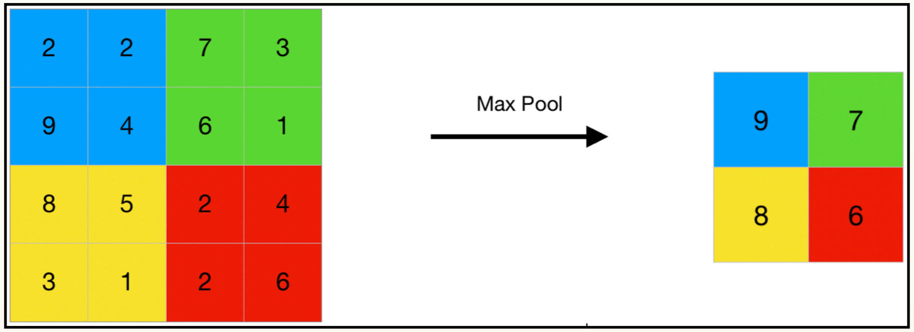


- For a given pixel, from how far away could it have accumulated information
- Central concept when designing neural networks in general
- Easily accessible for CNNs
- Needs to be big enough to capture the object

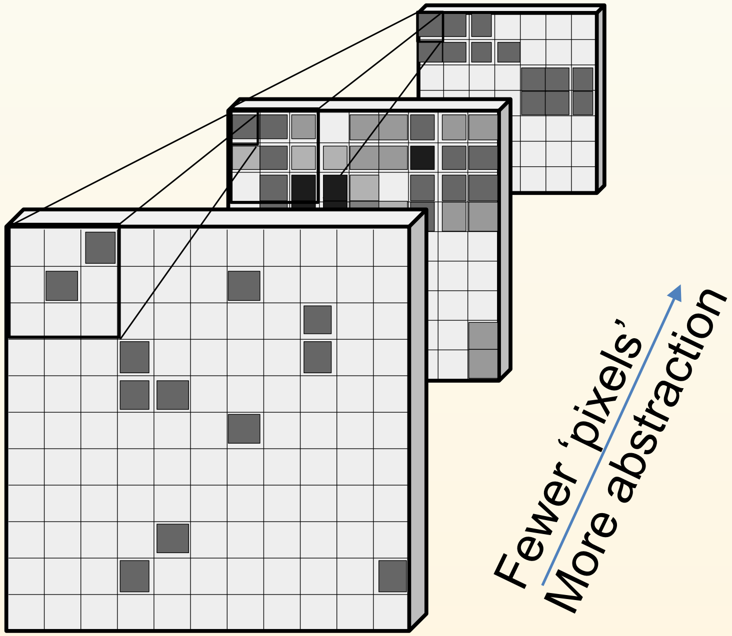
Our CNN toolbox

- CNN kernel
  - Learns filters

$$y_{j\alpha} = \theta\left(\sum_{\beta}^{N_F} \sum_{i}^{N_k} \omega_{i\alpha\beta}\, x_{I(j,i)\beta} - T_\alpha\right)$$



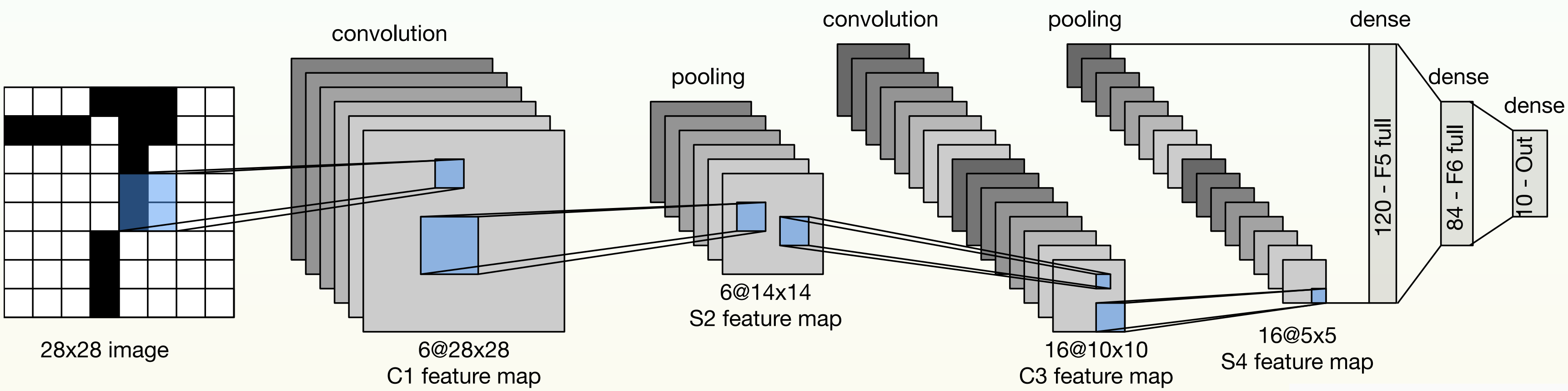
- Strides + Pooling
  - Build summaries

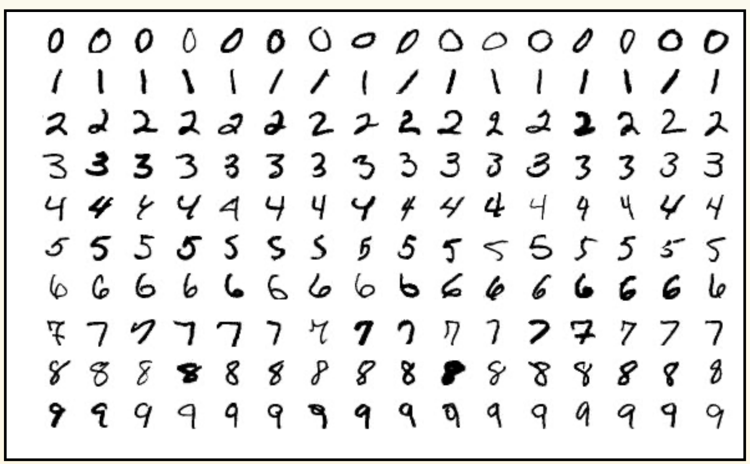


- Stack CNN layers
  - Abstraction

Example: LeNet (1998)

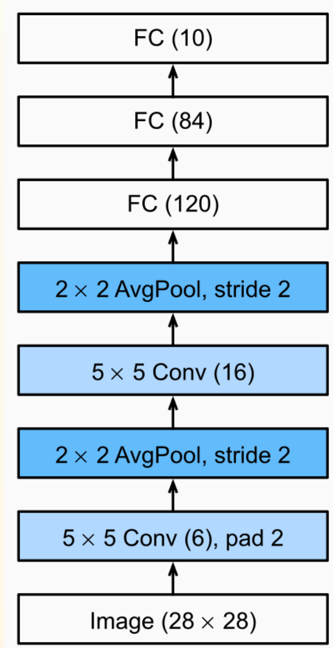
LeCun et al, Proceedings of the IEEE, 1998



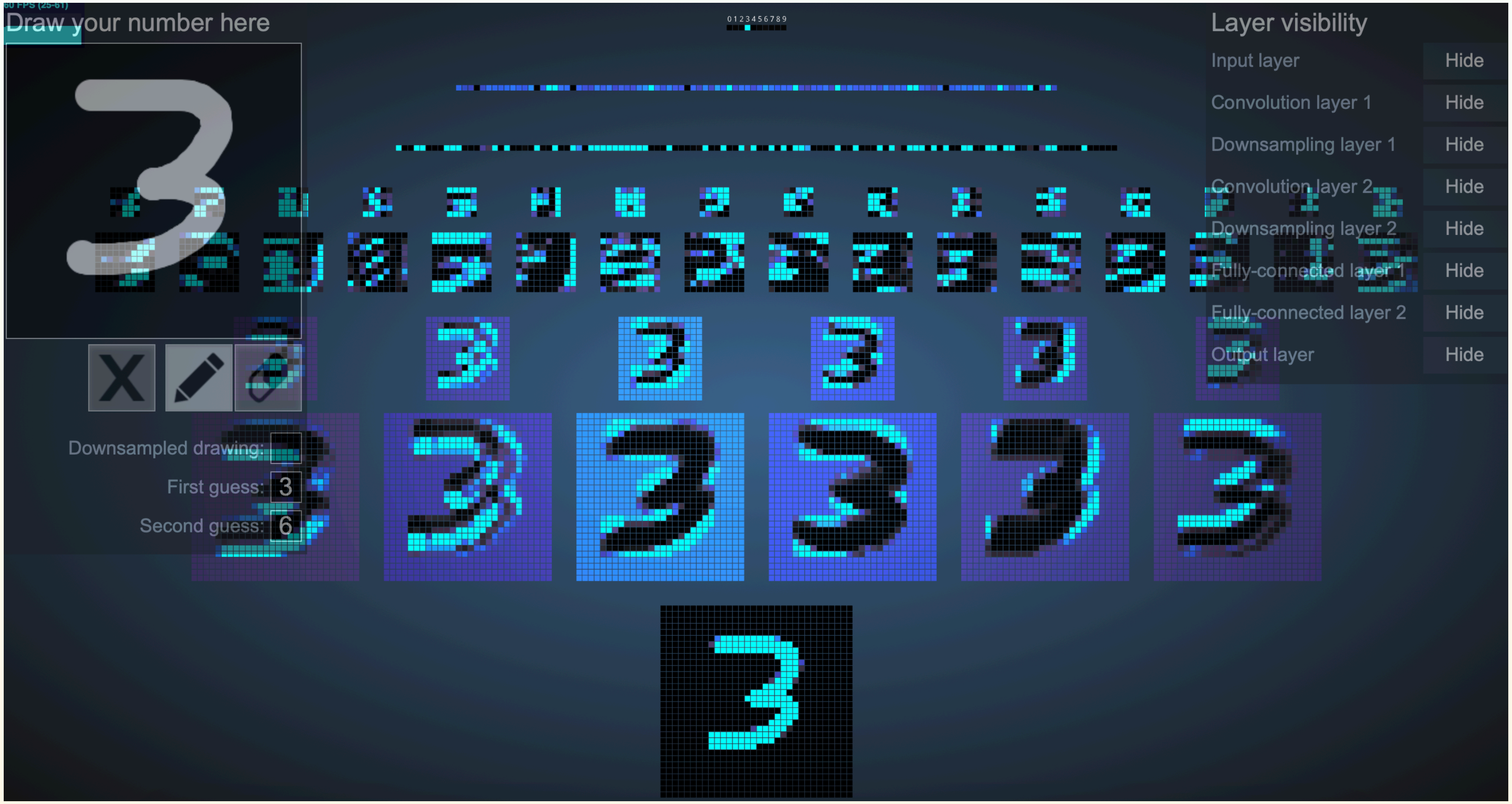


MNIST dataset

- Very early CNN ("the" CNN)
- Shows typical features of also modern classification CNNs: (pooling, pixel dims → feature dims, …)



Unboxing: we can directly visualise the filters



Try yourself: https://adamharley.com/nn_vis/cnn/2d.html

A. W. Harley, "An Interactive Node-Link Visualization of Convolutional Neural Networks," in ISVC, pages 867-877, 2015

CNNs are very powerful: fewer parameters

- In general the following statements hold:

R. Wolf

  - The more TPs the higher the risk to overtrain.
  - The larger the training dataset the smaller the risk to overtrain.
  - It is therefore also always possible to reduce the risk of overtraining by increasing the training dataset.
- A procedure that we have not discussed here, since it is irrelevant in particle physics is called *data augmentation*: there one artificially increases the training dataset by turning, stretching, mirroring individual samples of the training dataset.

- CNNs break down the large number of input pixels with a **much** smaller number of parameters
- Abstraction and pooling maintain expressivity

CNNs are very powerful: effective training sample

- In general the following statements hold:

  R. Wolf

  - The more TPs the higher the risk to overtrain.
  - The larger the training dataset the smaller the risk to overtrain.
  - It is therefore also always possible to reduce the risk of overtraining by increasing the training dataset.
- A procedure that we have not discussed here, since it is irrelevant in particle physics is called *data augmentation*: there one artificially increases the training dataset by turning, stretching, mirroring individual samples of the training dataset.

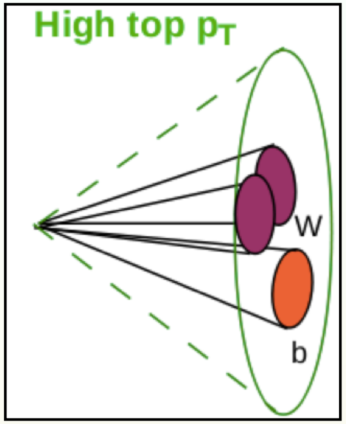
- The filter weights are shared for all j

$$y_{j\alpha} = \theta\left(\sum_{\beta}^{N_F}\sum_{i}^{N_k}\omega_{i\alpha\beta}\, x_{I(j,i)\beta} - T_\alpha\right)$$

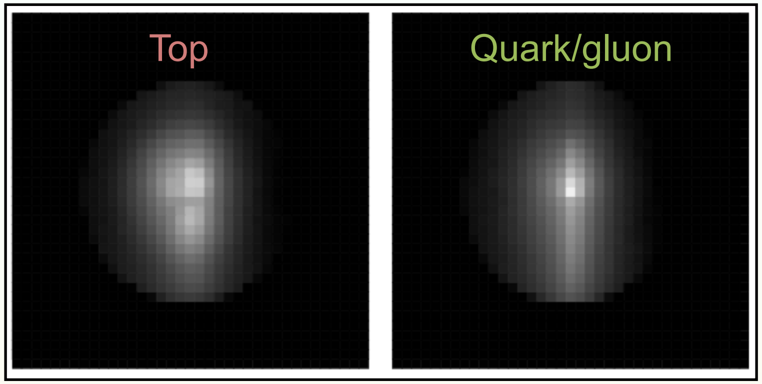
- They are trained for **every** y_j :
  - ω 'see' (sample size * number of pixels) training examples

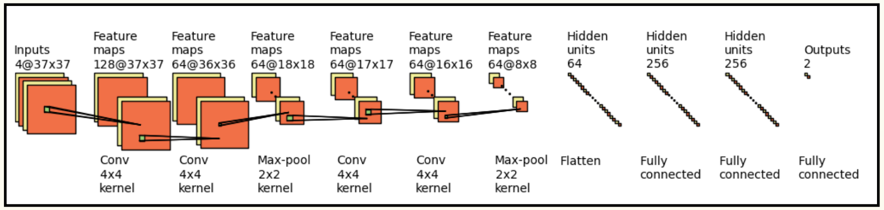
    Millions
- There are (almost) always **multiple** benefits from using the structure of the data

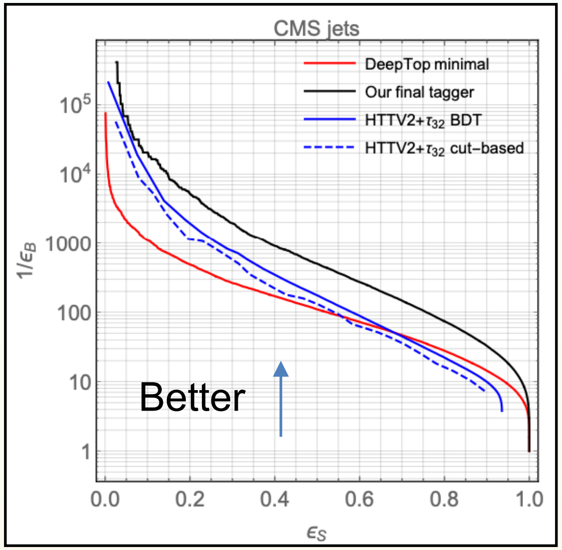
Physics examples: jet tagging



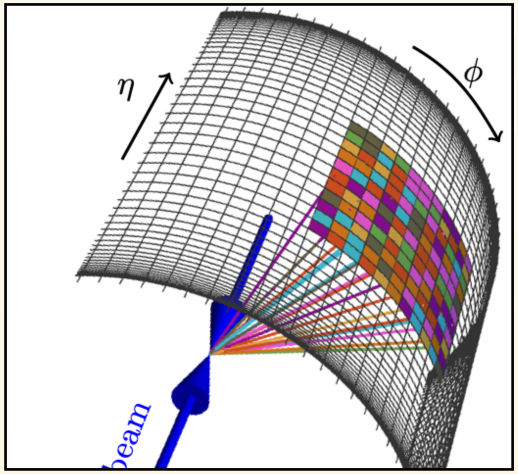
arxiv:1803.00107
(and many others)



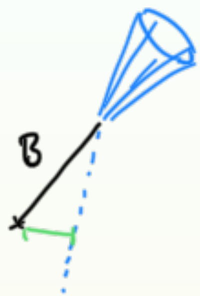


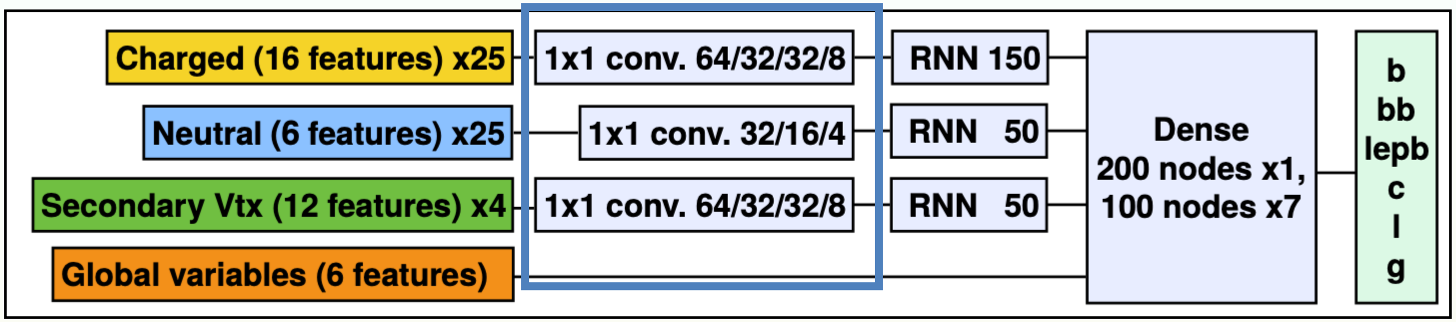


- Identifying origin of a jet very useful for many analyses
- Treat the jet deposits (e.g. in the calorimeter) as an image
- Performance gain over high-level variables

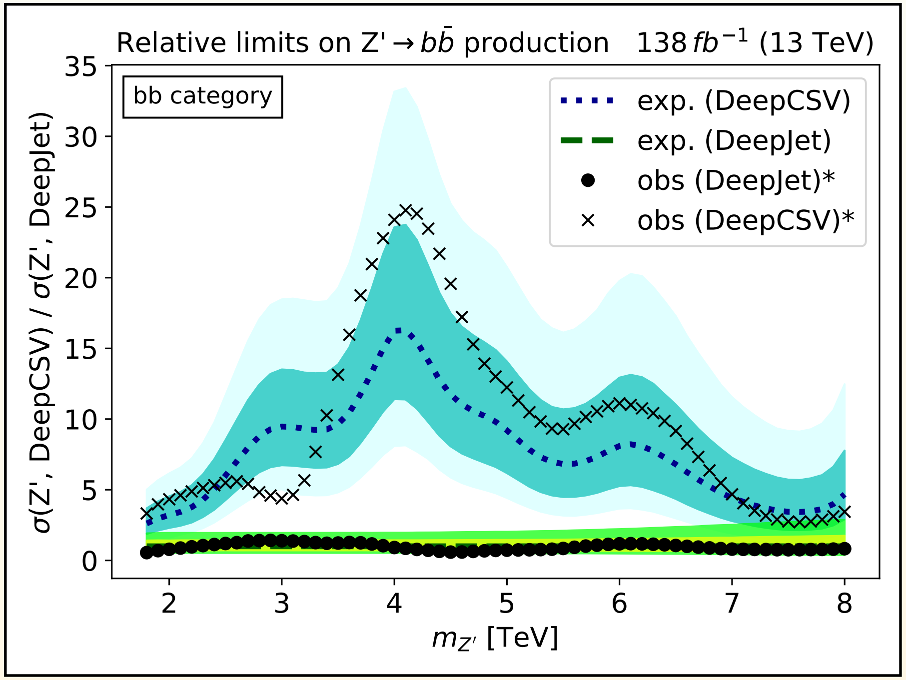


Structure matters: CNNs are not just for images





- Interpret all reconstructed **particles** in the jet as individual 'pixels' in a 1D image
- Pre-process using 1D 'CNNs'
  - Translation equivariance → particle equivariance
  - Enabled to use **all** jet constituents for the first time
  - Enormous performance gain in particular at high momentum
- Standard tagger in CMS
  - >>100 analyses



- **Gain ≈ up to decades more data taking!**

now ETP
arxiv:2008.10519

Summary

- Feed-forward NN can be powerful classifiers directly for analysis

- With great power comes great responsibility
  understand the inputs, their correlations, and the network response to them e.g. through Taylor expansion and beware of out-of-distribution effects

- Understanding and utilising the structure of the data is key

- CNN architectures combine translation equivariant feature detection, abstraction and pooling of information

- Stay tuned for next time:

"Attention is all you need"
featuring
"Everything is a graph"