

FliK Modul 2020

Convolutional Neural Networks

Steffen Seitz, Marvin Arnold & Markus Fritzsche

Prof. Ronald Tetzlaff

Dresden, 19-23.10.

Prequel

ML from a bayesian point of view

Prior:

Wahrscheinlichkeitsverteilung der Größe des Effekts, bevor man Daten untersucht

Posterior:

$$P(\text{Effekt}|\text{Daten}) = \frac{P(\text{Daten}|\text{Effekt}) \times P(\text{Effekt})}{\text{Konstante} *}$$

Falls prior = **Gleichverteilung** („flat prior“) = jeder Wert gleichwahrscheinlich

$$= \frac{P(\text{Daten}|\text{Effekt})}{\text{Konstante} *}$$

Prior → „Basisratenproblem“ in der Psychologie des Entscheidens:

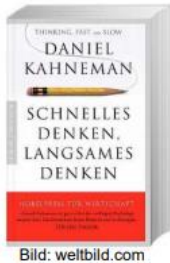


Bild: weltbild.com

= Ergebnis frequentistischer Statistik, aber bessere Interpretation mit **transparenter** Annahme (man weiß nichts über den Effekt, bevor man die Daten sieht)

Konstante dient nur dazu, dass das Ergebnis als Verteilung interpretiert werden kann; sorgt dafür, dass „Wahrscheinlichkeit über alle möglichen β -Werte“ = 1

Likelihood

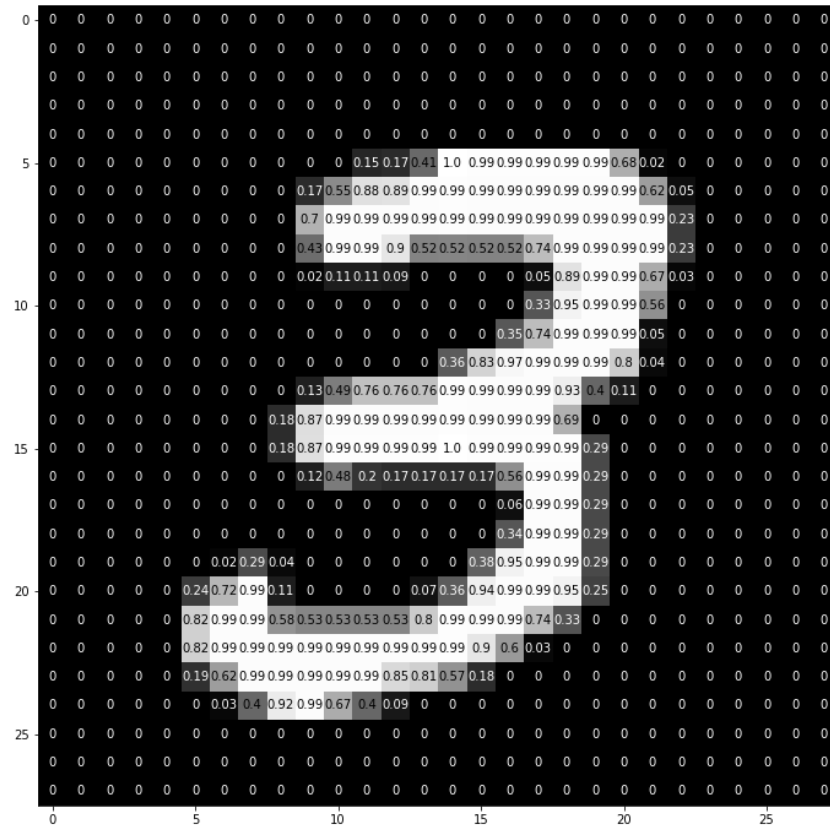
Neural Networks actually calculate (maximize) the **Likelihood**. Given Data → get **model** that best **explains** the data

In this context, the „Effekt“ is equivalent to the class we would like to detect and **prior** can be seen as the **network architecture**.

If enough **data** is available (i.g. not the case), any **prior** will **vanish** out. But „**bad**“ prior **don't exist**. (some just dont help much... :-))

© Michael Höfler TUD

Image Data



For a computer, images are just numbers. But there is one **prior** from years of image analysis that we can make use off:

Neighbour pixels are often correlated!

Priors Images

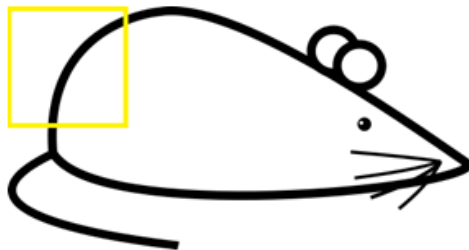


```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 04
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 37 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 23 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 24 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 89 44 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 34
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
```

In image analysis there is an algorithm that uses this effect to investigate specific areas of an image. This function is called (discrete) **convolution**. In classic CV research this was utilized together with (predefined) detectors or **filters**.



Original image



Visualization of the filter on the image

Example: *Sobel Filter* (Edge Detector)



Convolutional Neural Networks

Intuition

These filters can be **convoluted** with the **image** to **detect** the **feature** the filter was **designed** to detect by human image experts. In general a **filter** is just a matrix of numbers and the (discrete) convolution is simplified to a simple **vector-matrix** product – simple multiplication and summation.

Example: *Detecting a specific line*



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

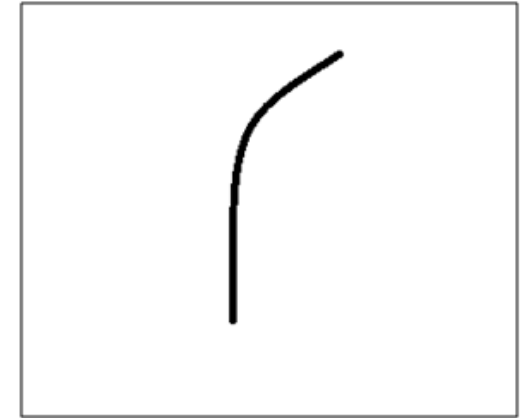
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = $(50 \cdot 30) + (50 \cdot 30) + (50 \cdot 30) + (20 \cdot 30) + (50 \cdot 30) = 6600$ (A large number!)

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter



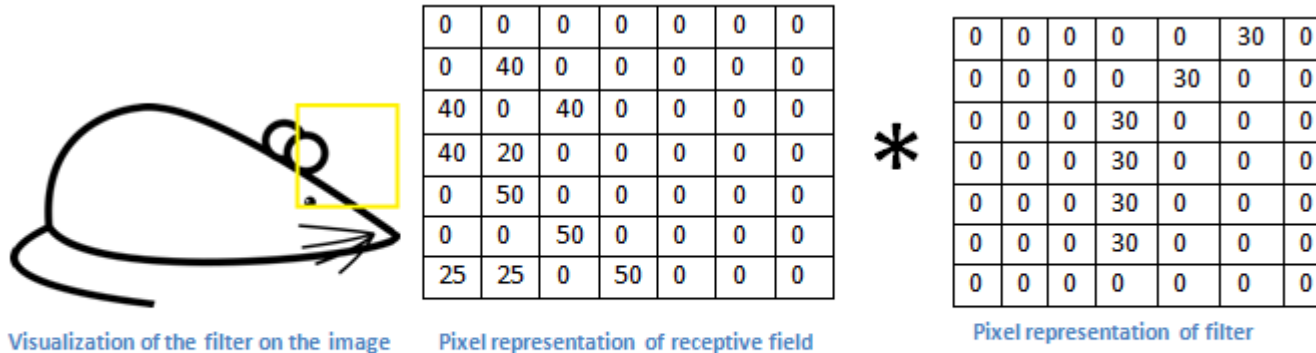
Visualization of the filter on the image

Basically, in the input image, if there is a **shape** that generally resembles the curve that this **filter is representing**, then all of the multiplications summed together will result in a **large value**!

Convolutional Neural Networks

Intuition

Now let's see what happens when we move our filter.



The value is much **lower**! This is because there was **nothing** in the image section that **responded** to the curve detector filter.

Multiplication and Summation = 0

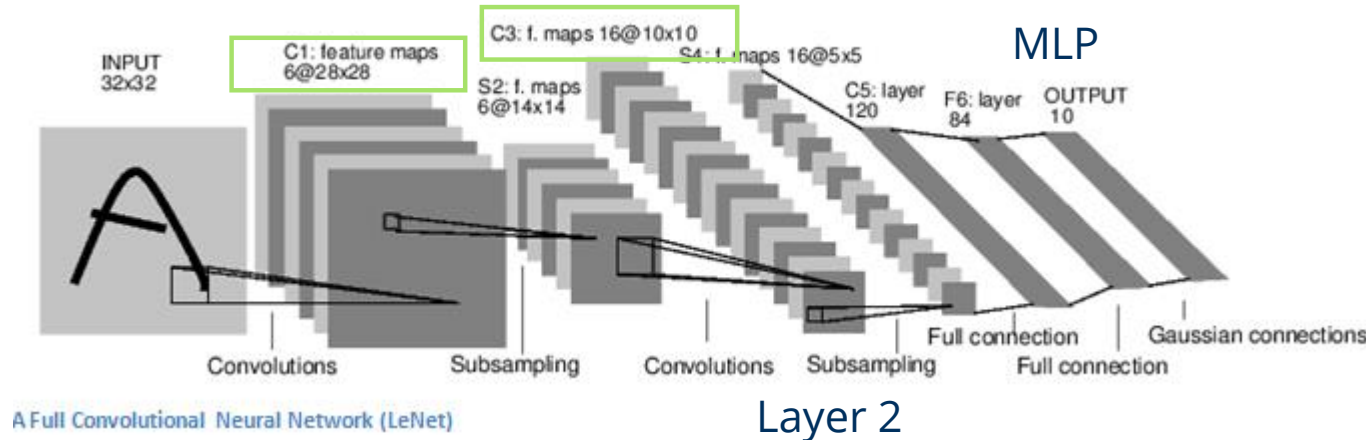
All previous filters we have seen, have been **handcrafted**. But the design of these filters is very **time consuming** and sometimes **not** even **possible** for human experts.

Idea: Use Neural Networks learn the filter!

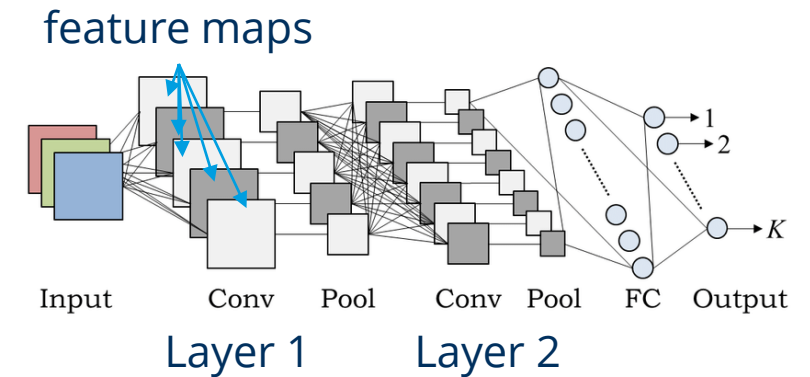
Convolutional Neural Networks

The go to **algorithm** for **images**!

First approach: **LeNet** (Yann LeCun, 1998)



Simple:

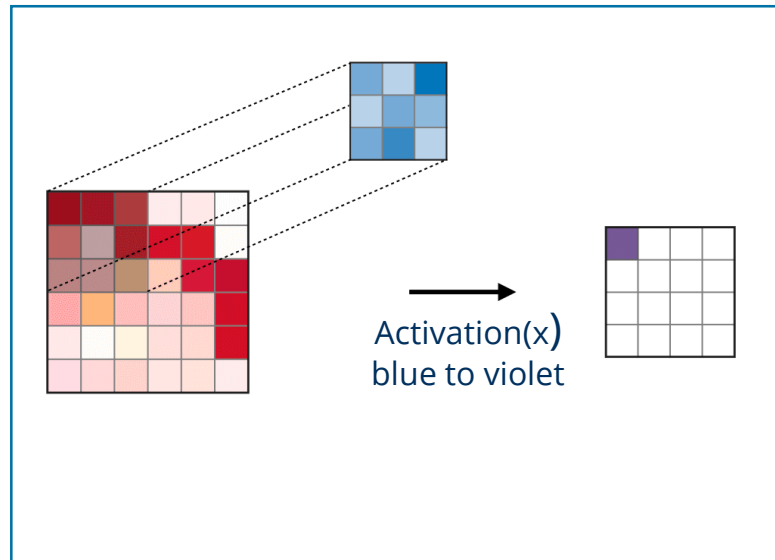


Interpret the **values** of our **filter** as the **weights** from our previous neural networks that we want to optimize. The forwards pass is exactly the already mentioned convolution of this initialized filter with our image. Usually multiple **featuremaps** are initialized and trained. This is **analog** to the **neurons** in the MLP case.

The feed forward path is the already mentioned **sliding convolution** of the filter over the image with two additions:

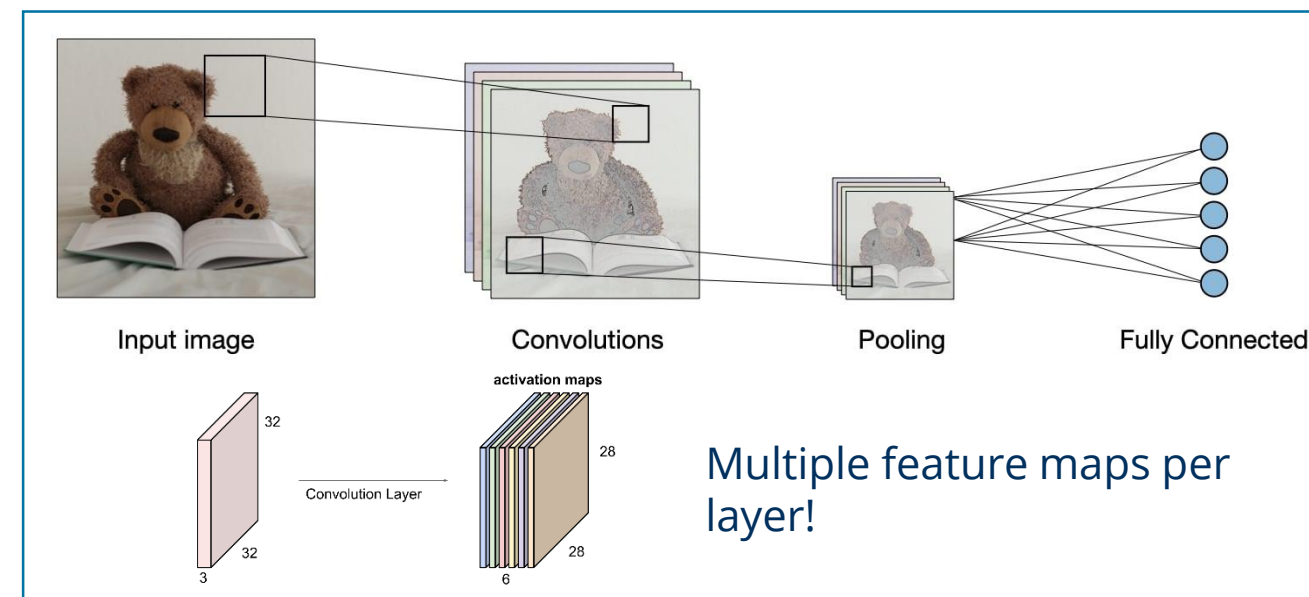
- 1) The **output** of the convolution = x is processed by an **activation function** (e.g. ReLu) before handing to the next layer.
- 2) After each Conv. layer, a **pooling crops** the **receptive field** for the next layer.

Convolution



Pooling

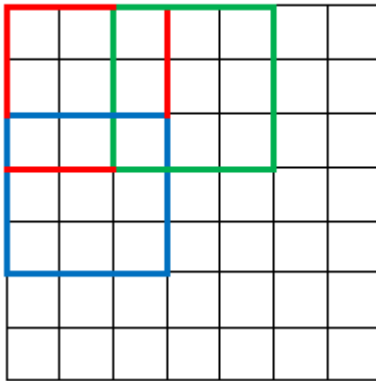
Type	Max pooling	Average pooling
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
Illustration		
Comments	<ul style="list-style-type: none"> Preserves detected features Most commonly used 	<ul style="list-style-type: none"> Downsamples feature map Used in LeNet



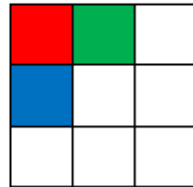
Convolutional Neural Networks

Stride & Padding

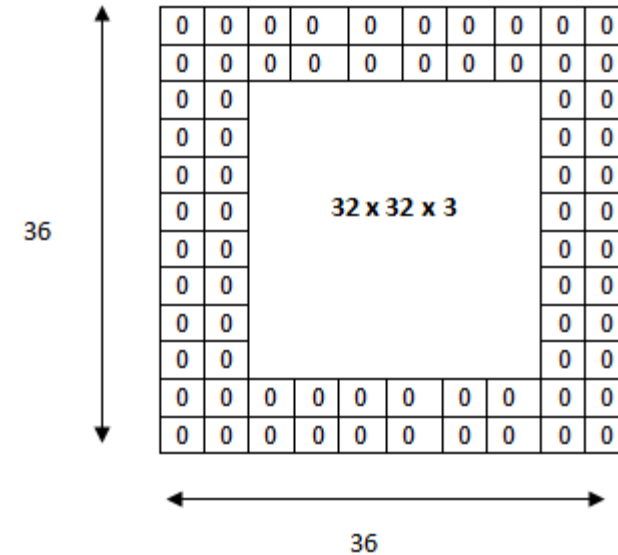
7 x 7 Input Volume



3 x 3 Output Volume



For a convolutional or a pooling operation, the stride S denotes the **number of pixels** by which the window moves after each operation



Zero-padding denotes the process of adding PP zeroes to each side of the boundaries of the input.

Cifar10

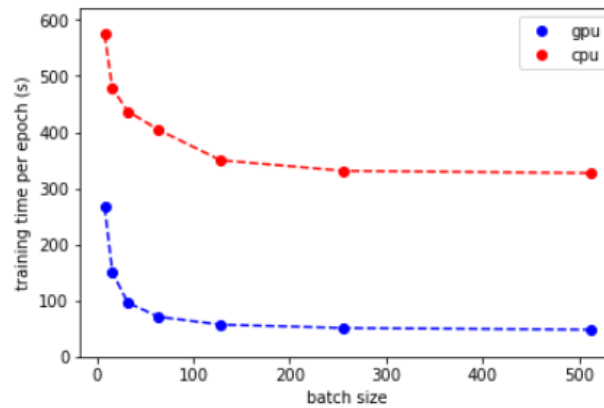
Our previous datasets are far too **easy** for Convnet so solve! Here, we will analyse Data from the **Cifar10** Dataset, a very well known image recognition **benchmark**.

For brevity we will start with only 2 classes in the Dataset: **Cats** and **Dogs**.

Training **time** has become a **limiting factor** in AI research today. But thankfully, **subsets** of **samples** in a training batch can be calculated in **parallel** on a GPU:

Remember:

We update weights after each batch!



32x32 pixel RGB images
10 classes
50,000 training images
10,000 test images

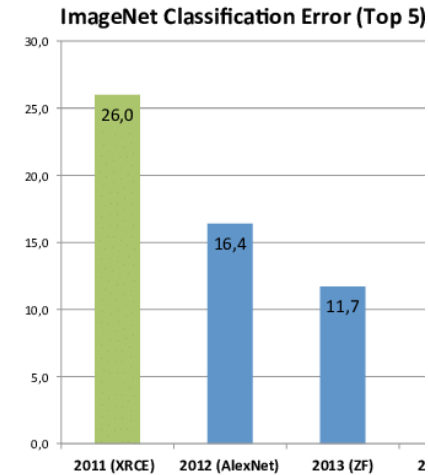


5. Exercise

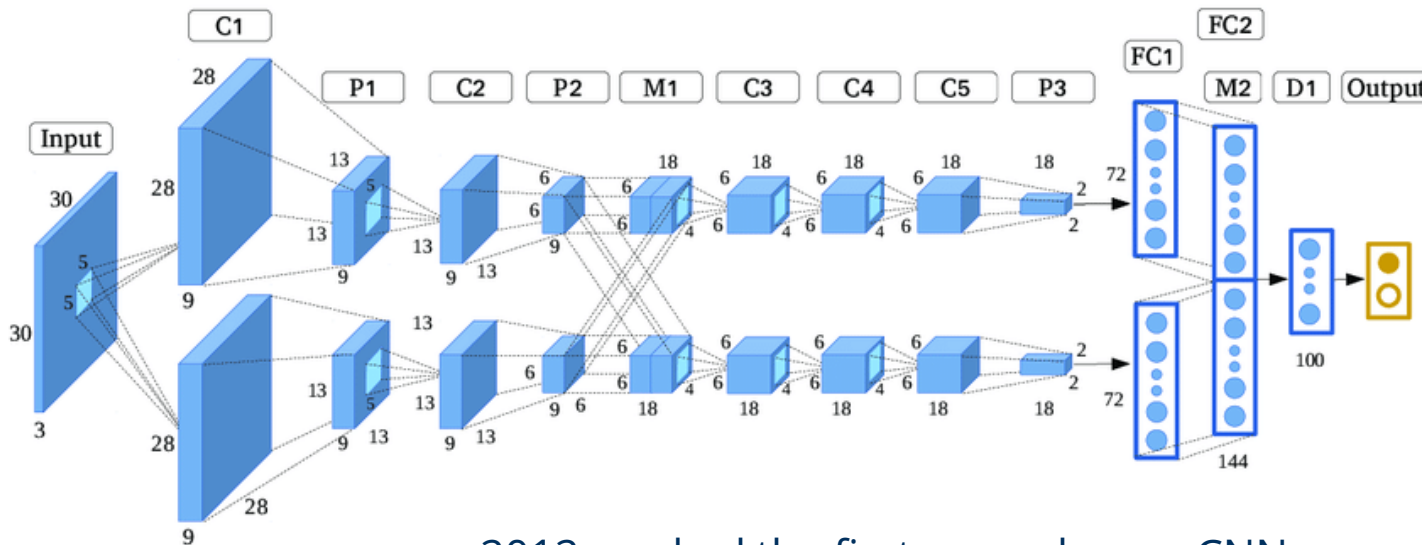
Let's train our first CNN classifier for cats and dogs with Keras!

ImageNet

Another famous dataset for benchmarking image recognition algorithms. Tiny **ImageNet** dataset has 100,000 images across 200 **classes**.



First remarkable CNN: **AlexNet** 2012



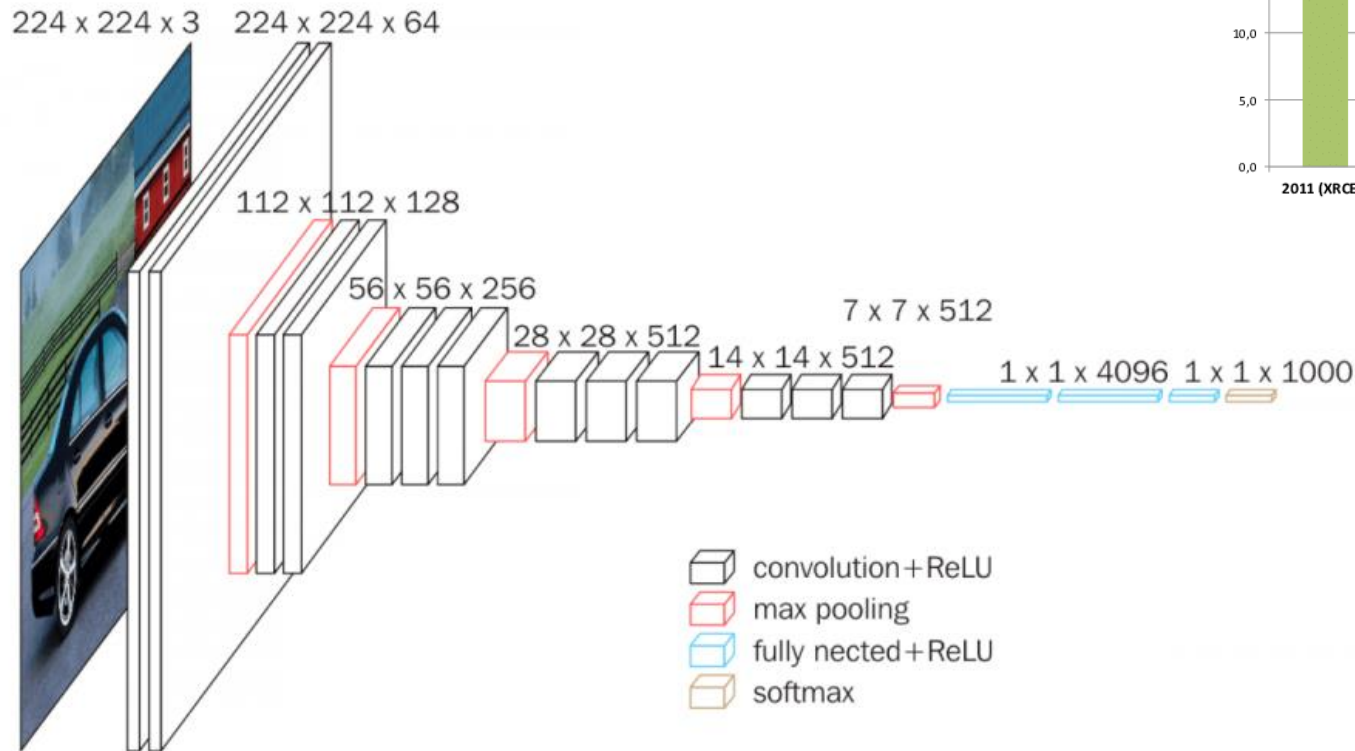
Comparison to LeNet:

- 2 Paths possible
- X – connections
- Filters number increasing with layer

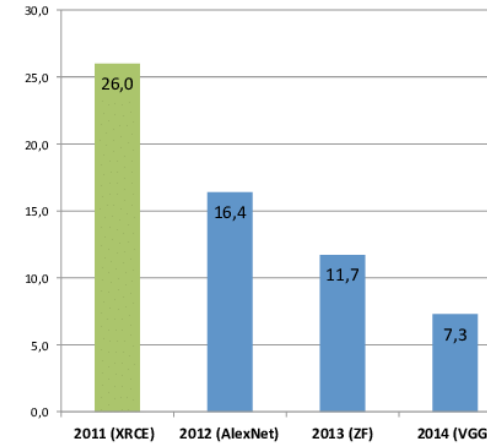
2012 marked the first year where a CNN was used to achieve a top 5 test error rate of 15.4%.

VGG16

Winner of 2014 with 19 Layers!



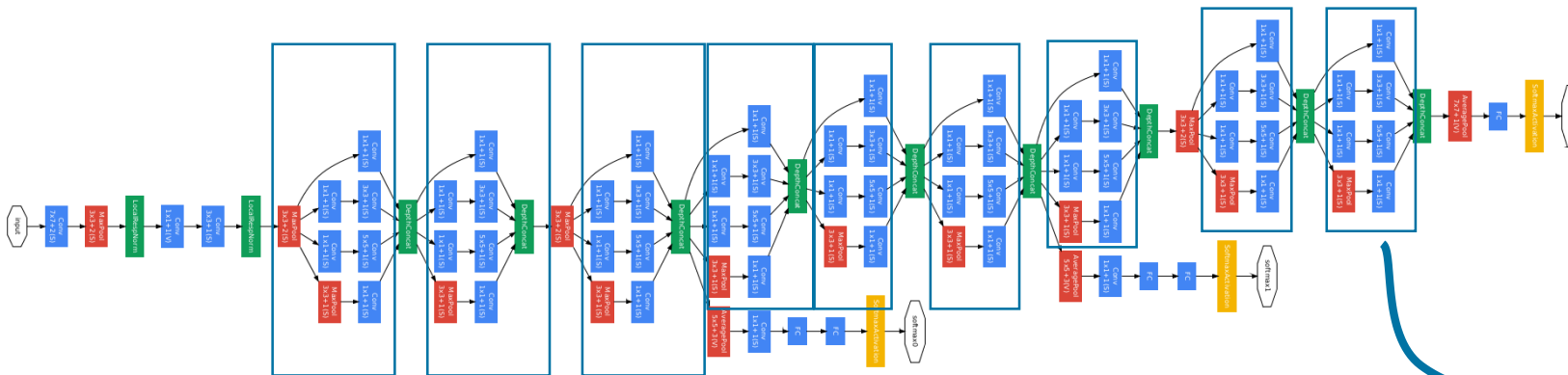
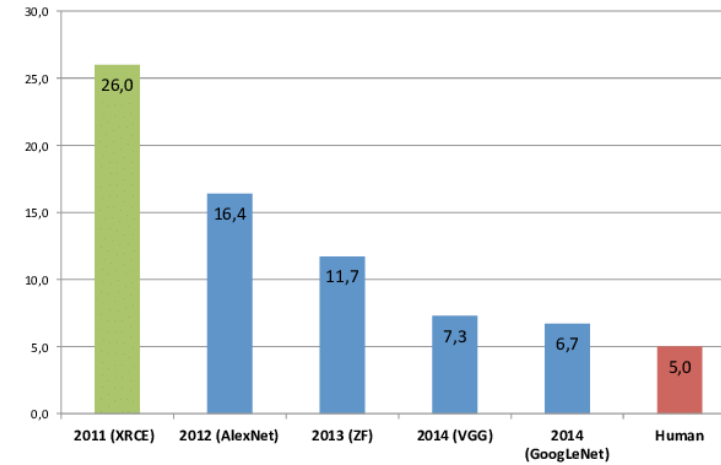
ImageNet Classification Error (Top 5)



Google LeNet (Inception v1)

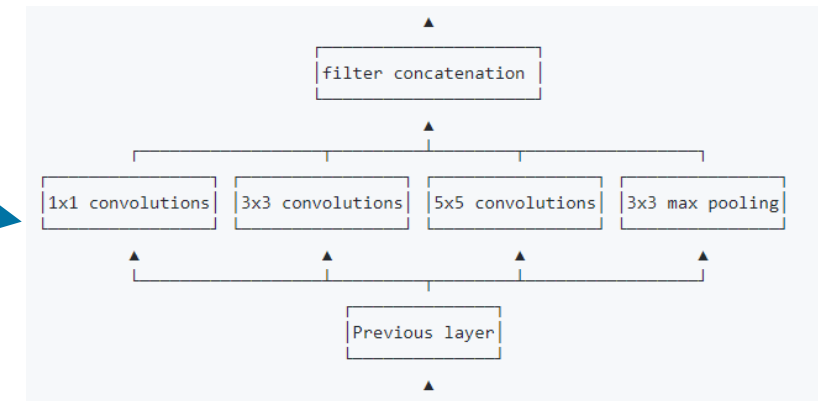


ImageNet Classification Error (Top 5)



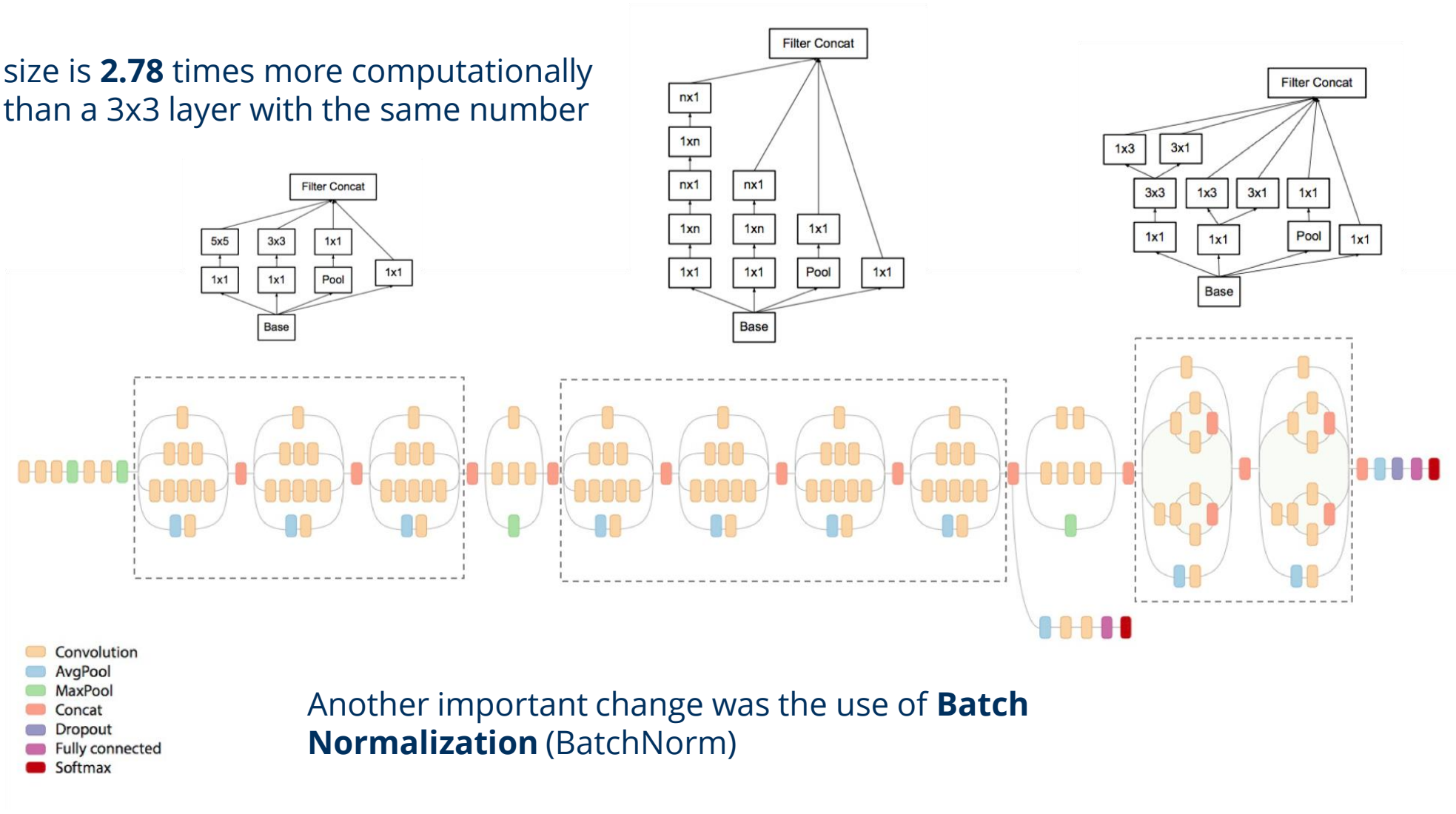
9 Inception modules, **22 layers** (blue)

Inception module



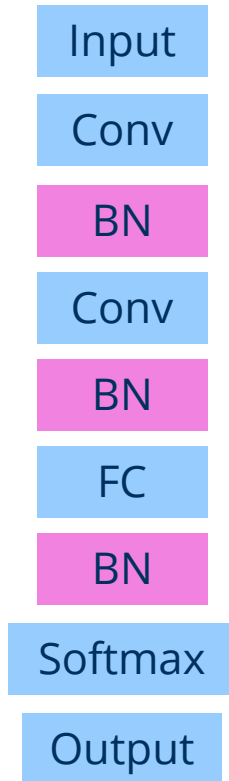
Google LeNet (Inception v2,3)

A 5x5 filter size is **2.78** times more computationally **expensive** than a 3x3 layer with the same number of filters!

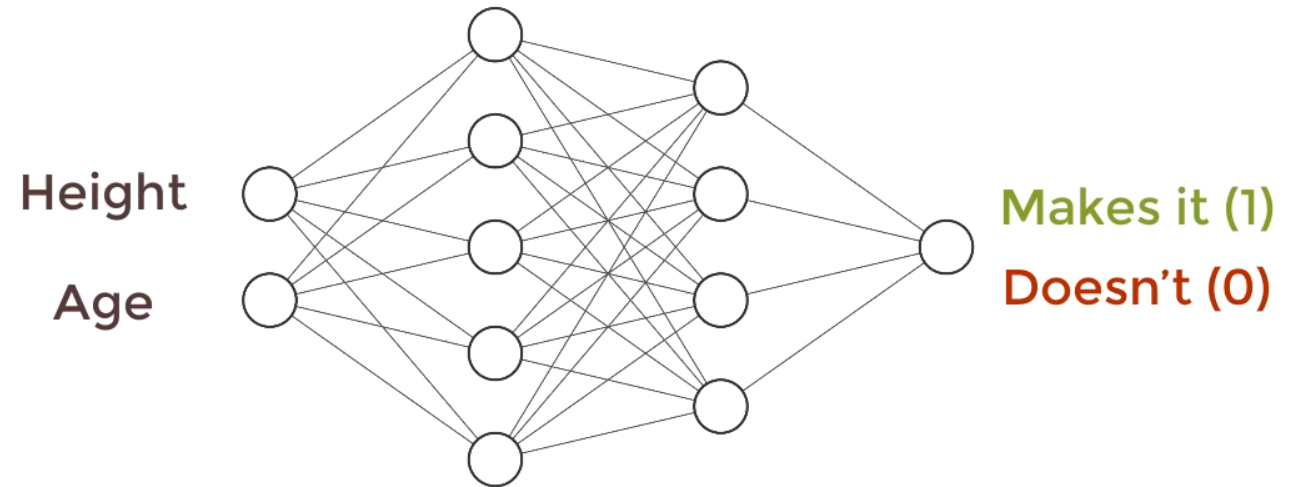


Another important change was the use of **Batch Normalization** (BatchNorm)

Batch Normalization

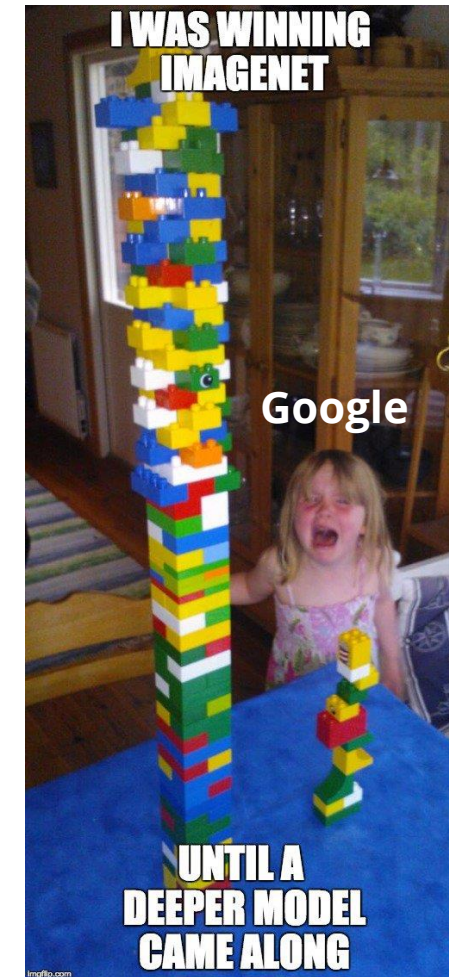
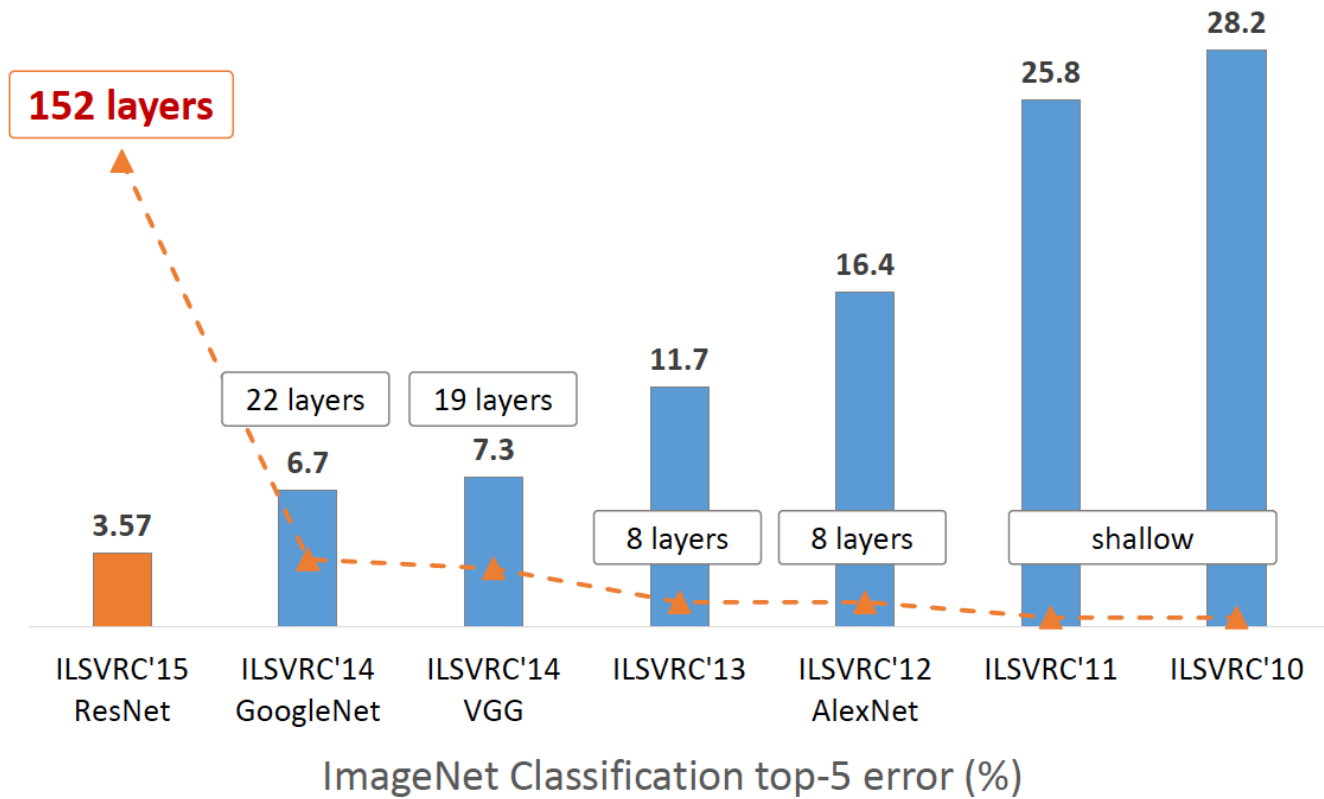


NBA Predictor



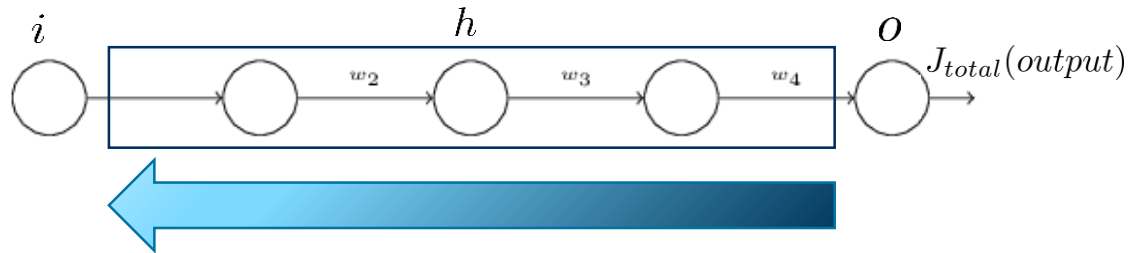
6. Exercise

Let's regularize our first CNN classifier with BatchNorm!

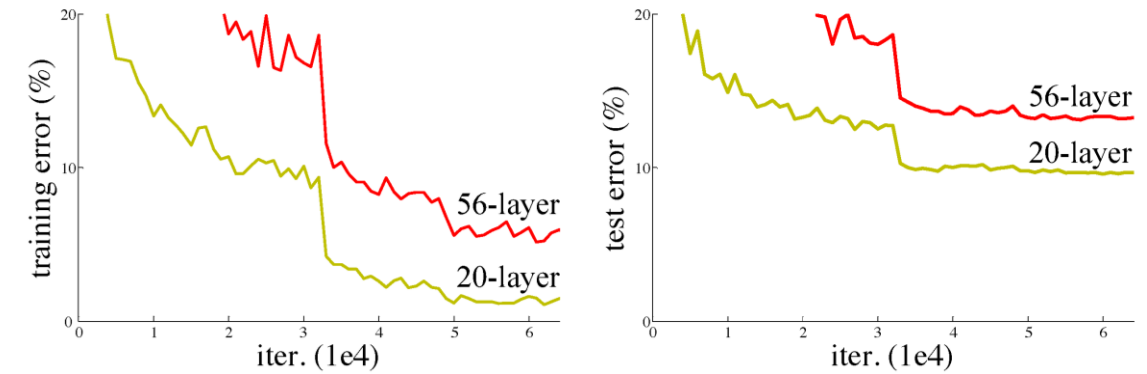


Vanishing Gradient

Is learning better networks as easy as stacking more layers?



Cifar10 for deeper VGG Network

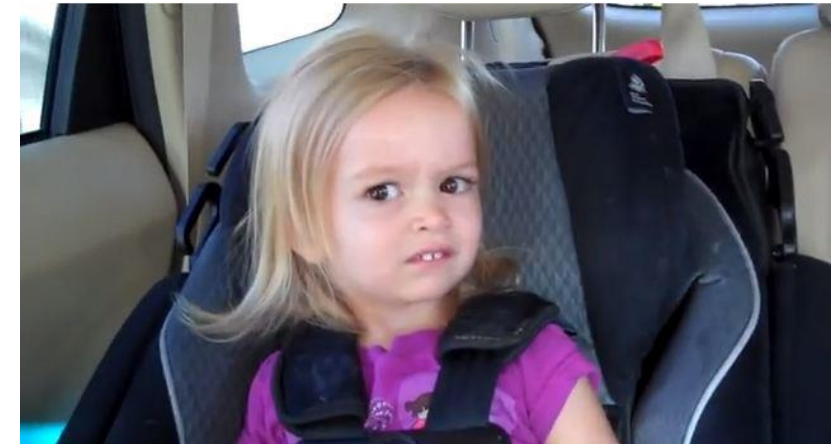


$$\frac{\partial J_{total}(output)}{\partial w_1} = \prod_{k=1}^K \underbrace{(w_i \sigma'(z_k))}_{\leq 0.25}$$

$K \dots$ Number of layers

mostly: $|w_i| < 1 \rightarrow$ Glorot/He: $\sigma(w) = 1/n$
 $\mu(w) = 0$

Regularization is also supposed to keep them small!



ResNet

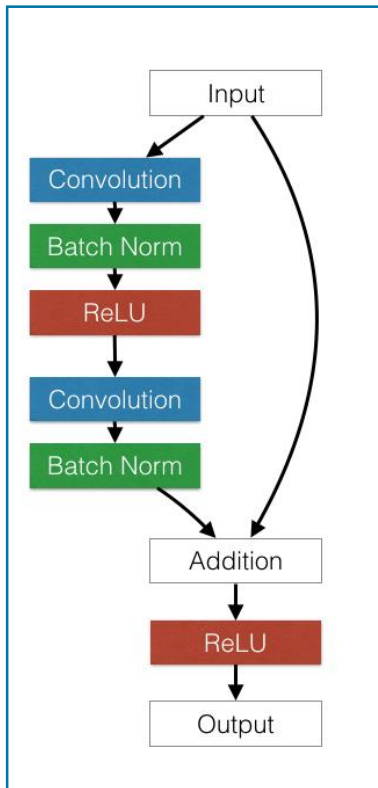
AlexNet, 8 layers
(ILSVRC 2012)



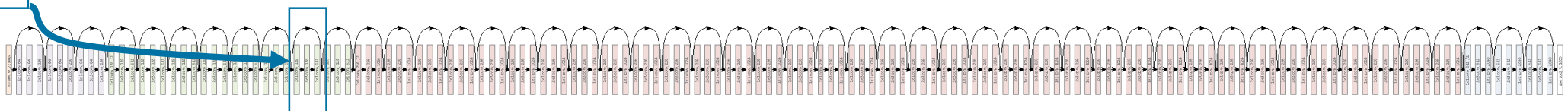
VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)



ResNet introduced **skip connections** that allowed a flexible gradient highway. It is not a fix to vanishing gradient but it allowed models more **resilient**, so larger models can be trained!



ResNet, **152 layers**
(ILSVRC 2015)

7. Exercise

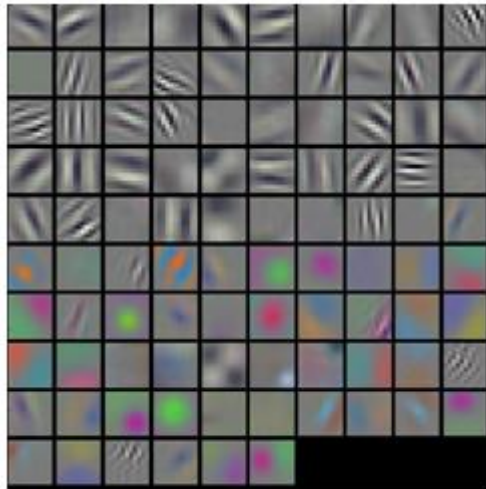
Let's train a ResNet an other CNN Classifiers on the full Cifar10 dataset!

Feature Visualization

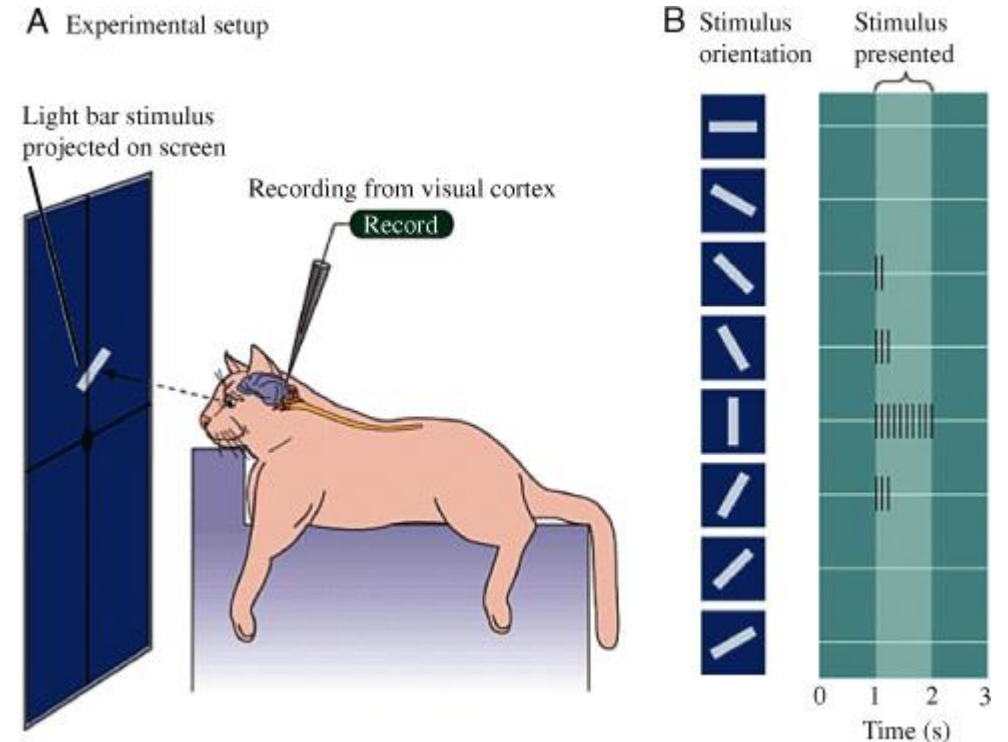
Activation Map plotting

Now that we know the basics, we can **plot** the output of the **feature maps activation** given an input image. This is a test how the respective layer is reacting and give a hint what the network might have learned while training.

The results are fascinating, especially if you compare them to the experiment from **Hubel** and **Wiesel** 1962.



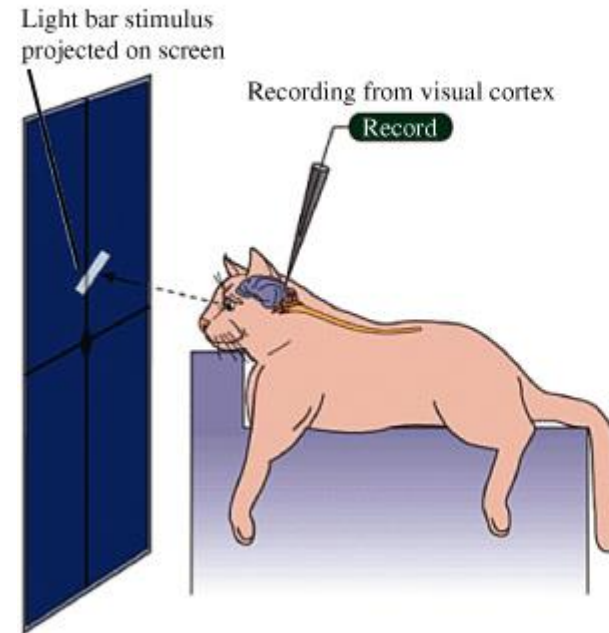
Visualizations of filters



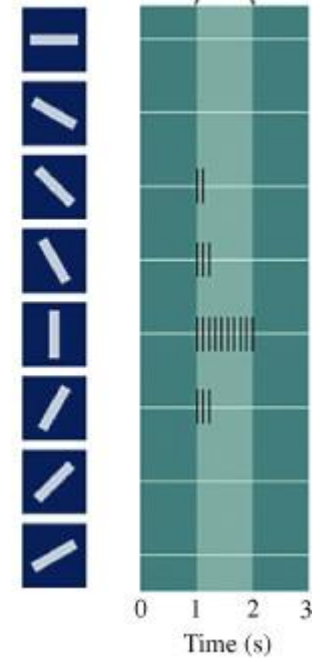
fascinating experiment by Hubel and Wiesel in 1962



A Experimental setup



B Stimulus orientation



8. Exercise

Let's plot our activation maps of the CNN classifier with Keras!

Feature Visualization

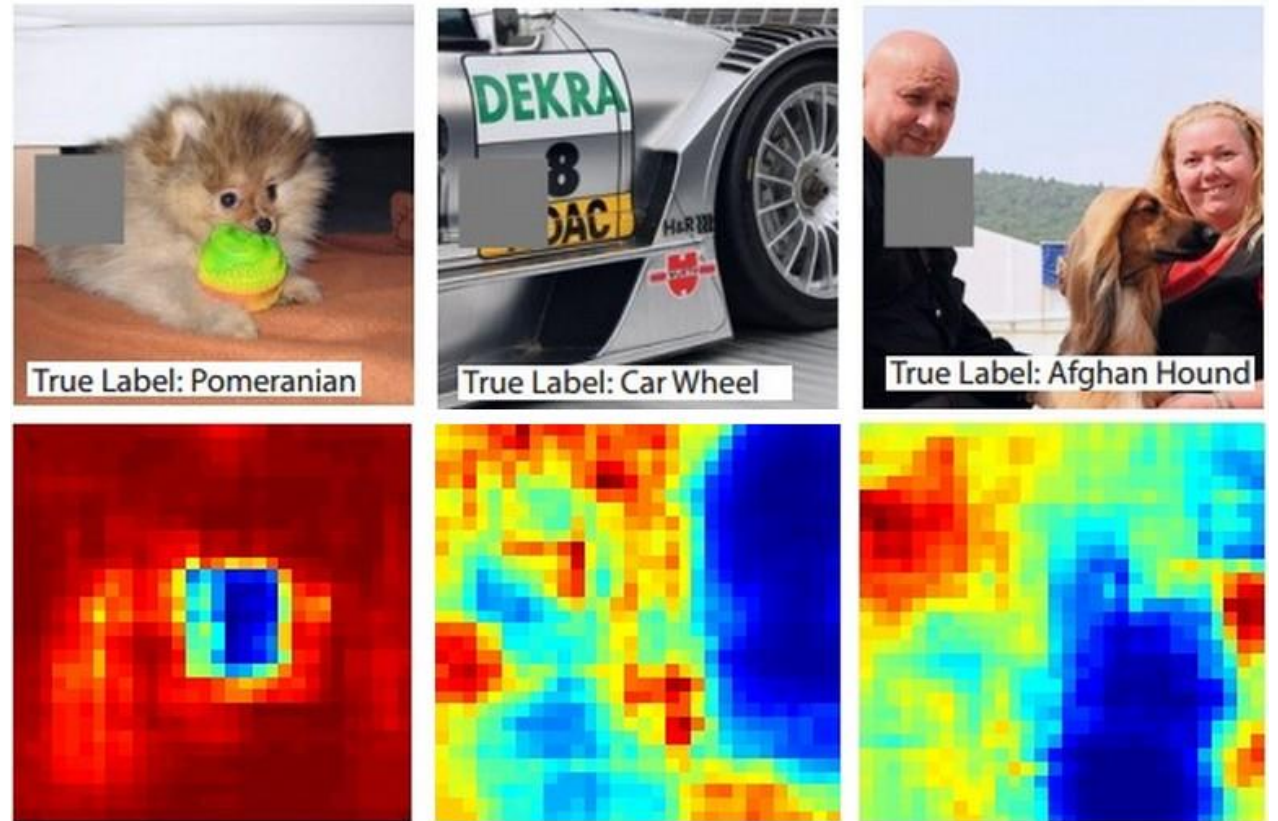
Occluding parts of the image

What part of an Image is **relevant** to a Neural Networks decision?

A native approach to this is the **occlusion** of specific sections in an Image.

Here we **iterate** over regions of the image, set a patch of the image to be all **zero**, and look at the **probability** of the class.

Matthew Zeiler:
Visualizing and Understanding Convolutional Networks (2013)

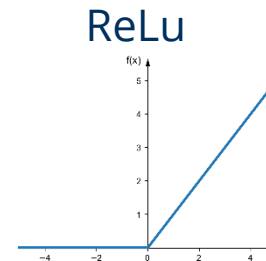
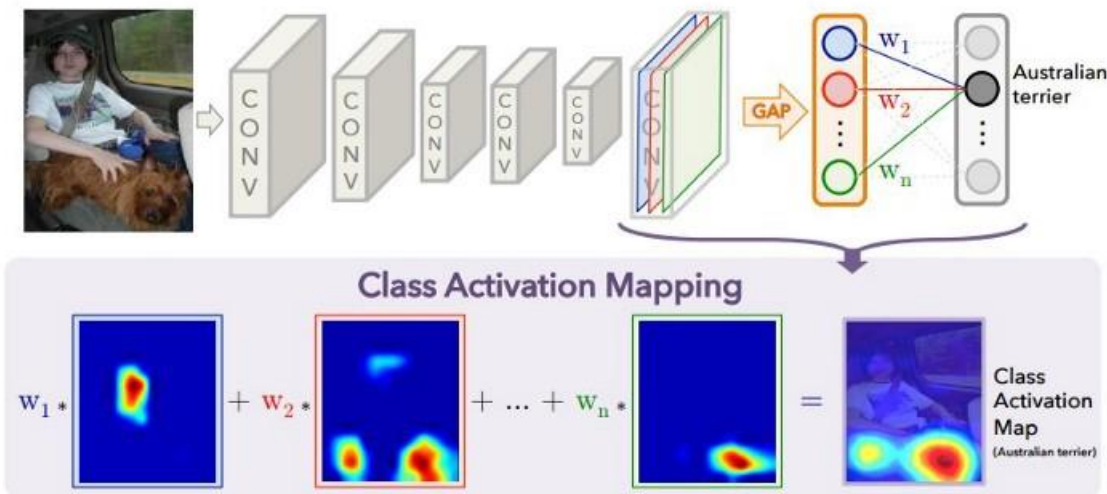


Class Activation Maps (CAM)

CAM and GradCam

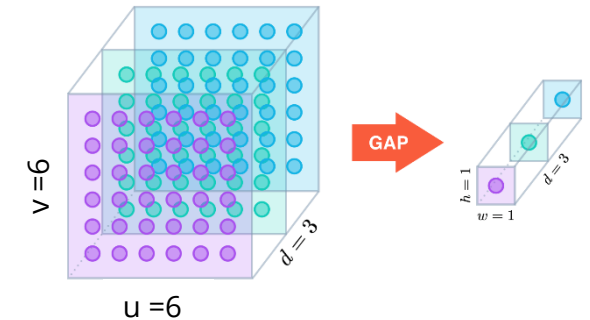
We have seen that Convolutional feature maps **retain** spatial information, which is **lost** in fully-connected layers (MLP). **Last Conv Layer** can be thought as the important **features** for the classification.

- Classificate from these feature Maps after using **GAP** and **sum** the **weighted positive** feature Maps =CAM (make them positive by ReLu(x))
- Compute the GAP pooled gradient of the last layer



ReLU because we are only interested in the **features** that have a **positive influence** on the class of interest

Global Average Pooling (GAP)



Number of FMaps $\rightarrow K$

Weight (Cam) $\rightarrow \alpha_k^c$

Feature Map $\rightarrow A^k$

$$L_{Grad-CAM}^c \sim \sum_{k=1}^K \alpha_k^c A^k = \text{CAM}$$

$$\alpha_k^c = \frac{1}{uv} \sum_{i=1}^u \sum_{j=1}^v \frac{dy^c}{dA_{i,j}^k}$$

GAP of Gradient (GradCam)

$$L_{Grad-CAM}^c = \text{ReLU} \left(\sum_{k=1}^K \alpha_k^c A^k \right) = \text{GradCAM}$$

9. Exercise

Lets train a feature visualisation algorithm! (GradCAM)