TU Clausthal

A Convolutional Neural Network for Image Classification of Cats and Dogs

Status update

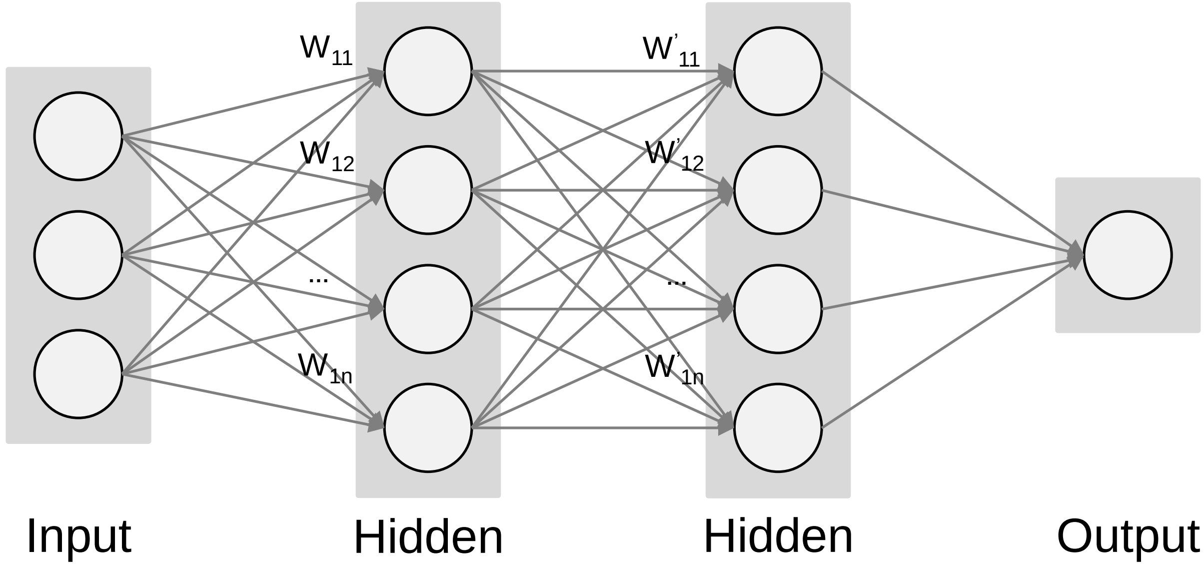
Aditya Raj, Sören Schleibaum
Institut für Informatik

Structure

- Neural Nets (NN)
- Convolution NN (CNN)
- Hyperparameters
- Problem
- Evaluation

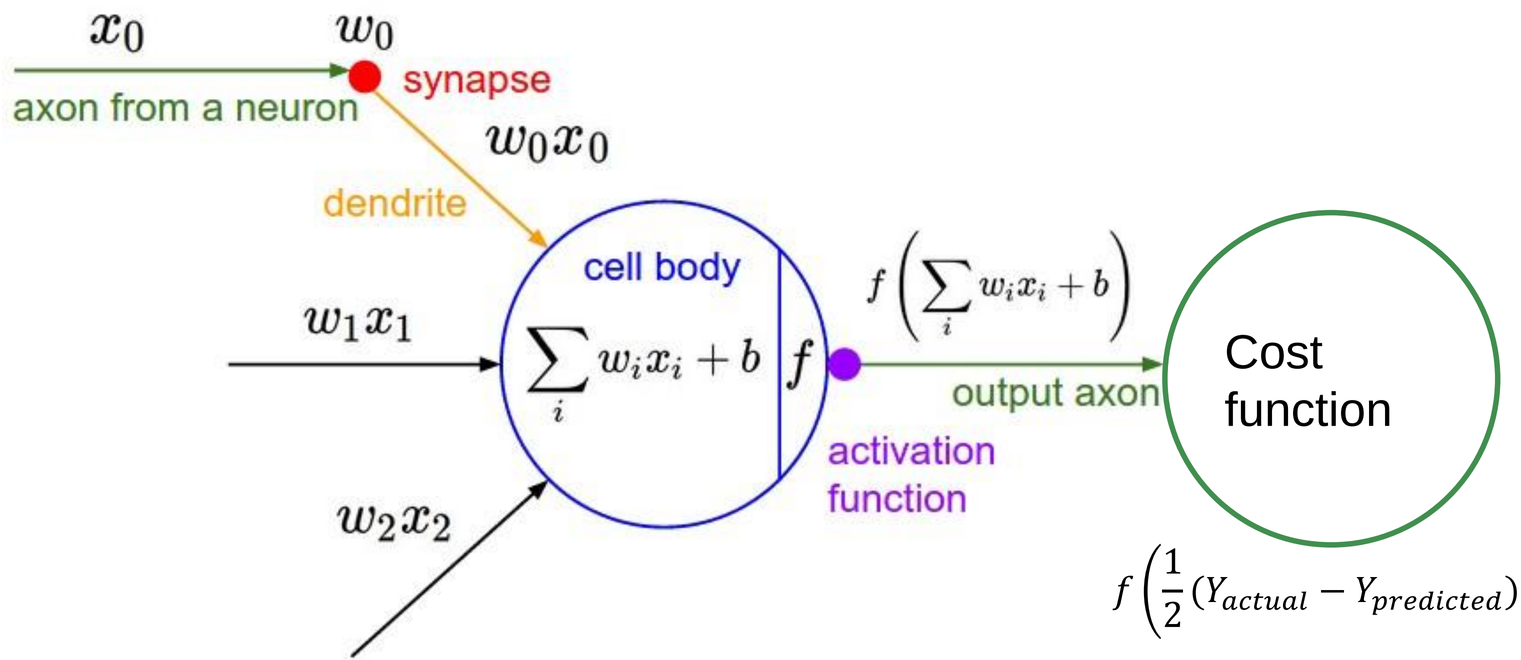
NEURAL NETS

Introduction – Neural Nets



Quelle: http://cs231n.github.io/convolutional-networks/

Neuron Model



$$f\left(\frac{1}{2}(Y_{actual} - Y_{predicted})^2\right)$$

Mathematical view

- Input, weights — $l_0 = X_i$, W = rand()
- Compute Sigmoid (activation function) — $l_1 = Sig(X_i \,.\, W)$
- Measure how much we missed (cost function) — Err = $l_0 - l_1$
- Multiply error by the Sigmoid slope — Δl_1 = Err*$\Delta\big(Sig(Err)\big)$
- Update weights — W = W+($l_0 . \Delta l_1$)

Activation Functions

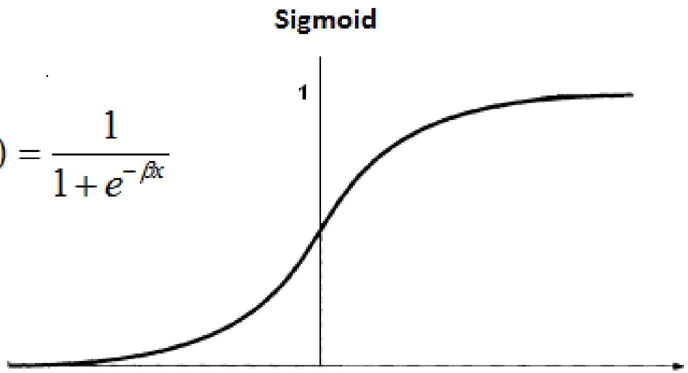
- Non-Linear
  - Sigmoid, tanh

- Continuous but not everywhere differentiable function
  - Relu

Activation functions - Sigmoid

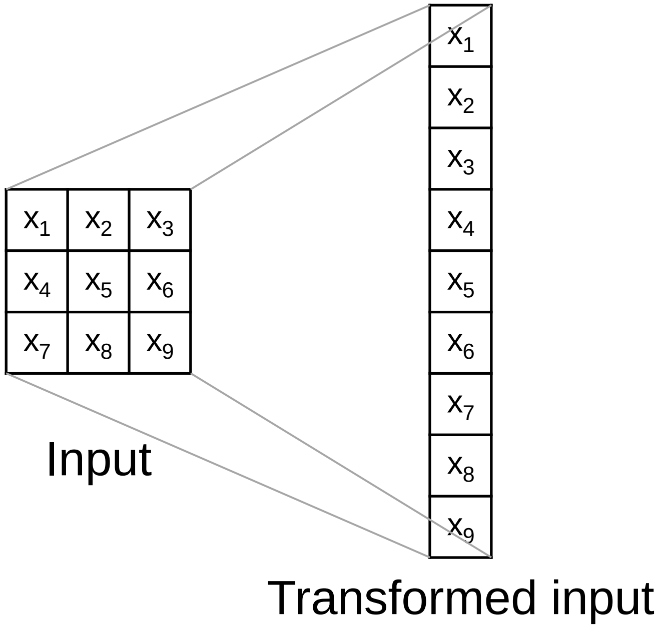
- Motivation
  - Not telling in which direction should we move in.
  - Non-differentiability at certain points

$$f(x) = \frac{1}{1+e^{-\beta x}}$$



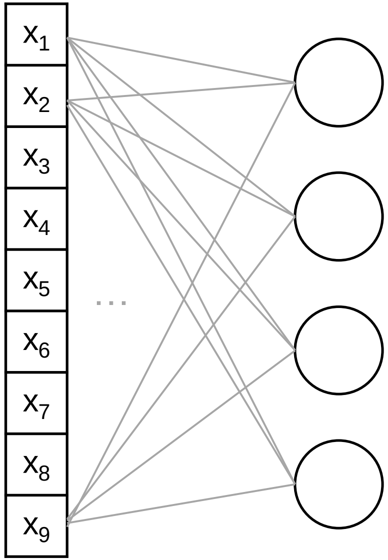
Motivation for CNN

- Number of parameters



Motivation for CNN

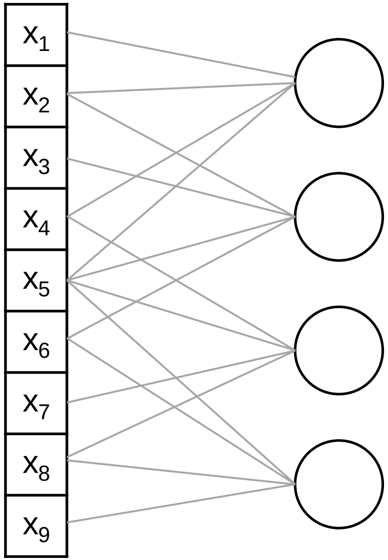
- NN
  - High number of params



x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9

Number of weights: 36

- CNN
  - Lower number of params



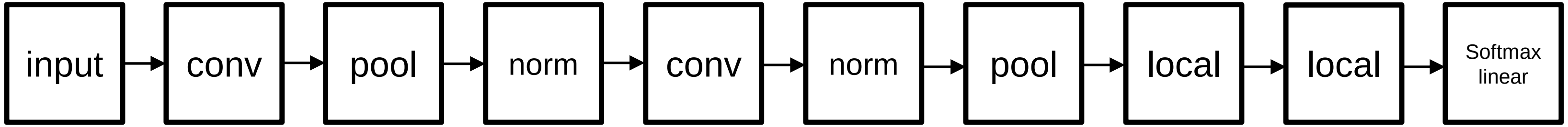
Number of weights: 4

CONVOLUTIONAL NN

TensorFlow

- Developed by Google Brain team
- Use cases
  - Handwritten patterns, image recognition, Word2Vec
- Input data
  - Audio, image, text
- Used techniques
  - Linear classifiers, NN, CNN

Structure of the CNN we used



Input layer

- Image cropping
- Distortions
  - Randomly flipping
  - Randomly changing brightness
  - Randomly changing contrast

input → conv → pool → norm → conv → norm → pool → local → local → Softmax linear

Convolutional layer - Filter

Input

0	0	0	0	0	0	0
0	2	1	0	0	2	0
0	2	2	2	0	2	0
0	1	2	0	2	0	0
0	2	1	2	**1**	**2**	**0**
0	1	2	1	**0**	**0**	**0**
0	0	0	0	**0**	**0**	**0**

Filter

0	0	0
0	2	1
0	2	2

Bias

1

Output

1

*1*0 + 2*0 + 0*0 +*
*0*0 + 0*2 + 0*1 +*
*0*0 + 0*2 + 0*2 +*
1 = ***1***

http://cs231n.github.io/convolutional-networks/

input → **conv** → pool → norm → **conv** → norm → pool → local → local → Softmax linear

Convolutional layer - Parameters

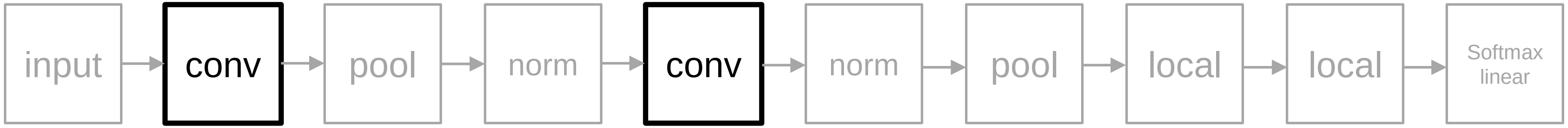
- Input volume size
- Number of filters
- Filter size
- Step size
- Zero padding

Input

0	0	0	0	0	0	0
0	2	1	0	0	2	0
0	2	2	2	0	2	0
0	1	2	0	2	0	0
0	2	1	2	**1**	**2**	**0**
0	1	2	1	**0**	**0**	**0**
0	0	0	0	**0**	**0**	**0**

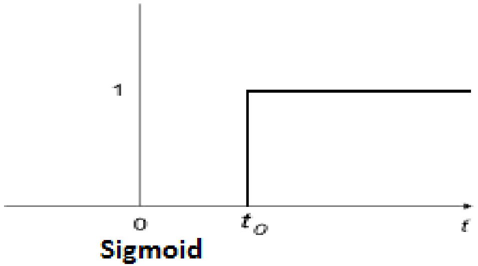
Filter

0	0	0
0	2	1
0	2	2

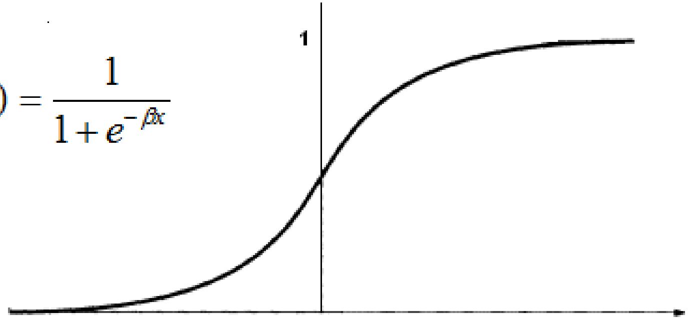


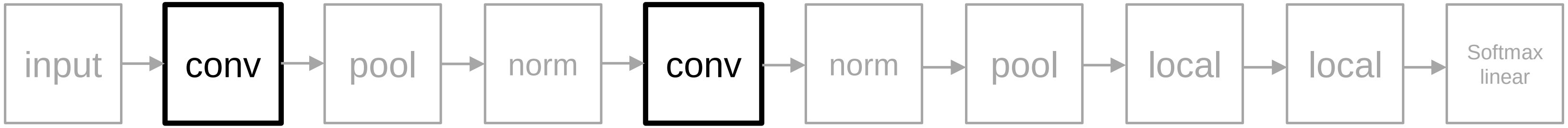
Convolutional layer – Activation function

- Sigmoid
  - Not telling in which direction should we move in.
  - Non-differentiability at certain points



$$f(x) = \frac{1}{1+e^{-\beta x}}$$





Convolutional layer – Activation function

- Rectified linear
  - $Element\ Wise: \max(0, x)$
  - Leaky Relu
  - If x < 0, Output = 0.01x.
  - Non-zero gradient when the input is negative

input → **conv** → pool → norm → **conv** → norm → pool → local → local → Softmax linear

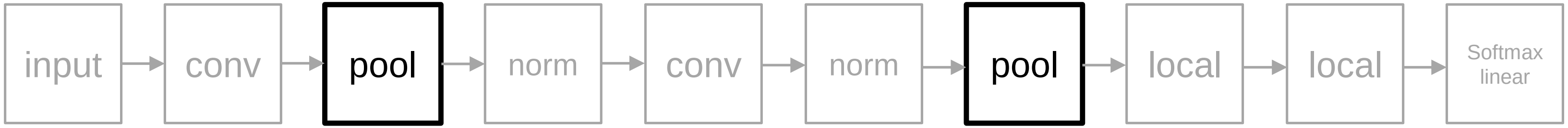
Pool layer – Max pooling

- Reduce the spatial dimension of an image

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

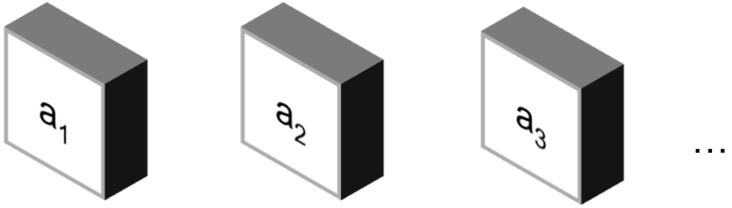
6	8
3	4

http://cs231n.github.io/convolutional-networks/



Norm layer

- 4D-array



- 3D-array of 1D-vector

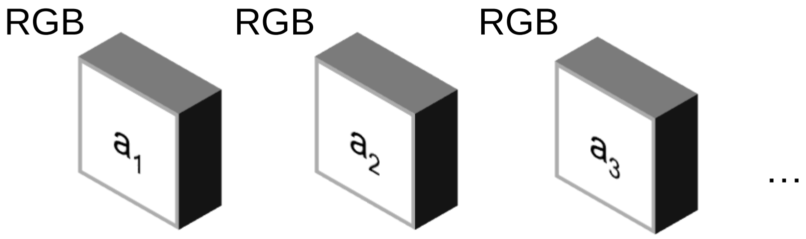
a_1	a_2	a_3	…

- Normalize each element of this1D-vector

input → conv → pool → **norm** → conv → **norm** → pool → local → local → Softmax linear

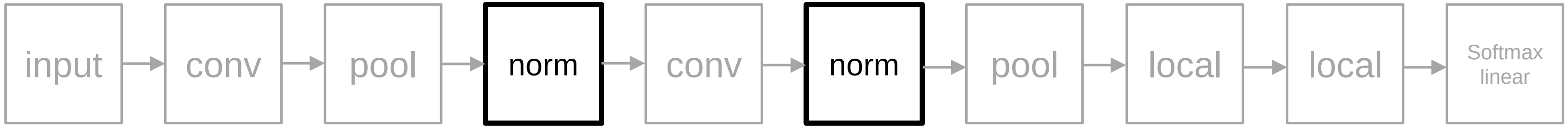
Norm layer

- Normalize each element of this1D-vector



$$a_1 = \left(\left(\frac{R}{\sqrt{R^2+G^2+B^2}}\right), \left(\frac{G}{\sqrt{R^2+G^2+B^2}}\right), \left(\frac{B}{\sqrt{R^2+G^2+B^2}}\right)\right)$$

a_1	a_2	a_3	…

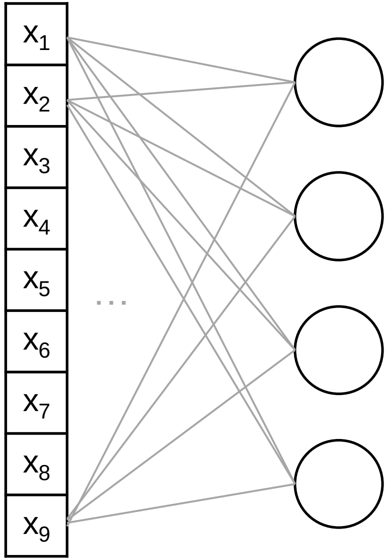


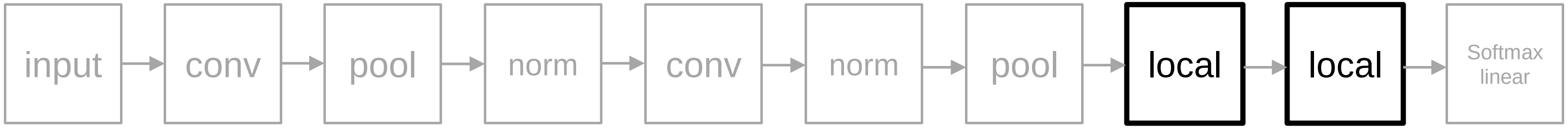
TU Clausthal

Local layer

- Fully connected

x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9





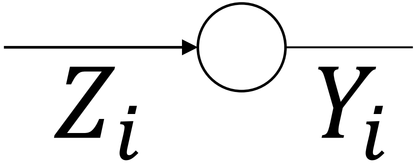
Softmax-linear layer

- Softmax output function
- Cost measure for softmax

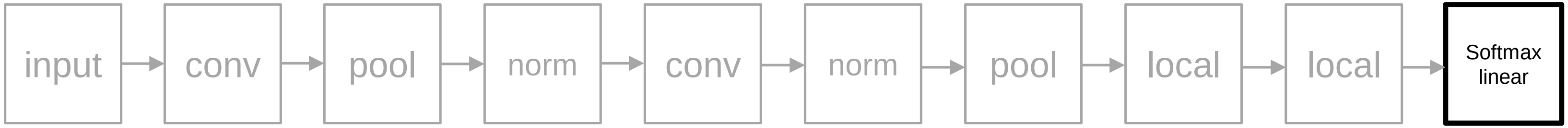
input → conv → pool → norm → conv → norm → pool → local → local → **Softmax linear**

Softmax output function

- Soft continuous version of Max Function
- Forces $\sum(Output\ of\ NN) = 1$.



$$Y_i = \frac{e^{Z_i}}{\sum_{j \in group} e^{Z_j}}$$



Softmax output function

- $\frac{\delta Y_i}{\delta Z_i} = Y_i\,(1 - Y_i)$
- Nice Simple derivative
- Even though Y_i depends of Z_i ,
  - Derivative
    - for an individual neuron
    - of an I/P in respect to O/P is just $Y_i\,(1 - Y_i)$

input → conv → pool → norm → conv → norm → pool → local → local → Softmax linear

Cost measure for softmax

- $C = -\sum_j t_j \log Y_j$
  - Negative log probability of correct answer
- Maximise the log probability of getting answer right

input → conv → pool → norm → conv → norm → pool → local → local → $\boxed{\text{Softmax linear}}$

HYPERPARAMETERS

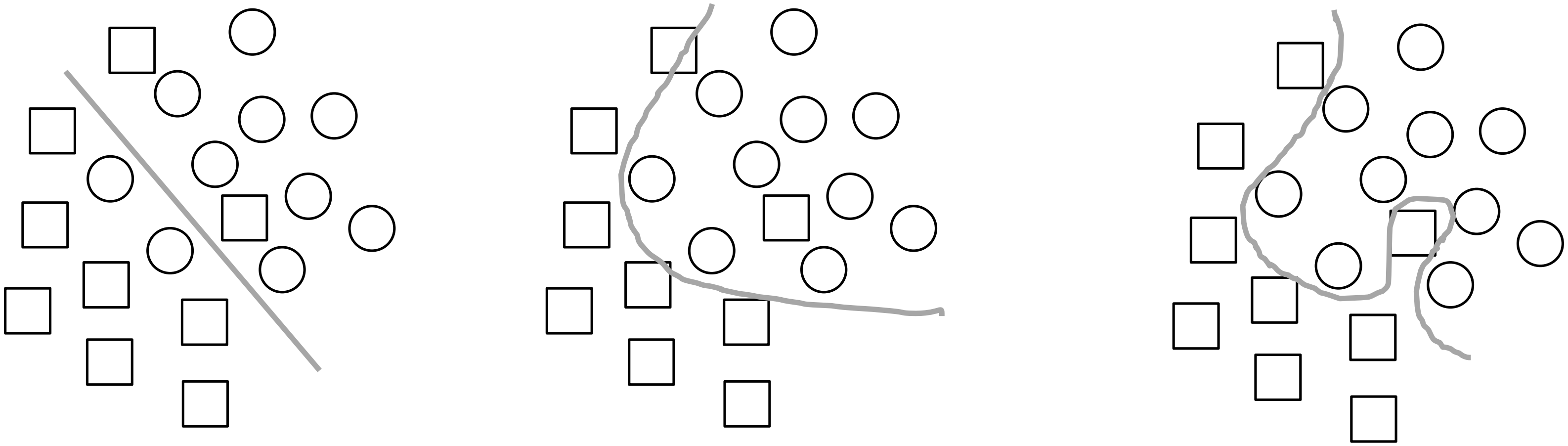
Learning Rate

- How fast the network trains
- High learning rate
  - Convergence or global minimum finding is problem
- Low learning rate
  - High training times

Learning Rate decay

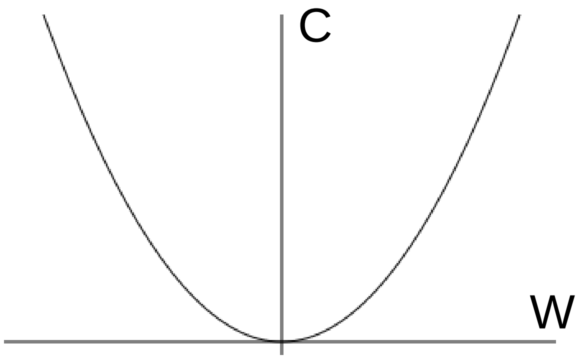
- Learning rate decay means the learning rate decreases over time
  - higher learning rate is well suited to get close to the global minimum
  - small learning rate is better at fine tuning the global minimum
- Several way
  - Exponential decay, reduction by factor of *n*
  - GoogleNet: Function to decrease the learning rate by 4%

Overfitting or Underfitting



Weight Penalty

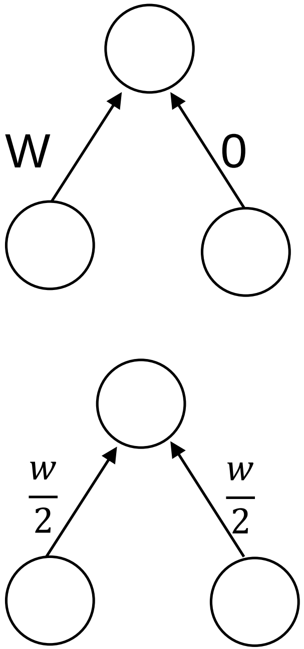
- Adding λ to penalise
  - Keeps weight small
  - Big error derivatives



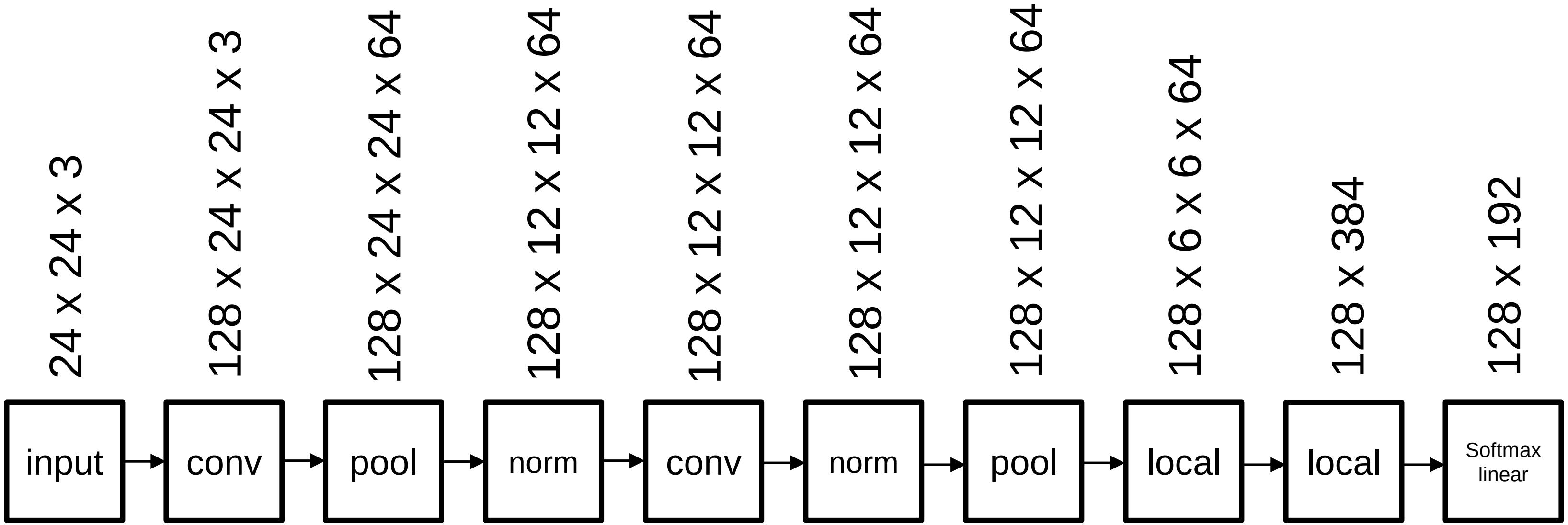
- $C = E + \frac{\lambda}{2}\sum_{i=1} w_i^2$
- $\frac{\partial C}{\partial w_i} = \frac{\partial E}{\partial w_i} + \lambda w_i$
- When $\frac{\partial C}{\partial w_i} = 0$;
  - $w_i = -\frac{1}{\lambda}\frac{\partial E}{\partial w_i}$
  - So, at minimum of cost function if $\frac{\partial E}{\partial w_i}$ is big, the weights are big

Weight Penalty - Advantages

- Preventing network from the weights it does not need
  - Don’t have a lot of weights not doing anything
  - So output changes more slowly as input changes.
- Putting half the weight on each and not on one



Structure of the CNN we used



Output: 128 x 2

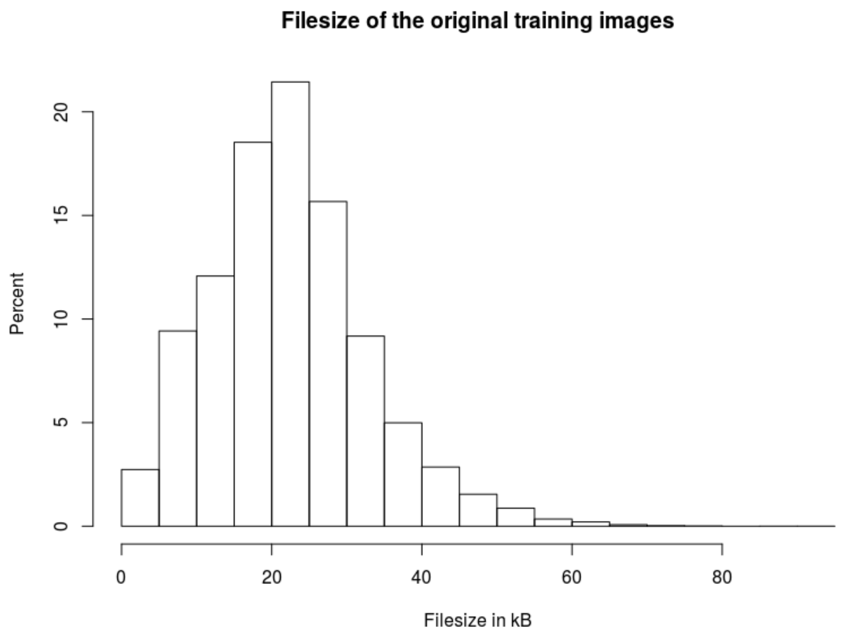
PROBLEM

The data

- Images of cats and dogs
- File format is *.jpg
- Color space is RGB

Data

- 25,000 images
  - 12,500 of dogs
  - 12,500 of cats
- Avg. file size
  - 22.34 kB



Train and test data

- Split data
  - Train data
    - 20,000 images (80 percent)
    - Divided into 5 batches containing 4,000 each
  - Test data
    - 5,000 images (20 percent)

Process images

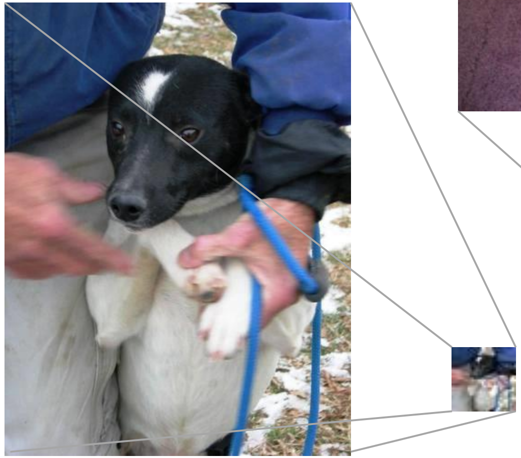
- Resize to 32 * 32 * 3 = 3,072
- Convert to array
  - 25,000 * 3,073

cat10.jpg

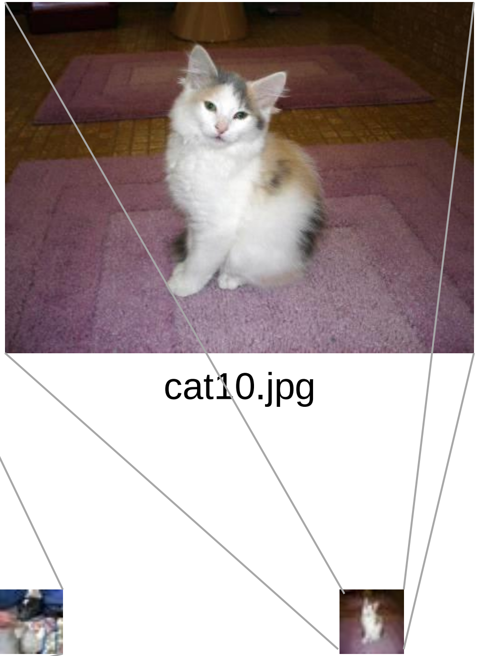
dog1.jpg

Process images

- Resize to 32 * 32 * 3
- Convert to array
  - 25,000 * 3,073
- Example
  - 1; 22; 11; 123; …
  - 0; 256; 255; 0; …

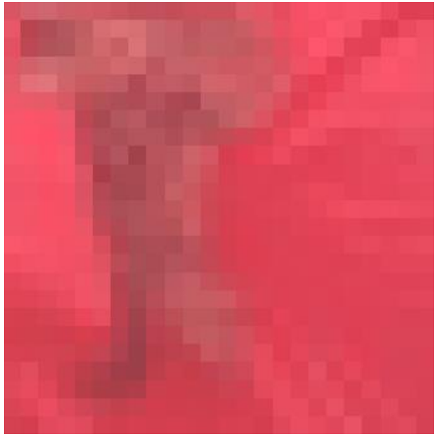


dog1.jpg

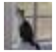


cat10.jpg

Random distorsion

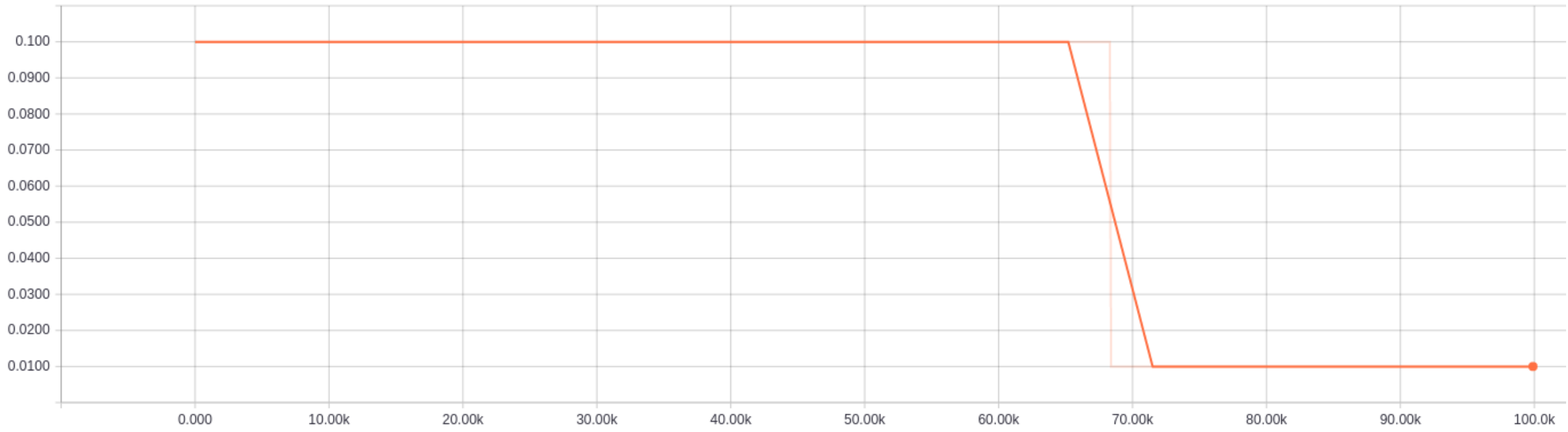





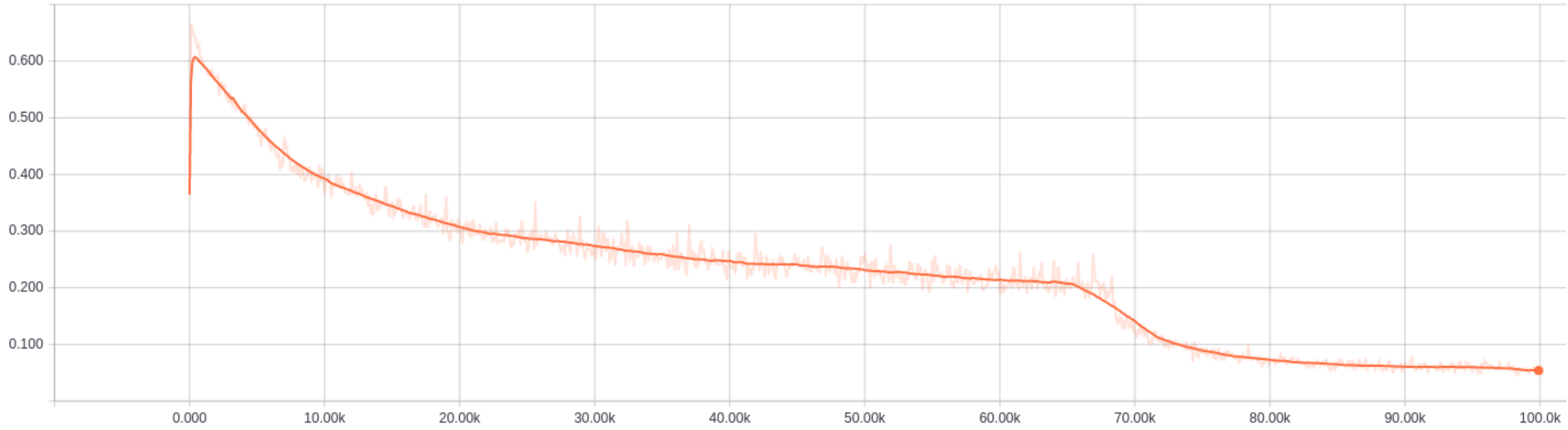



EVALUATION

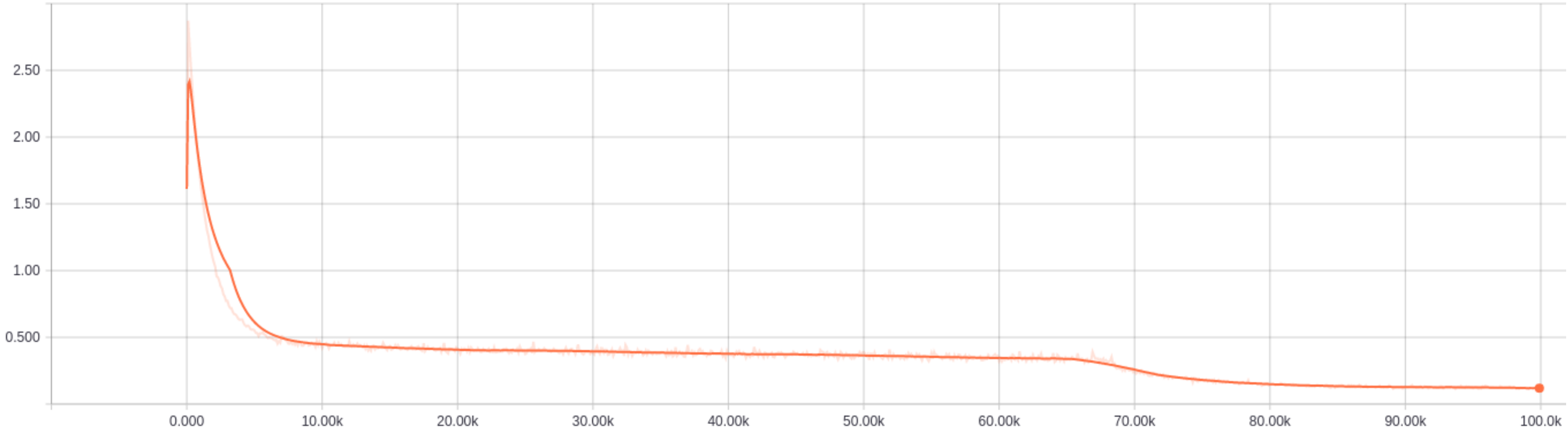
Learning rate



Cross-entropy



Total loss



Total loss after 100k steps roughly above 0.1

Summarize

- Understood a CNN
- Trained it
  - Total loss is decreasing over time
  - Precision just above 0.1

QUESTIONS

Quellen

- http://cs231n.github.io/convolutional-networks/
- https://www.tensorflow.org/tutorials/deep_cnn/
- Maas, Andrew L., Awni Y. Hannun, and Andrew Y. Ng. "Rectifier nonlinearities improve neural network acoustic models." *Proc. ICML*. Vol. 30. No. 1. 2013.