TU Clausthal

A Convolutional Neural Network for Image Classification of Cats and Dogs

Status update

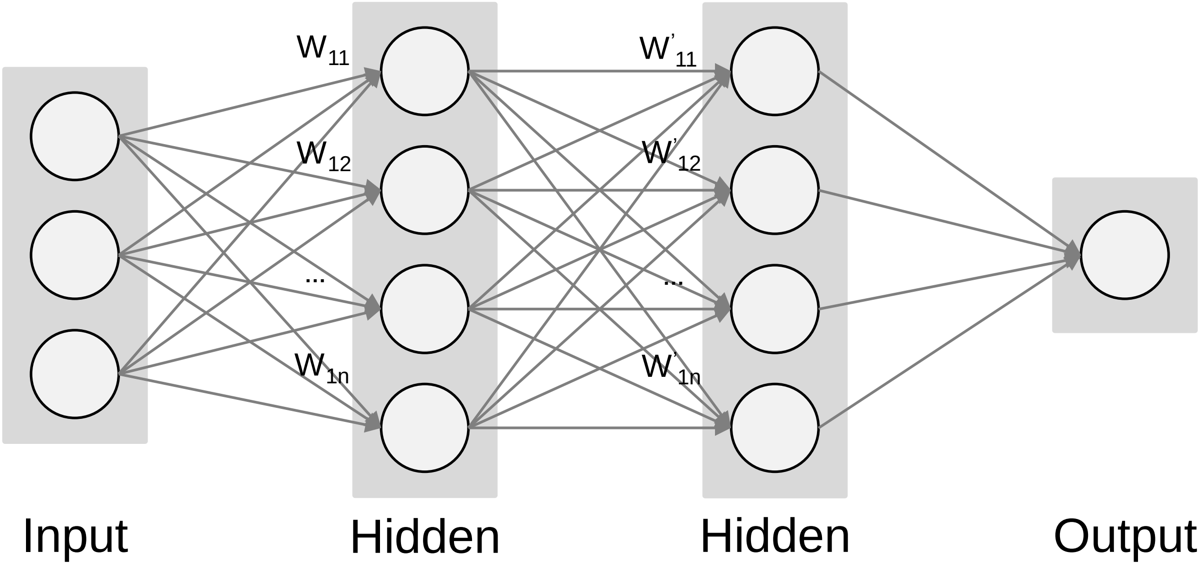
Aditya Raj, Sören Schleibaum
Institut für Informatik

Structure

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

NEURAL NETS

Introduction – Neural Nets



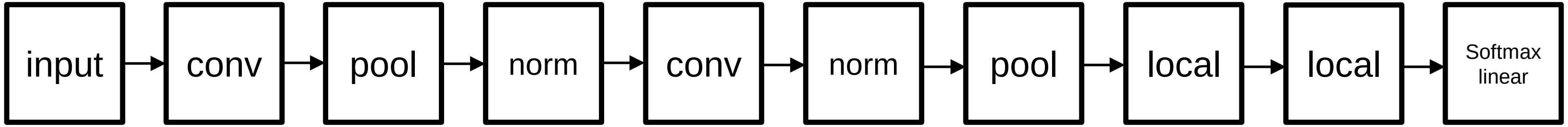
Quelle: http://cs231n.github.io/convolutional-networks/

CONVOLUTIONAL NN

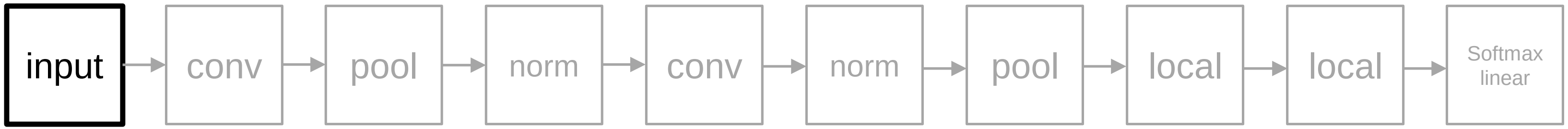
TensorFlow

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

Structure of the CNN we used



Input layer


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

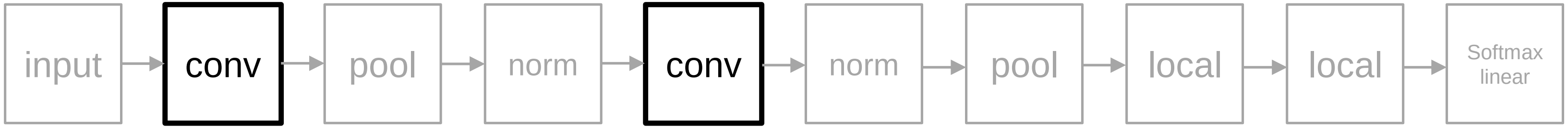
1

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/



Convolutional layer - Parameters

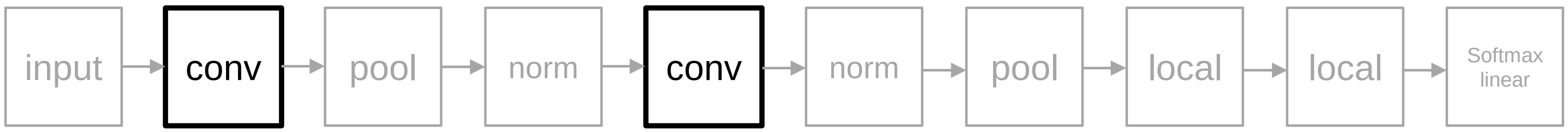
- Input volume size
- Number of filters
- Filter size
- Zero padding

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

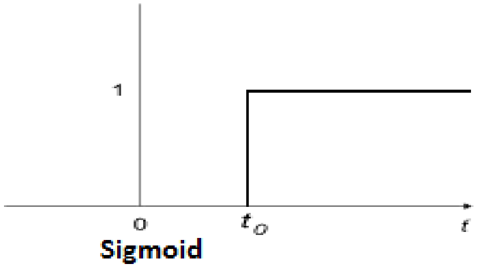
0	0	0
0	2	1
0	2	2



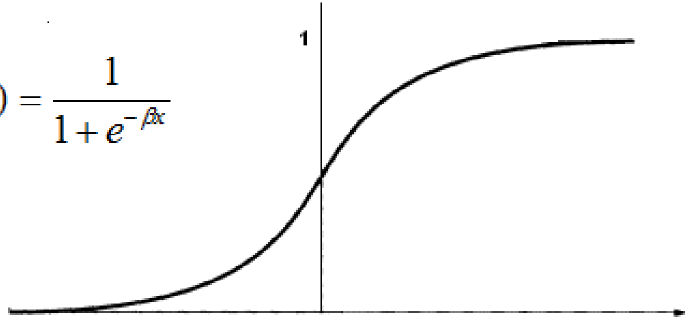
Aditya Raj, Sören Schleibaum
Institut für Informatik

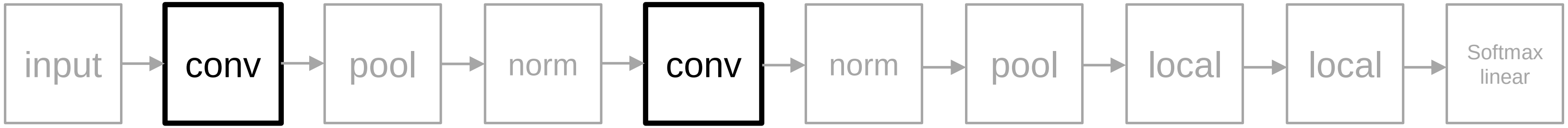
Convolutional layer – Activation function

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



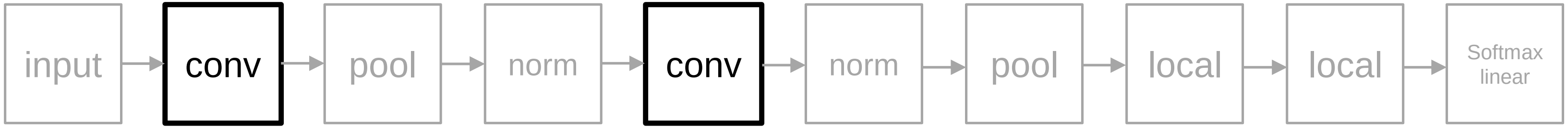
$$f(x) = \frac{1}{1+e^{-\beta x}}$$



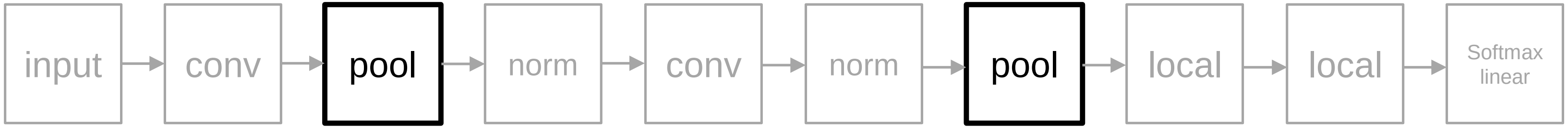


Convolutional layer – Activation function

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



Pool layer


input
conv
pool
norm
conv
norm
pool
local
local
Softmax
linear

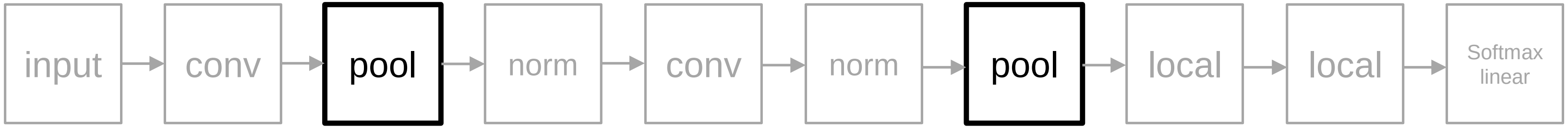
Pool layer – Max pooling

- Reduce the spatial dimension of an image

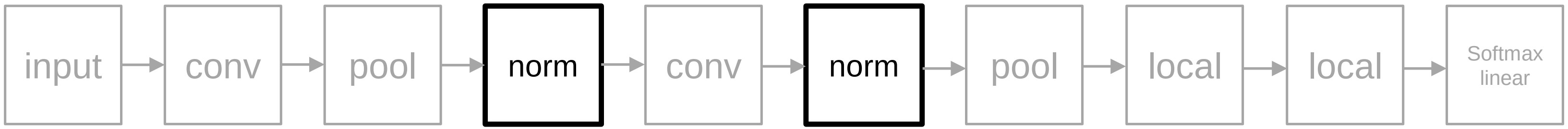
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

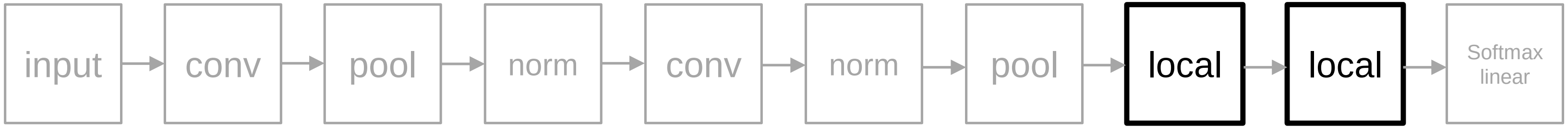
http://cs231n.github.io/convolutional-networks/



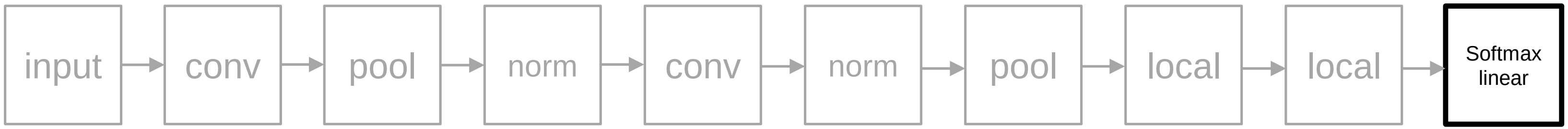
Norm layer


input
conv
pool
norm
conv
norm
pool
local
local
Softmax
linear

Local layer


input
conv
pool
norm
conv
norm
pool
local
local
Softmax
linear

Softmax-linear layer



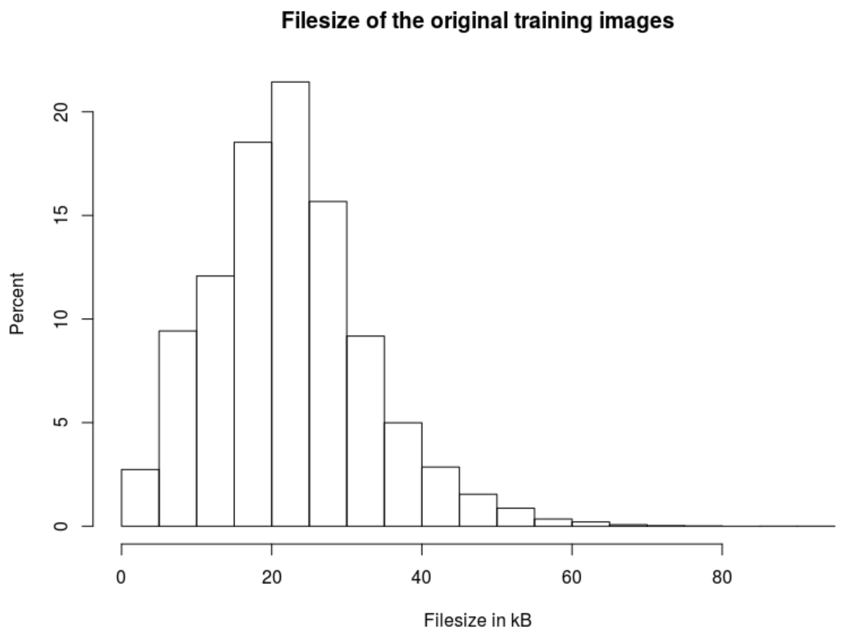
PROBLEM

The data

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

Training data

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



Test data

- 12,500 images
  - x of dogs
  - y of cats
  - $x + y = 12{,}500$

Process images

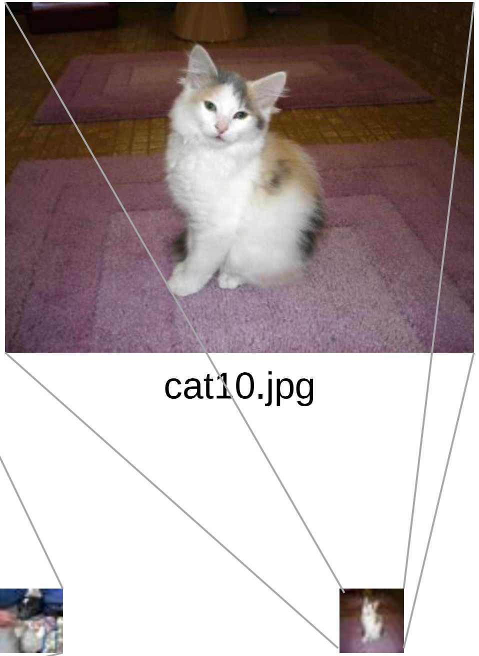
- Resize to 32 * 32 * 3
- Convert to array
  - 25,000 * 3,073

dog1.jpg

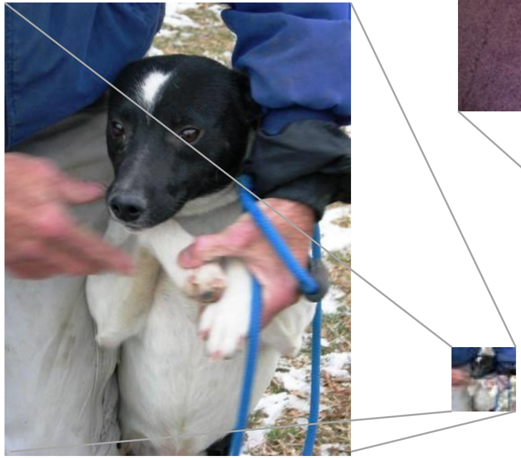
cat10.jpg

Process images

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



cat10.jpg



dog1.jpg

TU Clausthal

EVALUATION

AIMS

Aims

- Removing normalization layer

QUESTIONS

Quellen

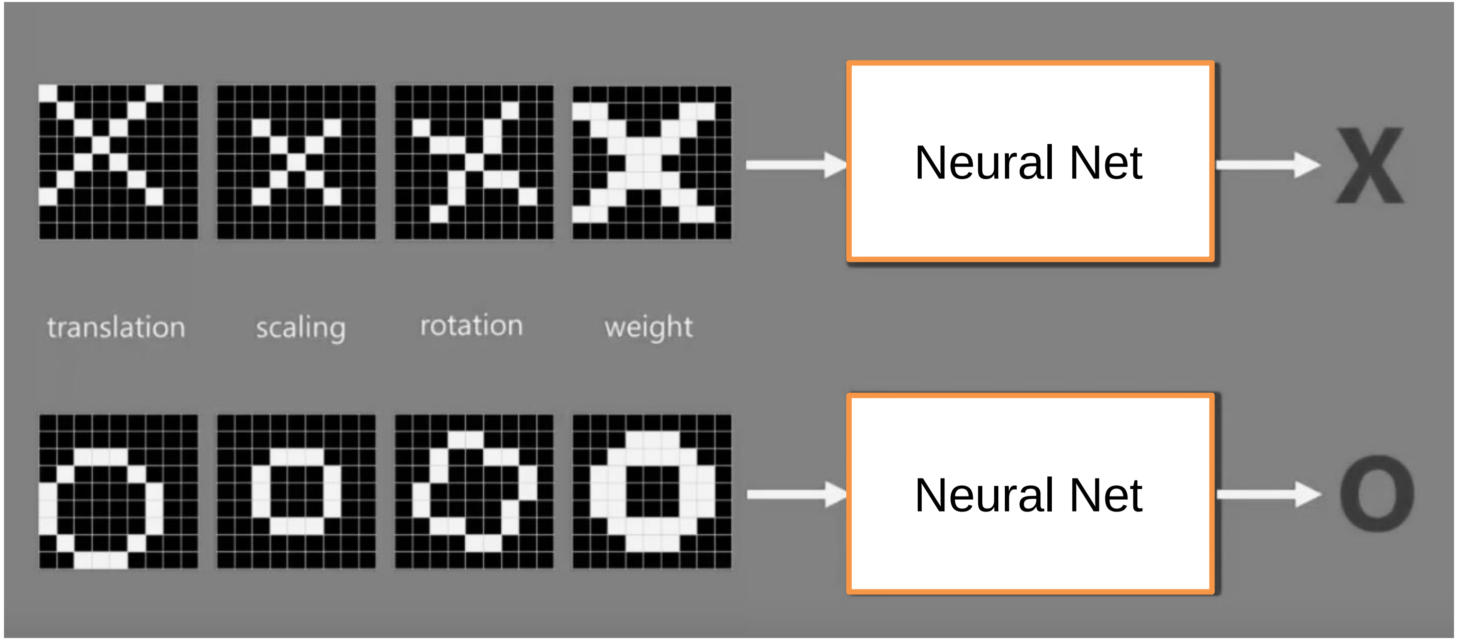
- http://cs231n.github.io/convolutional-networks/
- https://www.tensorflow.org/tutorials/deep_cnn/

- Maas, Andrew L., Awni Y. Hannun, and Andrew Y. Ng. "Rectifier nonlinearities improve neural network acoustic models." *Proc. ICML*. Vol. 30. No. 1. 2013.

Today‘s Talk

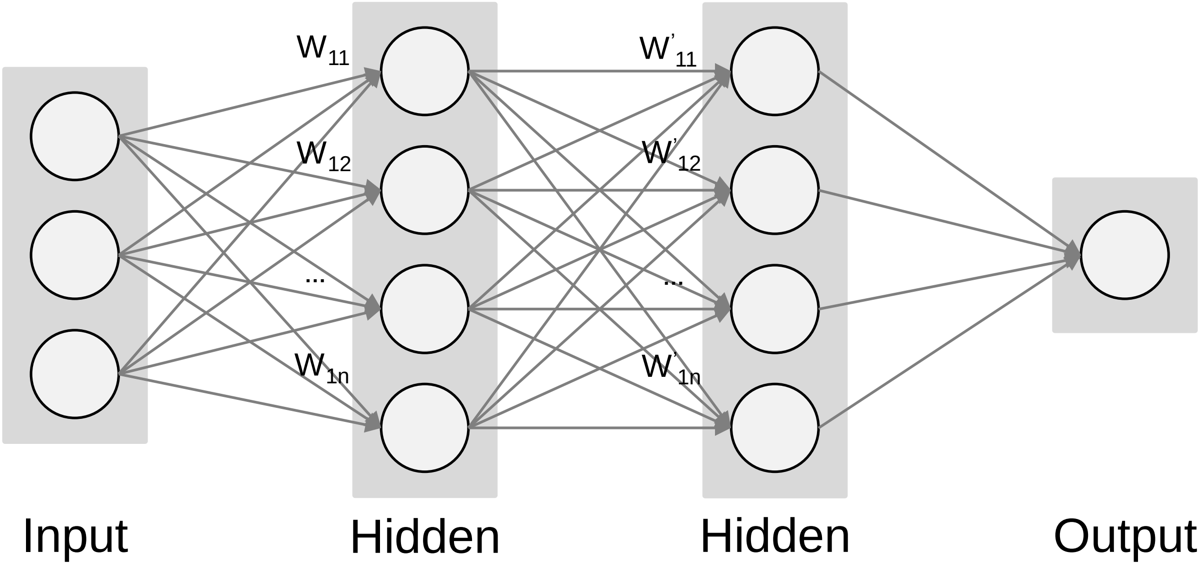
- Problem Statement
- Introduction to Deep Learning
  - Layers In Deep Learning
- TensorFlow
  - Methods
  - DataSets
  - Training Time
- TensorFlow (TF)
  - Data-Structure for TensorFlow
- Implementation in TF
  - Inputs
  - Prediction
  - Training
  - Evaluation
- Results
- Future Works

Problem Statement - Explained



Aditya Raj, Sören Schleibaum
Institut für Informatik

Introduction – Neural Nets



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

Mathematical view

- Input, Weights
- Compute Sigmoid
- Measure how much we missed called Err
- Multiply Err by the Sigmoid slope
- Update Weights

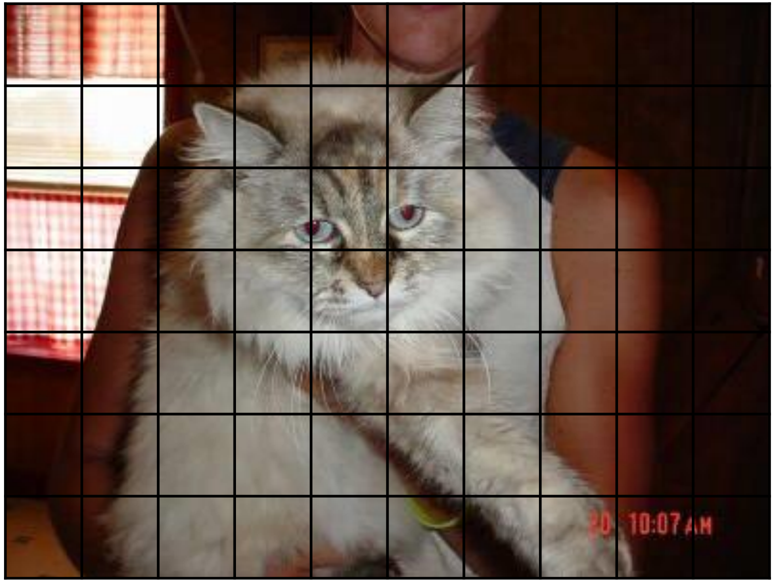
- $l_0 = X_i$, W = rand()
- $l_1 = Sig(X_i \, . W)$
- Err = $l_0 - l_1$
- Δl_1 = Err*$\Delta\left(Sig(Err)\right)$
- W = W+($l_0 . \Delta l_1$)

Gradient Descent

- Pushing down weights to push errors down
  - Move weights in negative direction of gradient
- $\Delta w_i = h\,(y - y')x_i$. *Perceptron Case*
  - *No Thresholding, finite convergence but in linear cases*
- $\Delta w_i = h\,(y - a)x_i$ *Activation Case*
  - *Thresholding, more robust to non-linear cases*
  - *Converge to a limit only to a local optimum*

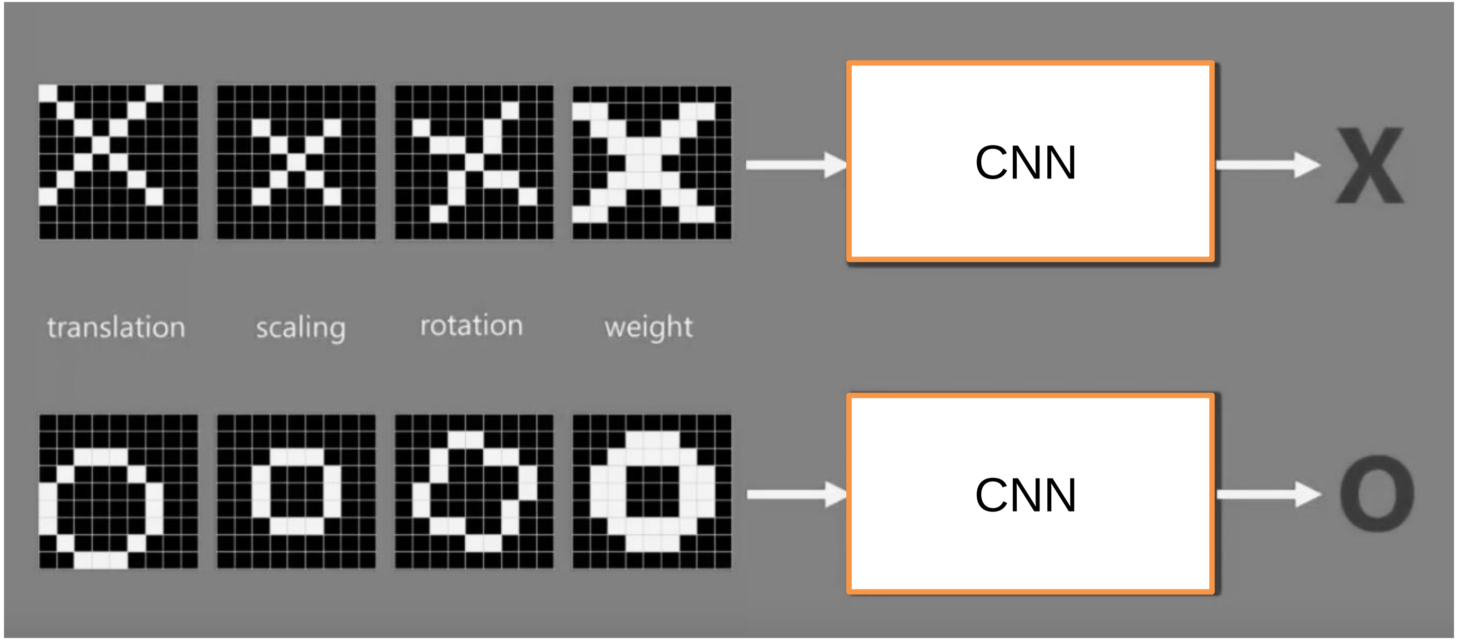
Why not just Neural Nets?

- Input 32*32*3 = 3072
- Weights 3072*N
- Biases N
- So,
  - Full connectivity is wasteful
  - Huge number of parameters
  - Loss of spatial information
    - How 3072 input signals represent 32*32*3 matrix ?
  - Deconvolution



Aditya Raj, Sören Schleibaum
Institut für Informatik

Convolutional Neural Network (CNN)



CNN Layers

- INPUT
- CONV
- RELU
- POOL
- FC

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

Output

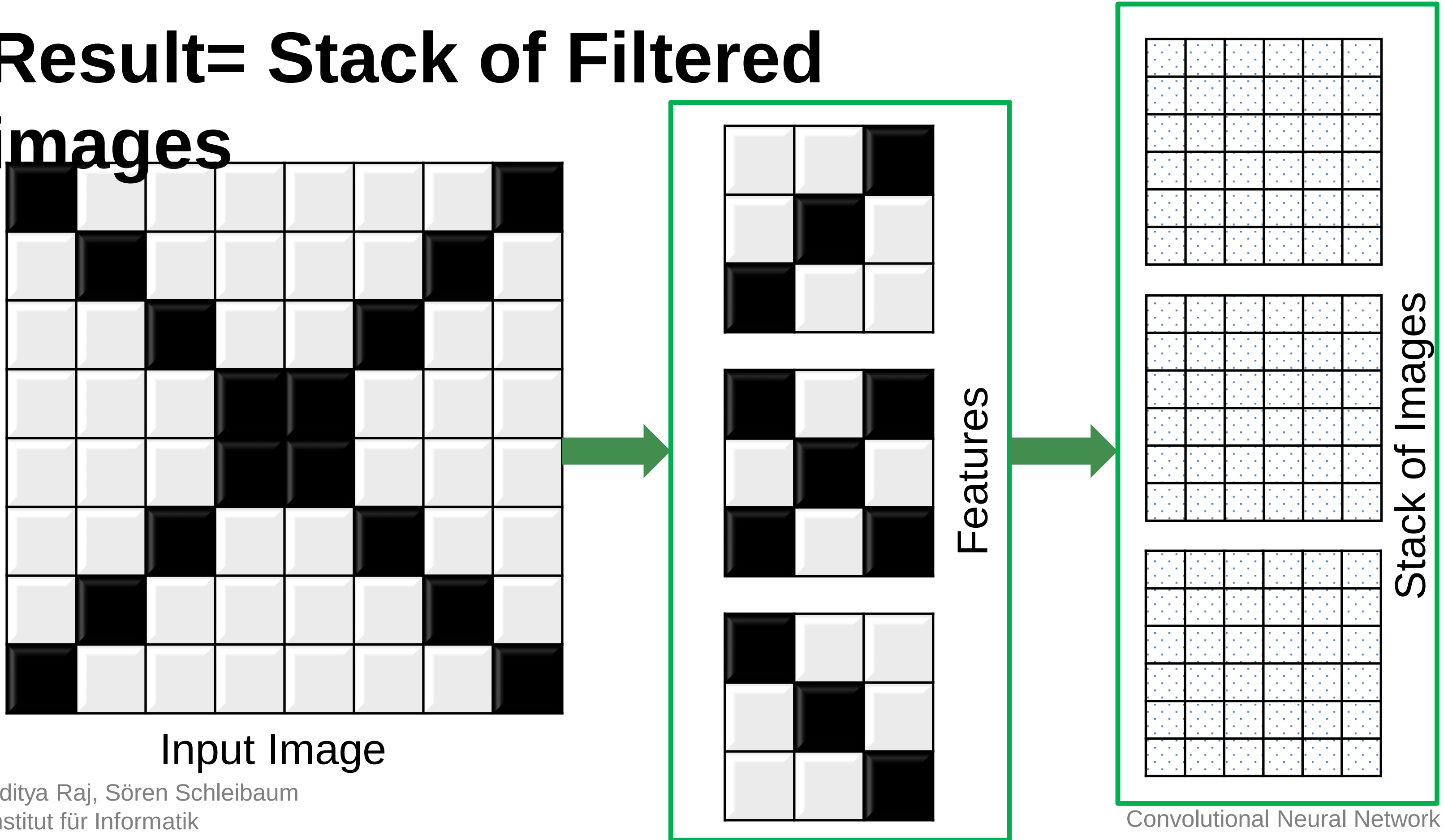
1

Bias

1

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

Result= Stack of Filtered images



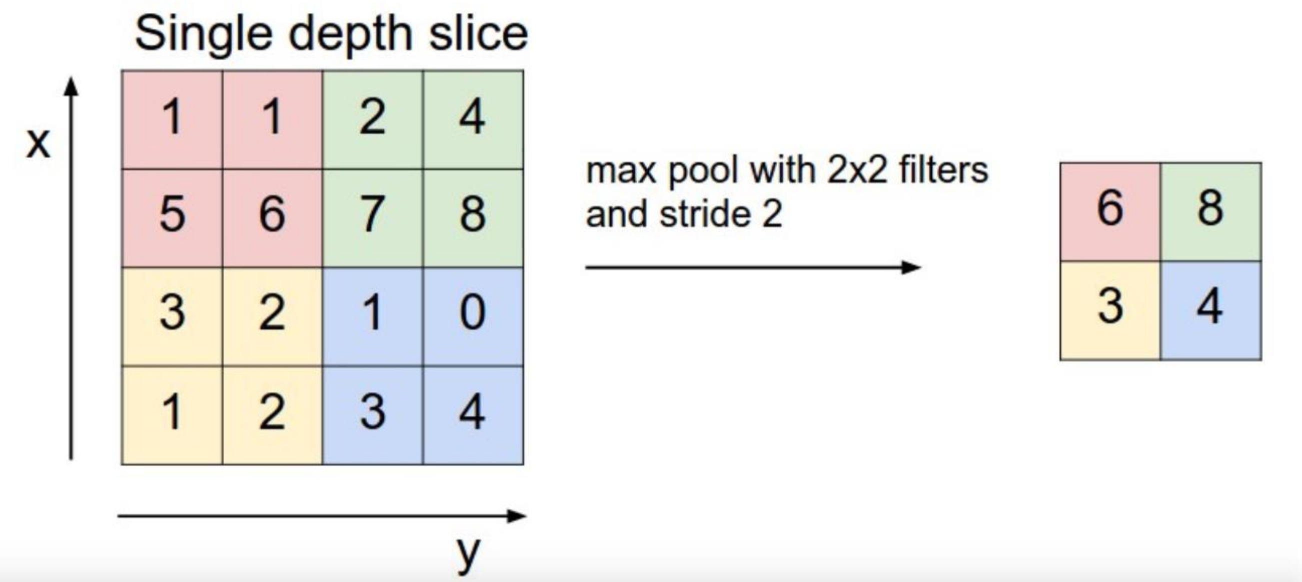
Aditya Raj, Sören Schleibaum
Institut für Informatik

Convolutional Filter Size

- Uneven dimensions such as 3*3, 5*5 …
  - To reduce spatial dimension
    - Padding can undo dimensionality reduction
- Number of conv layers:
  - More conv layers with small filters
    - This makes the decision function more discriminative

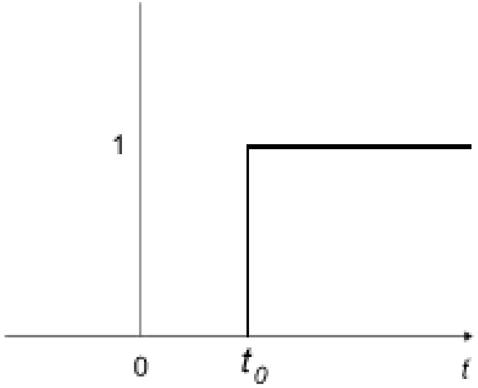
CNN – POOL

- reduce the spatial dimension of an image



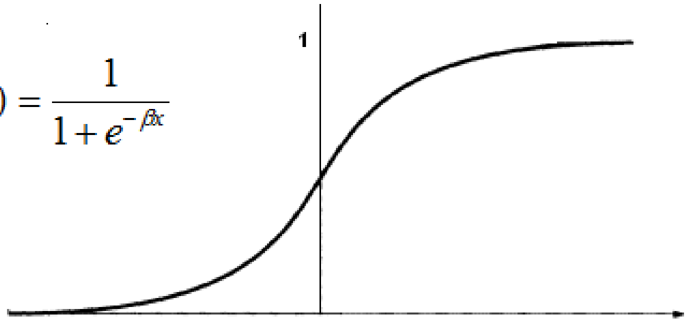
Activation functions

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



Sigmoid

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



CNN – RELU

- $Element\ Wise: \max(0, x)$
- Leaky ReLU by by Maas et al.
  - if x < 0, Output = 0.01x.
  - non-zero gradient when the input is negative

CNN Parameters

- Can be trained on an endless amount of parameters:
  - learning rate, learning rate decay
  - momentum
  - filter size, number of convolutional layers
  - activation functions (relu, leakyrelu)
  - weight decay and dropouts

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 ways to do it:
  - Exponential decay, reduction by factor of n
  - GoogLeNet: function to decrease the learning rate by 4%

Momentum

- Rolling ball gains speed downwards the hill
- So, velocity to the learning rate in a given direction
  - With consistent gradient
- Convolutional neural networks commonly use a momentum value of 0.9

Batch Normalization (BN)

- *batch normalization* of the input to the activation function of each neuron
- normalizing the training batch after certain layers
  - Reducing amount of retraining
  - input to the activation function across each training batch has a mean of 0 and a variance of 1.
- Example
  - $BN\ of\ \sigma(Wx + b) = \sigma\big(BN(Wx + b)\big)\ Where\ BN = \frac{X_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$

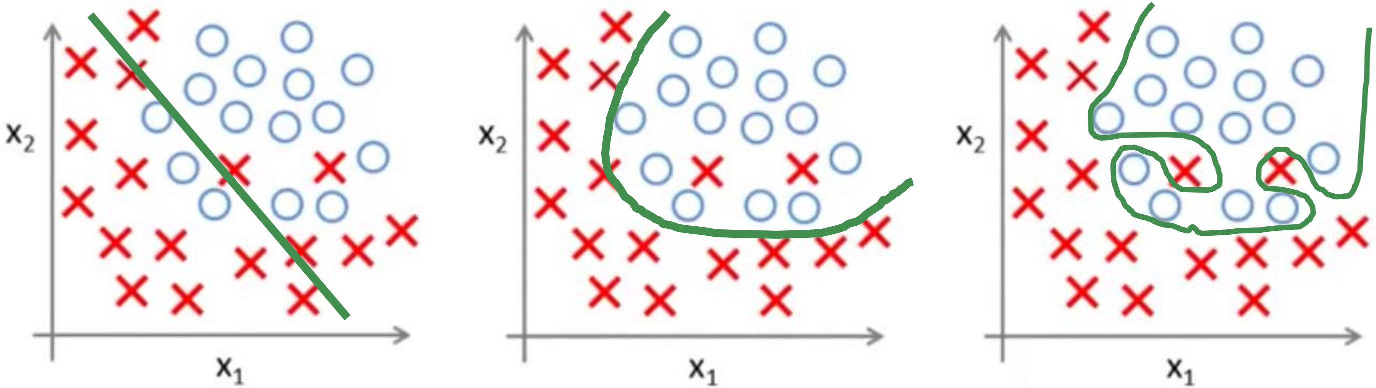
Is BN enough ?

- No:
  - Activation function limited to a prescribed normal distribution
- Adding γ and β -> Learnable parameters
  - γ , undoes the batch normalizing transform
  - β, a new shift parameter

$$BN = \gamma\left(\frac{X_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}\right) + \beta$$

Overfitting vs Underfitting

Example: Logistic regression



- $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

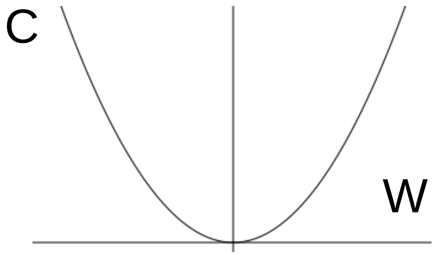
  (g = sigmoid function)

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2$
$+\theta_3 x_1^2 + \theta_4 x_2^2$
$+\theta_5 x_1 x_2)$

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2$
$+\theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2$
$+\theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \ldots)$

Weight Penalty

- Adding extra term to cost function to penalise
  - Keeps weight small
  - Big error derivatives

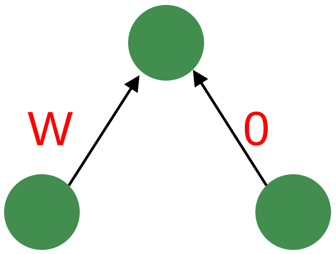


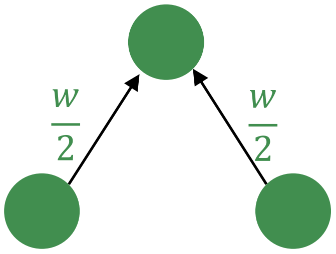
- $C = E + \frac{\lambda}{2}\sum_i {w^i}^2$
- $\frac{\partial C}{\partial w_i} = \frac{\partial E}{\partial w_i} + \lambda w_i$
- When $\frac{\partial C}{\partial w_i}$ =0;
  - $w_i = -\frac{1}{\lambda}\frac{\partial E}{\partial w_i}$
  - So, at minimum of Cost function if $\frac{\partial E}{\partial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

TU Clausthal

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





Cifar10 Weight Decay

- weight_decay = tf.mul(tf.nn.l2_loss(var), wd, name='weight_loss')
- tf.add_to_collection('losses', weight_decay)

Weight Penal

TensorFlow

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

Limitations

- 150 images viewed
  - 1 duplicate
  - 1 wrong content
- Training lasts long
- Reduced image size

Activation Functions

- Non-Linear
  - sigmoid, tanh, elu, softplus, and softsign
- continuous but not everywhere differentiable functions
  - relu, relu6, crelu and relu_x

TF Implementation: Prediction

- Cifar10.Inference()
  - Conv1: convolution and rectified linear activation.
  - Pool1: max pooling.
  - Norm1: local response normalization.
  - Local3: fully connected layer with rectified linear activation.
  - Local4: fully connected layer with rectified linear activation.
  - Softmax_linear: Linear transformation to produce logits.

Cost Functions

- Squared Error Measure

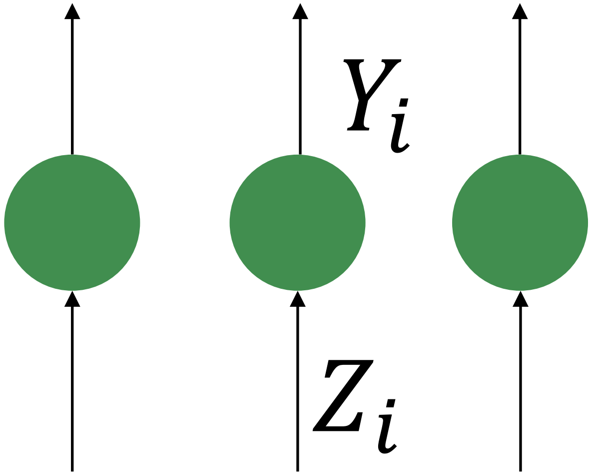
- Softmax Cross-entropy Function

Squared Error Measure Function

- $Error = \frac{1}{2}(Y_{actual} - Y_{predicted})^2$
- Drawbacks
  - No gradient to get from 0.000…1 to 1.
    - To do so it will take quite longer.
  - Deprives NN of probability information.

Softmax Output Function

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



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

Derivative Softmax

- $\frac{\delta Y_i}{\delta Z_i} = Y_i\,(1 - Y_i)$
- 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 $Y_i\,(1 - Y_i)$

Cost Measure for Softmax Output Function

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

Advantages to Squared Error Measure

- $C = -\sum_j t_j \log\ Y_j$
- Very big gradient when:
  - Target value is 1.
  - Actual output is 0.
- Balance between
  - Steepness of $\frac{dC}{dy}$ and
  - Flatness of $\frac{dy}{dZ}$

$$\frac{\partial C}{\partial Z_j} = \sum_j \frac{\partial C}{\partial y_j}\frac{\partial y_j}{\partial Z_j}$$

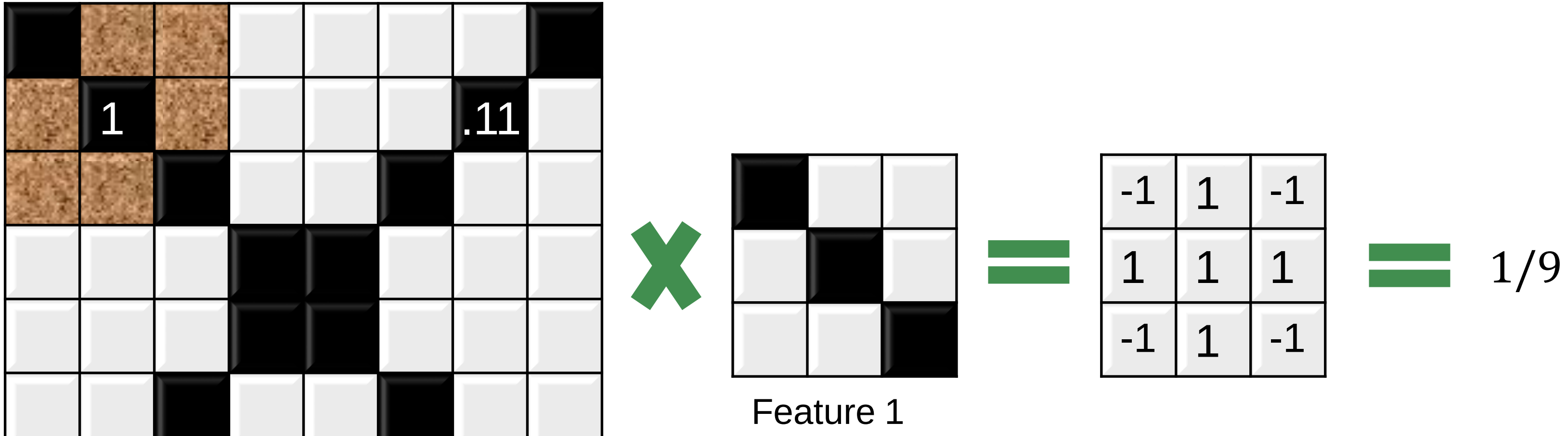
Hyperparameters

- Learning Rate

Results

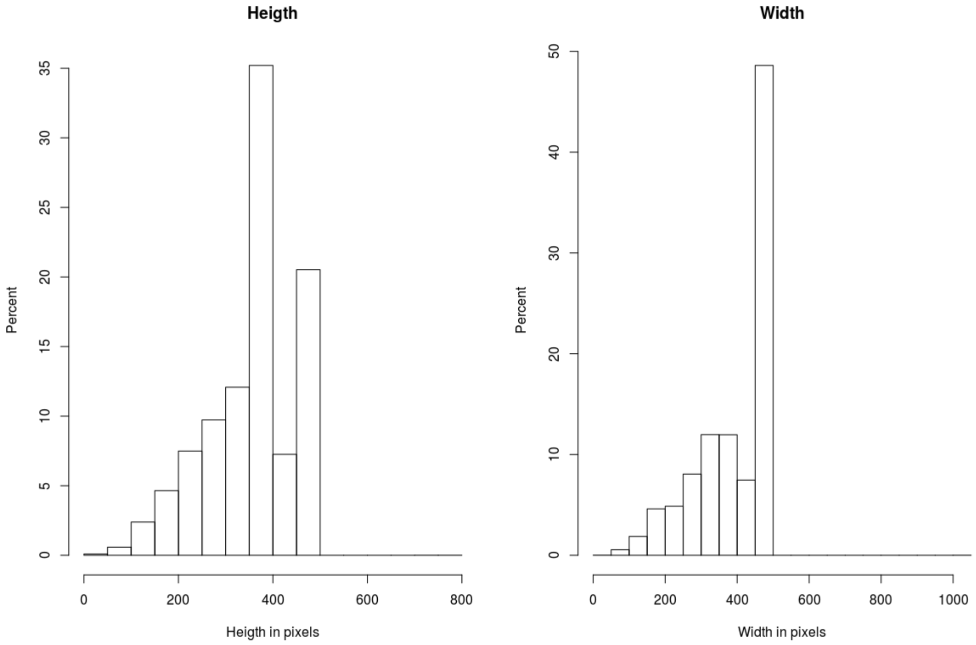
- Precision 0.83

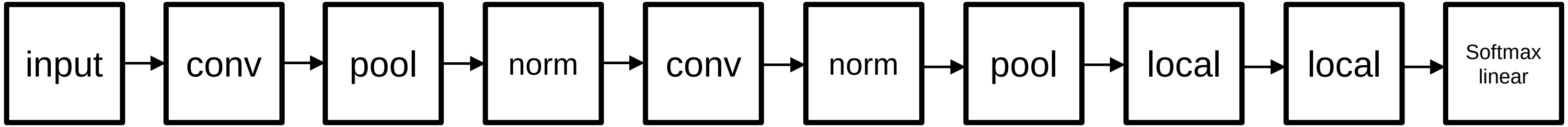
CNN – Conv Layer



Problem Statement

Ratio


Heigth
Percent
0
5
10
15
20
25
30
35
0
200
400
600
800
Heigth in pixels
Width
Percent
0
10
20
30
40
50
0
200
400
600
800
1000
Width in pixels


input
conv
pool
norm
conv
norm
pool
local
local
Softmax linear