CAS Machine Intelligence: Deep Learning
Week 4

Beate Sick

Institut für Datenanalyse und Prozessdesign

Zürcher Hochschule für Angewandte Wissenschaften

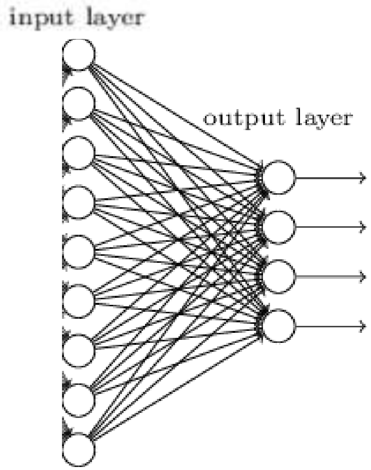
Beate.Sick@zhaw.ch

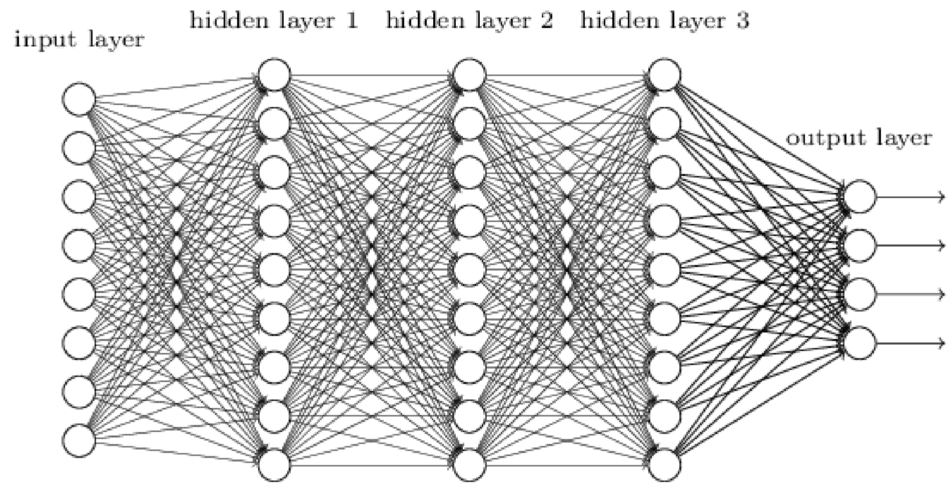
Outline

- Recap fully connected NN
- Keras intro (recap)
- Homework: Questions and results
- Motivation of <u>c</u>onvolutional <u>n</u>eural <u>n</u>etworks (CNNs)
- What is convolution?
- How is convolution performed over several channels/stack of images?
- How does a classical CNN look like?
- Do a CNN yourself

We will go from fully connected NNs to CNNs

Fully connected Neural Networks (fcNN) without and with hidden layers:





Convolutional Neural Network:

input conv1 pool1 conv2 pool2 layer3 output

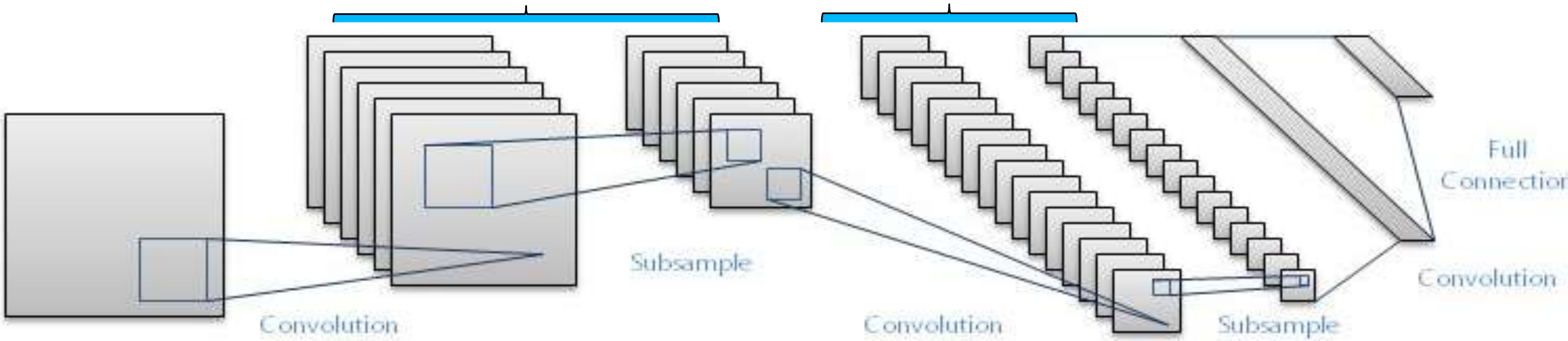


Image credits: http://neuralnetworksanddeeplearning.com/chap6.html, http://www.cemetech.net/projects/ee/scouter/thesis.pdf

At the end of the day

Develop a DL model to solve this task:

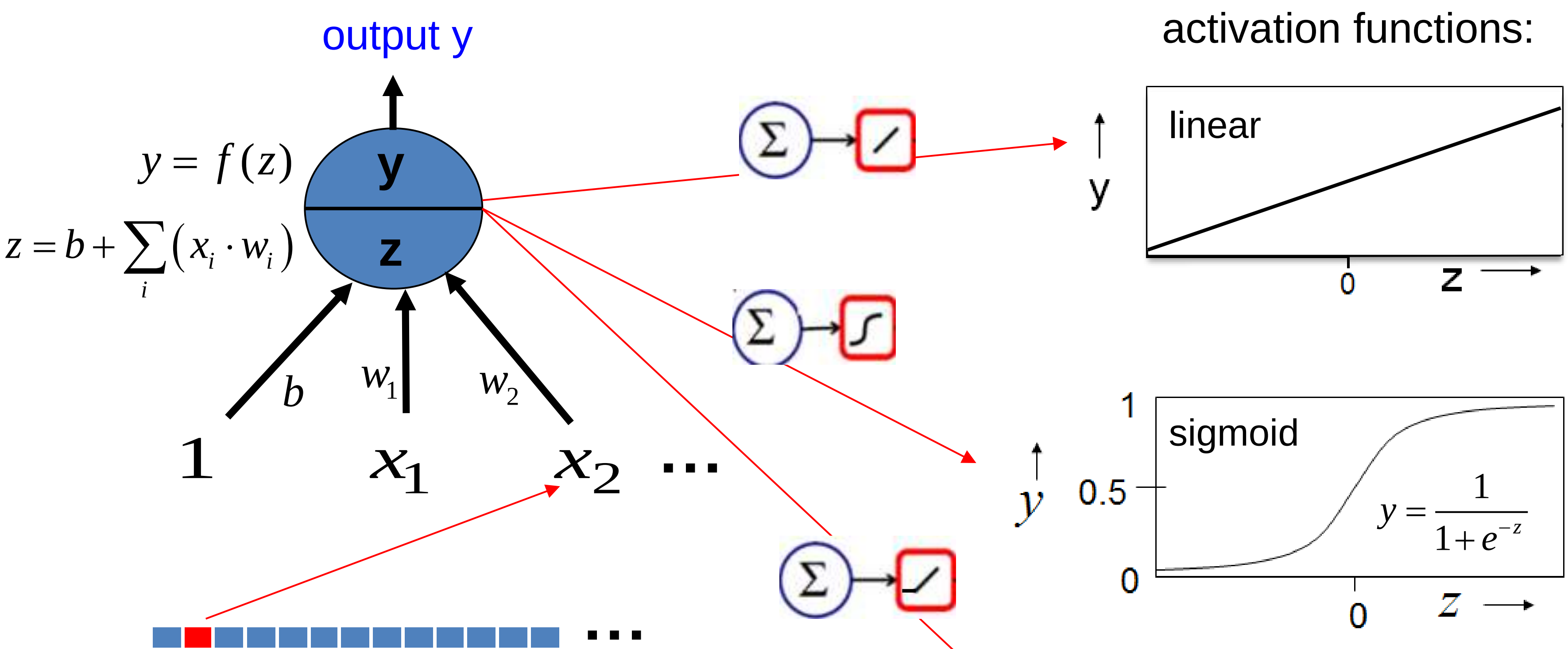
For a given image from the internet, decide which out of 8 celebrities is on the image.

Example images:

Label: Steve Jobs (entrepreneur)

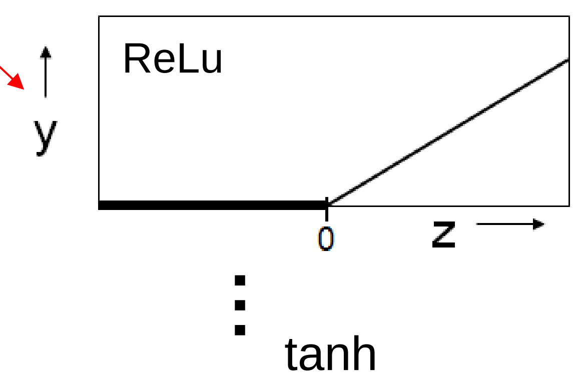
Label: Emma Stone (actress)

Recap: computations performed in a neuron



Input
feature vector

Features can come from different sources (not only from images)



2 linear layers can be replaced by 1 linear layer

-> we can not go deep with linear layers!

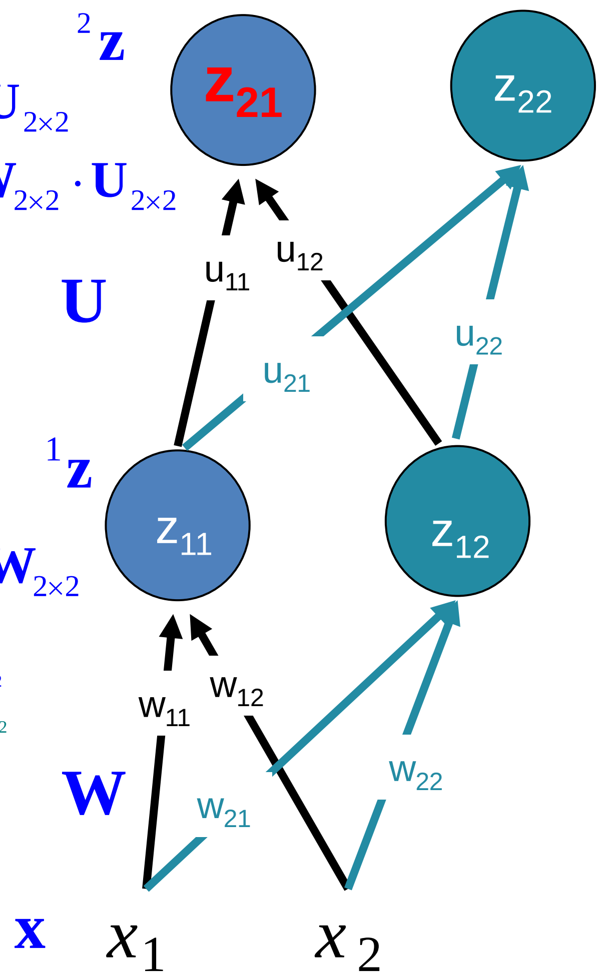
$$z_{21} = z_{11}\cdot u_{11} + z_{12}\cdot u_{12} = (w_{11}\cdot x_1 + w_{12}\cdot x_2)\cdot u_{11} + (w_{21}\cdot x_1 + w_{22}\cdot x_2)\cdot u_{12}$$
$$= x_1\cdot(w_{11}\cdot u_{11} + w_{21}\cdot u_{12}) + x_2\cdot(w_{12}\cdot u_{11} + w_{22}\cdot u_{12})$$

The activations of two stacked linear layers are linear functions of the inputs x_i that can also be achieved with one linear layer.

$${}^{2}\mathbf{z}_{1\times2} = {}^{1}\mathbf{z}_{1\times2}\cdot\mathbf{U}_{2\times2}$$
$$= \mathbf{x}_{1\times2}\cdot\mathbf{W}_{2\times2}\cdot\mathbf{U}_{2\times2}$$

$${}^{1}\mathbf{z}_{1\times2} = \mathbf{x}_{1\times2}\cdot\mathbf{W}_{2\times2}$$

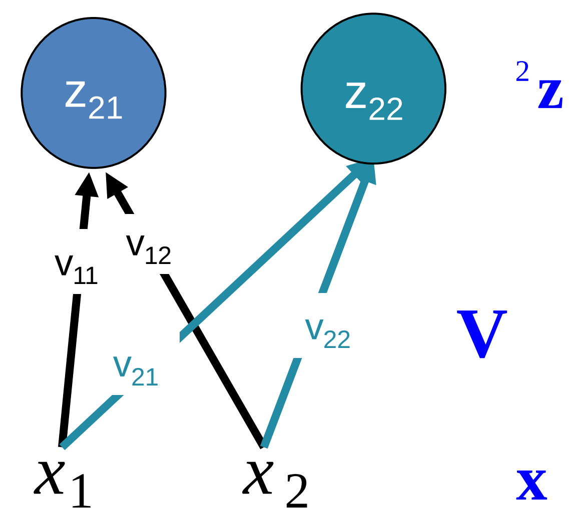
$$z_{11} = w_{11}\cdot x_1 + w_{12}\cdot x_2$$
$$z_{12} = w_{21}\cdot x_1 + w_{22}\cdot x_2$$



=

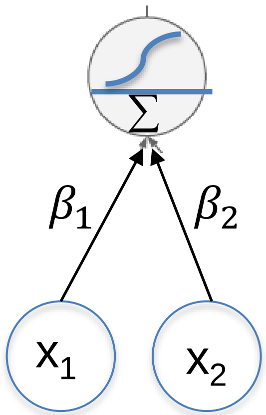
$$z_{21} = v_{11}\cdot x_1 + v_{12}\cdot x_2$$
$$v_{11} = w_{11}\cdot u_{11} + w_{21}\cdot u_{12}$$
$$v_{12} = w_{12}\cdot u_{11} + w_{22}\cdot u_{12}$$

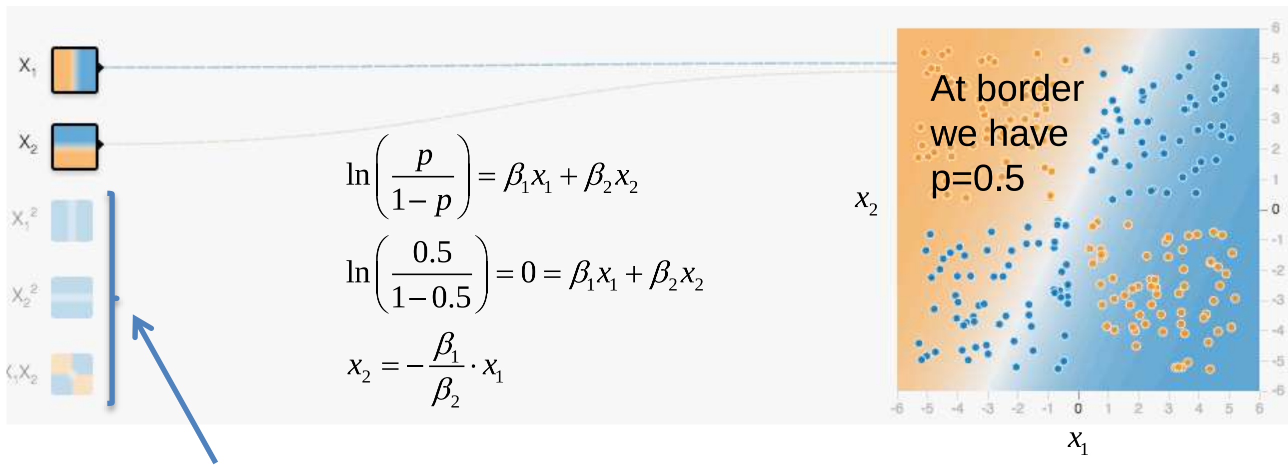
$${}^{2}\mathbf{z}_{1\times2} = {}^{1}\mathbf{x}_{1\times2}\cdot\mathbf{V}_{2\times2}$$
$$\text{with}\ \ \mathbf{V}_{2\times2} = \mathbf{W}_{2\times2}\cdot\mathbf{U}_{2\times2}$$



NN can only go deep with non-linear activation functions

NN **without** hidden layer and sigmoid activation function yields linear separation curve.





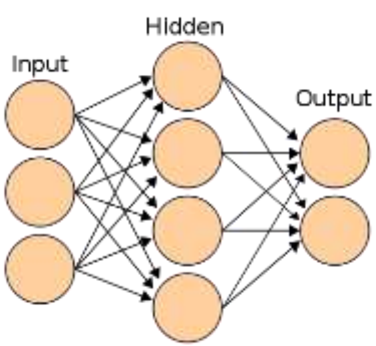
$$\ln\left(\frac{p}{1-p}\right) = \beta_1 x_1 + \beta_2 x_2$$

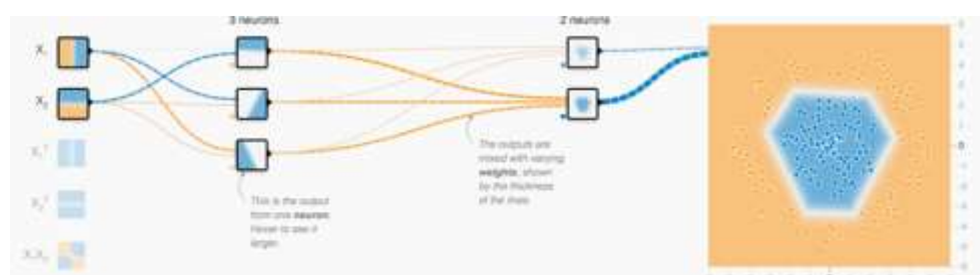
$$\ln\left(\frac{0.5}{1-0.5}\right) = 0 = \beta_1 x_1 + \beta_2 x_2$$

$$x_2 = -\frac{\beta_1}{\beta_2} \cdot x_1$$

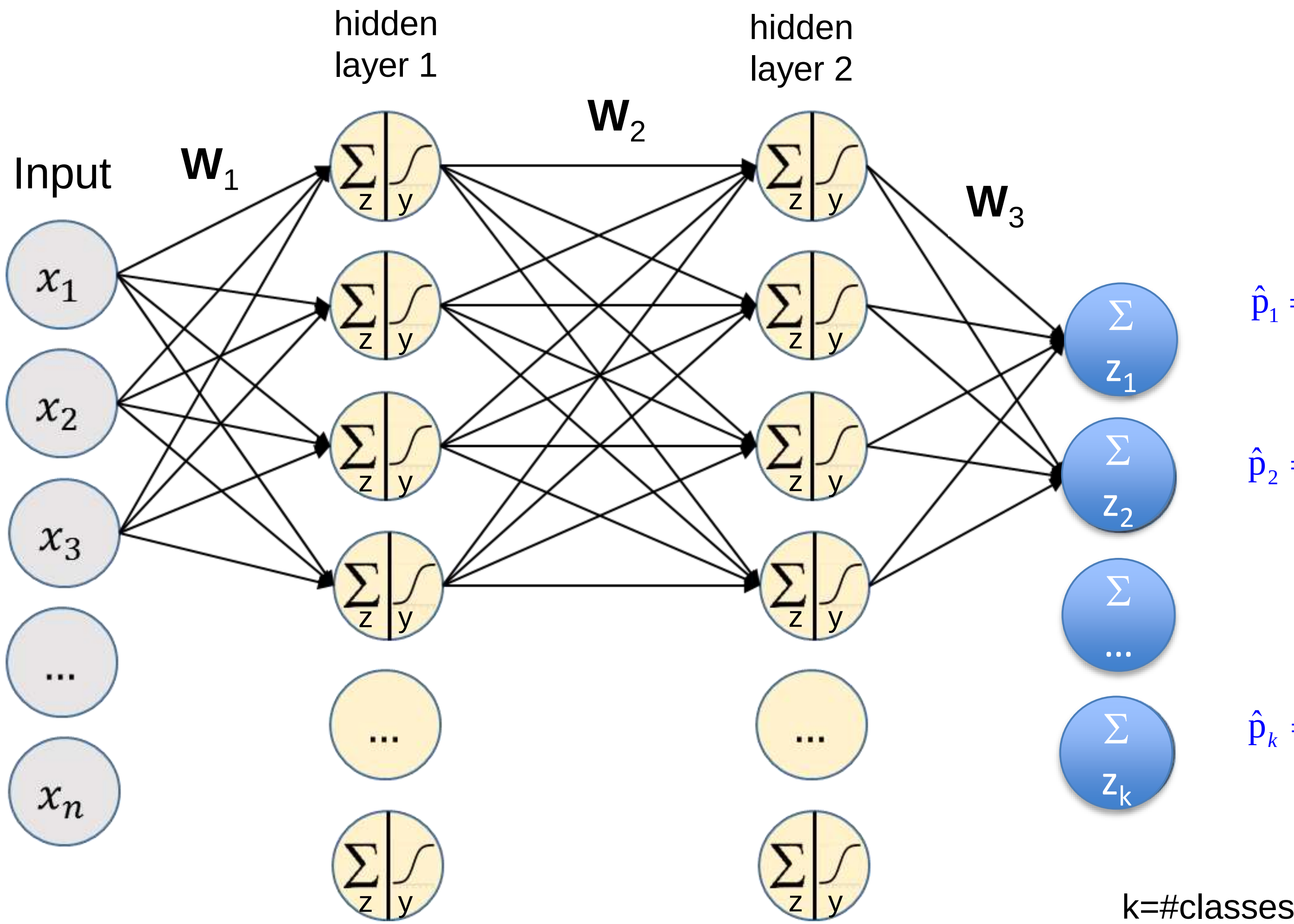
"handcrafted feature"

NN with ≥1 hidden layer and sigmoid activation function yields arbitrary separation curve.





A fully conneted neural networks with 2 hidden layers



$$\hat{p}_1 = \frac{e^{z_1}}{\sum_{j=1}^{k} e^{z_j}}$$

$$\hat{p}_2 = \frac{e^{z_2}}{\sum_{j=1}^{k} e^{z_j}}$$

$$\vdots$$

$$\hat{p}_k = \frac{e^{z_k}}{\sum_{j=1}^{k} e^{z_j}}$$

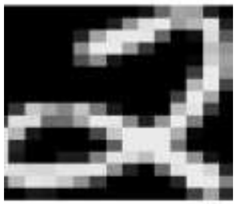
$$z = b + \sum_i (x_i \cdot w_i) \quad y = f(z)$$

Remark: weight values in the weight matrices that are learned during training.

What is going on in our 1 layer fully connected NN?

k=28

k=28



Flatten to vector with k^2 elements

input
$\mathbf{x}^t$

$$\begin{bmatrix} 24.0 \\ -5.1 \\ 12.2 \\ 89.9 \\ \vdots \\ 3.2 \end{bmatrix}$$

$$\mathbf{x}_{(1,k^2)} \cdot \mathbf{W}_{(k^2,2)} + \mathbf{b}_{(1,2)} = \mathbf{z}_{(1,2)}$$

Score or logit
$\mathbf{z}^t$

$$\begin{bmatrix} 2.3 \\ 0.1 \end{bmatrix}$$

$$\hat{\mathrm{p}}_k = \frac{e^{z_k}}{\sum_j e^{z_j}}$$

Softmax or probability
$\mathbf{p}$=S($\mathbf{z}$)

$$\begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix}$$

1-hot labels
$\mathbf{y}$

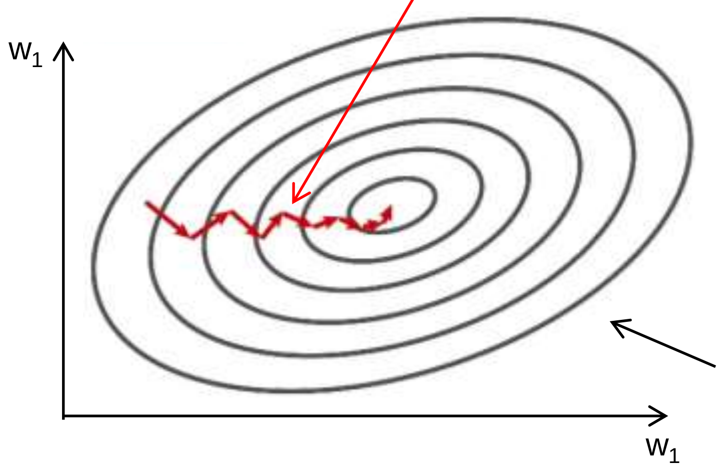
$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{matrix} \text{“2”} \\ \text{“1”} \end{matrix}$$

$D(\mathbf{p},\mathbf{y})$ cross-entropy

$$= -\sum_{k=1}^{2} y_k \cdot \log\left(p_k\right)$$

Take step in direction of descent gradient:
(the gradient is oriented orthogonal to contour lines)

$$w_i^{(t)} = w_i^{(t-1)} - \varepsilon^{(t)} \left. \frac{\partial C(\mathbf{w})}{\partial w_i} \right|_{w_i = w_i^{(t-1)}}$$



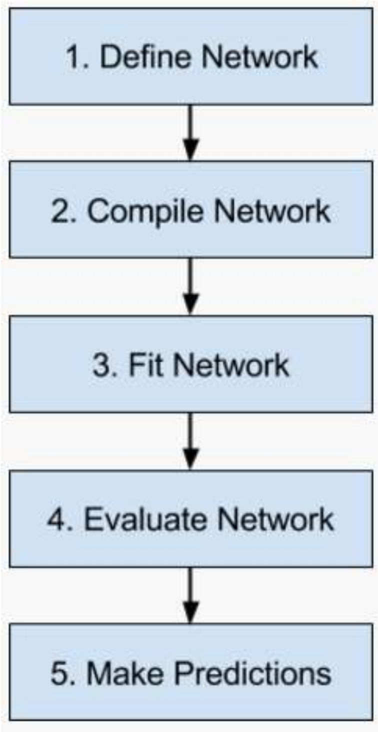
$\mathrm{C}\left(w_1, w_2\right)$

Cost C or Loss = cross-entropy averaged over all images in mini-batch

$$\mathrm{C} = \frac{1}{N} \sum_i D(\mathbf{p}_i, \mathbf{y}_i)$$

Keras: A high level API with best practice defaults

Keras workflow



- Keras is a high-level NN API for TensorFlow and Theano.
- Is now shipped with TF (or can be imported as python library)
- Can be mixed with TF code
- Offers many predefined layers
- Each layer has a default "best practice choices of parameters"
- Allows for easy and fast prototyping (define only key parameters)
- Supports fcNN, CNNs, RNNs ...
- Supports arbitrary connectivity schemes and NN architectures

Keras: import keras and the required layers

Keras provides two ways to define a model:

1) Sequential API: simple, good for linear stack of layers
2) Functional API: flexible, required for complicated architectures

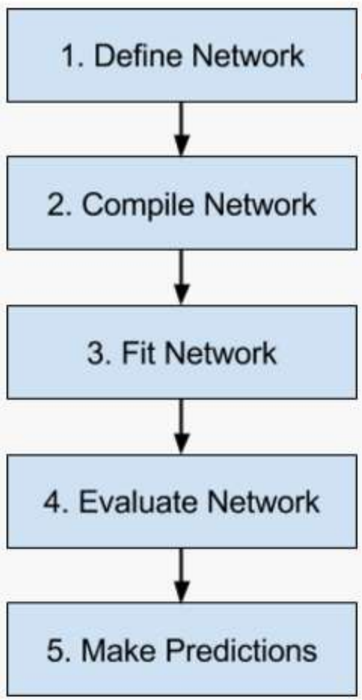
```
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, BatchNormalization
```

For documentation see:

https://keras.io/

Remark: Keras can be used as API for theano and tf and which differ in the shape of expectedtensors – i.e. #channels is in tf at last position and in theano not.

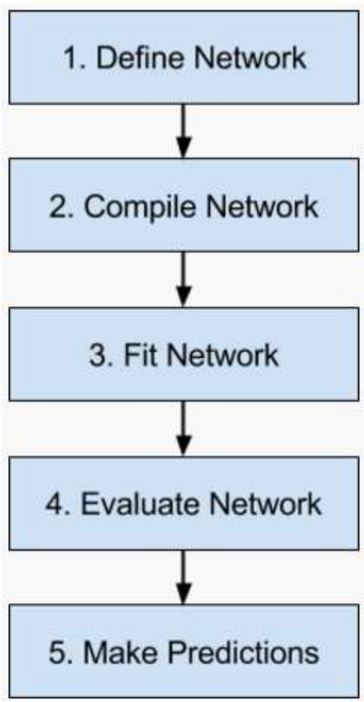
Keras: A high level API with best practice defaults



Number of neurons in (first)
hidden dense layers
(will be input to next layer)

```python
### Model with default initialization

model = Sequential()

model.add(Dense(500, batch_input_shape=(None, 784)))
model.add(Activation('sigmoid'))

model.add(Dense(50))
model.add(Activation('sigmoid'))

model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])
```

Keras: A high level API with best practice defaults



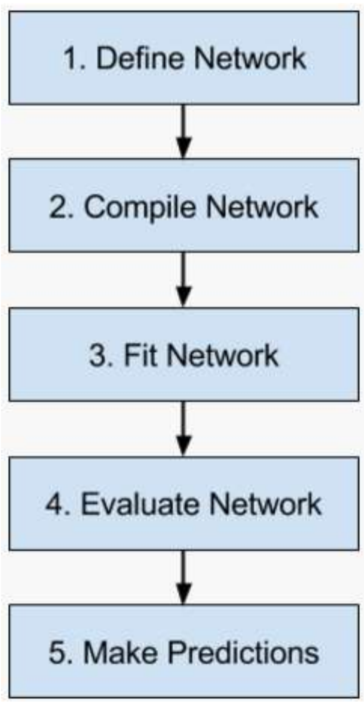
loss function that will be minimized

```
model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])
```

easiest optimizer is SGD (stochastic gradient descent)

Which metrics besides 'loss' do we want to collect during training

Keras: A high level API with best practice defaults



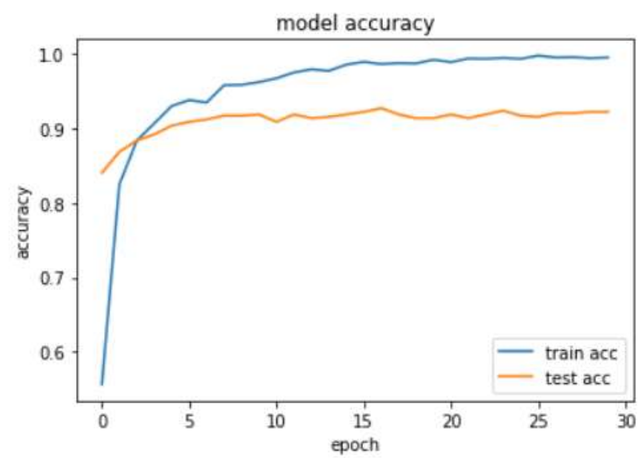

```python
# define information required for tensorboard
tensorboard = keras.callbacks.TensorBoard(
    log_dir='tensorboard/mnist_small/' + name + '/',
    write_graph=True,
    histogram_freq=1)

# train the model, memorize training history
history = model.fit(X[0:2400],
          convertToOneHot(y[0:2400],10),
          epochs=30,
          batch_size=128,
          callbacks=[tensorboard],
          validation_data=[X[2400:3000],
                           convertToOneHot(y[2400:3000],10)])
```

input data (train)
output/label (train)

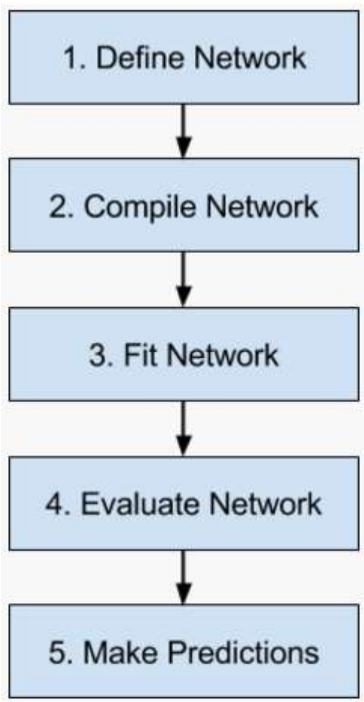
How and how often provide training data

In history we memorize development of loss and metrics achieved in successive training steps

```python
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train acc', 'test acc'], loc='lower right')
plt.show()
```



Keras: A high level API with best practice defaults




```
model.evaluate(X[0:2000], convertToOneHot(y[0:2000],10))
```

```
2000/2000 [==============================] - 0s 215us/step
```

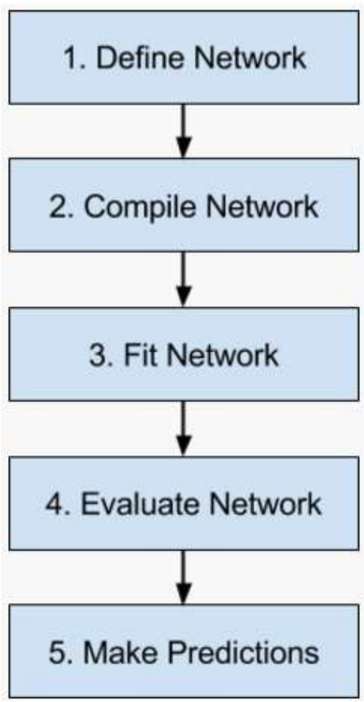
```
[2.5549037284851073, 0.1085]
```

loss

metrics: accuracy

```
model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])
```

Keras: A high level API with best practice defaults




```
model.predict_classes(X[0:10])
```

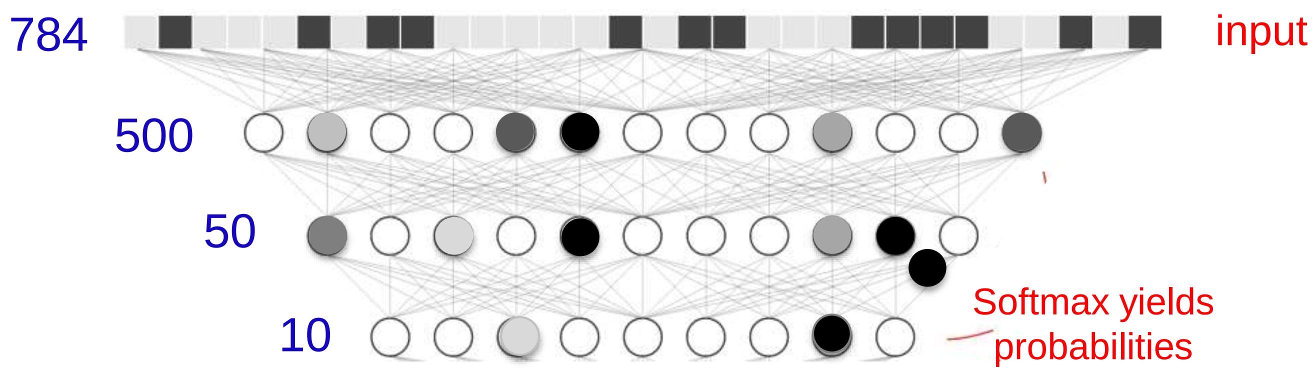
```
array([0, 9, 9, 1, 6, 9, 8, 9, 0, 0])
```

predicted class labels of the first 10 images

Keras: gives nice summary of model architecture

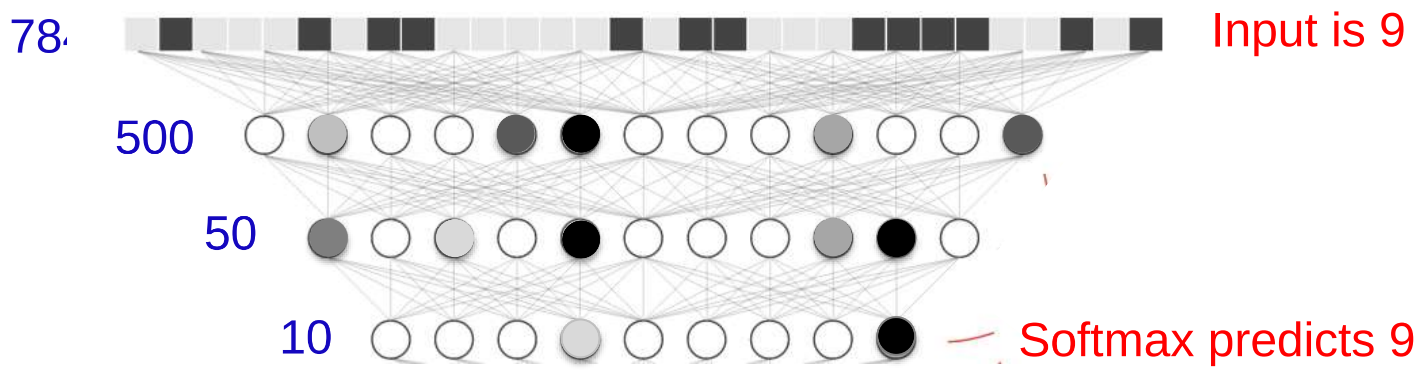
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 500)               392500
_________________________________________________________________
activation_1 (Activation)    (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                25050
_________________________________________________________________
activation_2 (Activation)    (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
=================================================================
Total params: 418,060
Trainable params: 418,060
Non-trainable params: 0
_________________________________________________________________
```

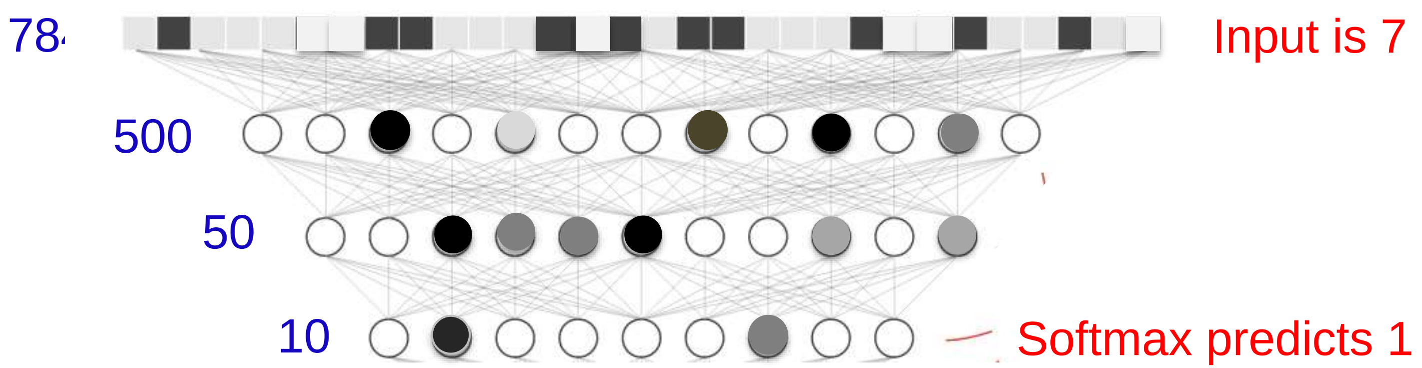


Cheated: we should have 10 (not 9) nodes ;-)

A trained NN has fixed weights and input specific activation patterns in each layer

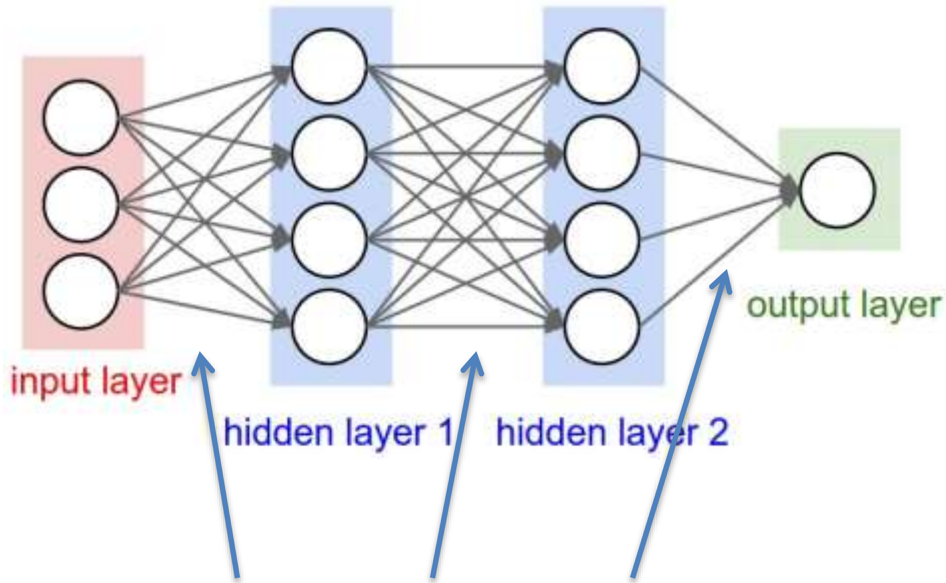


Cheated: we should have 10 (not 9) nodes ;-)



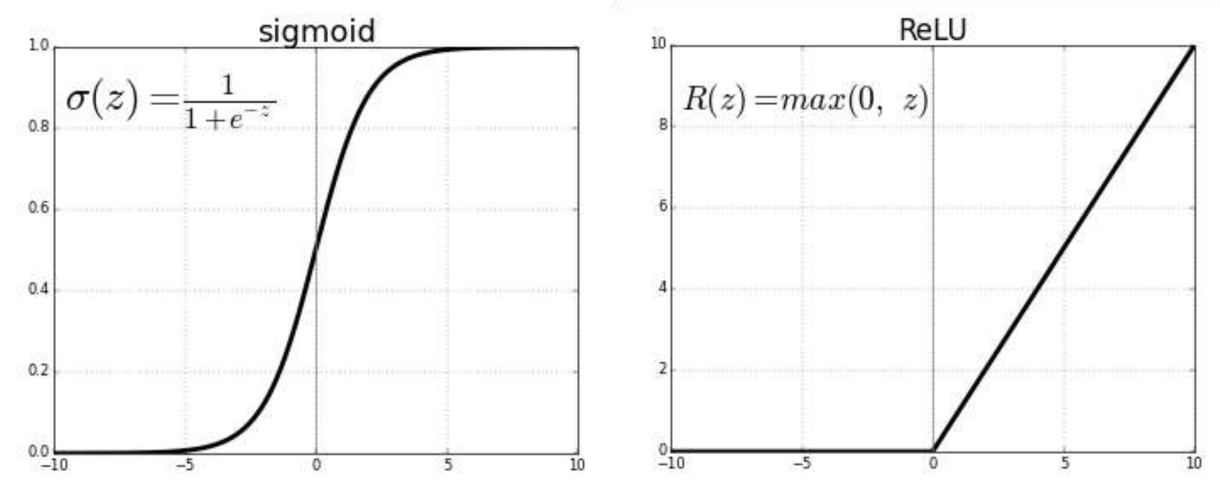
Cheated: we should have 10 (not 9) nodes ;-)

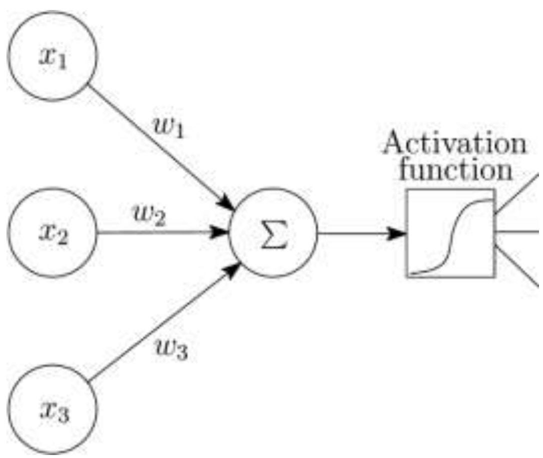
Question in home work:
How to initialize the weights? Does it matter?



How to initialize the weights?

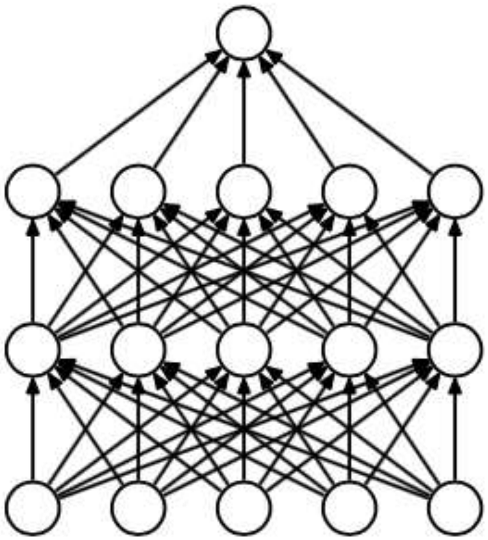
Question in home work:
Which activation function should we use? Does it matter?



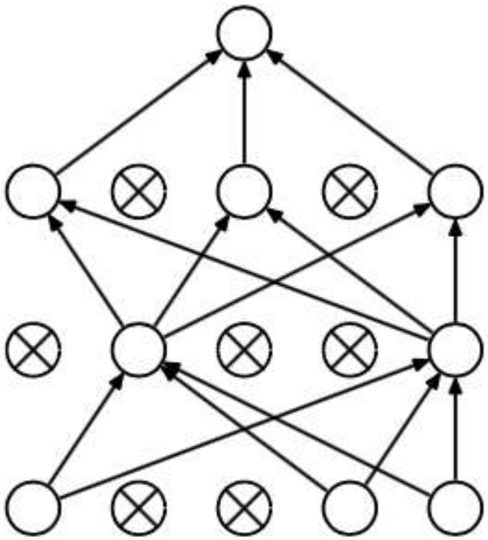


Question in home work: Dropout – does it help?

- After each weight update, we randomly "delete" a certain percentage of neurons, which will not be updated in the next step – than repeat.
- In each training step we optimize a slightly different NN model.

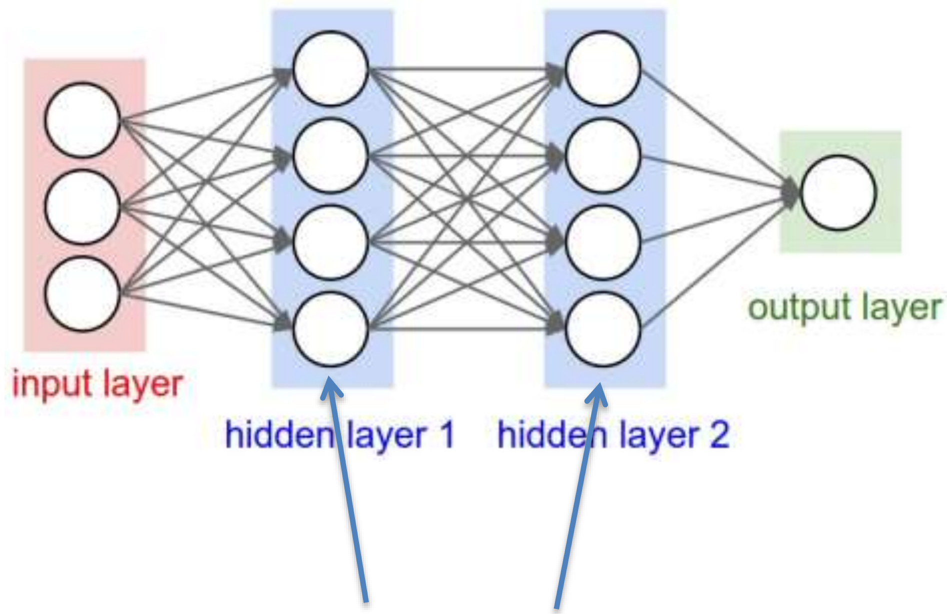


(a) Standard Neural Net

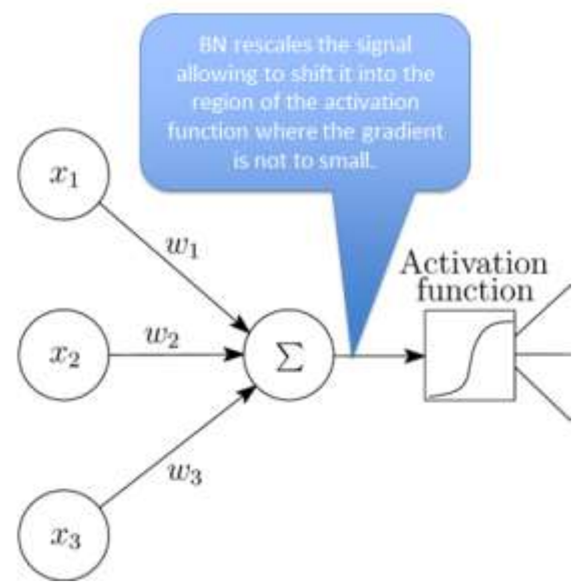


(b) After applying dropout.

Question in home work: Should we allow the NN to normalize the data between layers (batch_norm)? Does it matter?

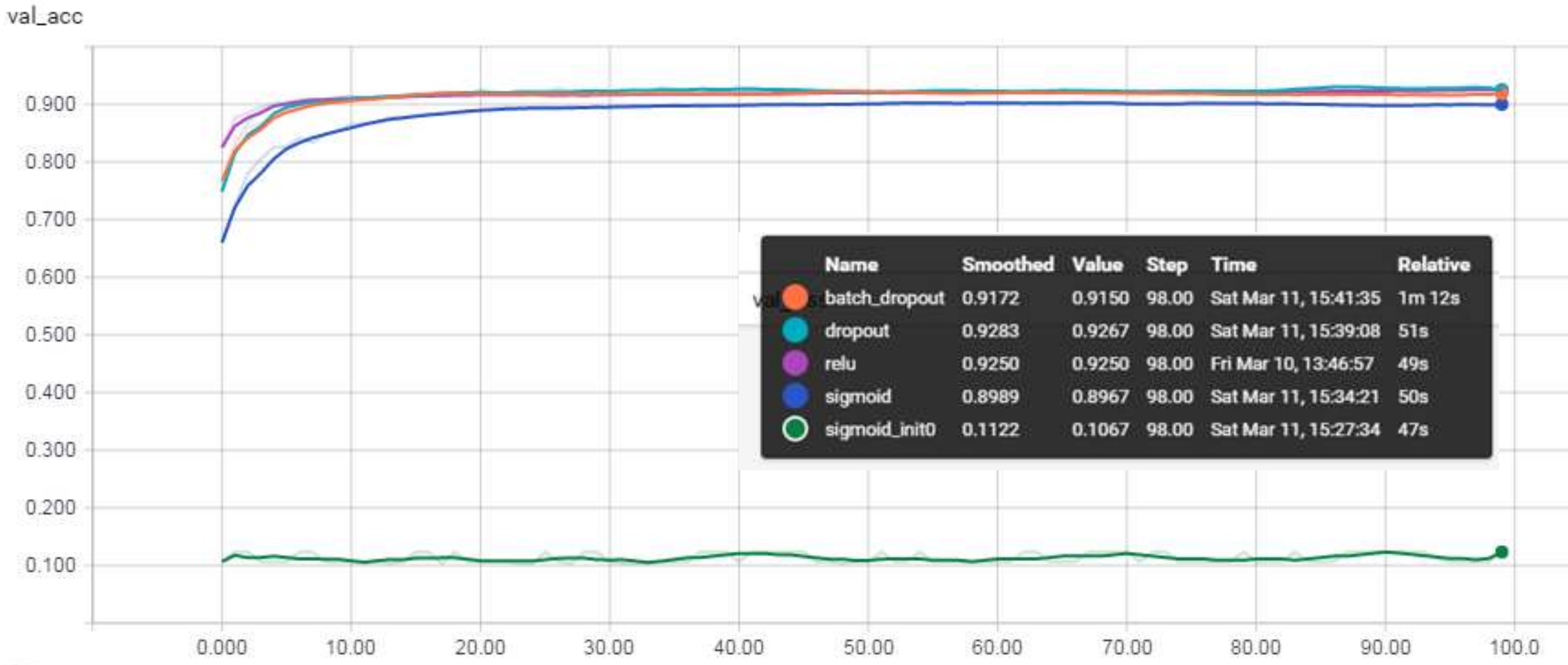


Should we allow the NN to normalize intermediate data (activations), so that they have mean=0 and sd=1?



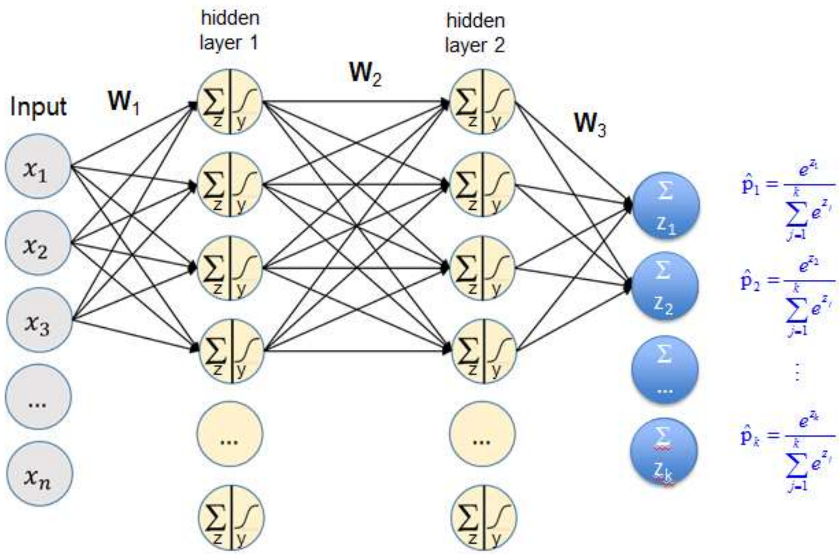
Home work – main result

With this small training set of 4000 images and 100 epochs training we get the best test accuracy of ~92% when working with random initialization (reducing weights if number of input data increases), ReLu, dropout and BN (here BN does not improve things – in many applications it does!).



Remark: we will later discuss why BatchNorm and Dropout help.

Bad idea: initializing all weights with the same value



$z = b + \sum_i (x_i \cdot w_i) \qquad y = f(z)$

$$w_i^{(t)} = w_i^{(t-1)} - \varepsilon^{(t)} \left. \frac{\partial C(\mathbf{w})}{\partial w_i} \right|_{w_i = w_i^{(t-1)}}$$

chain rule:

$$\left. \frac{\partial C}{\partial w_{2n}} \right|_{\text{current NN values}} = \left. \frac{\partial C}{\partial p_k} \right|_{cv} \cdot \left. \frac{\partial \hat{p}_k}{\partial z_{2m}} \right|_{cs} \cdot \left. \frac{\partial z_{2m}}{\partial y_{2m}} \right|_{cv} \cdot \left. \frac{\partial y_{2m}}{\partial z_{1n}} \right|_{cv} \cdot \left. \frac{\partial z_{1n}}{\partial w_{2n}} \right|_{cv}$$

Forward pass: initialize all weights with the same value

⇒ all units get the same values $y_j = f(z_j) = f(b + \sum_i x_i w_i)$

⇒ … all outputs are the same.. (Initializing all weights=0 will give all units the value 0!)

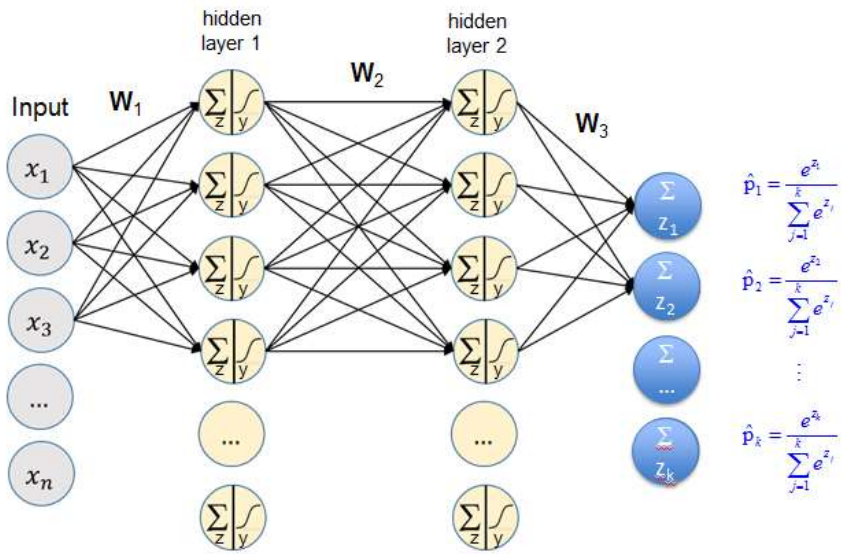
Backward pass: all weights and units have same values & all functions same

⇒ all gradients are the same

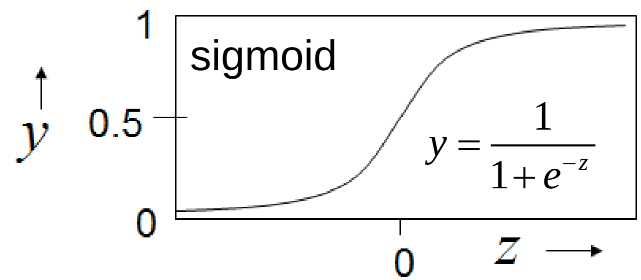
⇒ all weights get the same update and get again the same value!

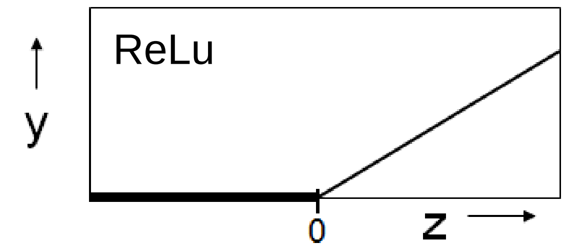
⇒ no learning

Bad idea: initializing with high values



$z = b + \sum_i (x_i \cdot w_i) \quad y = f(z)$





Initialize weights with partly large values.
⇒ large absolute z-values
⇒ flat parts of activation function

⇒ According chain rule we multiply with $\frac{\partial y}{\partial z} \approx 0$
⇒ gradient is zero we cannot update the weights ⇒ no learning

$$w_i^{(t)} = w_i^{(t-1)} - \varepsilon^{(t)} \left. \frac{\partial C(\mathbf{w})}{\partial w_i} \right|_{w_i = w_i^{(t-1)}}$$

What is the default initializer in Keras?

Also in conv layer 'glorot_uniform' is used as default initializer.

Dense https://keras.io/layers/core/ [source]

```
keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_i
```

glorot_uniform Recommended 2010 by Glorot & Bengio http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

```
glorot_uniform(seed=None)
```

Glorot uniform initializer, also called Xavier uniform initializer.

guarantees random small numbers

It draws samples from a uniform distribution within [-limit, limit] where `limit` is `sqrt(6 / (fan_in + fan_out))` where `fan_in` is the number of input units in the weight tensor and `fan_out` is the number of output units in the weight tensor.

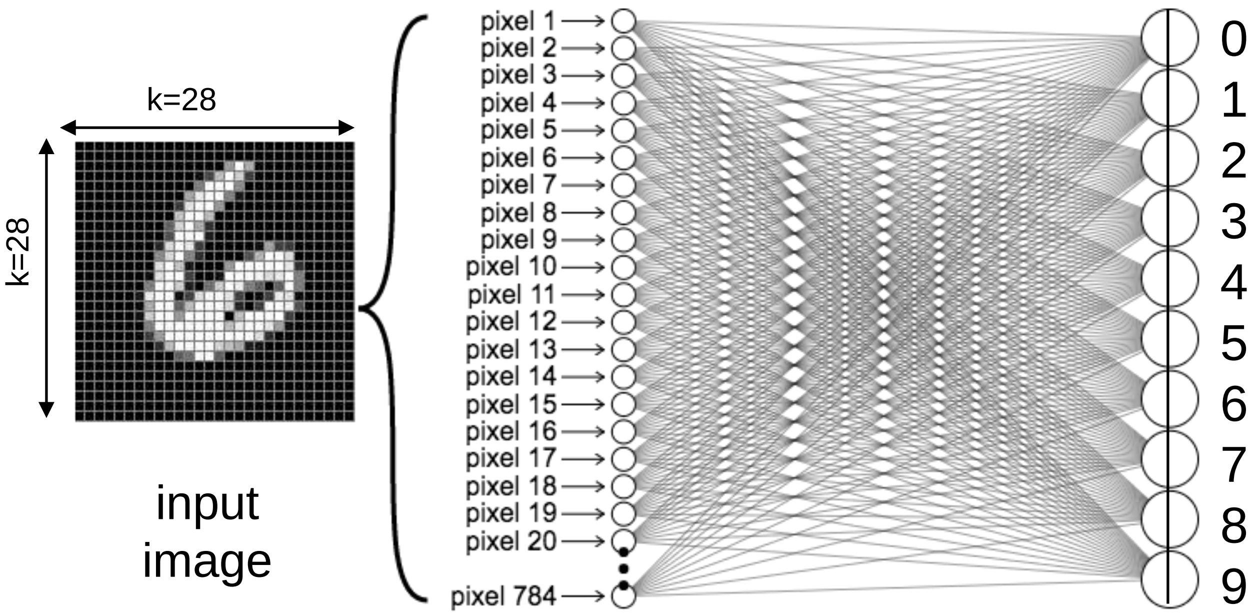
he_normal Great initializer for CNN with ReLu – see He's 2015 paper https://arxiv.org/abs/1502.01852

```
he_normal(seed=None)
```

He normal initializer.

It draws samples from a truncated normal distribution centered on 0 with `stddev = sqrt(2 / fan_in)` where `fan_in` is the number of input units in the weight tensor.

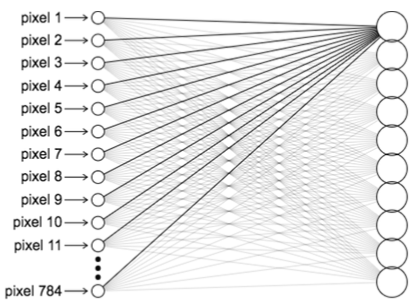
MNIST classification with a 1 layer fully connected NN



$$z_j = \sum_i (x_i \cdot w_{ij})$$

$$p_j = \frac{e^{z_j}}{\sum_{k=0}^{9} e^{z_k}}$$

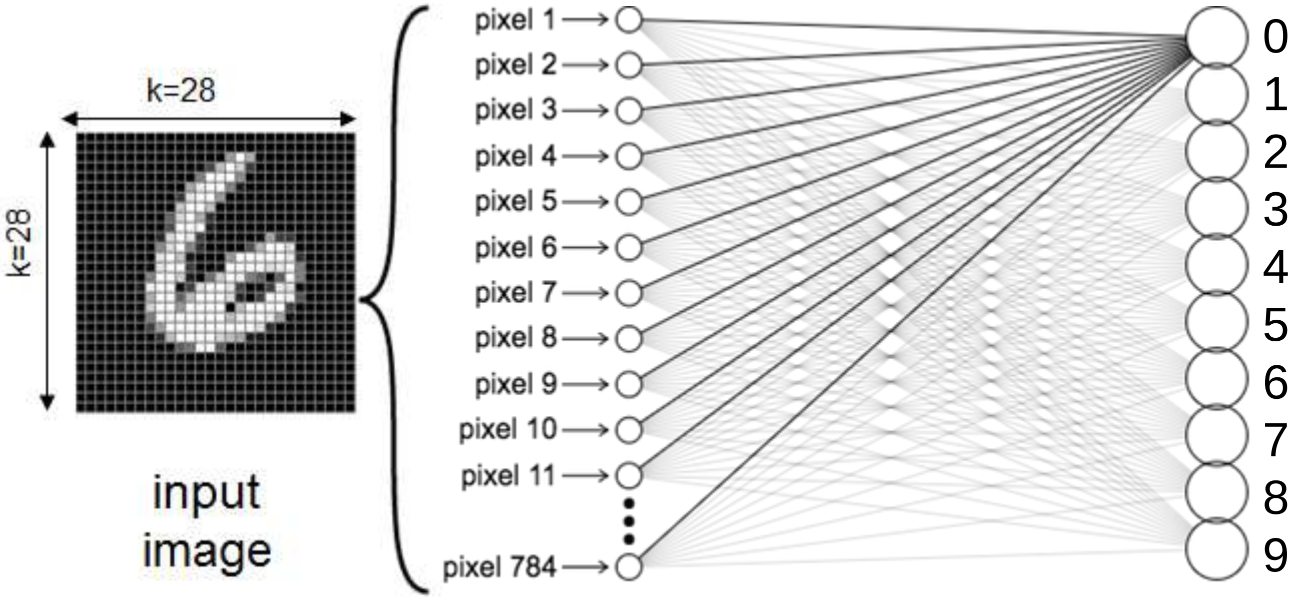
Each output node z_j gets 784 inputs connected via 784 weights which (usually the weights are different for each node) can also be arranged as 28x28 weight image



$$\begin{matrix} W_1 & W_2 & W_3 & \cdots & W_{28} \\ W_{29} & W_{30} & W_{31} & & W_{56} \\ W_{57} & W_{58} & W_{59} & & W_{84} \\ \vdots & & & \ddots & \\ W_{757} & W_{758} & W_{759} & & W_{784} \end{matrix}$$

Image credits: https://ml4a.github.io/ml4a/looking_inside_neural_nets/

MNIST classification with a 1 layer fc NN



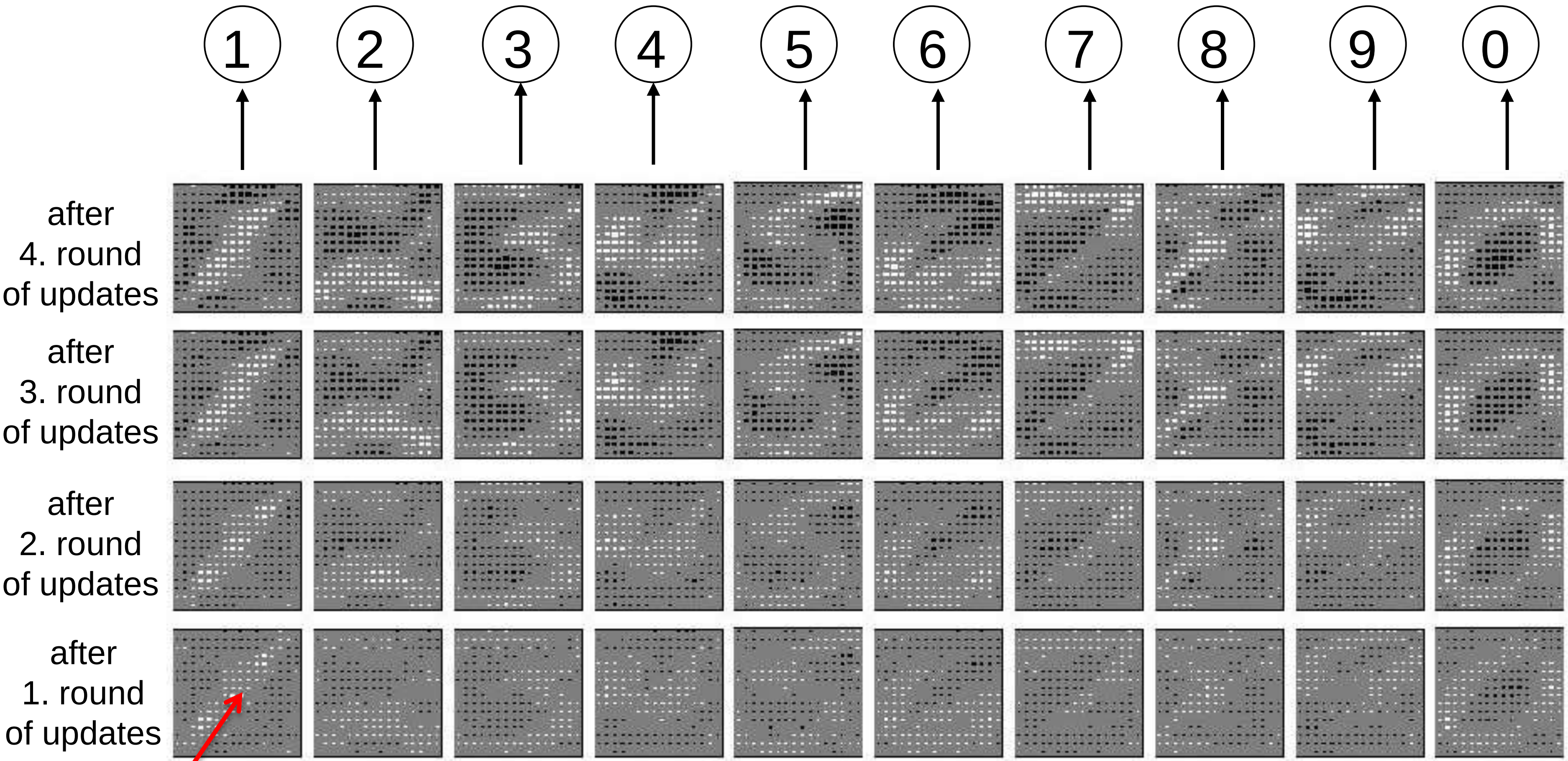
$$p_j = \frac{e^{z_j}}{\sum_{k=0}^{9} e^{z_k}}$$

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

- For the correct digit we want to get a high probability p_j $\Rightarrow$ we need a large z_j
- z_j is large if at large signal pixels x the w are also large and at small signal pixels x the w are also small
- For maximal p_j the j^th weight image looks like the input image!

Image credits: https://ml4a.github.io/ml4a/looking_inside_neural_nets/

Displaying learned weights at different steps of training via backpropagation

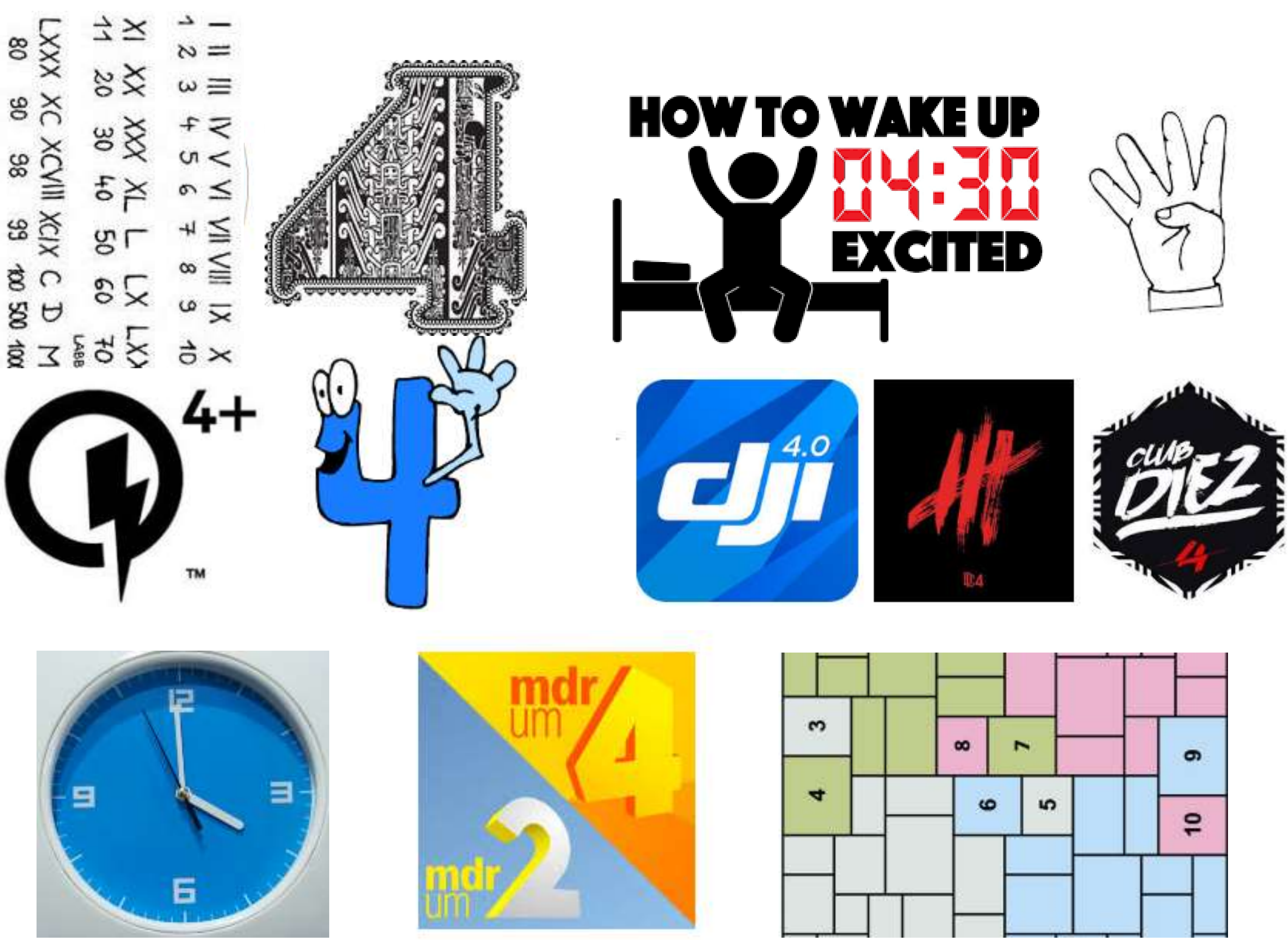


A 1 layer fully connected NN learns rigid template for each class!

"weight images"

Slide (modified) credits: Hinton, coursera on NN lecture 1

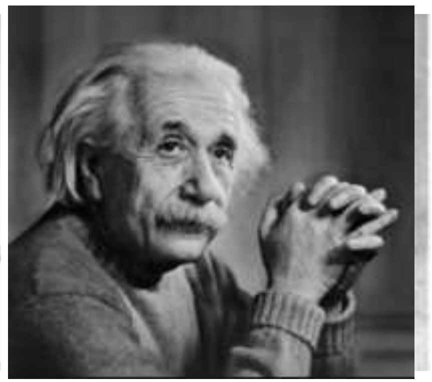
What is wrong with a rigid template? Find the 4!




Break

Some tasks are hard - how can solve them?

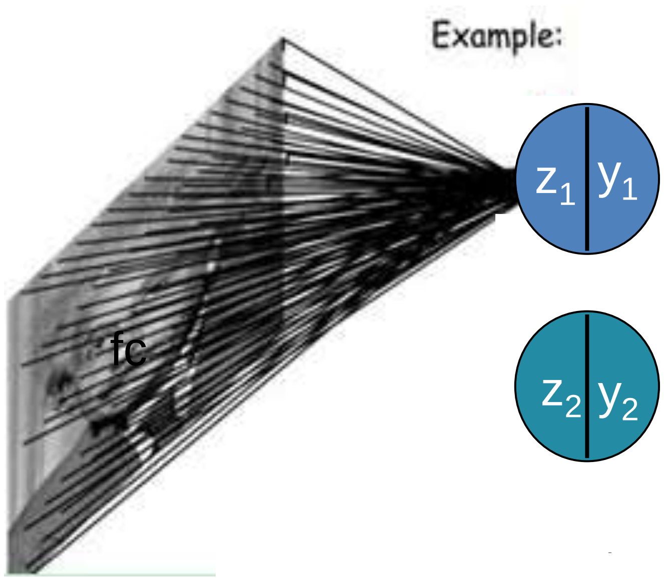
The training data consists of many different pictures of Oliver Dürr and Albert Einstein



How to distinguish Oliver from Einstein?

A hard task ! ☺

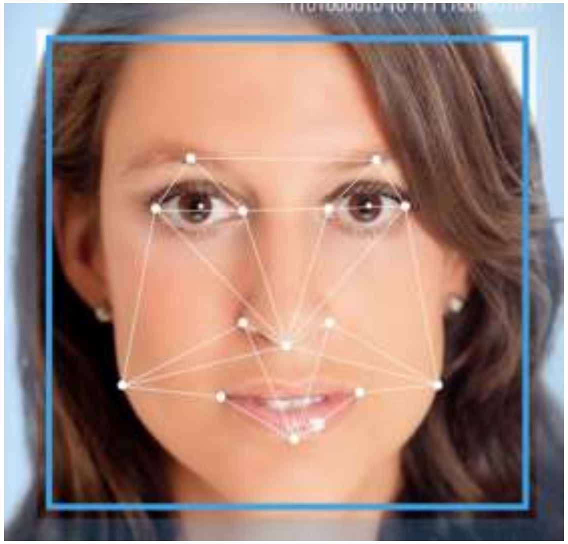
At least with only 2 neurons…



A rigid template is not enough!

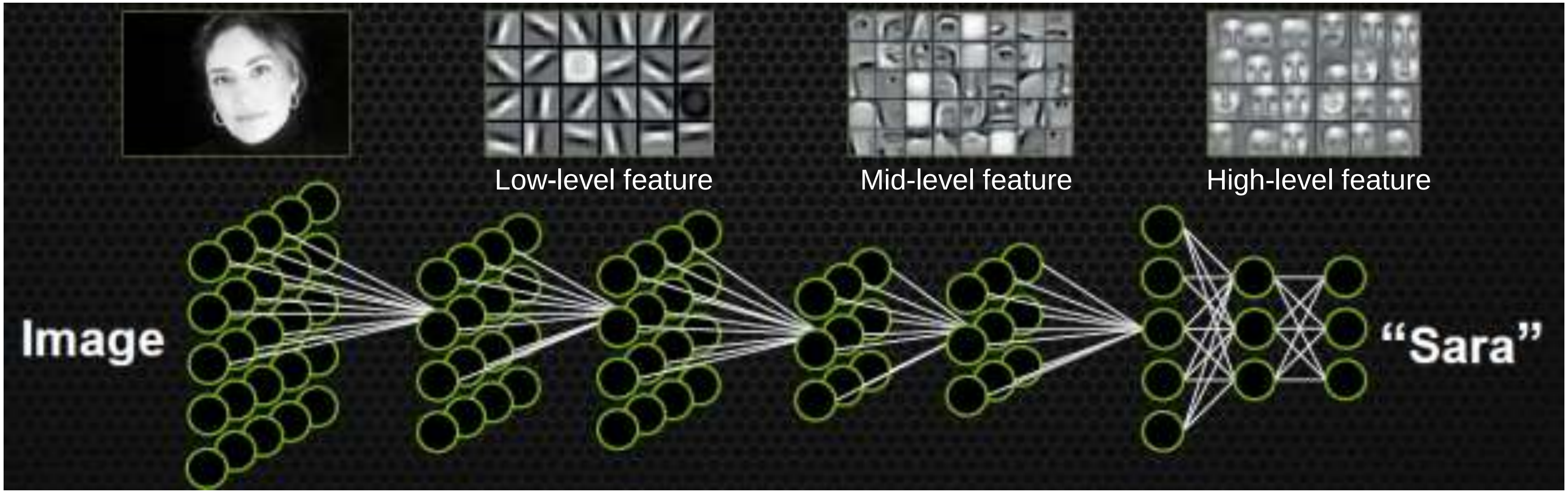
We should go beyond fc NN!

How to do computer vision? Traditional and DL approach

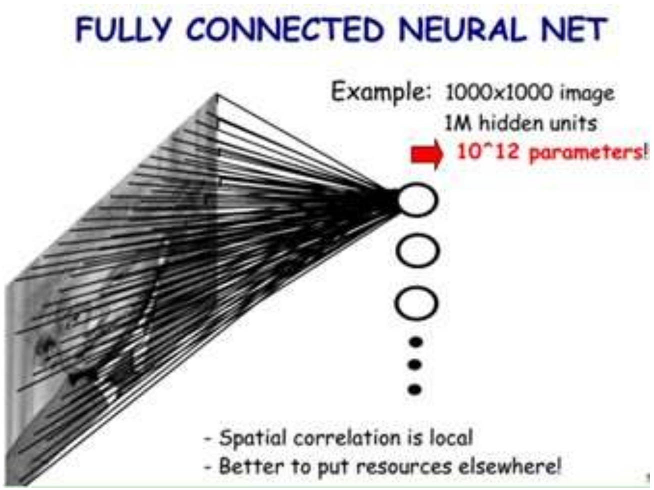


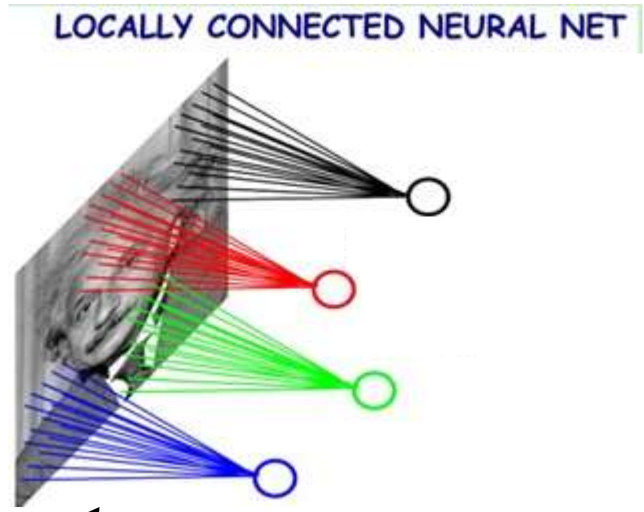
Traditional:
Extract handcrafted features & use these features to train / fit a model (e.g. logistic regression) and use fitted model to perform classification/prediction.

Deep learning using CNNs:
Based on variants of deep neural networks which learn during training/fitting appropriate hierarchical features e.g. from images. The fitted model (artificial neural net) can be used for classification/prediction.



Convolution extracts local information using few weights





Shared weights:

by using the same weights for each patch of the image we need much less parameters than in the fully connected NN and get from each patch the same kind of local feature information such as the presence of a edge.

Convolutional networks use neighborhood information and replicated local feature extraction

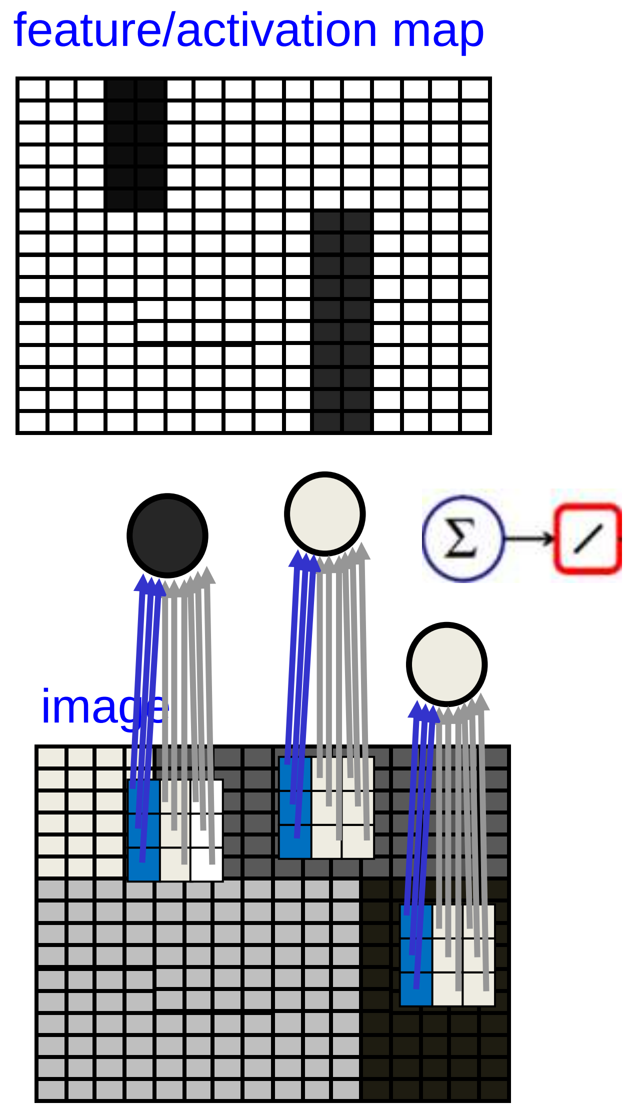
In a locally connected network the calculation rule

$$z = b + \sum_i (x_i \cdot w_i)$$

Corresponds to convolution of a filter with the image and the pattern of weights represent a filter.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

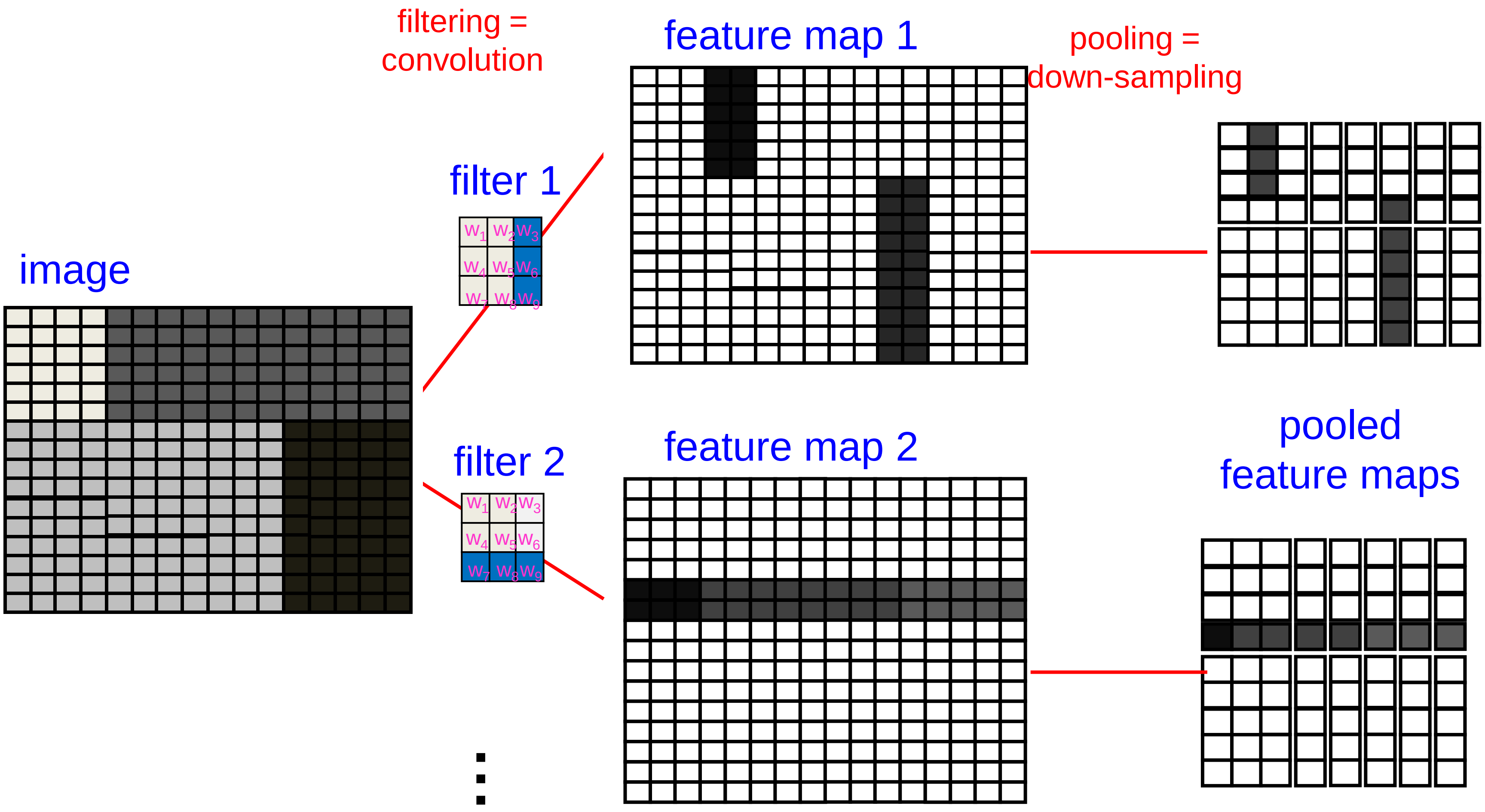
0.9	0.1	0.1
0.9	0.1	0.1
0.9	0.1	0.1



The filter is applied at each position of the image and it can be shown that the result is maximal if the image pattern corresponds to the weight pattern.

The results form again an image called feature map (=activation map) which shows at which position the feature is present.

Convolutional networks use neighborhood information and replicated local feature extraction



The weights of each filter are randomly initiated and then adapted during the training.

Convolutional networks
Building Block 1: Convolution

What is convolution?

At each position of the kernel we perform an elementwise dot-product between intensitiy values of the underlying input image

input image $\mathbf{X}$

35	40	41	45	50
40	40	42	46	52
42	46	50	55	55
48	52	56	58	60
56	60	65	70	75

×

kernel (or filter)

$\mathbf{W}$

0	1	0
0	0	0
0	0	1

=

output

		100		

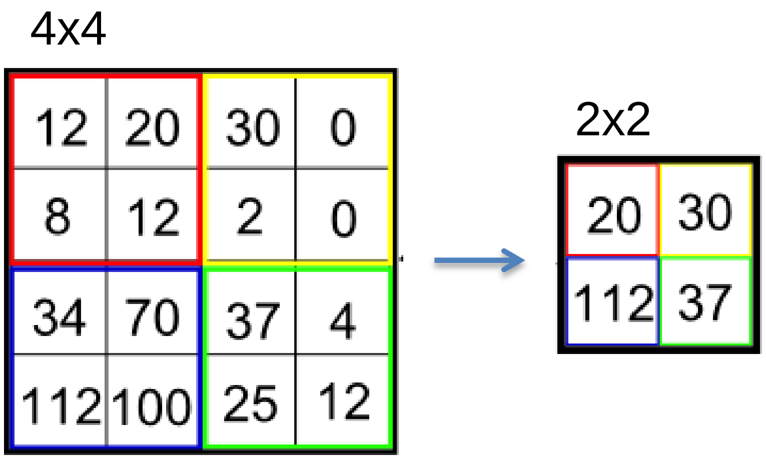
The 9 weights w_{ij} are called kernel (or filter)

$$z = b + \sum_i x_i w_i$$

In deep-learning the weights are not fixed, they are learned!

Image credits: http://docs.gimp.org/en/plug-in-convmatrix.html

Convolutional networks
Building Block 2: (Max-) pooling

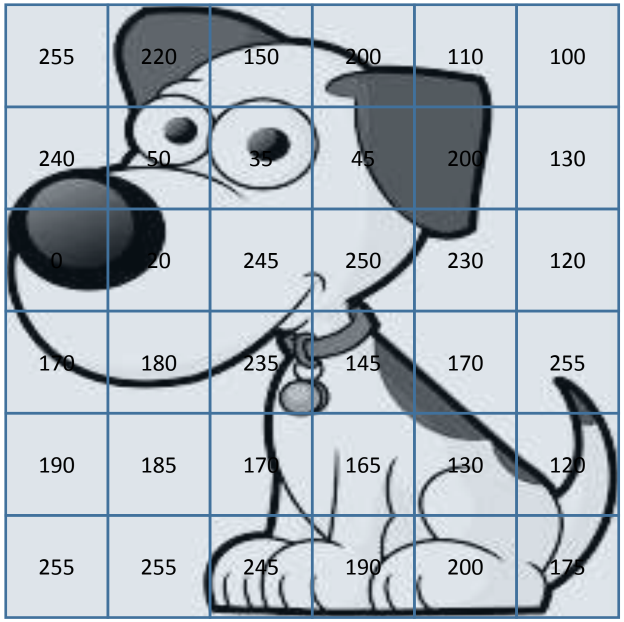


Simply join e.g. 2x2 adjacent pixels in one, e.g. by picking the maximum of all 4 pixel values (max-pooling) or determining the average of the
4 pixel values (average pooling)

Hinton: „The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster“

Convolutional neural networks

Input image 6x6x1



255	220	150	200	110	100
240	50	35	45	200	130
0	20	245	250	230	120
170	180	235	145	170	255
190	185	170	165	130	120
255	255	245	190	200	175

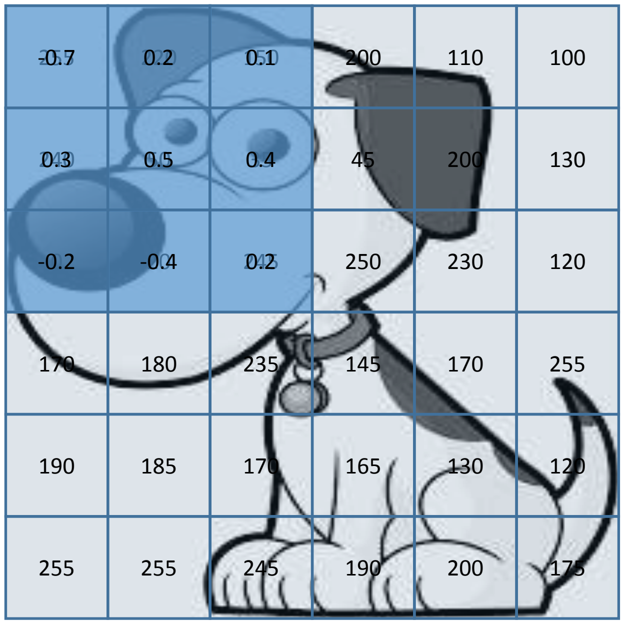
-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

Convolutional neural networks

Input image 6x6x1



Feature map
4x4x1

32.5

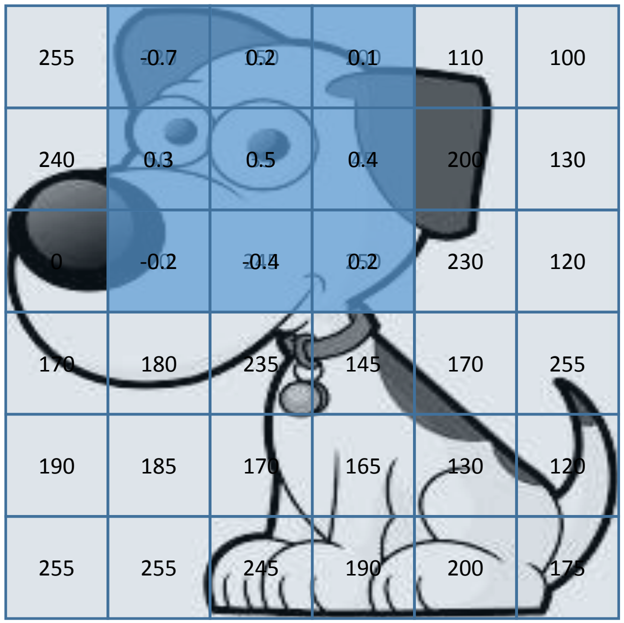
-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

Convolutional neural networks

Input image 6x6x1



Feature map
4x4x1

32.5	-105.5

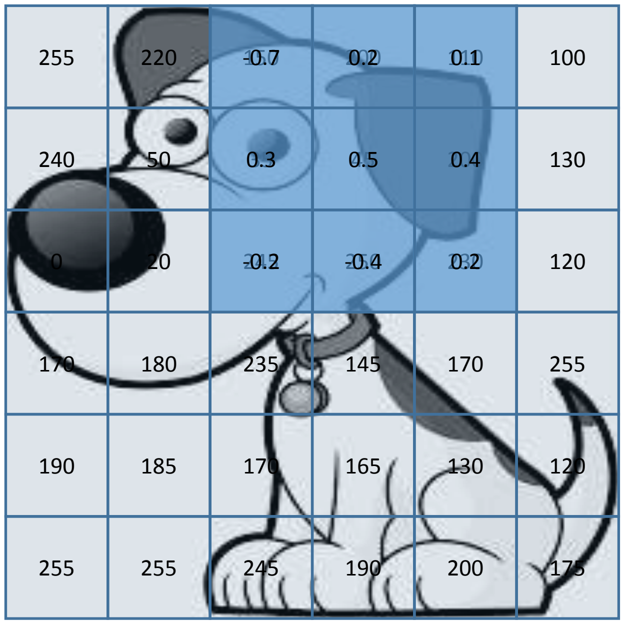
-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

Convolutional neural networks

Input image 6x6x1



255	220	-0.7	0.2	0.1	100
240	50	0.3	0.5	0.4	130
0	20	-0.2	-0.4	0.2	120
170	180	235	145	170	255
190	185	170	165	130	120
255	255	245	190	200	175

Feature map 4x4x1

32.5	-105.5	185.5

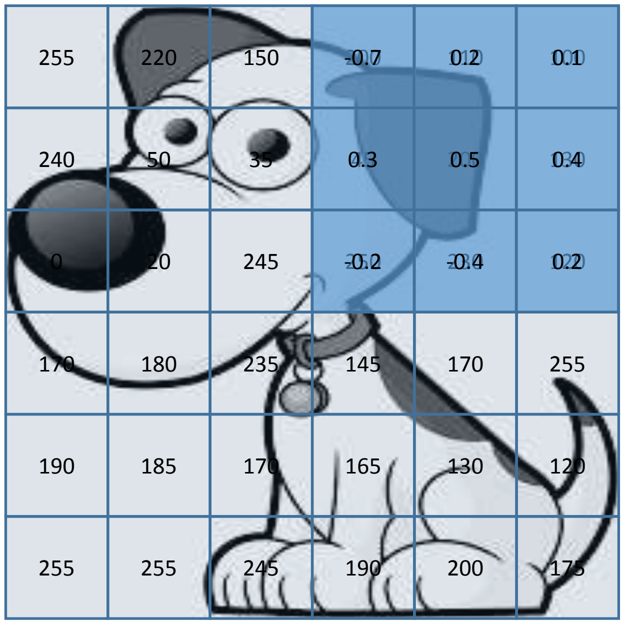
-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

Convolutional neural networks

Input image 6x6x1



Feature map
4x4x1

32.5	-105.5	185.5	54

-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

Convolutional neural networks

Input image 6x6x1

255	220	150	200	110	100
-0.7	0.2	0.1	45	200	130
0.3	0.5	0.4	250	230	120
-0.2	-0.4	0.2	145	170	255
190	185	170	165	130	120
255	255	245	190	200	175

Feature map 4x4x1

32.5	-105.5	185.5	54
-105.5			

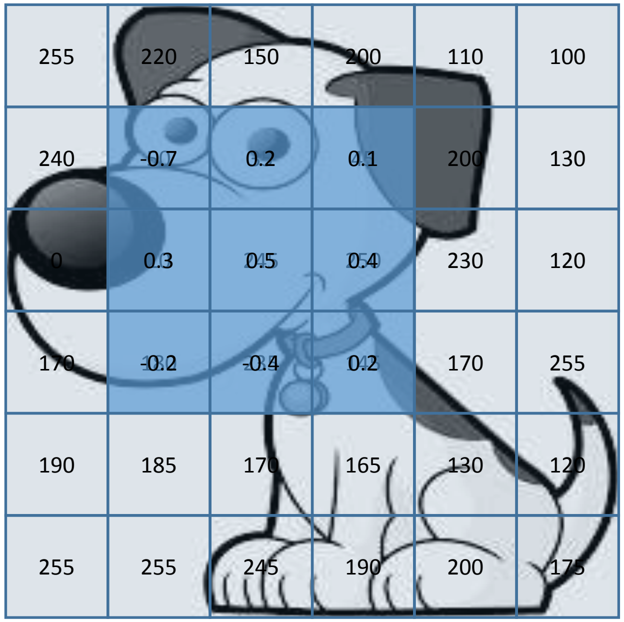
-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

Convolutional neural networks

Input image 6x6x1



255	220	150	200	110	100
240	-0.7	0.2	0.1	200	130
0	0.3	0.5	0.4	230	120
170	-0.2	-0.4	0.2	170	255
190	185	170	165	130	120
255	255	245	190	200	175

Feature map 4x4x1

32.5	-105.5	185.5	54
-105.5	104		

-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

Convolutional neural networks

Input image 6x6x1

255	220	150	200	110	100
240	50	-0.7	0.2	0.1	130
0	20	0.3	0.5	0.4	120
170	180	-0.2	-0.4	0.2	255
190	185	170	165	130	120
255	255	245	190	200	175

Feature map 4x4x1

32.5	-105.5	185.5	54
-105.5	104	217.5	

-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

Convolutional neural networks

Input image 6x6x1

255	220	150	200	110	100
240	50	35	-0.7	0.2	0.1
0	20	245	0.3	0.5	0.4
170	180	235	-0.2	-0.4	0.2
190	185	170	165	130	120
255	255	245	190	200	175

Feature map
4x4x1

32.5	-105.5	185.5	54
-105.5	104	217.5	31

-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

Convolutional neural networks

Input image 6x6x1

255	220	150	200	110	100
240	50	35	45	200	130
-0.7	0.2	0.1	250	230	120
0.3	0.5	0.4	145	170	255
-0.2	-0.4	0.2	165	130	120
255	255	245	190	200	175

Feature map 4x4x1

32.5	-105.5	185.5	54
-105.5	104	217.5	31
-44			

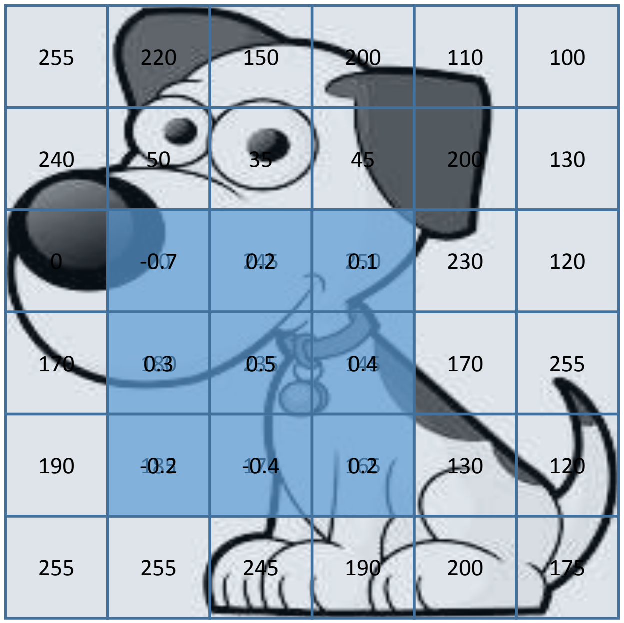
-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

Convolutional neural networks

Input image 6x6x1



255	220	150	200	110	100
240	50	35	45	200	130
0	-0.7	0.2	0.1	230	120
170	0.3	0.5	0.4	170	255
190	-0.2	-0.4	0.2	130	120
255	255	245	190	200	175

Feature map 4x4x1

32.5	-105.5	185.5	54
-105.5	104	217.5	31
-44	224		

-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

Convolutional neural networks

Input image 6x6x1

255	220	150	200	110	100
240	50	35	45	200	130
0	20	-0.7	0.2	0.1	120
170	180	0.3	0.5	0.4	255
190	185	-0.2	-0.4	0.2	120
255	255	245	190	200	175

Feature map 4x4x1

32.5	-105.5	185.5	54
-105.5	104	217.5	31
-44	224	38.5	

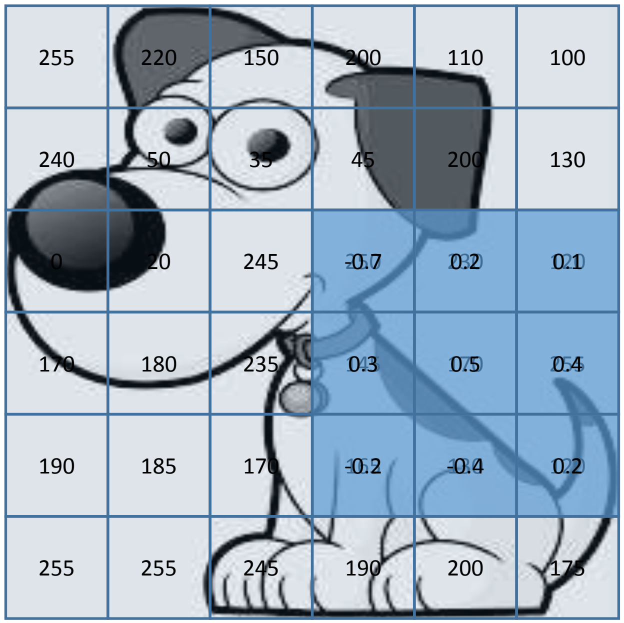
-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

Convolutional neural networks

Input image 6x6x1



255	220	150	200	110	100
240	50	35	45	200	130
0	20	245	-0.7	0.2	0.1
170	180	235	0.3	0.5	0.4
190	185	170	-0.2	-0.4	0.2
255	255	245	190	200	175

Feature map 4x4x1

32.5	-105.5	185.5	54
-105.5	104	217.5	31
-44	224	38.5	-18

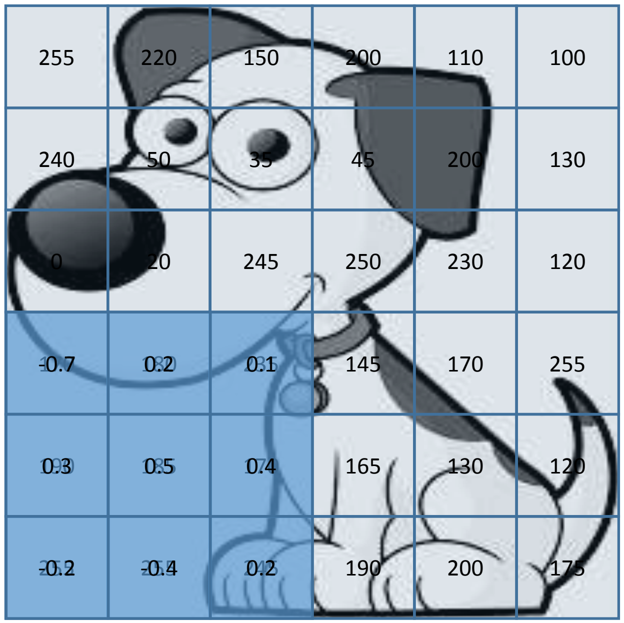
-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

Convolutional neural networks

Input image 6x6x1



255	220	150	200	110	100
240	50	35	45	200	130
0	20	245	250	230	120
-0.7	0.2	0.1	145	170	255
0.3	0.5	0.4	165	130	120
-0.2	-0.4	0.2	190	200	175

Feature map 4x4x1

32.5	-105.5	185.5	54
-105.5	104	217.5	31
-44	224	38.5	-18
-60.5			

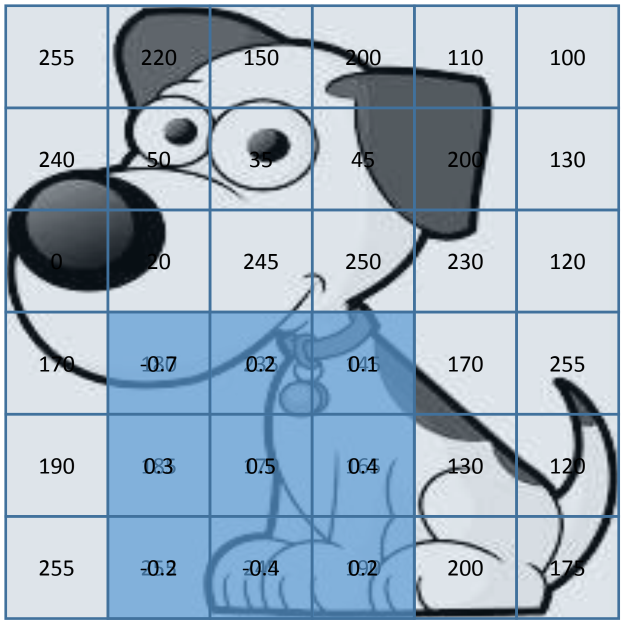
-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

Convolutional neural networks

Input image 6x6x1



255	220	150	200	110	100
240	50	35	45	200	130
0	20	245	250	230	120
170	-0.7	0.2	0.1	170	255
190	0.3	0.5	0.4	130	120
255	-0.2	-0.4	0.2	200	175

Feature map
4x4x1

32.5	-105.5	185.5	54
-105.5	104	217.5	31
-44	224	38.5	-18
-60.5	213.5		

-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

Convolutional neural networks

Input image 6x6x1

255	220	150	200	110	100
240	50	35	45	200	130
0	20	245	250	230	120
170	180	-0.7	0.2	0.1	255
190	185	0.3	0.5	0.4	120
255	255	-0.2	-0.4	0.2	175

Feature map
4x4x1

32.5	-105.5	185.5	54
-105.5	104	217.5	31
-44	224	38.5	-18
-60.5	213.5	52.5	

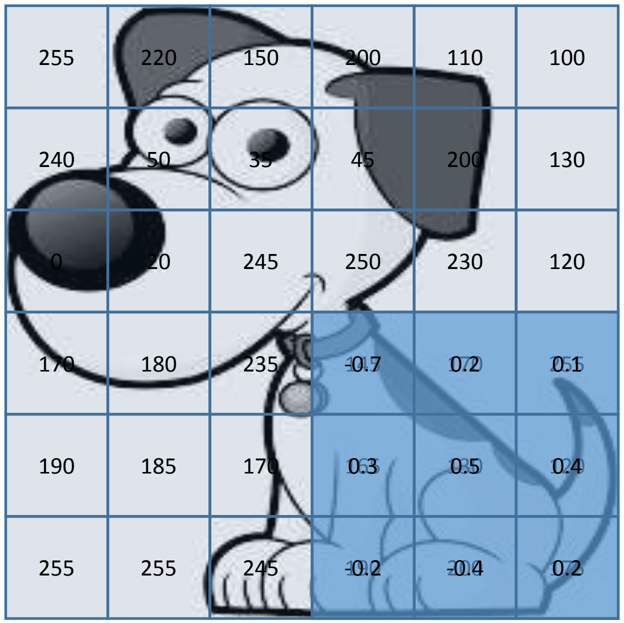
-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i \left(x_i \cdot w_{ij} \right)$$

Convolutional neural networks

Input image 6x6x1



Feature map 4x4x1

32.5	-105.5	185.5	54
-105.5	104	217.5	31
-44	224	38.5	-18
-60.5	213.5	52.5	37.5

-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

Convolutional neural networks

Input image 6x6x1

255	220	150	200	110	100
240	50	35	45	200	130
0	20	245	250	230	120
170	180	235	-0.7	0.2	0.1
190	185	170	0.3	0.5	0.4
255	255	245	-0.2	-0.4	0.2

Feature map 4x4x1

32.5	-105.5	185.5	54
-105.5	104	217.5	31
-44	224	38.5	-18
-60.5	213.5	52.5	37.5

Relu

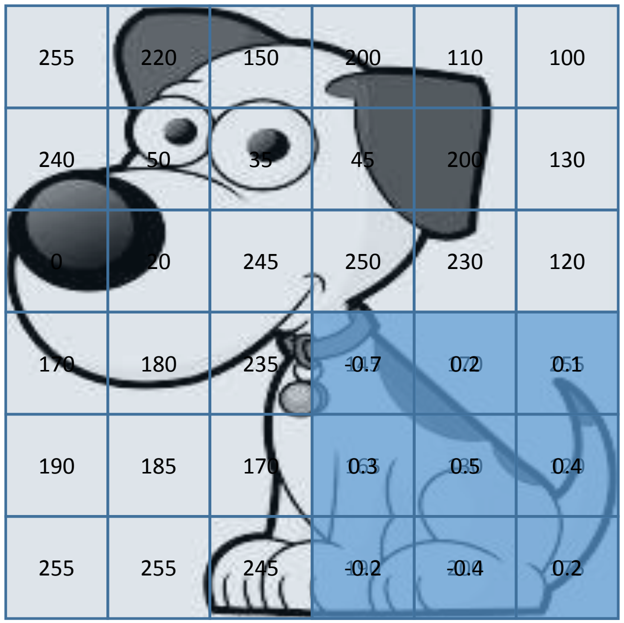
32.5	0	185.5	54
0	104	217.5	31
0	224	38.5	0
0	213.5	52.5	37.5

-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

Convolutional neural networks

Input image 6x6x1



255	220	150	200	110	100
240	50	35	45	200	130
0	20	245	250	230	120
170	180	235	-0.7	0.2	0.1
190	185	170	0.3	0.5	0.4
255	255	245	-0.2	-0.4	0.2

Feature map 4x4x1

32.5	-105.5	185.5	54
-105.5	104	217.5	31
-44	224	38.5	-18
-60.5	213.5	52.5	37.5

Relu

32.5	0	185.5	54
0	104	217.5	31
0	224	38.5	0
0	213.5	52.5	37.5

Maxpool (2x2x1)

104

-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

Convolutional neural networks

Input image 6x6x1

255	220	150	200	110	100
240	50	35	45	200	130
0	20	245	250	230	120
170	180	235	-0.7	0.2	0.1
190	185	170	0.3	0.5	0.4
255	255	245	-0.2	-0.4	0.2

Feature map 4x4x1

32.5	-105.5	185.5	54
-105.5	104	217.5	31
-44	224	38.5	-18
-60.5	213.5	52.5	37.5

Relu

32.5	0	185.5	54
0	104	217.5	31
0	224	38.5	0
0	213.5	52.5	37.5

Maxpool (2x2x1)

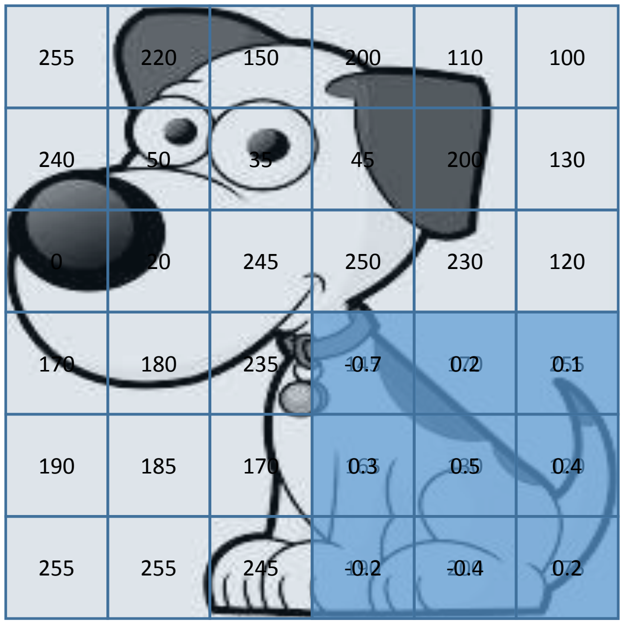
104	217.5

-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

Convolutional neural networks

Input image 6x6x1



3x3 filter

-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

Feature map 4x4x1

32.5	-105.5	185.5	54
-105.5	104	217.5	31
-44	224	38.5	-18
-60.5	213.5	52.5	37.5

Relu

32.5	0	185.5	54
0	104	217.5	31
0	224	38.5	0
0	213.5	52.5	37.5

Maxpool (2x2x1)

104	217.5
224	

Convolutional neural networks

Input image 6x6x1

255	220	150	200	110	100
240	50	35	45	200	130
0	20	245	250	230	120
170	180	235	-0.7	0.2	0.1
190	185	170	0.3	0.5	0.4
255	255	245	-0.2	-0.4	0.2

Feature map 4x4x1

32.5	-105.5	185.5	54
-105.5	104	217.5	31
-44	224	38.5	-18
-60.5	213.5	52.5	37.5

Relu

32.5	0	185.5	54
0	104	217.5	31
0	224	38.5	0
0	213.5	52.5	37.5

Maxpool (2x2x1)

104	217.5
224	52.5

-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

One kernel or filter searches for specific local feature

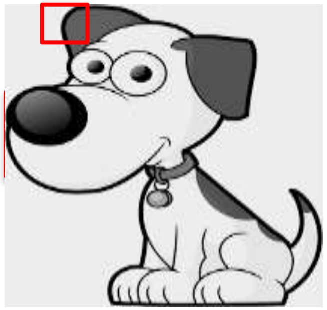


image patch

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

*

filter/kernel: curve detector

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

=6600

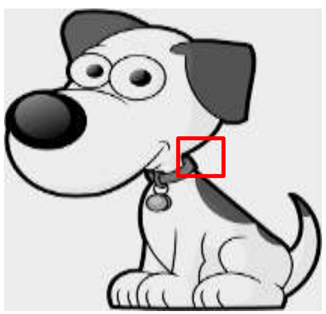


image patch

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

filter/kernel: curve detector

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

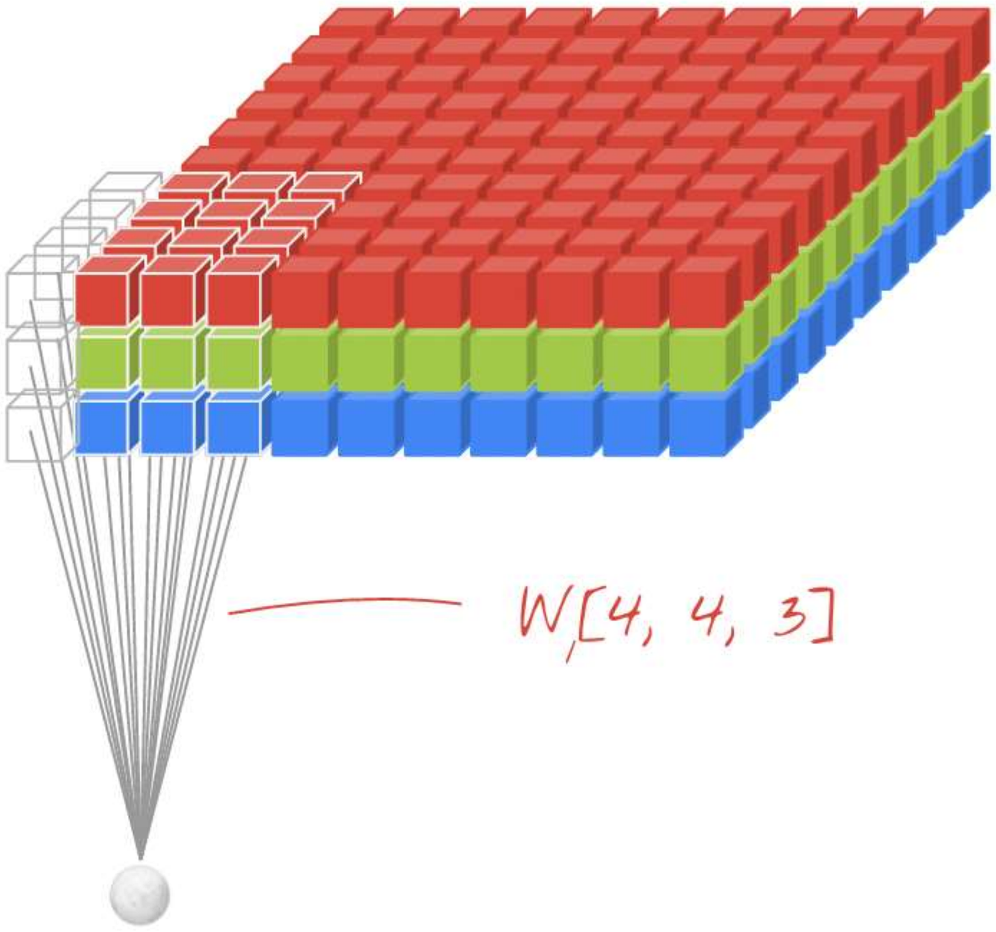
Pixel representation of filter

=0

We get a large resulting value if the filter resembles the pattern in the image patch on which the filter was applied.

credits: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

Animated convolution with 3 input channels



Animation credits: M.Gorner, https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#10

Convolution with 3 input channels and k output channels

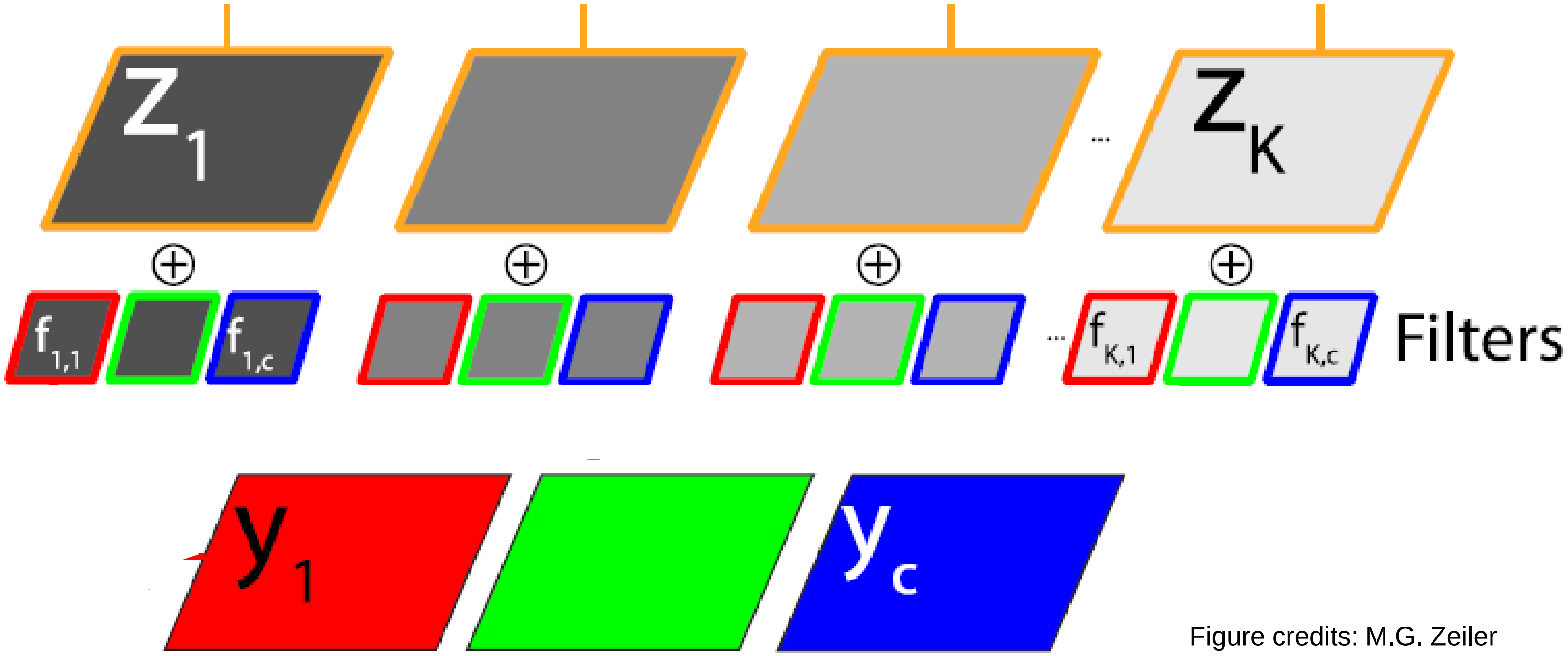


Figure credits: M.G. Zeiler

$$z_k = b + \sum_j \sum_i \left(y_{ji} \cdot {}^k w_{ji} \right) = b + \left(\sum_i \left(y_{1i} \cdot {}^k w_{1i} \right) + \sum_i \left(y_{2i} \cdot {}^k w_{2i} \right) + \sum_i \left(y_{3i} \cdot {}^k w_{3i} \right) \right)$$

We can imagine to have for each input channel y_c a channel specific filter f_c and get one output activation map per channel and the we add them element-wise.

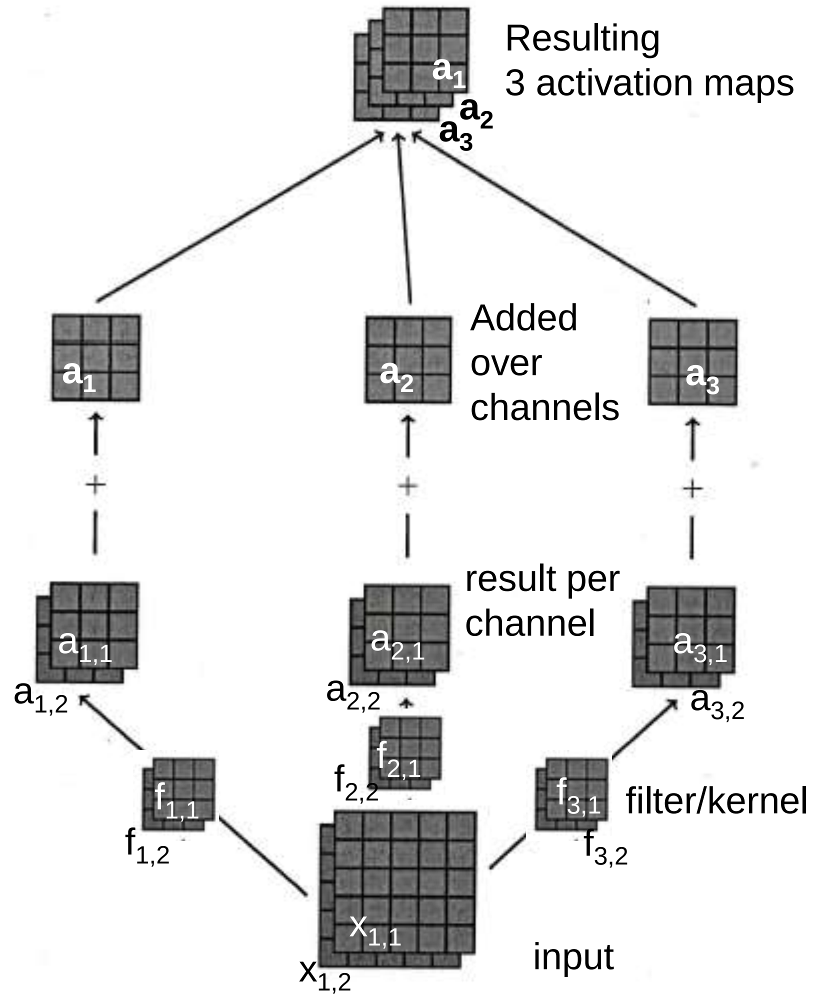
Convolution with 2 input channels and 3 output channels

Here, we need 3-times 2 filters.

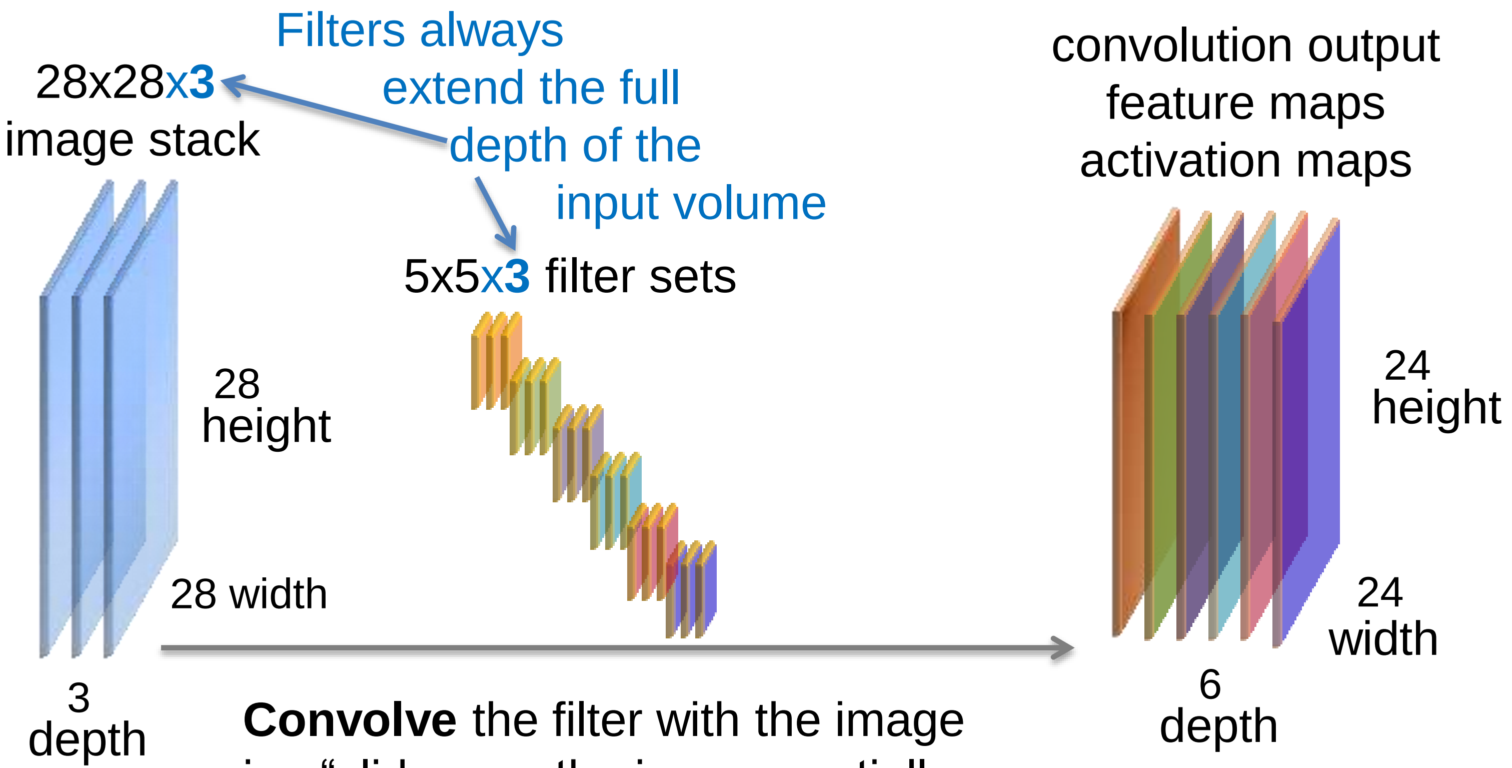
To get the **first output channel** we convolve the first filter $\mathbf{f}_{1,1}$ of the **first** 2-filter-block with the first input channel $\mathbf{x}_{1,1}$ and the second filter $\mathbf{f}_{1,2}$ of the **first** 2-filter-block with the second input channel $\mathbf{x}_{1,2}$. Then we sum the channel-wise resulting activation maps $a_{1,1}$ and $a_{1,2}$ at each position to get the first output channel a_1.

To get the **second output channel** we convolve $\mathbf{f}_{2,1}$ with the $\mathbf{x}_{1,1}$ and $\mathbf{f}_{2,2}$ with the $\mathbf{x}_{1,2}$. Elementwise adding of the results yields a_2.

To get the **third output channel** we convolve $\mathbf{f}_{3,1}$ with the $\mathbf{x}_{1,1}$ and $\mathbf{f}_{3,2}$ with the $\mathbf{x}_{1,2}$. Elementwise adding of the results yields a_3.



Convolution continued

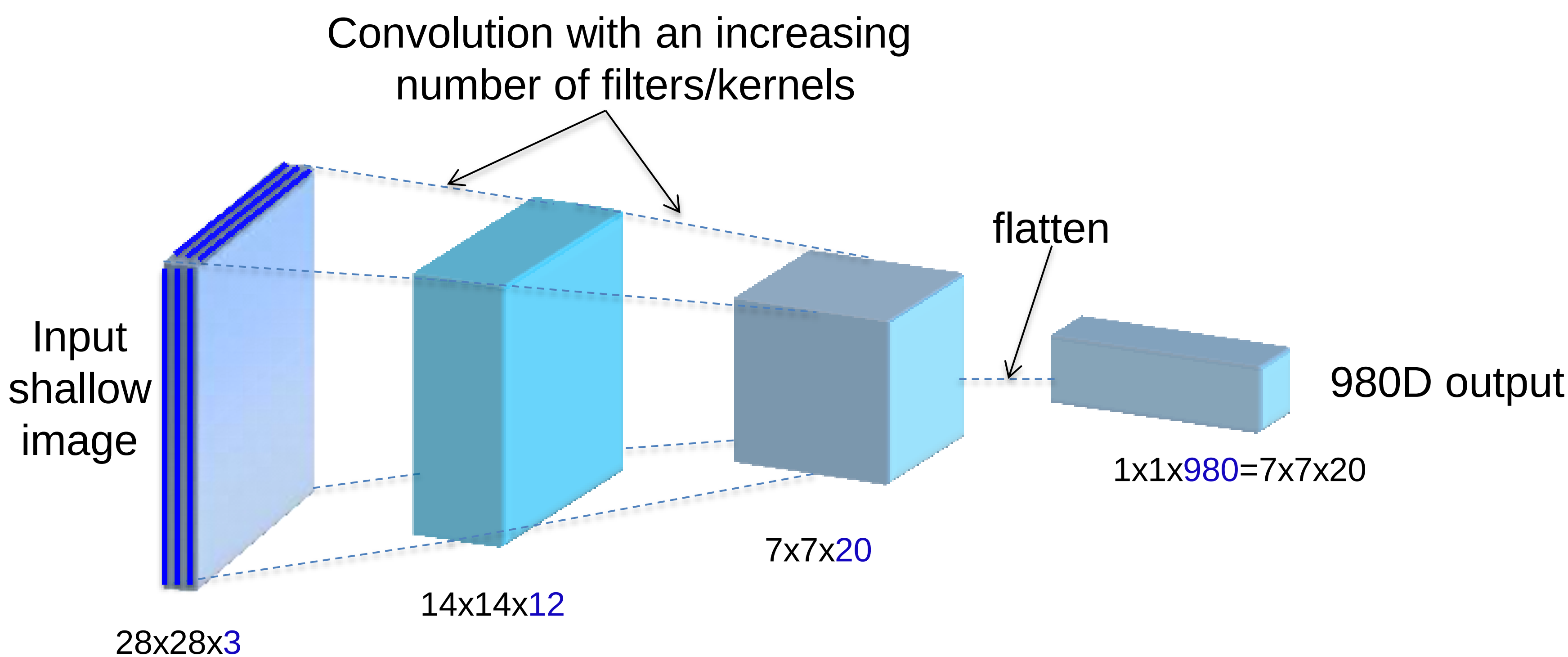


Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products at each position"

Output size in case of
6 5x5x3 filter
no zero-padding
stride=1

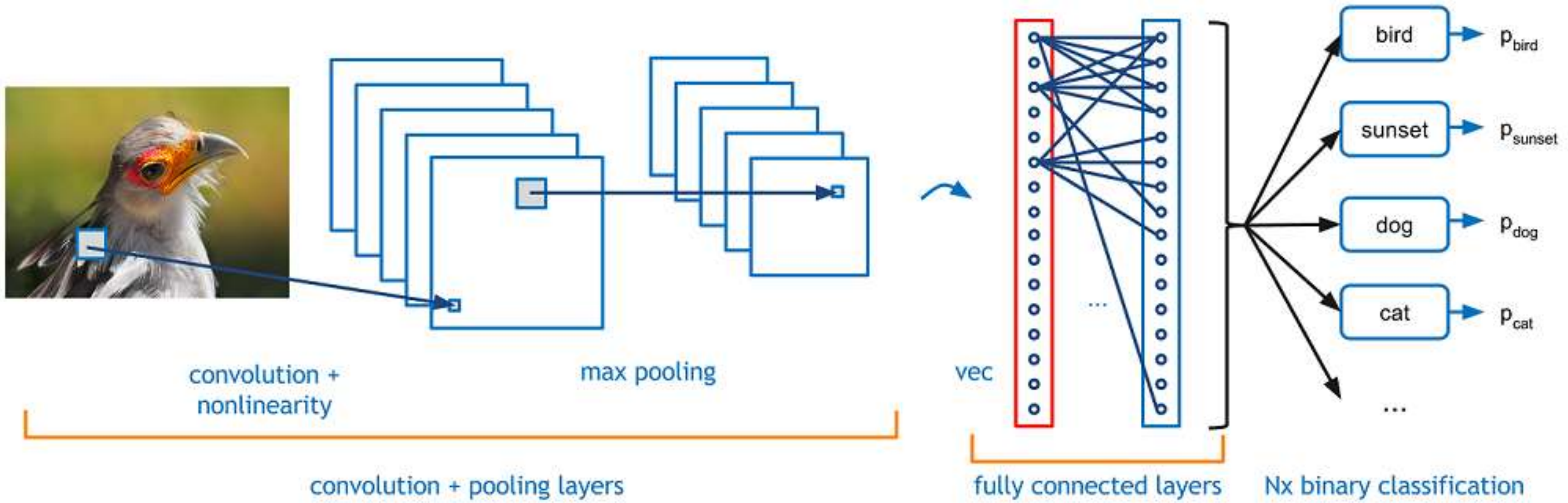
Output: 24x24x6

Typical shape of a classical CNN



Spatial resolution is decreased e.g. via max-pooling while more abstract image features are detected in deeper layers.

A classical CNN has fc layers at the end



In a classical CNN we start with convolution layers and end with fc layers.

The task of the convolutional layers is to extract useful features from the image which might be appear at arbitrary positions in the image.

The task of the fc layer is to use these extracted features for classification.

Image credits:
https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

Fast prototyping CNNs with Keras

For a plain CNN we can use Keras' Sequential API

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, BatchNormalization
from keras.layers import Convolution2D, MaxPooling2D, Flatten
import keras
```

For documentation see:

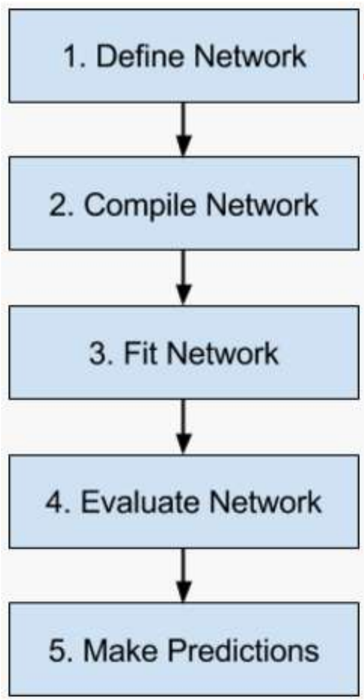
https://keras.io/layers/core/

https://keras.io/layers/convolutional/

https://keras.io/layers/pooling/

Keras: A high level API with best practice defaults

Number filter/feature maps (activation maps) in first hidden layers




```
model = Sequential()
# this applies 32 convolution filters of size 3x3 each.

model.add(Convolution2D(32, (3, 3), border_mode='same', input_shape=(28, 28, 1)))
model.add(Activation('relu'))

model.add(Flatten())

model.add(Dense(10))
model.add(Activation('softmax'))
```

Will arrange the 32x28x28 = 25088 neurons, that are located in 32 feature maps of dim 28x28, in one vector.

```
model.summary()
```

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
convolution2d_2 (Convolution2D)  (None, 28, 28, 32)    320         convolution2d_input_2[0][0]
____________________________________________________________________________________________________
activation_3 (Activation)        (None, 28, 28, 32)    0           convolution2d_2[0][0]
____________________________________________________________________________________________________
flatten_2 (Flatten)              (None, 25088)         0           activation_3[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 10)            250890      flatten_2[0][0]
____________________________________________________________________________________________________
activation_4 (Activation)        (None, 10)            0           dense_2[0][0]
====================================================================================================
```

Exercise on setting up a simple CNN in keras

Check out the architecture of the CNN described in [“live cnn in browser”]()

And fill in the pieces to get a Keras code for a model with this architecture

write a digit

Check out the default parameters and the keras syntax starting from:

https://keras.io/layers/convolutional/   https://keras.io/layers/core/   or google…

```
model = Sequential()
model.add(Conv2D(filters= ...,
                 kernel_size=(..., ...),
                 input_shape=(..., ..., ...))
model.add(Activation('...'))
model.add(Conv2D(filters= ...,
                 kernel_size=(..., ...))
model.add(Activation(...))
model.add(MaxPooling2D(pool_size=(..., ...)))
model.add(Dropout(...))
model.add(Flatten())
model.add(Dense(...))
model.add(Activation('...'))
model.add(Dropout(...))
model.add(Dense(...))
model.add(Activation('softmax'))
```

Exercise on setting up a simple CNN in keras

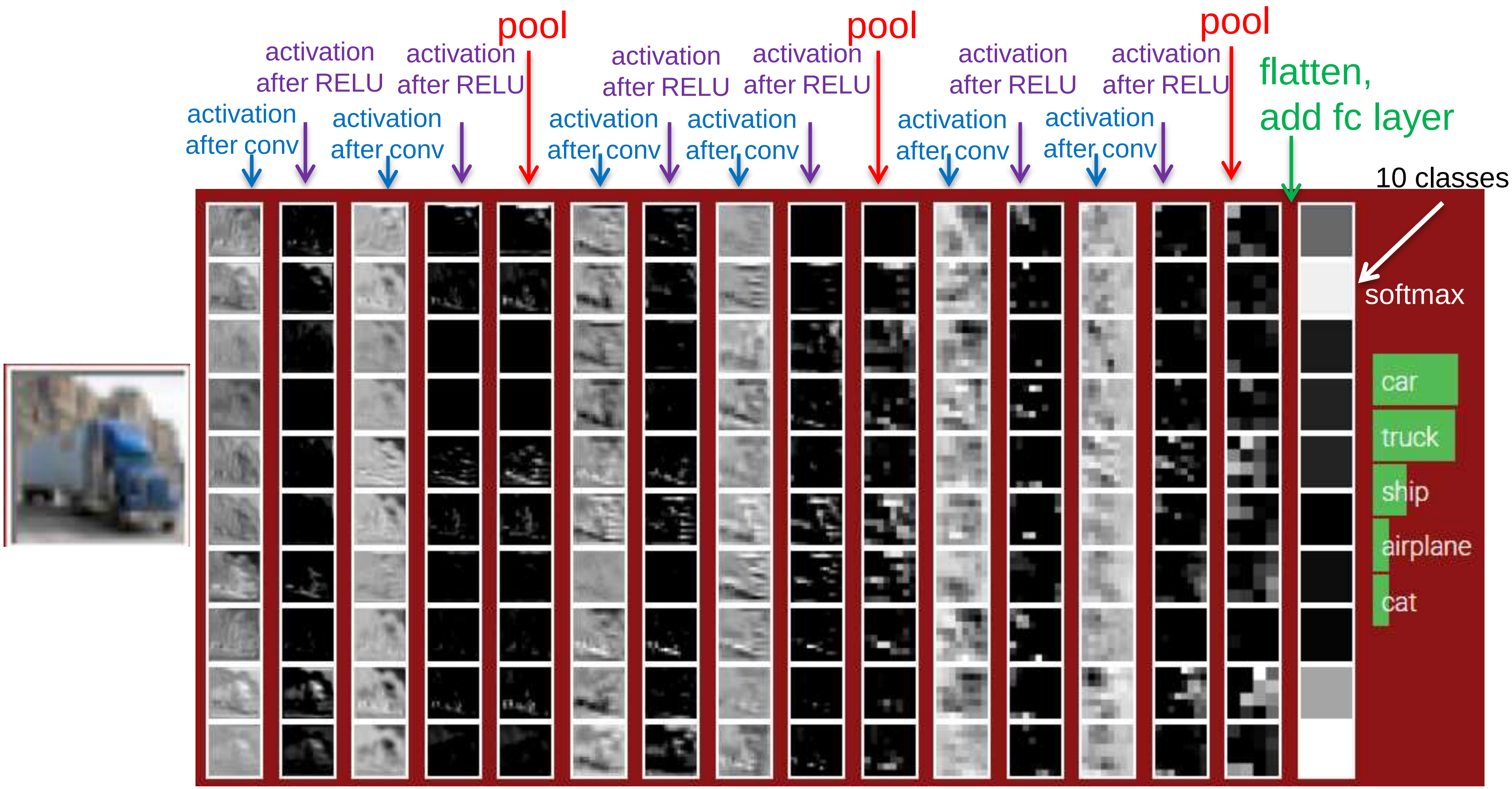
Check out the architecture of the CNN described in [“live cnn in browser”]()
And fill in the pieces to get a Keras code for a model with this architecture

Solution:

```
model = Sequential()
model.add(Conv2D(filters=32,
                 kernel_size=(3, 3),
                 input_shape=(28,28,1))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```


Coffee
Break

Appearance of activation/feature maps in different layers



Activation maps give insight on the spatial positions where the filter pattern was found in the input one layer below (in higher layers activation maps have no easy interpretation) -> only the activation maps in the first hidden layer correspond directly to features of the input image.

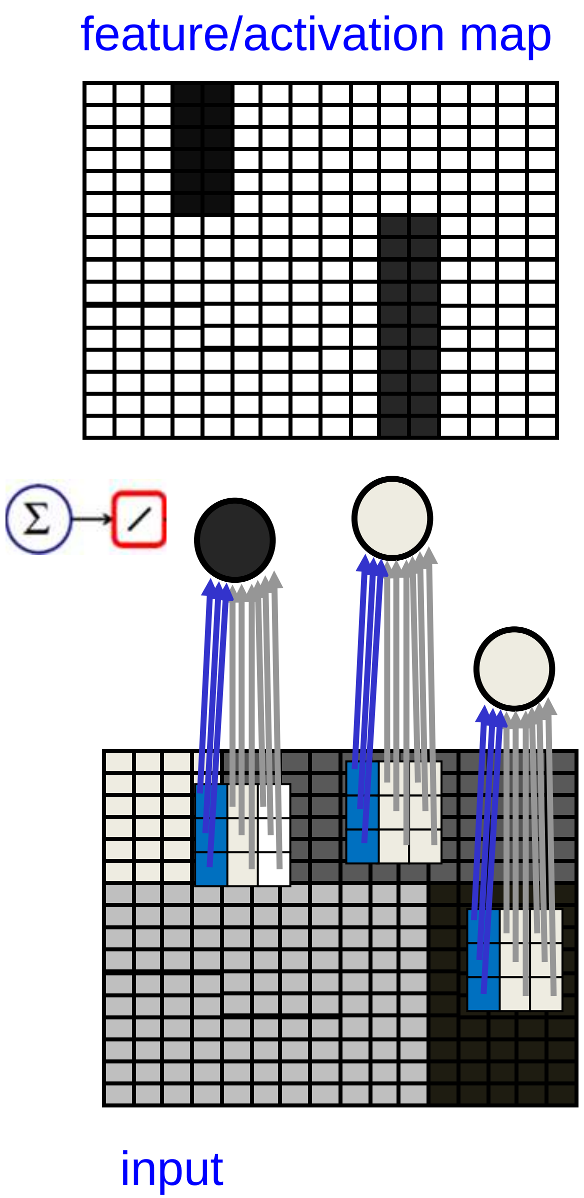
http://cs231n.stanford.edu/

See next lecture for understanding higher layers.

Exercise: use CNN for mnist classification

- Work through the instructions in 07 and 08 CNN Exercises **in day4** and use the ipython notebooks that are referred to.

Standardizing data is more important in CNNs than fcNN



Since we share weights in CNNs – only one filter is required per feature map - the weights in a filter should be appropriate for each patch of the input image, i.e. yielding inputs to the activation function that are party in their sweet spot.

To ensure that different image patches or even different pixels yield comparable ranges of values as input to the filter we need to standardize the input pixel-wise (we also restrict the variation to get small activations corresponding to uncertain classifications in the beginning of the training).

```
print(X_train.shape)
print(X_val.shape)
```

```
(4000, 28, 28, 1)
(1000, 28, 28, 1)
```

```
# here we center and standardize the data per pixel
# calculate mean, std over all training images at each pixel position
X_mean = np.mean( X_train, axis = 0)
X_std = np.std( X_train, axis = 0)

X_train = (X_train - X_mean ) / (X_std + 0.0001)
X_val = (X_val - X_mean ) / (X_std + 0.0001)
```

Wrapping up today's story
Why going from fully connected NN to convolutional NN?

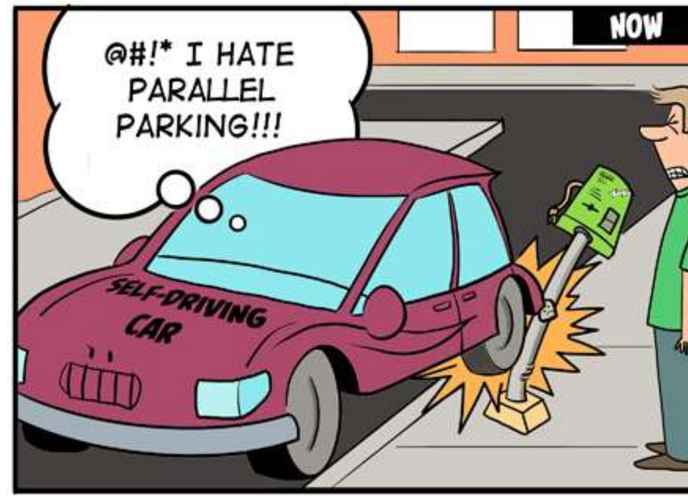
- Most images are much too complicated to be captured by simple template matches of whole shapes (1 layer fc NN).
- Many fully connected layer correspond to many rigid templates.
- We need to learn the features that a image is composed of.
- The classification should not depend on the location of the object in the image
- We want to exploit the information that is contained in the neighborhood structure of pixels in a image.
- Next week we will see how to get insight to the hierarchical features learned by the CNN.

What kind of tasks can be tackled by CNNs?

Convolutional Neural Nets are used for **detecting patterns** in images, videos, sounds and texts.

Where are CNNs used already?

- Recommendation at Spotify, Amazon …
  (http://benanne.github.io/2014/08/05/spotify-cnns.html)
- Google, Facebook for image interpretation
  e.g. PlaNet—Photo Geolocation
  (http://arxiv.org/abs/1602.05314)
- Who else is using CNNs?
  (https://www.quora.com/Apart-from-Google-Facebook-who-is-commercially-using-deep-recurrent-convolutional-neural-networks)



Spotify amazon Google facebook …

Your project to be presented in week 8 ?

Homework: Do some real stuff

Team-up for your first real DL project:

Develop a DL model to solve this task:

For a given image, decide which out of 8 celebrities is on the image.

Data:

For each of the 8 celebrities you get 250 images in the training data set, 50 images in the validation data set and 50 images in test data set.

Special challenge:

The images come from the OXFORD VGG Face dataset. The images were derived from the internet and automatically labeled. The data set contains also mislabeled images or ambiguous images.

Example images:

Label: Steve Jobs (entrepreneur)

Label: Emma Stone (actress)