

EE3-25: Deep Learning

Krystian Mikolajczyk & Carlo Ciliberto

Department of Electrical and Electronic Engineering
Imperial College London

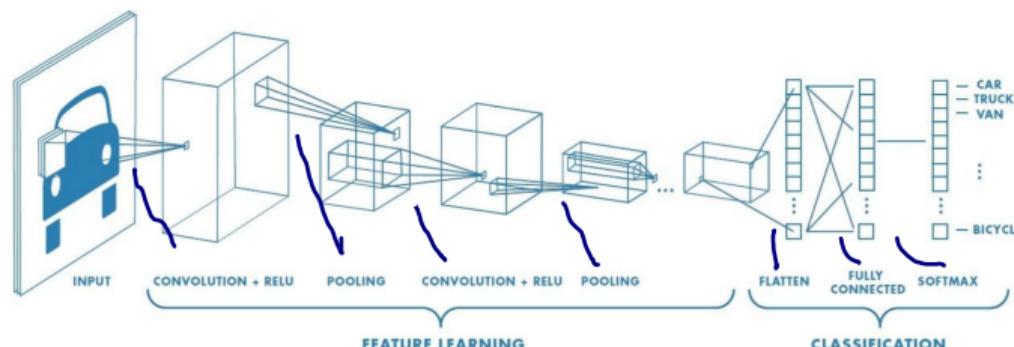


IN CS, IT CAN BE HARD TO EXPLAIN
THE DIFFERENCE BETWEEN THE EASY
AND THE VIRTUALLY IMPOSSIBLE.

Convolutional Neural Networks

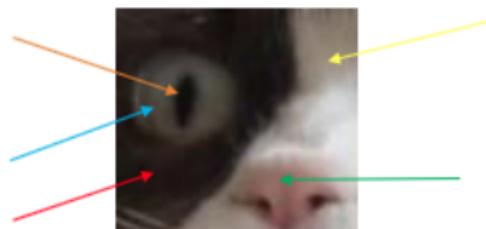
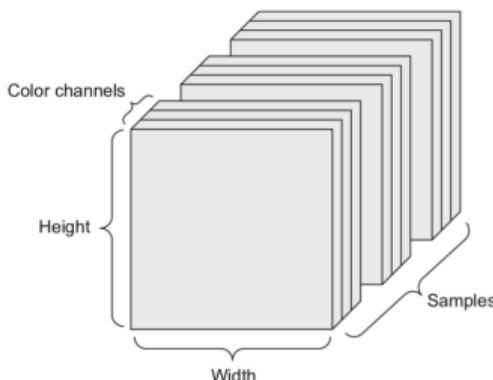
Type of neural network with a special architecture

- Convolution
- Filters, strides, padding
- Pooling
- FC layer
- Loss layer
- Practicals



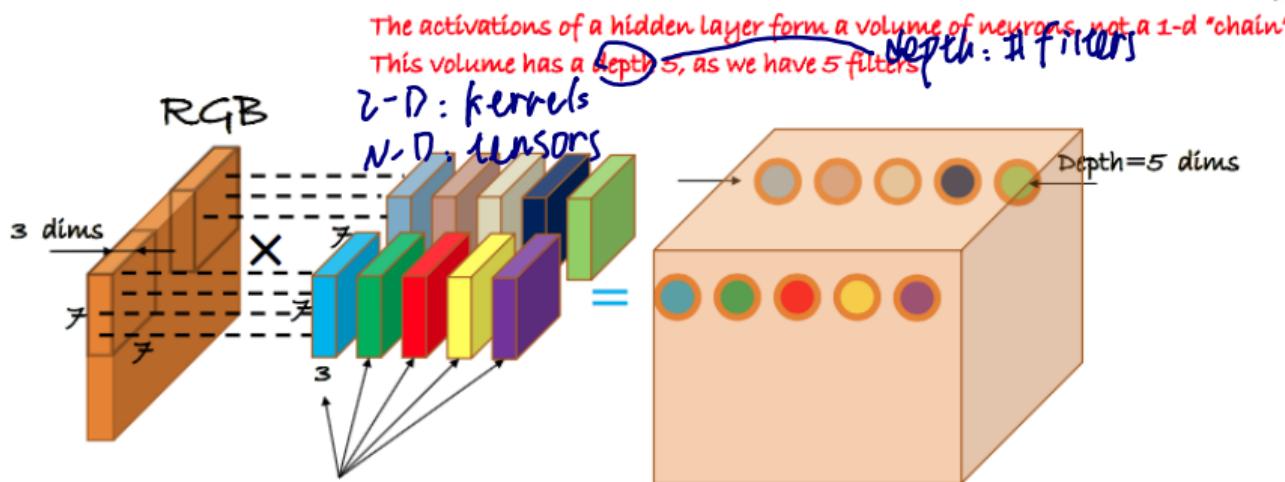
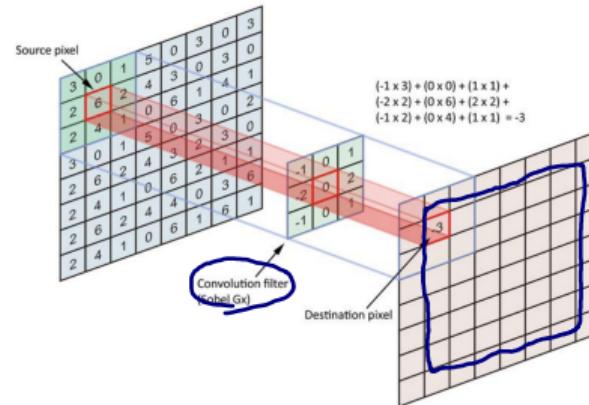
CNN Input Data

- Neural network architecture optimised for k-D data e.g. images
 - RGB, hyperspectral, etc.
- Images are represented in 4D tensors: (samples, height, width, channels)
- Neighbouring variables are locally correlated



CNN Neurons (filters, kernels, tensors)

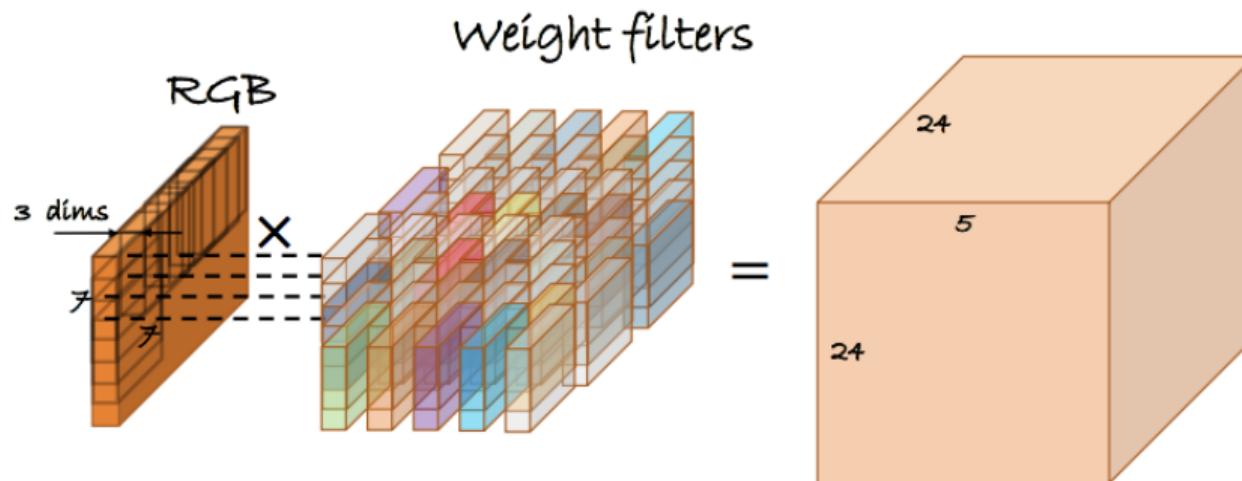
- Spatially large input data
 - Many neurons distributed spatially
 - Many weights per neurons



How many weights for these 5 neurons?

CNN Neurons

- Covering whole 2D input with different filters



Assume the image is $30 \times 30 \times 3$.

1 filter every pixel (stride = 1)

How many parameters in total?

24 filters along the x axis

24 filters along the y axis

Depth of 5

$7 \times 7 \times 3$ parameters per filter

$13 \times$ parameters in total

how far the filter moves from one position to another.

CNN Neurons

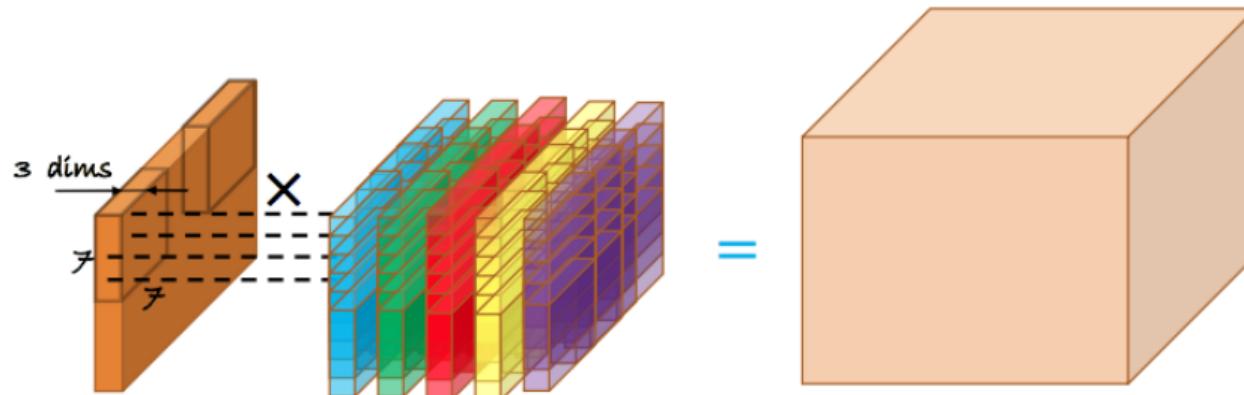
- Many parameters to train
 - ▶ With an image of only 30×30 pixels and a single hidden layer of depth 5 we need $85k$ parameters
 - ▶ With a 256×256 image we need 46×10^6 parameters
- Problems
 - ▶ How to fit a model with that many parameters ?
 - ▶ Where to find the data for training such a model ?

Are all these weights necessary?

CNN shared filters

- If the data repeats, why not repeat filters?

• Sliding window (filter)
reuse filters



Assume the image is $30 \times 30 \times 3$.

1 column of filters common across the image.

How many parameters in total?

$$\frac{\text{Depth of 5} \times 7 \times 7 \times 3 \text{ parameters per filter}}{735 \text{ parameters in total}}$$

Convolution

- Convolution operator in continuous space

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

- a weighted average of the input f according to the weighting (filter, kernel, tensor) w at each point in time t or space

- Discrete convolution

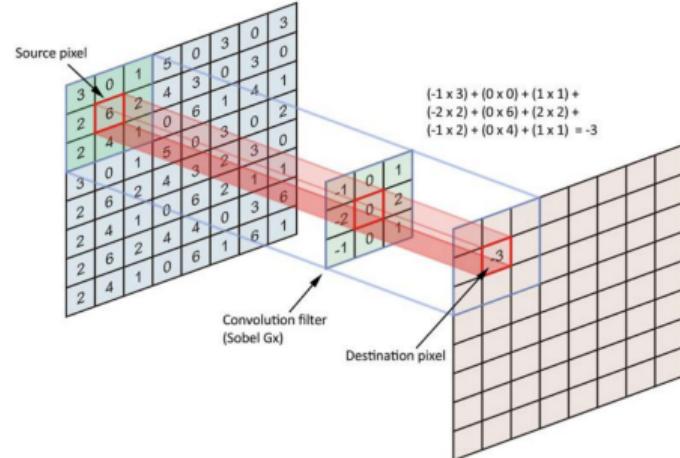
$$(f * w)(t) = \sum_{-\infty}^{+\infty} f(\tau)w(t - \tau)$$

- In practice both signal and kernel are finite

$$1D (\mathbf{x} * F)(i) = \sum_m \mathbf{x}(m)F(i - m)$$

$$2D (\mathbf{x} * F)(i, i) = \sum_m \sum_n \mathbf{x}(m, n)F(i - m, j - n)$$

$$\text{or } \sum_m \sum_n \mathbf{x}(i + m, j + n)F(m, n)$$



16 * 0	24 * -1	32	23	0	16	24 * 0	32 * -1	23	-14	
47 * 1	18 * 0	26	0	0	47	18 * 1	26 * 0	50	0	
68	12	9	68	12	9	68	12	9	50	0
16	24	32	23	0	16	24	32	23	-14	
47 * 0	18 * -1	26	50	0	47	18 * 0	26 * -1	50	-14	
68 * 1	12 * 0	9	68	12 * 1	9 * 0	68	12 * 1	9 * 0	50	-14

Convolutional Layer

Convolution

- The size of filter F is its receptive field *spatial extent of data*
- The result of a convolution is a feature map
- A layer can be formed from several channels (feature maps)
 - Typically, there are many filters and feature maps for the same input
 - RGB image has 3 channels
- Filters (kernels) are therefore multi dimensional tensors (3 for input image)
- The convolution is applied across all channels in the input layer
$$\sum_m \sum_n \sum_c x(i + m, j + n, c) F(m, n, c)$$
- The convolution is usually combined with a shared bias (one per channel) and input into an activation function

Strides

- Step of the shift when applying the filter (kernel, tensor)
- Stride 1x1 shifts by 1 pixel at a time
 - Most frequently used
 - Patches are heavily overlapping
- Stride 2x2 skips one patch horizontally and vertically
 - Convolution with downsampling

Filter F=2x2 with stride S=1x1

23	-12	16	90
12	32	12	45
-1	7	8	9
-2	-12	14	56

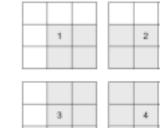
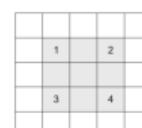
23	-12	16	90
12	32	12	45
-1	7	8	9
-2	-12	14	56

23	-12	16	90
12	32	12	45
-1	7	8	9
-2	-12	14	56

F=2x2 with S=2x2

23	-12	16	90
12	32	12	45
-1	7	8	9
-2	-12	14	56

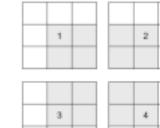
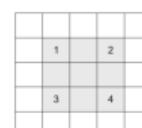
23	-12	16	90
12	32	12	45
-1	7	8	9
-2	-12	14	56



F=3x3 with S=2x2

23	-12	16	90
12	32	12	45
-1	7	8	9
-2	-12	14	56

23	-12	16	90
12	32	12	45
-1	7	8	9
-2	-12	14	56



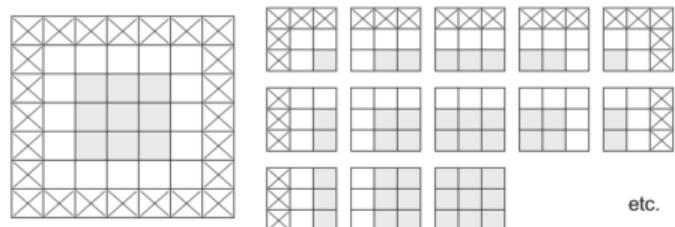
Padding

- VALID - no padding
size of output equals the size of input.
- SAME - adds $P = F - 1$ zeros to the input to constrain the output size to be the same for unit strides, $H_o = H_i$.
 - For odd-sized kernels, add $P = \lceil F/2 \rceil$ zeros on both sides of the input
- FULL - adds $P = 2(F - 1)$ zeros to the input ($F - 1$ zeros on both sides), resulting in an output height of $H_i + 2(F - 1)$

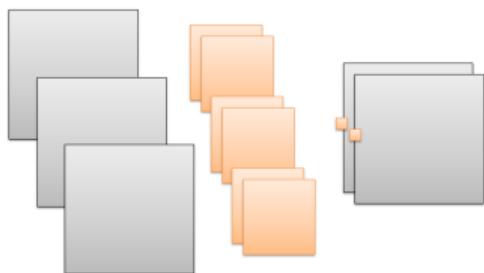
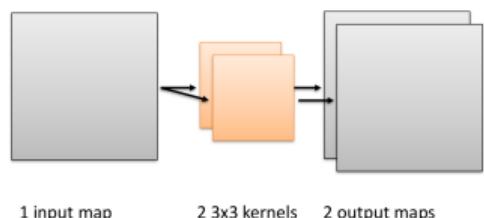
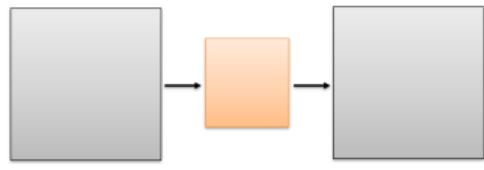
X							
-12	16	90	34				
32	12	45	-8				
7	8	9	12				
-12	14	56	18				

X with 1-padding of zeros							
0	0	0	0	0	0	0	0
0	-12	16	90	34	0		
0	32	12	45	-8	0		
0	7	8	9	12	0		
0	-12	14	56	18	0		
0	0	0	0	0	0		

X with 2-padding of zeros							
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	-12	16	90	34	0	0	0
0	32	12	45	-8	0	0	0
0	7	8	9	12	0	0	0
0	-12	14	56	18	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



Convolutional tensors and kernels (filters)



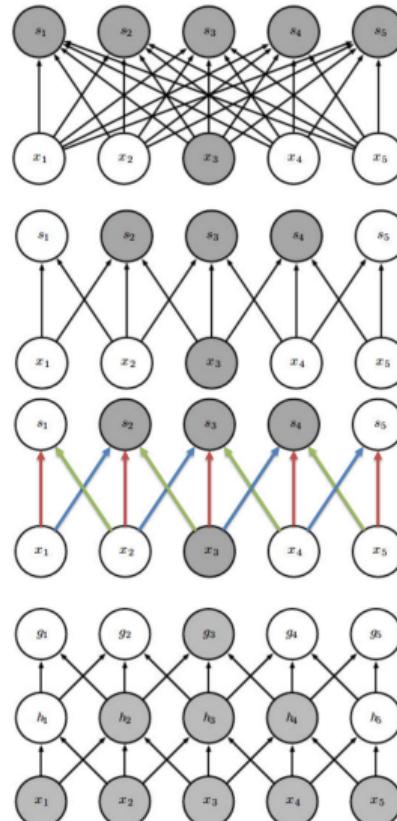
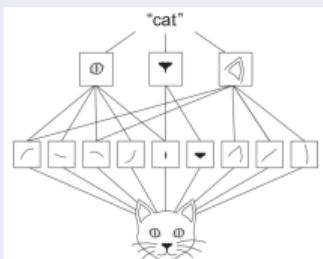
Input Volume (+pad 1) (7x7x3)	Filter W0 (3x3x3)	Filter W1 (3x3x3)	Output Volume (3x3x2)
$x[:, :, 0]$	$w0[:, :, 0]$	$w1[:, :, 0]$	$o[:, :, 0]$
$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 1 & 0 \\ 0 & 2 & 2 & 0 & 1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 & -1 \\ -1 & 0 & 0 \\ -1 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -6 & -7 & -5 \\ -9 & -6 & -9 \\ 3 & -5 & -8 \end{bmatrix}$
$\begin{bmatrix} 0 & 2 & 2 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 2 & 0 \\ 0 & 2 & 1 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 1 \\ -1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 2 & 3 & -2 \\ 7 & 4 & 1 \\ 5 & 5 & 7 \end{bmatrix}$
$x[:, :, 1]$	$w0[:, :, 1]$	$w1[:, :, 1]$	$b1[:, :, 0]$
$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 2 & 2 & 0 \\ 0 & 0 & 1 & 1 & 0 & 2 & 0 \\ 0 & 1 & 2 & 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & 1 \\ 0 & 0 & 0 \\ 1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{bmatrix}$	0
$x[:, :, 2]$	$w0[:, :, 2]$	$w1[:, :, 2]$	
$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 2 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 & 0 & 2 & 0 \\ 0 & 2 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 & 1 \\ 0 & 0 & 0 \\ 1 & -1 & -1 \end{bmatrix}$		
	$b0[:, :, 0]$		
	1		

Convolutional Neural Networks

CNN properties

relevant position?

- Sparse connectivity, neurons act locally
- Shared weights, reduced complexity
- Translational invariance (not rotation)
- Suitable for locally correlated neighbours (patterns)
- Enables some flexibility in input size
- Hierarchical structure of pattern detectors

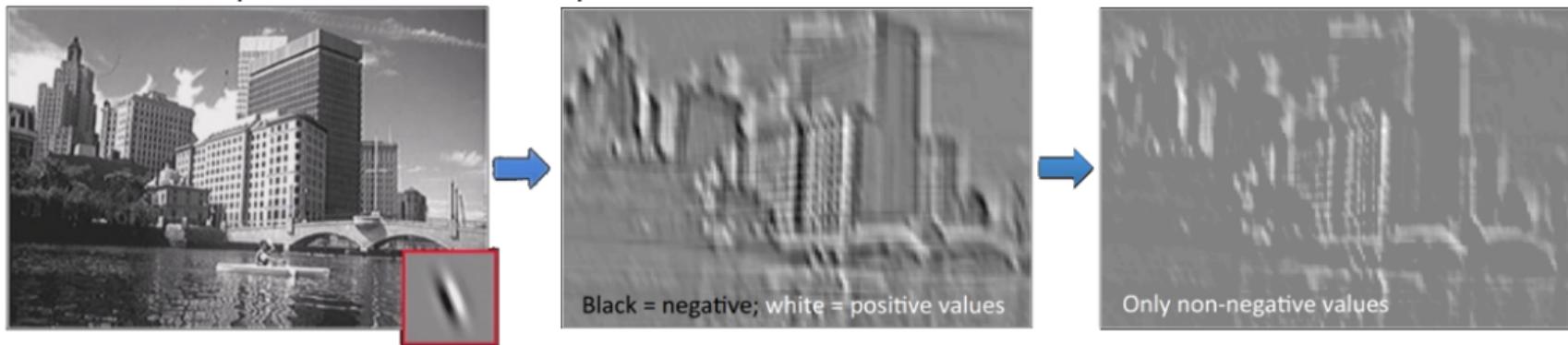


CNN layer parameters

- Input: a volume of size height \times width \times depth $H_i \times W_i \times D_i$
- Parameters:
 - Number of filters K ,
 - Their spatial extent F ,
 - Stride S ,
 - zero padding P
- Output: a volume of size height \times width \times depth $H_o \times W_o \times D_o$
 - $W_o = (W_i - F + 2P)/S + 1$
 - $H_o = (H_i - F + 2P)/S + 1$
 - $D_o = K$
- With parameter sharing, there are $F \cdot F \cdot D_i$ weights per filter, i.e. $F \cdot F \cdot D_i \cdot K$ weights and K biases
- Common settings $F \in \{2, \dots, 15\}$, $S = 2$, $P = 0$,

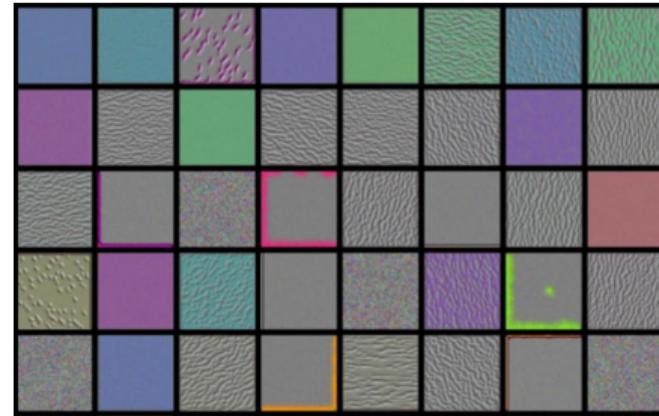
CNN layer example

- Filter output and ReLU example

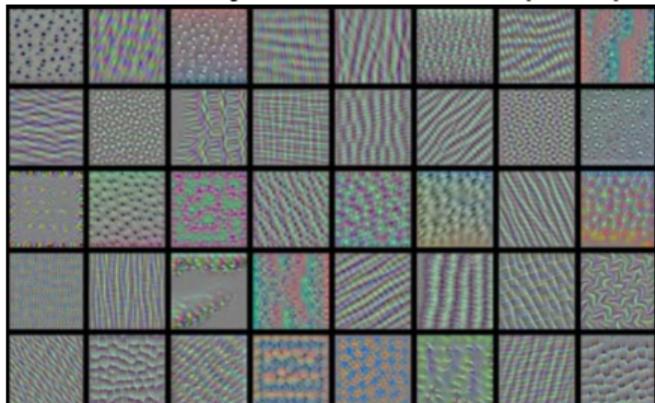


Convolutional Filters/kernels

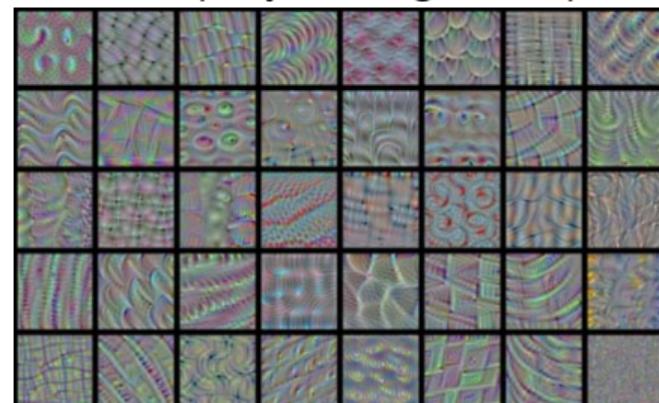
Shallow (initial) layers - low level patterns



Intermediate layers - more complex patterns



Deep layers - high level patterns



Pooling layer (aggregate/downsample)

- A function that aggregates multiple inputs into a single value
- A summary statistic over regions of the input space identified by a sliding window
 - Non-linear downsampling on activation maps
- Similar to the convolution but the linear transformation is replaced with a pooling operation
- Reduces the size of the layer output (dimensionality)
 - Keeps only the most important information for the next layer
 - Robust against small shifts
 - Faster computations
- Pooling discards information

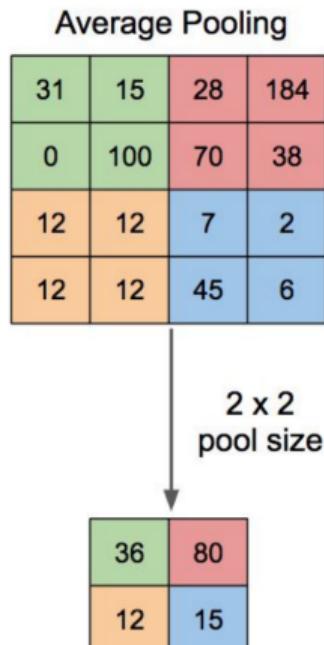
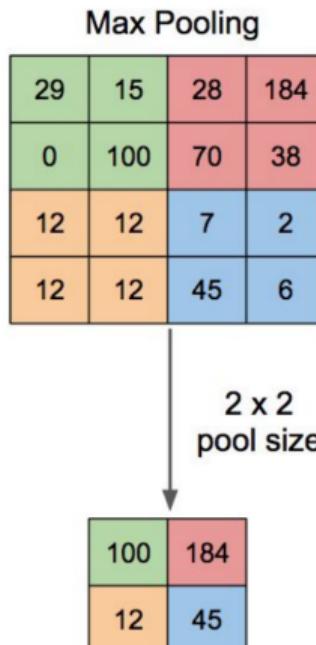


Pooling methods

- Max pooling

- Most common in 2D
- Filters of size 2×2 applied with a stride of 2, takes a max over 4 numbers, discards 75% of the activations.
- Downsamples at every depth slice (feature map) by 2 both, width & height
- The depth dimension remains unchanged
- Has been shown to work well in practice

- Average pooling and L2-norm pooling have also been used
- The gradient is only propagated through the input pixel that contributes to the output value



Pooling parameters

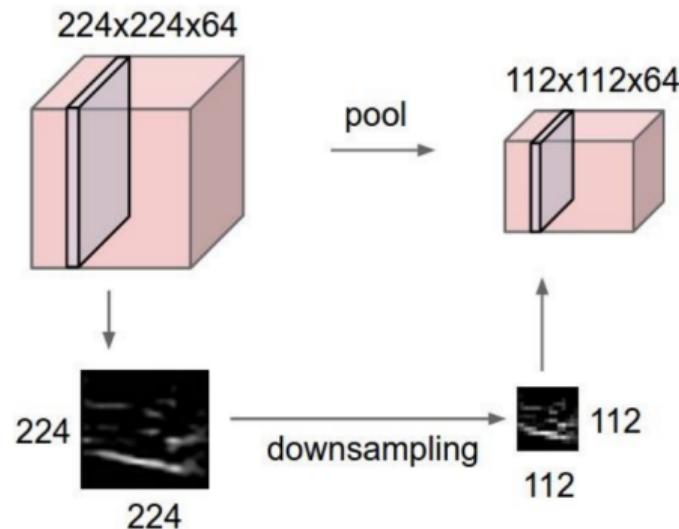
- Input: a volume of size height \times width \times depth $H_i \times W_i \times D_i$
- Parameters: Spatial extent of the filter F , stride S , input padding P
- Output: a volume of size height \times width \times depth $H_o \times W_o \times D_o$

$$\triangleright W_o = (W_i - F)/S + 1$$

$$\triangleright H_o = (H_i - F)/S + 1$$

$$\triangleright D_o = D_i$$

- Common settings $F \in \{2, \dots, 4\}$, $S = 2$, $P = 0$
- Typically no zero-padding in pooling layers
- Useful for handling inputs of varying size
- No new learnable parameters



Pooling examples

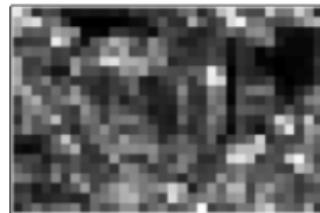
- Max Pooling
- Average Pooling



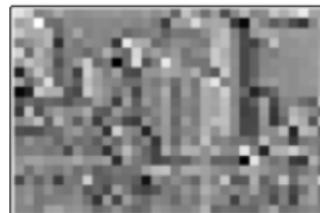
ReLU output



Max



Average



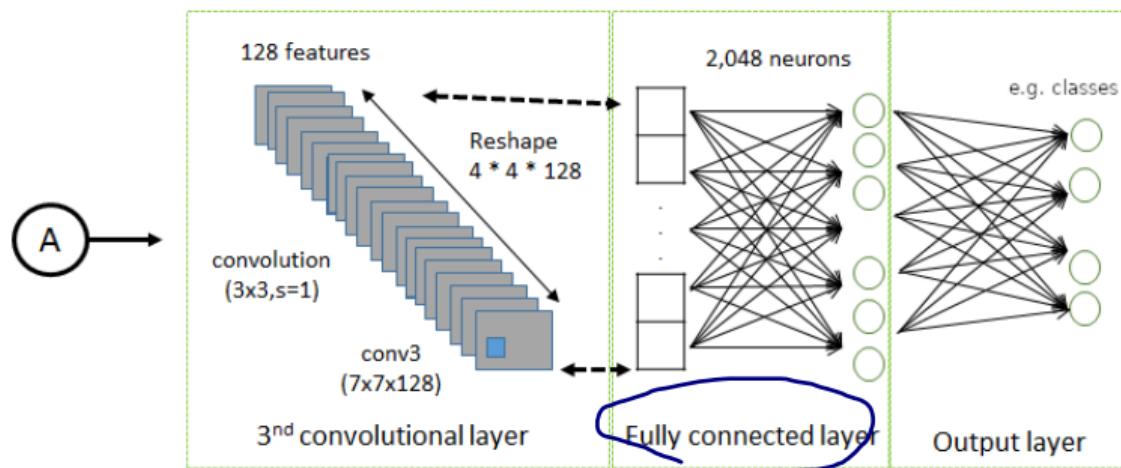
Fully Connected layer

- Basic neural network (MLP)
- Global view on all features
- Used as the final learning phase, which maps extracted features to desired outputs
- Usually adaptive to classification/encoding tasks, with typical sequence

Convolution → Pooling → Flattening → Full connection → Sigmoid/Softmax/Loss

- Flattening reshapes feature maps (2D arrays) from a conv. layer to form an input into an FC layer

- Common output is a vector, which is then passed through softmax to represent confidence of classification

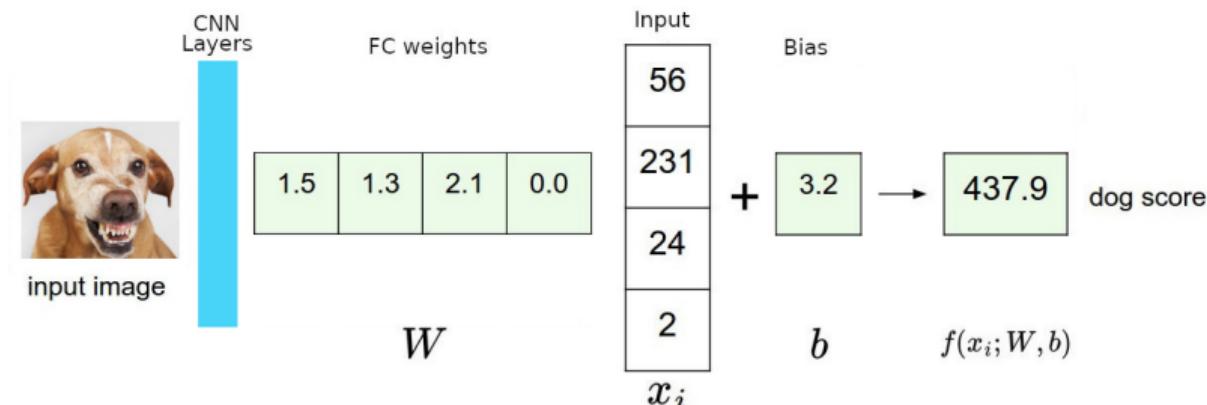
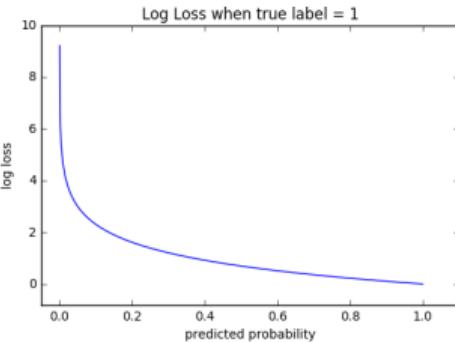


Loss layer

- Needed for training the network
- Inputs: Network predictions and ground truth (samples from target function)
- Output: Prediction error (scalar)
- Loss function: depends on the task and data
- Classification
 - Softmax (sigmoid) activation function followed by the cross-entropy loss
- Regression
 - L2, L1, Smooth L1

Loss layer - Binary Classification

- For ground truth label y and network prediction \hat{y}
 - with $y \in \{0, 1\}$ $\hat{y} = \frac{e^{W_k x_k + b}}{e^{W_k x_k + b} + 1}$ (sigmoid)
 - Can be interpreted as probability
- Cross-entropy loss function $\ell = -y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$



- Loss $\hat{y} = \frac{e^{437.9}}{e^{437.9} + 1} \approx 1$, $\ell = -1 \log(1) + (1 - 1) \log(1 - 1) = 0$
 - note that $\hat{y} \in (0, 1)$ is never 0 or 1 probability

Loss layer - Multiclass Classification

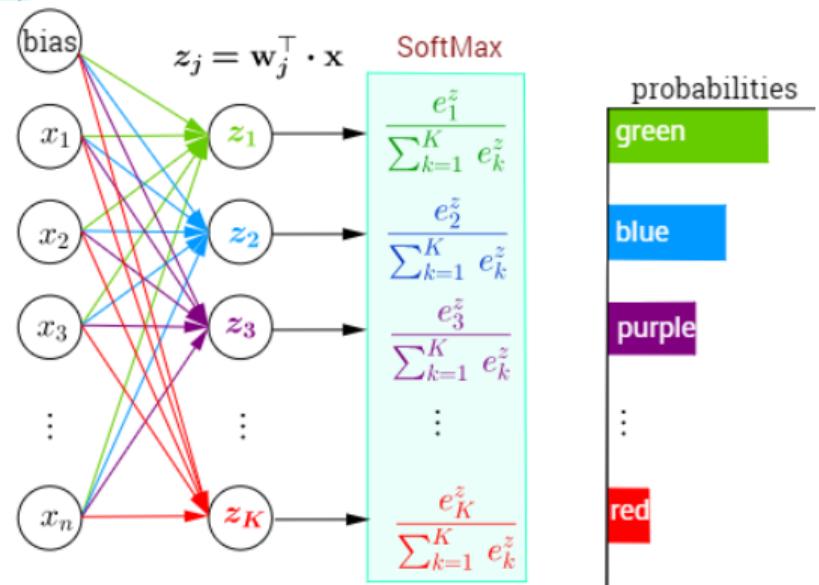
- For multi label ground truth y and prediction $\hat{y} \in \mathbb{R}$
- Softmax $\hat{y}_k = \frac{e^{W_k x_k + b}}{\sum_i e^{W_i x_i + b}}$ and $y_k \in \{0, 1\}$
- Categorical (multiclass) cross entropy error $\ell = -y_k \log(\hat{y}_k)$

- Better than classification error

or MSE

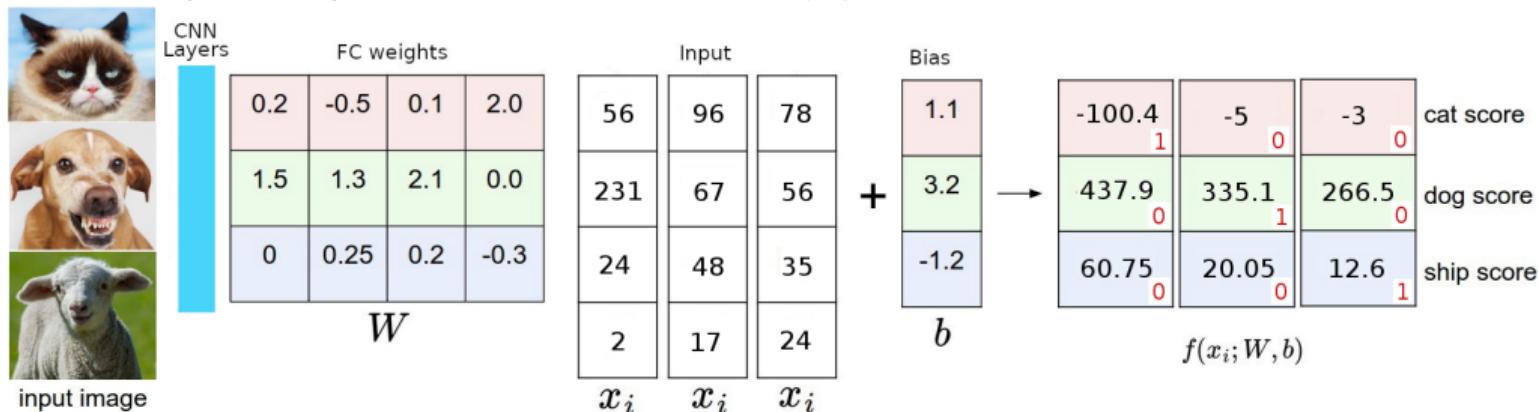
- Classification Error:
ignores confidence
- Mean Squared Error:
too much emphasis on
incorrect outputs

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$



Loss layer - Multiclass Classification

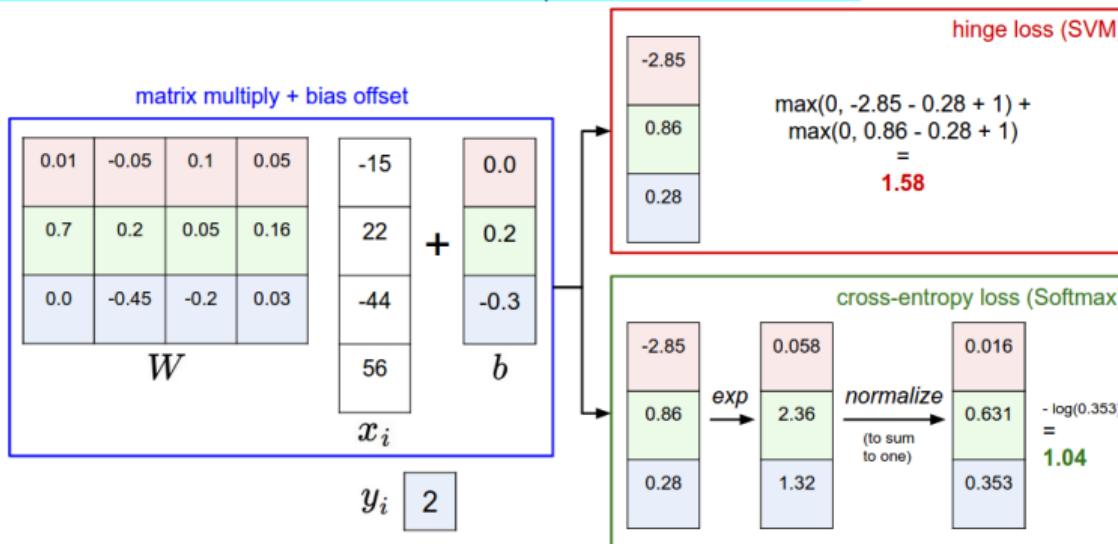
- For multi label ground truth y and prediction $\hat{y} \in \mathbb{R}$
- Softmax $\hat{y}_k = \frac{e^{W_k x_k + b}}{\sum_i e^{W_i x_i + b}}$
- Categorical (multiclass) cross entropy error $\ell = -y_k \log(\hat{y}_k)$



- cat score $\hat{y}_{1,1} = \frac{e^{-100.4}}{\sum_i e^{-100.4} + e^{437.9} + e^{60.75}}$
- Average Cross Entropy (ACE) loss $\mathcal{L} = \frac{1}{3}[-(1 \log(\hat{y}_{1,1}) + 1 \log(\hat{y}_{2,2}) + 1 \log(\hat{y}_{3,3}))]$

Loss layer - Multiclass Classification

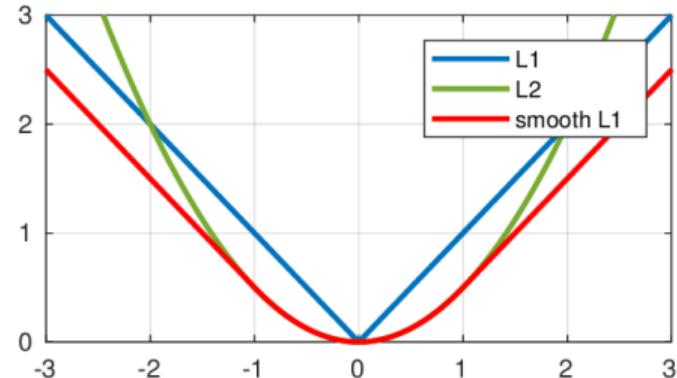
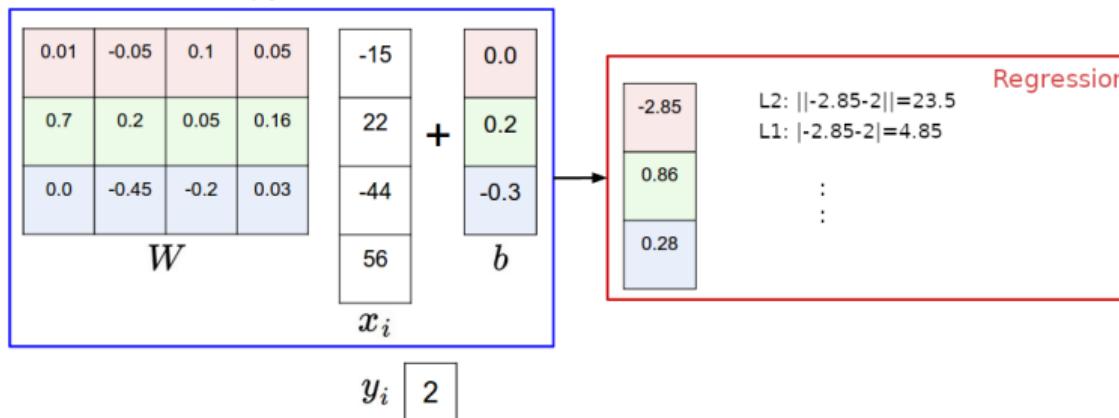
- For multi label ground truth $y \in \mathbb{N}$ and prediction $\hat{y} \in \mathbb{R}$
- Softmax $\hat{y}_k = \frac{e^{W_k x_k + b}}{\sum_i e^{W_i x_i + b}}$, with cross entropy error $\ell = -y_k \log(\hat{y}_k)$
- Hinge loss for one data sample from class k , $\ell = \sum_{i \neq k} \max(0, \hat{y}_i - \hat{y}_k + 1)$



Loss layer - Regression

- For ground truth value $y \in \mathbb{R}$ and prediction $\hat{y} \in \mathbb{R}$
- L2 Loss: $\ell_{L2} = \|\mathbf{w}\mathbf{x} + b - y\| = (\mathbf{w}\mathbf{x} + b - y)^2$
- Mean Squared Error: $MSE = \frac{1}{N} \ell_{L2}$
- L1 Loss: $\ell_{L1} = |\mathbf{w}\mathbf{x} + b - y|$
- Mean Absolute Error: $MAE = \frac{1}{N} \ell_{L1}$
- Smooth L1 Loss (Huber): $\ell_{L1} = 0.5, \forall \ell_{L1} \geq 0.5 \text{ otherwise } 0.5\ell_{L2}$
 - less sensitive to outliers than $L2$

matrix multiply + bias offset



Practical hints

- Number of filters
 - The number of feature maps directly controls network capacity
 - Depends on the number of available examples and the complexity of the task
 - Computing activations is more expensive than for MLP (convolution operation) even though the neuron is much smaller than in MLP
 - Initial layers (large inputs) will have less filters than later layers (small but more feature maps),
 - The number of filters can be set to equalise the number of computations and to preserve information at each layer e.g. number of activations.
- Filter shape (size)
 - Depends on the type of data
 - Set to create abstractions at the proper scale
 - Can be larger in first layers eg. 11x11, smaller later 5x5.

Practical hints

- Pooling

- Typical 2x2 or no max-pooling
- The trend is to use smaller filters, deeper architectures and abandon pooling
- Large input data may allow 4x4 but this may be discarding too much information.

- Batch normalisation of each feature map

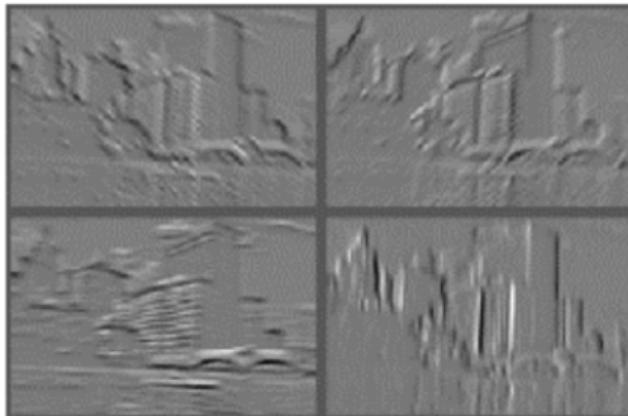
- e.g. input data in a 2D ConvNet has shape $[N, H, W, C]$, with
 - ★ N is the number of examples in the minibatch
 - ★ $H \times W$ are image height and width
 - ★ C is the number of channels
- Standard BN would compute $H \times W \times C$ means and std devs to normalise each feature separately at each spatial location
- BN in ConvNets instead computes C means and std devs and normalises jointly for all locations (feature map)
 - ★ to respect the structure (spatial patterns)

Practical hints

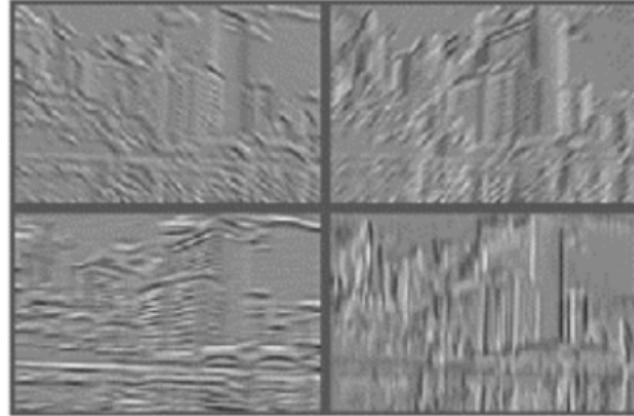
- Batch normalisation example

- Per feature map (output of one filter)
- Zero-mean, one standard deviation
- Enhanced details with BN

Without Batch Normalisation



With Batch Normalisation



- Loss

- Classification - Softmax
- Regression - L2, Smooth L1

CNN architecture

- Typical sequence of layers in shallow networks

- $INPUT \rightarrow [CONV \rightarrow Norm \rightarrow RELU \rightarrow POOL]^{(\times 2)} \rightarrow FC \rightarrow RELU \rightarrow FC \rightarrow OUTPUT$
- $INPUT \rightarrow [CONV \rightarrow RELU \rightarrow CONV \rightarrow RELU \rightarrow POOL]^{(\times 3)} \rightarrow [FC \rightarrow RELU]^{(\times 2)} \rightarrow FC \rightarrow OUTPUT$

- Code example

```
def makeModel(nb_filters):  
    model = Sequential()  
    model.add(Conv2D(nb_filters, kernel_size, input_shape=(patchsize,patchsize,3), padding = "same"))  
    model.add(Activation('relu'))  
    model.add(MaxPooling2D(pool_size = pool_size))  
    model.add(Conv2D(nb_filters*2, kernel_size, padding = "same"))  
    model.add(Activation('relu'))  
    model.add(MaxPooling2D(pool_size = pool_size))  
    model.add(Conv2D(nb_filters*4, kernel_size, padding = "same"))  
    model.add(Activation('relu'))  
    model.add(MaxPooling2D(pool_size = pool_size))  
    model.add(AveragePooling2D(pool_size = pool_size))  
    model.add(Flatten())  
    model.add(Dense(128)) # generate a fully connected layer with 128 outputs (arbitrary value)  
    model.add(Activation('relu'))  
    model.add(Dropout(0.5))  
    model.add(Dense(3)) # output layer  
    model.add(Activation('softmax'))
```

CNN Summary

- Convolution
- Filters, strides, padding
- Pooling
- FC layer
- Loss layer
- Practicals