Applied Machine Learning

Convolutional Neural Networks

Siamak Ravanbakhsh

COMP 551 (winter 2020)

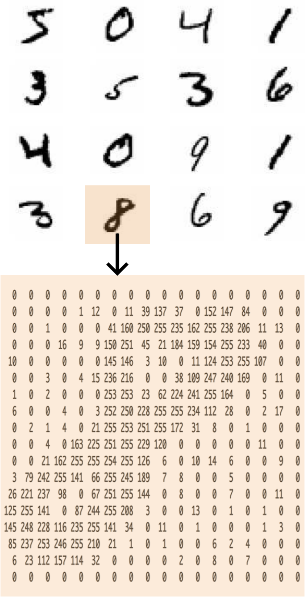
Learning objectives

understand the convolution layer and the architecture of conv-net

- its inductive bias
- its derivation from fully connected layer
- different types of convolution

MLP and image data

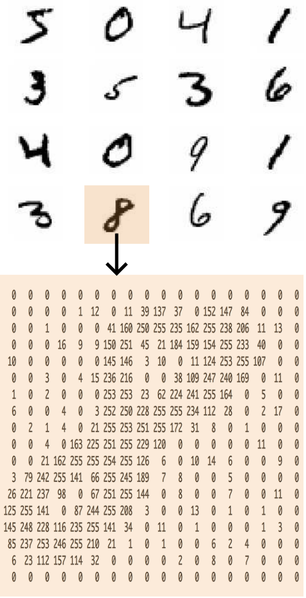
we can apply an MLP to image data



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

image:https://medium.com/@rajatjain0807/machine-learning-6ecde3bfd2f4

MLP and image data



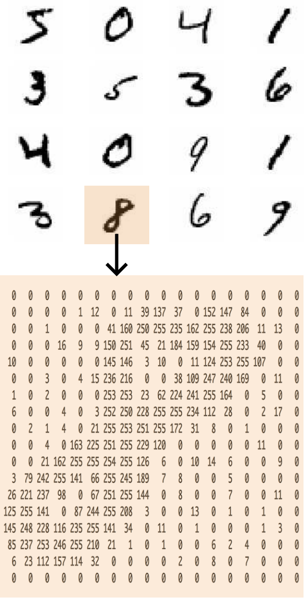
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

we can apply an MLP to image data

> first vectorize the input $x \rightarrow \mathrm{vec}(x) \in \mathbb{R}^{784}$
>
> feed it to the MLP (with L layers) and predict the labels
>
> $$\mathrm{softmax} \circ W^{\{L\}} \circ \ldots \circ \mathrm{ReLU} \circ W^{\{1\}} \mathrm{vect}(x)$$

image:https://medium.com/@rajatjain0807/machine-learning-6ecde3bfd2f4

MLP and image data



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

we can apply an MLP to image data

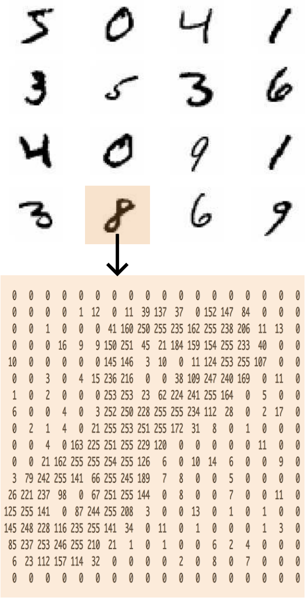
> first vectorize the input $x \to \mathrm{vec}(x) \in \mathbb{R}^{784}$
>
> feed it to the MLP (with L layers) and predict the labels
>
> $$\mathrm{softmax} \circ W^{\{L\}} \circ \ldots \circ \mathrm{ReLU} \circ W^{\{1\}} \mathrm{vect}(x)$$

the model knows nothing about the image structure

we could shuffle all pixels and learn an MLP with similar performance

image:https://medium.com/@rajatjain0807/machine-learning-6ecde3bfd2f4

MLP and image data



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

we can apply an MLP to image data

> first vectorize the input $x \rightarrow \mathrm{vec}(x) \in \mathbb{R}^{784}$
>
> feed it to the MLP (with L layers) and predict the labels
>
> $$\mathrm{softmax} \circ W^{\{L\}} \circ \ldots \circ \mathrm{ReLU} \circ W^{\{1\}} \mathrm{vect}(x)$$

the model knows nothing about the image structure

we could shuffle all pixels and learn an MLP with similar performance

how to bias the model, so that it "knows" its input is image?

image is like 2D version of sequence data

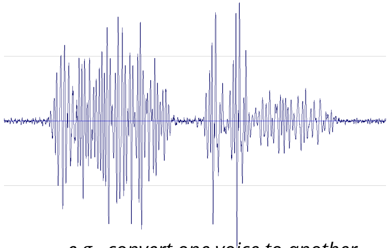
lets find the right model for sequence first...

image:https://medium.com/@rajatjain0807/machine-learning-6ecde3bfd2f4

Parameter-sharing

suppose we want to convert one sequence to another $\mathbb{R}^D \rightarrow \mathbb{R}^D$

suppose we have a dataset of input-output pairs $\{(x^{(n)}, y^{(n)})\}_n$

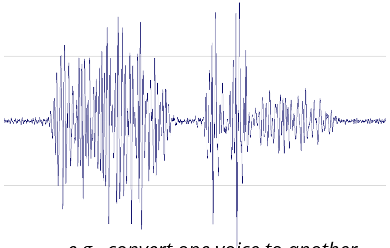


e.g., convert one voice to another

Parameter-sharing

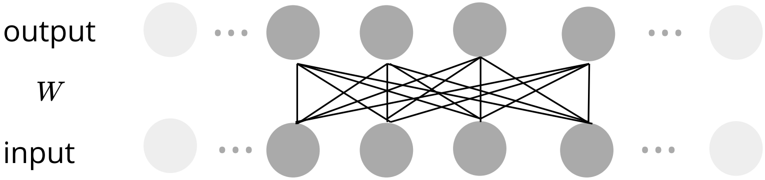
suppose we want to convert one sequence to another $\mathbb{R}^D \to \mathbb{R}^D$

suppose we have a dataset of input-output pairs $\{(x^{(n)}, y^{(n)})\}_n$



e.g., convert one voice to another

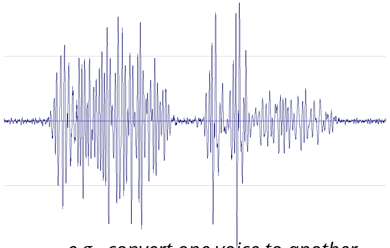
consider only a single layer $y = g(Wx)$



Parameter-sharing

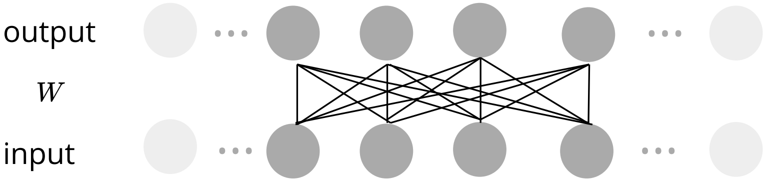
suppose we want to convert one sequence to another $\mathbb{R}^D \to \mathbb{R}^D$

suppose we have a dataset of input-output pairs $\{(x^{(n)}, y^{(n)})\}_n$



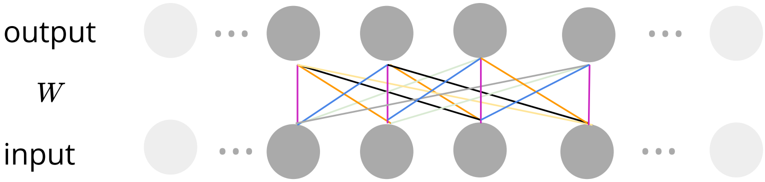
e.g., convert one voice to another

consider only a single layer $y = g(Wx)$



we may assume, each output unit is the same function shifted along the sequence

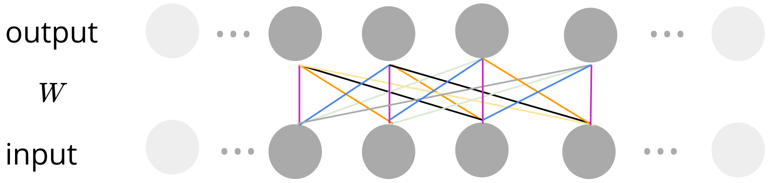
when is this a good assumption?



elements of w of the same color are tied together (parameter-sharing)

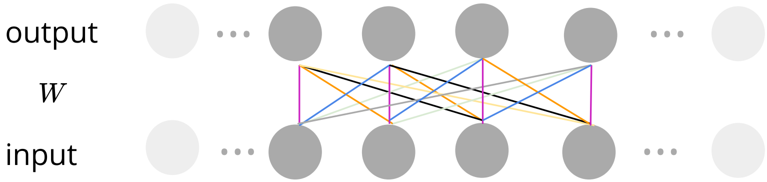
Locality & sparse weight

we may assume, each output unit is the same function shifted along the sequence

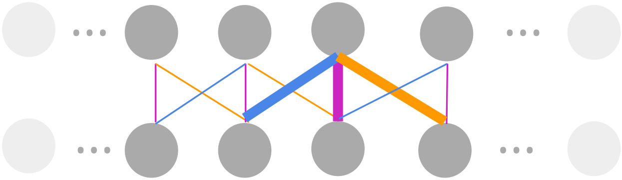


Locality & sparse weight

we may assume, each output unit is the same function shifted along the sequence

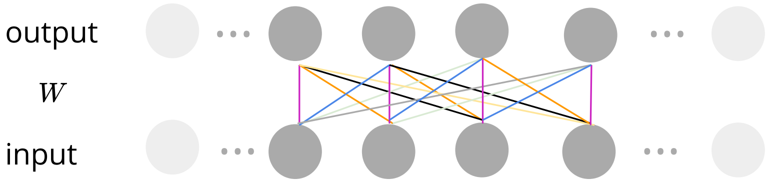


we may further assume each output is a **local** function of input

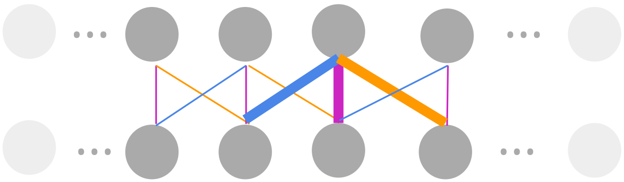


Locality & sparse weight

we may assume, each output unit is the same function shifted along the sequence



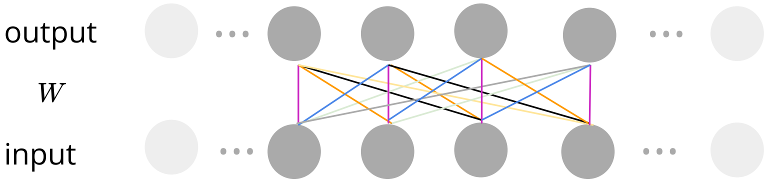
we may further assume each output is a **local** function of input



size of the receptive field is 3

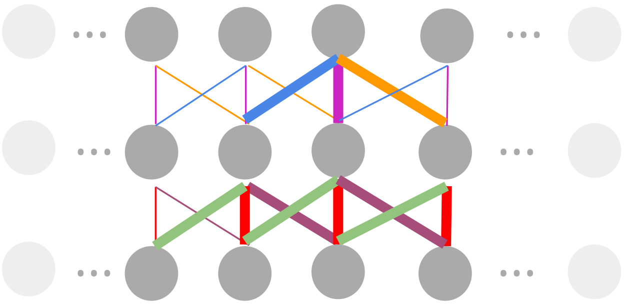
Locality & sparse weight

we may assume, each output unit is the same function shifted along the sequence



we may further assume each output is a **local** function of input

larger **receptive field** with multiple layers

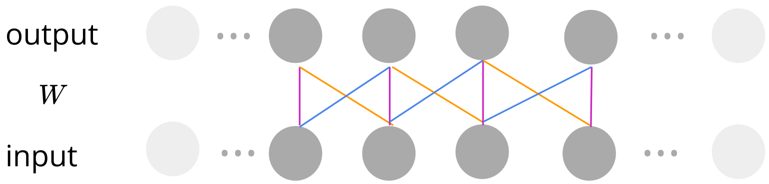


size of the receptive field is 3

size of the receptive field is 5

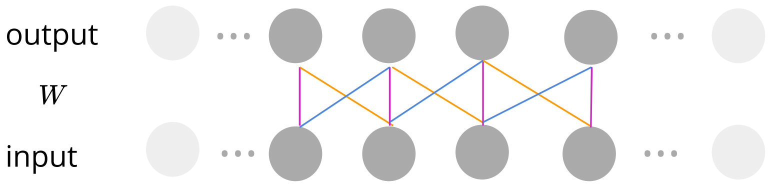
Cross-correlation (1D)

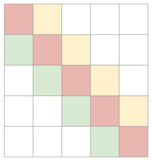
we may further assume each output is a local function of input



Cross-correlation (1D)

we may further assume each output is a local function of input

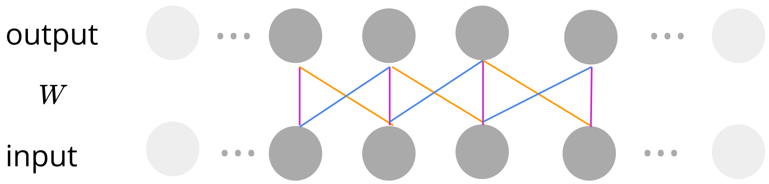


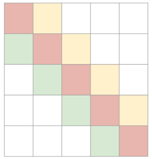


parameter-sharing in W
W is very sparse

Cross-correlation (1D)

we may further assume each output is a local function of input



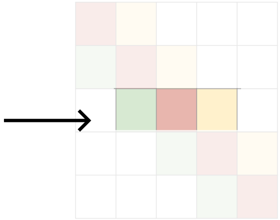


parameter-sharing in W

W is very sparse

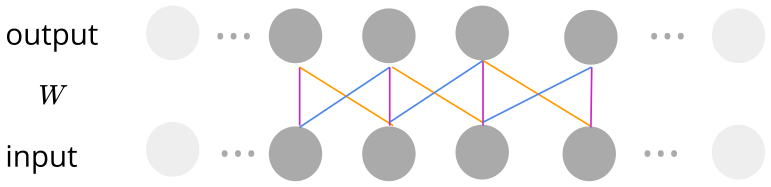
instead of the whole matrix we can keep the one set of nonzero values

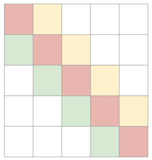
$$w = [w_1, \ldots, w_K] = [W_{c,c-\lfloor \frac{K}{2} \rfloor}, \ldots, W_{c,c+\lfloor \frac{K}{2} \rfloor}]$$



Cross-correlation (1D)

we may further assume each output is a local function of input

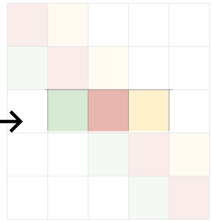




parameter-sharing in W
W is very sparse

instead of the whole matrix we can keep the one set of nonzero values

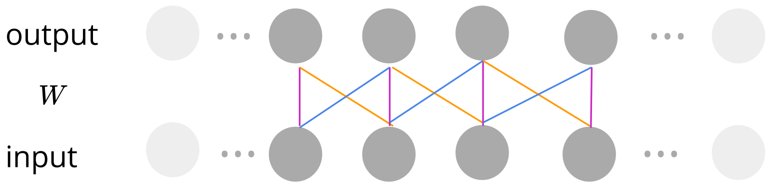
$$w = [w_1, \dots, w_K] = [W_{c,c-\lfloor \frac{K}{2} \rfloor}, \dots, W_{c,c+\lfloor \frac{K}{2} \rfloor}] \longrightarrow$$

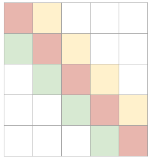


we can write matrix multiplication as **cross-correlation** of w and x

Cross-correlation (1D)

we may further assume each output is a local function of input

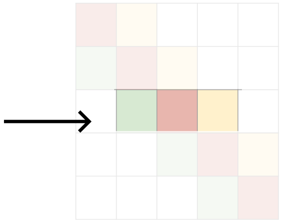




parameter-sharing in W
W is very sparse

instead of the whole matrix we can keep the one set of nonzero values

$$w = [w_1, \dots, w_K] = [W_{c,c-\lfloor \frac{K}{2} \rfloor}, \dots, W_{c,c+\lfloor \frac{K}{2} \rfloor}]$$

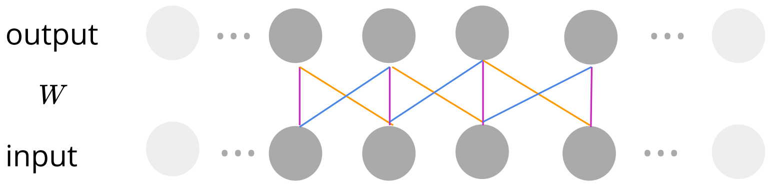


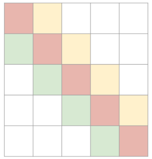
we can write matrix multiplication as **cross-correlation** of w and x

$$y_c = g\left(\sum_{d=1}^{D} W_{c,d} x_d \right)$$

Cross-correlation (1D)

we may further assume each output is a local function of input

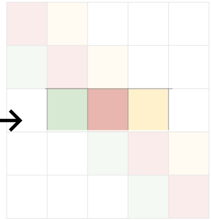




parameter-sharing in W
W is very sparse

instead of the whole matrix we can keep the one set of nonzero values

$$w = [w_1, \dots, w_K] = [W_{c,c-\lfloor \frac{K}{2} \rfloor}, \dots, W_{c,c+\lfloor \frac{K}{2} \rfloor}] \longrightarrow$$

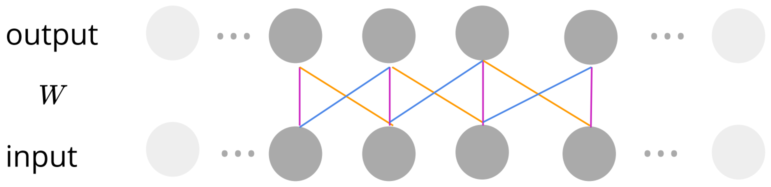


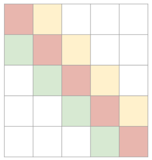
we can write matrix multiplication as **cross-correlation** of w and x

$$y_c = g\left(\sum_{d=1}^{D} W_{c,d} x_d \right) = g\left(\sum_{k=1}^{K} w_k x_{c-\lfloor \frac{K}{2} \rfloor + k} \right)$$

Cross-correlation (1D)

we may further assume each output is a local function of input

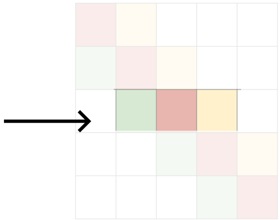




parameter-sharing in W
W is very sparse

instead of the whole matrix we can keep the one set of nonzero values

$$w = [w_1, \ldots, w_K] = [W_{c,c-\lfloor \frac{K}{2} \rfloor}, \ldots, W_{c,c+\lfloor \frac{K}{2} \rfloor}]$$



we can write matrix multiplication as **cross-correlation** of w and x

$$y_c = g\left(\sum_{d=1}^{D} W_{c,d} x_d\right) = g\left(\sum_{k=1}^{K} w_k x_{c-\lfloor \frac{K}{2} \rfloor + k}\right)$$

slide 🟩🟥🟨 on the input, calculate inner product and apply the nonlinearity

Convolution (1D)

Cross-correlation is similar to convolution

Convolution (1D)

Cross-correlation is similar to convolution

Cross-correlation $y(c) = \sum_{k=-\infty}^{\infty} w(k) x(c+k)$

w is called the filter or kernel

ignoring the activation (for simpler notation)
assuming w and x are zero for any index outside the input and filter bound

Convolution (1D)

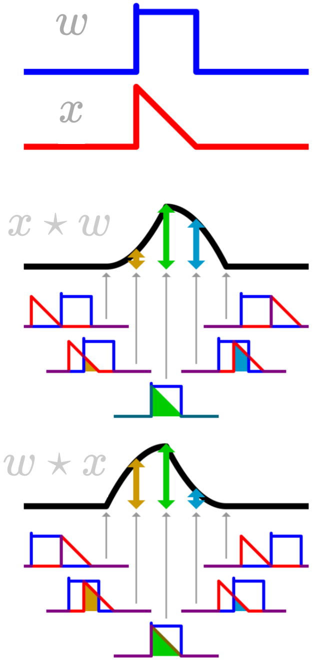
Cross-correlation is similar to convolution

Cross-correlation $y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c+k)$ $w \star x$

w is called the filter or kernel

ignoring the activation (for simpler notation)
assuming w and x are zero for any index outside the input and filter bound

Cross-correlation



Convolution (1D)

Cross-correlation is similar to convolution

Cross-correlation $y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c+k)$ $w \star x$

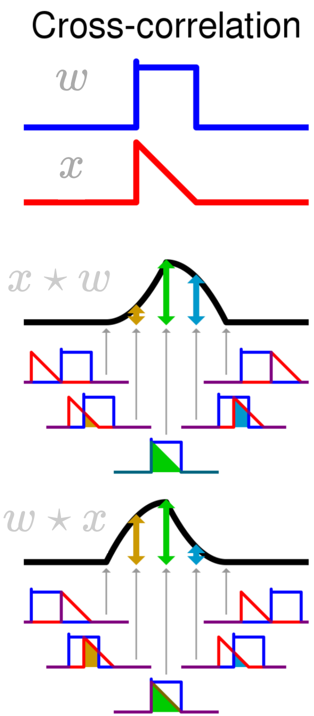
w is called the filter or kernel

ignoring the activation (for simpler notation)
assuming w and x are zero for any index outside the input and filter bound

Convolution flips w or x (to be commutative)

$$y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c-k)$$

$w * x$



Convolution (1D)

Cross-correlation is similar to convolution

Cross-correlation $y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c+k)$ $w \star x$

w is called the filter or kernel

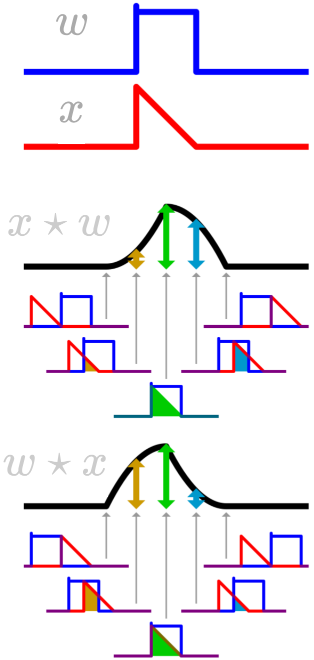
ignoring the activation (for simpler notation)
assuming w and x are zero for any index outside the input and filter bound

Convolution flips w or x (to be commutative)

$$y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c-k) = \sum_{d=-\infty}^{\infty} w(c-d)x(d)$$

$w * x$ *change of variable* $x * w$

Cross-correlation



Convolution (1D)

Cross-correlation is similar to convolution

Cross-correlation $y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c+k)$ $w \star x$

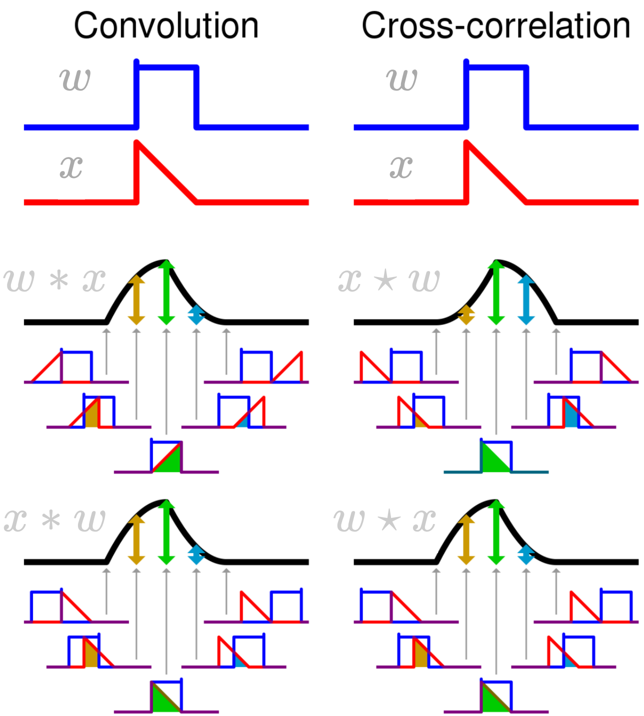
w is called the filter or kernel

ignoring the activation (for simpler notation)
assuming w and x are zero for any index outside the input and filter bound

Convolution flips w or x (to be commutative)

$$y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c-k) = \sum_{d=-\infty}^{\infty} w(c-d)x(d)$$

$w * x$ *change of variable* $x * w$



Convolution (1D)

Cross-correlation is similar to convolution

Cross-correlation $y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c+k)$ $w \star x$

w is called the filter or kernel

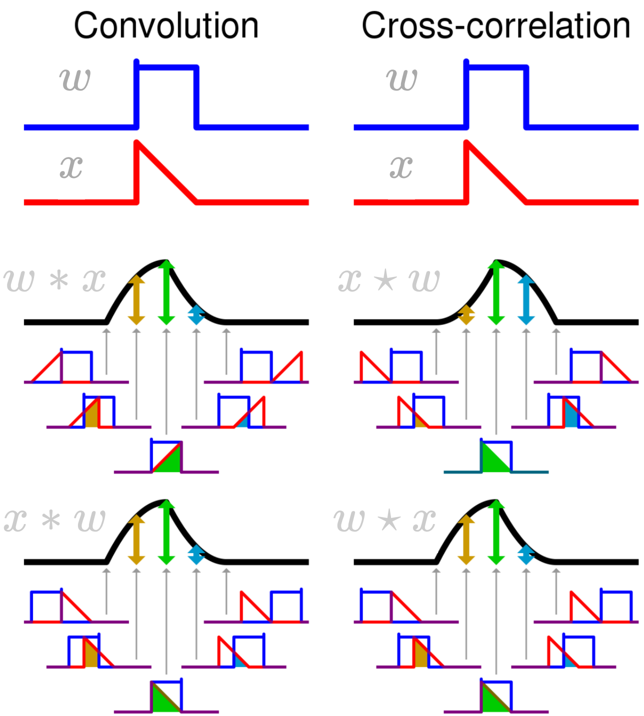
ignoring the activation (for simpler notation)
assuming w and x are zero for any index outside the input and filter bound

Convolution flips w or x (to be commutative)

$$y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c-k) = \sum_{d=-\infty}^{\infty} w(c-d)x(d)$$

$w * x$ *change of variable* $x * w$

since we learn w, flipping it makes no difference



Convolution (1D)

Cross-correlation is similar to convolution

Cross-correlation $y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c+k)$ $w \star x$

w is called the filter or kernel

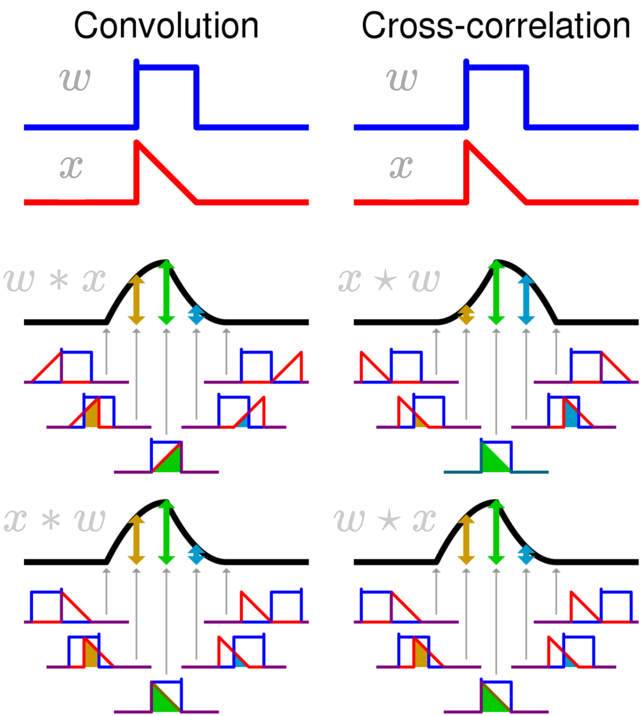
ignoring the activation (for simpler notation)
assuming w and x are zero for any index outside the input and filter bound

Convolution flips w or x (to be commutative)

$$y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c-k) = \sum_{d=-\infty}^{\infty} w(c-d)x(d)$$

$w * x$ *change of variable* $x * w$

since we learn w, flipping it makes no difference
in practice, we use cross correlation rather than convolution



Convolution (1D)

Cross-correlation is similar to convolution

Cross-correlation $y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c+k)$ $w \star x$

w is called the filter or kernel

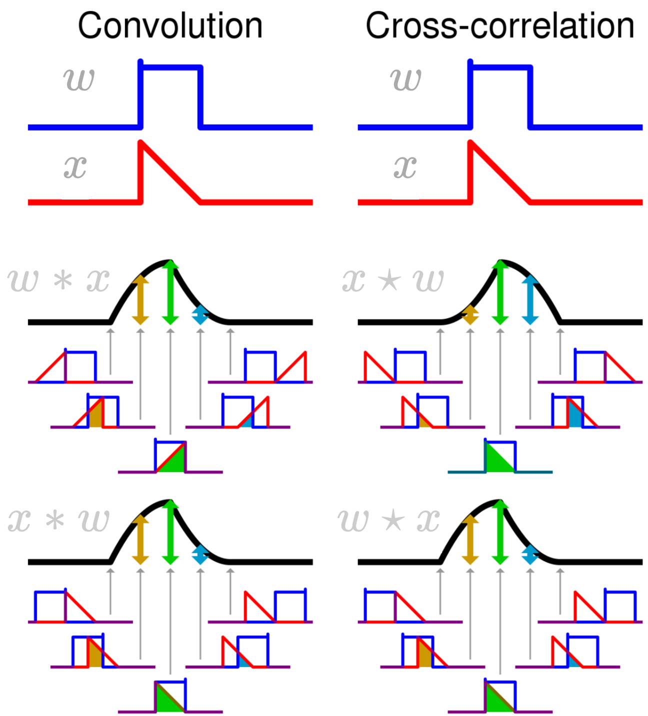
ignoring the activation (for simpler notation)
assuming w and x are zero for any index outside the input and filter bound

Convolution flips w or x (to be commutative)

$$y(c) = \sum_{k=-\infty}^{\infty} w(k)x(c-k) = \sum_{d=-\infty}^{\infty} w(c-d)x(d)$$

$w * x$ *change of variable* $x * w$

since we learn w, flipping it makes no difference
in practice, we use cross correlation rather than convolution
convolution is equivariant wrt translation
-- *i.e.,* shifting **x**, shifts **w*x**



Convolution (2D)

similar idea of parameter-sharing and locality extends to 2 dimension *(i.e. image data)*

$$y_{d_1,d_2} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} x_{d_1+k_1-1,d_2+k_2-1} w_{k_1,k_2}$$

Convolution **(2D)**

similar idea of parameter-sharing and locality extends to 2 dimension *(i.e. image data)*

$$y_{d_1,d_2} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} x_{d_1+k_1-1,d_2+k_2-1} w_{k_1,k_2}$$

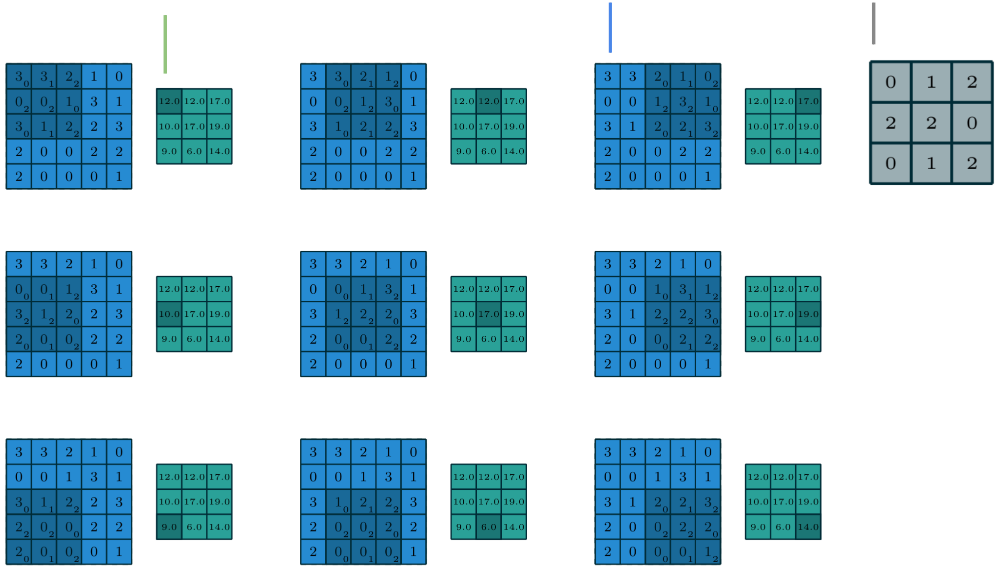
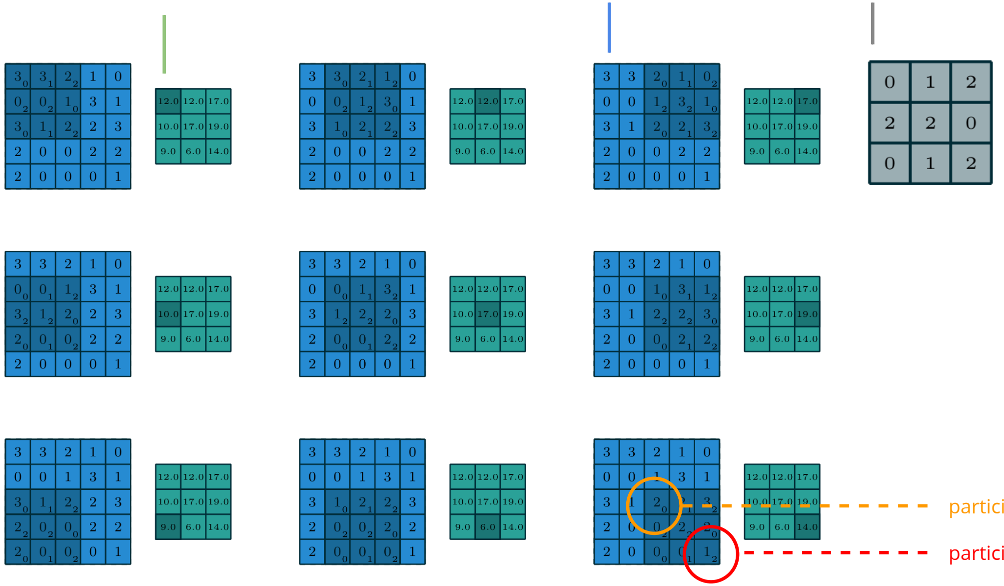


image credit: Vincent Dumoulin, Francesco Visin

Convolution (2D)

similar idea of parameter-sharing and locality extends to 2 dimension *(i.e. image data)*

$$y_{d_1,d_2} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} x_{d_1+k_1-1,d_2+k_2-1} w_{k_1,k_2}$$



3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

0	1	2
2	2	0
0	1	2

participates in all outputs

participates in a single output

this is related to the borders

image credit: Vincent Dumoulin, Francesco Visin

Convolution (2D)

similar idea of parameter-sharing and locality extends to 2 dimension *(i.e. image data)*

$$y_{d_1,d_2} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} x_{d_1+k_1-1,d_2+k_2-1} w_{k_1,k_2}$$

there are different ways of handling the borders

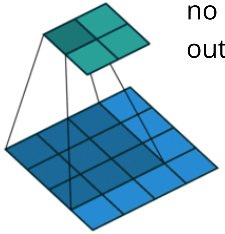
image credit: Vincent Dumoulin, Francesco Visin

Convolution (2D)

similar idea of parameter-sharing and locality extends to 2 dimension *(i.e. image data)*

$$y_{d_1,d_2} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} x_{d_1+k_1-1,d_2+k_2-1} w_{k_1,k_2}$$

there are different ways of handling the borders



no padding at all (valid)
output is small than input (how much?)

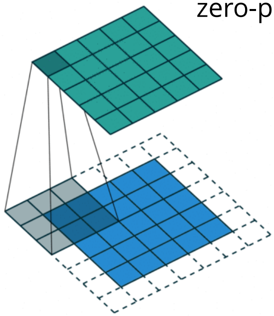
image credit: Vincent Dumoulin, Francesco Visin

Convolution (2D)

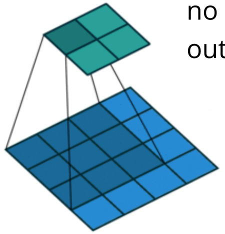
similar idea of parameter-sharing and locality extends to 2 dimension *(i.e. image data)*

$$y_{d_1,d_2} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} x_{d_1+k_1-1,d_2+k_2-1} w_{k_1,k_2}$$

there are different ways of handling the borders



zero-pad the input, to keep the output dims similar to input (same)



no padding at all (valid)
output is small than input (how much?)

image credit: Vincent Dumoulin, Francesco Visin

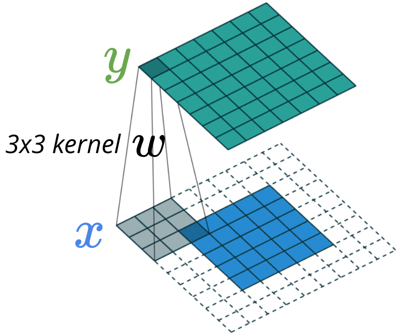
Convolution (2D)

similar idea of parameter-sharing and locality extends to 2 dimension *(i.e. image data)*

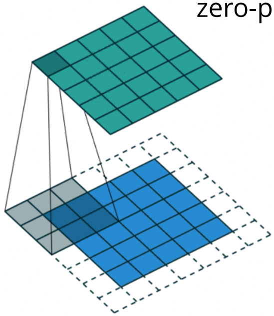
$$y_{d_1,d_2} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} x_{d_1+k_1-1,d_2+k_2-1} w_{k_1,k_2}$$

there are different ways of handling the borders

zero-pad the input, and produce all non-zero outputs (full)
output is larger than input (by how much?)
each input participates in the same number of output elements



zero-pad the input, to keep the output dims similar to input (same)



no padding at all (valid)
output is small than input (how much?)

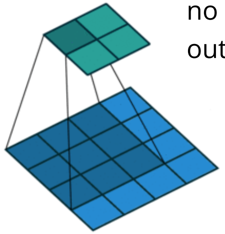


image credit: Vincent Dumoulin, Francesco Visin

Convolution (2D)

similar idea of parameter-sharing and locality extends to 2 dimension *(i.e. image data)*

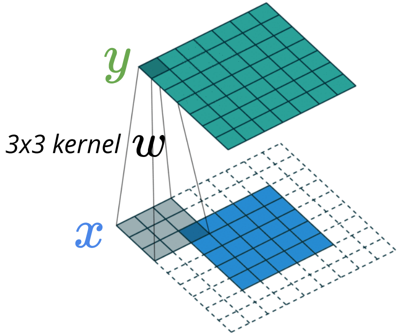
$$y_{d_1,d_2} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} x_{d_1+k_1-1,d_2+k_2-1} w_{k_1,k_2}$$

there are different ways of handling the borders

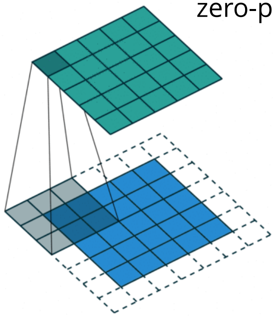
zero-pad the input, and produce all non-zero outputs (full)
output is larger than input (by how much?)
each input participates in the same number of output elements

output length (for one dimension)

$$\lfloor D + \text{padding} - K + 1 \rfloor$$



zero-pad the input, to keep the output dims similar to input (same)



no padding at all (valid)
output is small than input (how much?)

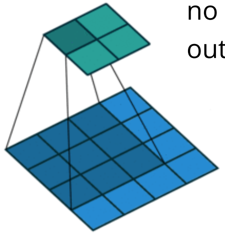


image credit: Vincent Dumoulin, Francesco Visin

Pooling

sometimes we would like to reduce the size of output *e.g., from D x D to D/2 x D/2*

Pooling

sometimes we would like to reduce the size of output *e.g., from D x D to D/2 x D/2*

a combination of pooling and downsampling is used

Pooling

sometimes we would like to reduce the size of output *e.g., from D x D to D/2 x D/2*

a combination of pooling and downsampling is used

1. calculate the output $\tilde{y}_d = g\left(\sum_{k=1}^{K} x_{d+k-1} w_k\right)$

Pooling

sometimes we would like to reduce the size of output *e.g., from D x D to D/2 x D/2*

a combination of pooling and downsampling is used

1. calculate the output $\tilde{y}_d = g\left(\sum_{k=1}^{K} x_{d+k-1} w_k\right)$
2. aggregate the output over different regions

$$y_d = \text{pool}\{\tilde{y}_d, \ldots, \tilde{y}_{d+p}\}$$

two common aggregation functions are ***max*** *and* ***mean***

Pooling

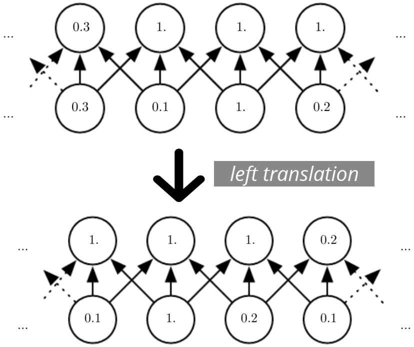
sometimes we would like to reduce the size of output *e.g., from D x D to D/2 x D/2*

a combination of pooling and downsampling is used

1. calculate the output $\tilde{y}_d = g\left(\sum_{k=1}^{K} x_{d+k-1} w_k\right)$
2. aggregate the output over different regions

$$y_d = \text{pool}\{\tilde{y}_d, \ldots, \tilde{y}_{d+p}\}$$

two common aggregation functions are ***max*** *and* ***mean***
pooling results in some degree of invariance to translation



left translation

Pooling

sometimes we would like to reduce the size of output *e.g., from D x D to D/2 x D/2*

a combination of pooling and downsampling is used

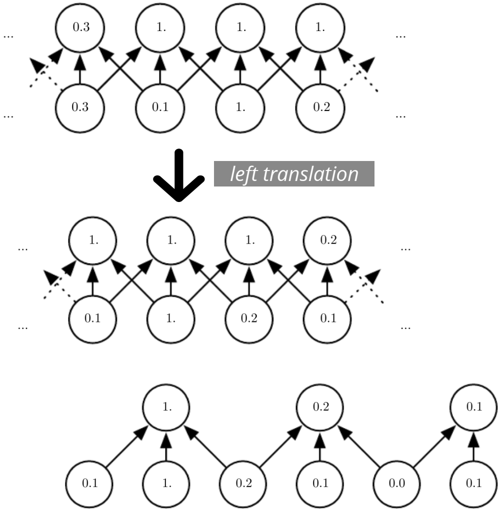
1. calculate the output $\tilde{y}_d = g\left(\sum_{k=1}^{K} x_{d+k-1} w_k\right)$

2. aggregate the output over different regions

$$y_d = \text{pool}\{\tilde{y}_d, \ldots, \tilde{y}_{d+p}\}$$

*two common aggregation functions are **max** and **mean***
pooling results in some degree of invariance to translation

3. often this is followed by subsampling using the same step size



Pooling

sometimes we would like to reduce the size of output *e.g., from D x D to D/2 x D/2*

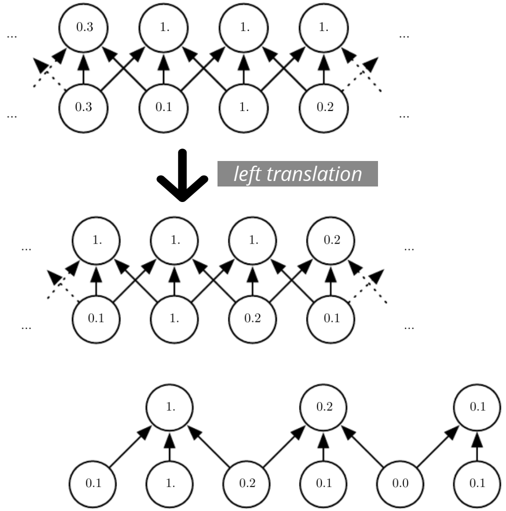
a combination of pooling and downsampling is used

1. calculate the output $\tilde{y}_d = g\left(\sum_{k=1}^{K} x_{d+k-1} w_k\right)$
2. aggregate the output over different regions

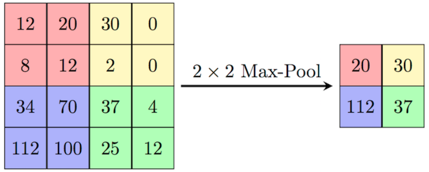
$$y_d = \text{pool}\{\tilde{y}_d, \ldots, \tilde{y}_{d+p}\}$$

*two common aggregation functions are **max** and **mean***
pooling results in some degree of invariance to translation

3. often this is followed by subsampling using the same step size



the same idea extends to higher dimensions



12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

$\xrightarrow{2 \times 2 \text{ Max-Pool}}$

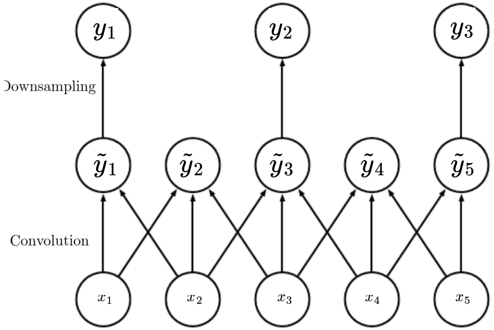
20	30
112	37

Strided convolution

alternatively we can directly subsample the output

$\tilde{y}_d = g\big(\sum_{k=1}^{K} x_{(d-1)+k} w_k\big)$

$y_d = \tilde{y}_{dp}$

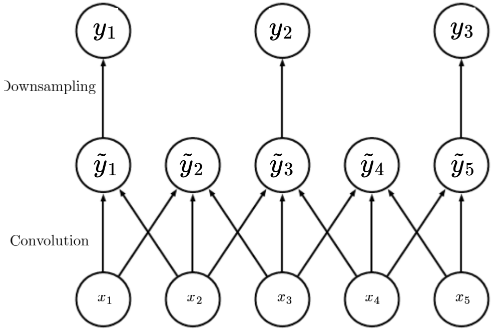


Strided convolution

alternatively we can directly subsample the output

$\tilde{y}_d = g\left(\sum_{k=1}^{K} x_{(d-1)+k} w_k\right)$

$y_d = \tilde{y}_{dp}$



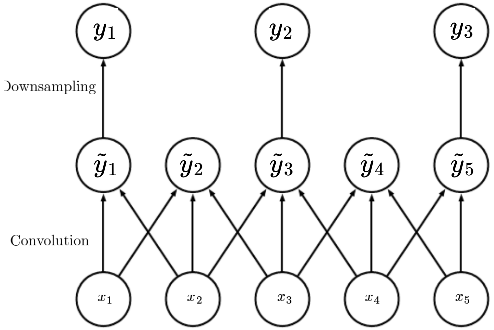
equivalent to →

Strided convolution

alternatively we can directly subsample the output

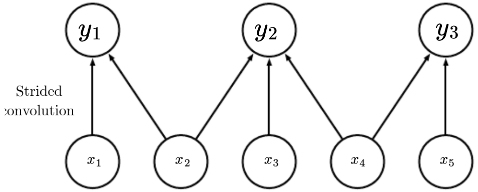
$$\tilde{y}_d = g\left(\sum_{k=1}^{K} x_{(d-1)+k} w_k \right)$$

$$y_d = \tilde{y}_{dp}$$



equivalent to →

$$\tilde{y}_d = g\left(\sum_{k=1}^{K} x_{p(d-1)+k} w_k \right)$$



Strided convolution

the same idea extends to higher dimensions

$$y_{d_1,d_2} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} x_{p_1(d_1-1)+k_1, p_2(d_2-1)+k_2} w_{k_1,k_2}$$

different step-sizes for different dimensions

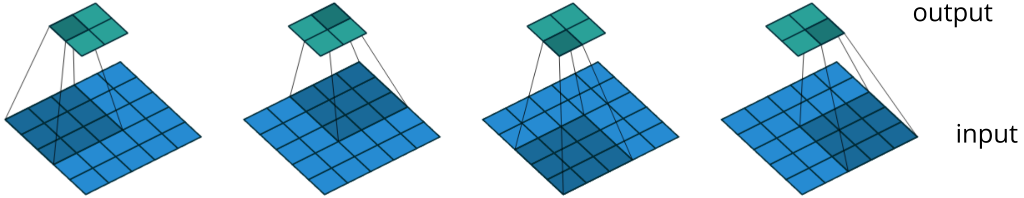


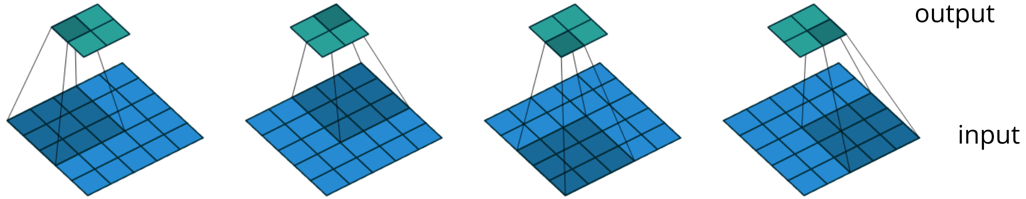
image: Dumoulin & Visin'16

Strided convolution

the same idea extends to higher dimensions

$$y_{d_1,d_2} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} x_{p_1(d_1-1)+k_1, p_2(d_2-1)+k_2} w_{k_1,k_2}$$

different step-sizes for different dimensions



with padding

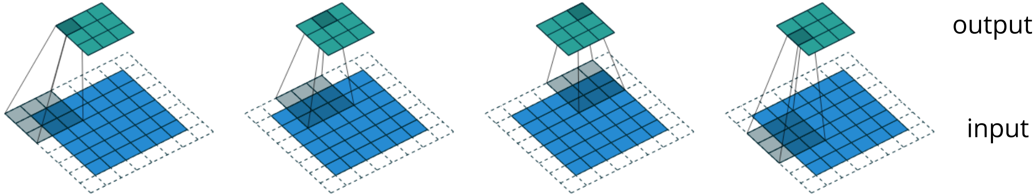


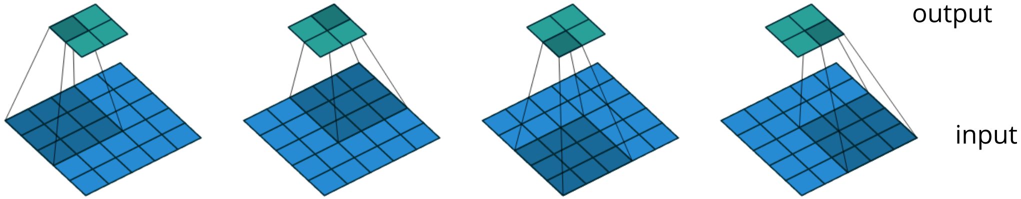
image: Dumoulin & Visin'16

Strided convolution

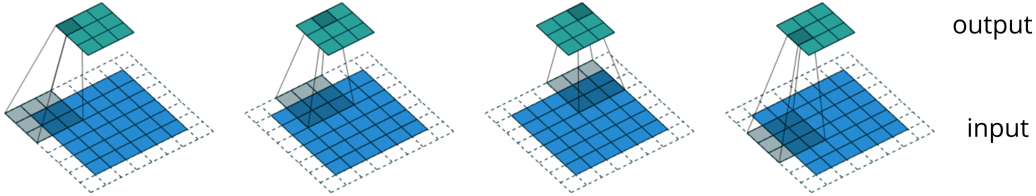
the same idea extends to higher dimensions

$$y_{d_1,d_2} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} x_{p_1(d_1-1)+k_1, p_2(d_2-1)+k_2} w_{k_1,k_2}$$

different step-sizes for different dimensions



with padding



output length (for one dimension)

$$\left\lfloor \frac{D+\text{padding}-K}{\text{stride}} + 1 \right\rfloor$$

image: Dumoulin & Visin'16

Channels

so far we assumed a single input and output sequence or image

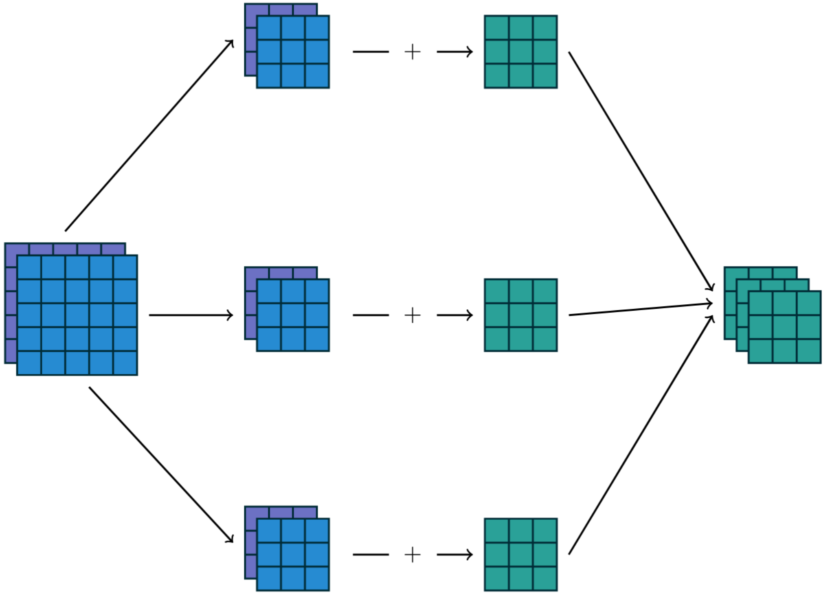
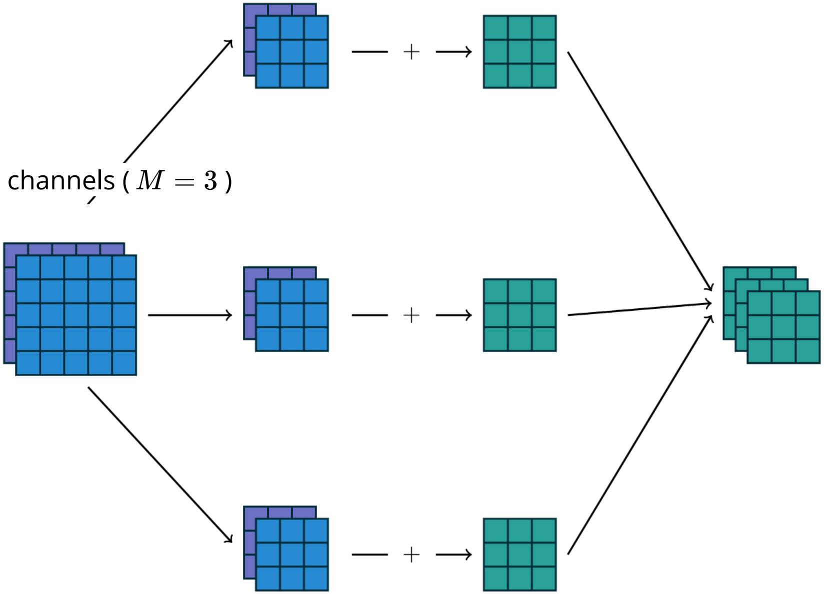


image: Dumoulin & Visin'16

Channels

so far we assumed a single input and output sequence or image



with RGB data, we have 3 input channels ($M = 3$)

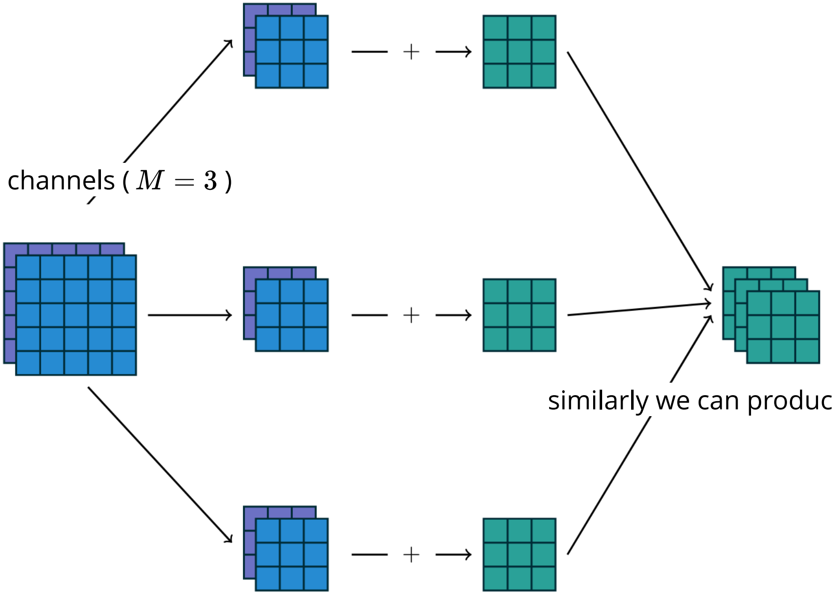
this example: 2 input channels

$x \in \mathbb{R}^{M \times D_1 \times D_2}$

image: Dumoulin & Visin'16

Channels

so far we assumed a single input and output sequence or image



with RGB data, we have 3 input channels ($M = 3$)

this example: 2 input channels

$x \in \mathbb{R}^{M \times D_1 \times D_2}$

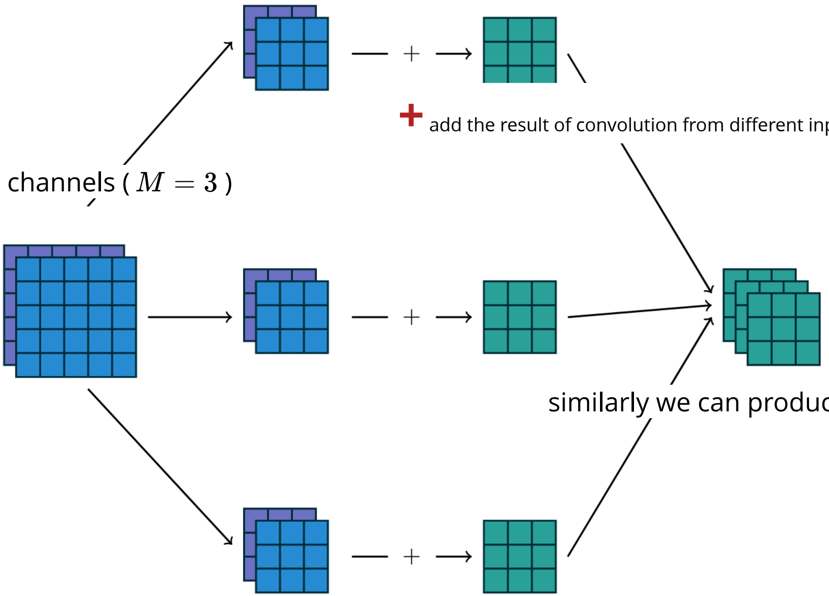
similarly we can produce multiple output channels $M' = 3$

$y \in \mathbb{R}^{M' \times D'_1 \times D'_2}$

image: Dumoulin & Visin'16

Channels

so far we assumed a single input and output sequence or image



we have one $K_1 \times K_2$ filters per input-output channel combination $w \in \mathbb{R}^{M \times M' \times K_1 \times K_2}$

➕ add the result of convolution from different input channels

with RGB data, we have 3 input channels ($M = 3$)

this example: 2 input channels

$x \in \mathbb{R}^{M \times D_1 \times D_2}$

similarly we can produce multiple output channels $M' = 3$

$y \in \mathbb{R}^{M' \times D'_1 \times D'_2}$

image: Dumoulin & Visin'16

Channels

so far we assumed a single input and output sequence or image

image: https://cs231n.github.io/convolutional-networks/

Channels

so far we assumed a single input and output sequence or image

we can also add a *bias parameter (b),* one per each output channel $\boldsymbol{b} \in \mathbb{R}^{M'}$

image: https://cs231n.github.io/convolutional-networks/

Channels

so far we assumed a single input and output sequence or image

we can also add a *bias parameter (b),* one per each output channel $b \in \mathbb{R}^{M'}$

$$y_{m',d_1,d_2} = g\left(\sum_{m=1}^{M} \sum_{k_1} \sum_{k_2} w_{m,m',k_1,k_2}\, x_{m,d_1+k_1-1,d_2+k_2-1} + b_{m'}\right)$$

$x \in \mathbb{R}^{M \times D_1 \times D_2}$

$y \in \mathbb{R}^{M' \times D'_1 \times D'_2}$

$w \in \mathbb{R}^{M \times M' \times K_1 \times K_2}$

image: https://cs231n.github.io/convolutional-networks/

Channels

so far we assumed a single input and output sequence or image

we can also add a *bias parameter (b),* one per each output channel $b \in \mathbb{R}^{M'}$

$$y_{m',d_1,d_2} = g\left(\sum_{m=1}^{M} \sum_{k_1} \sum_{k_2} w_{m,m',k_1,k_2}\, x_{m,d_1+k_1-1,d_2+k_2-1} + b_{m'} \right)$$

$y \in \mathbb{R}^{M' \times D_1' \times D_2'}$

$w \in \mathbb{R}^{M \times M' \times K_1 \times K_2}$

$x \in \mathbb{R}^{M \times D_1 \times D_2}$

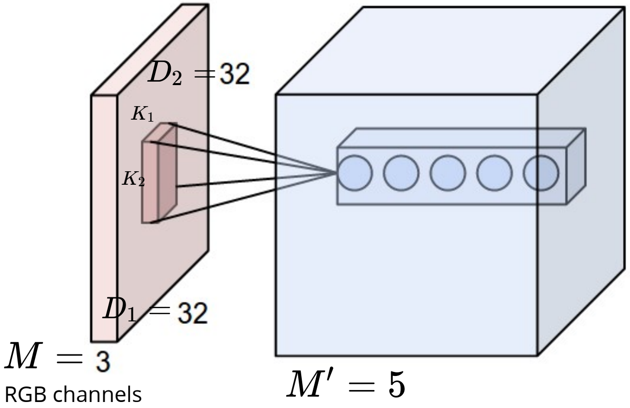


image: https://cs231n.github.io/convolutional-networks/

Convolutional Neural Network (CNN)

CNN or convnet is a neural network with convolutional layers (so it's a special type of MLP)

Convolutional Neural Network (CNN)

CNN or convnet is a neural network with convolutional layers (so it's a special type of MLP)

it could be applied to 1D sequence, 2D image or 3D volumetric data

Convolutional Neural Network (CNN)

CNN or convnet is a neural network with convolutional layers (so it's a special type of MLP)

it could be applied to 1D sequence, 2D image or 3D volumetric data

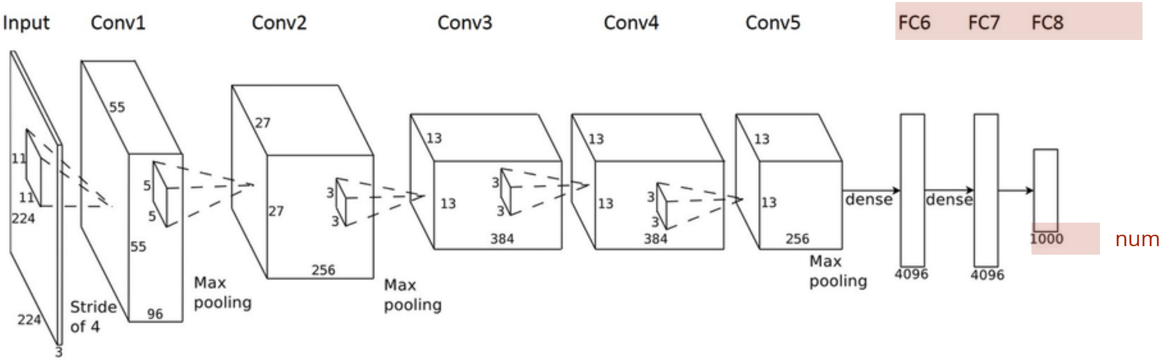
example: conv-net architecture (derived from AlexNet) for image classification

Convolutional Neural Network (CNN)

CNN or convnet is a neural network with convolutional layers (so it's a special type of MLP)

it could be applied to 1D sequence, 2D image or 3D volumetric data

example: conv-net architecture (derived from AlexNet) for image classification



fully connected layers

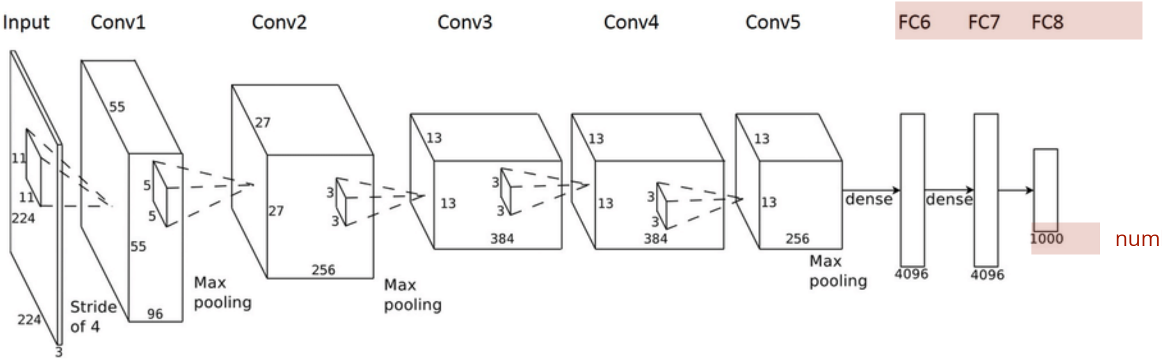
number of classes

Convolutional Neural Network (CNN)

CNN or convnet is a neural network with convolutional layers (so it's a special type of MLP)

it could be applied to 1D sequence, 2D image or 3D volumetric data

example: conv-net architecture (derived from AlexNet) for image classification

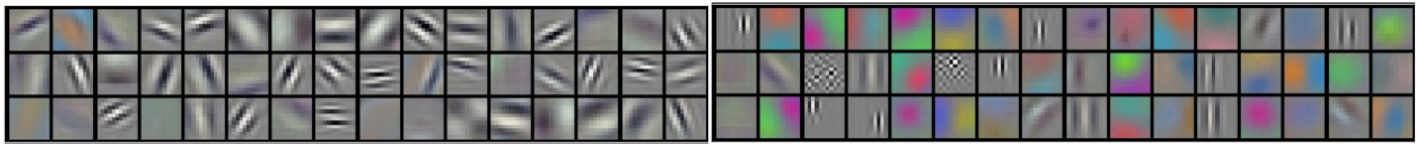


fully connected layers

number of classes

↓ visualization of the convolution kernel at the first layer 11x11x3x96

96 filters, each one is 11x11x3. each of these is responsible for one of 96 feature maps in the second layer

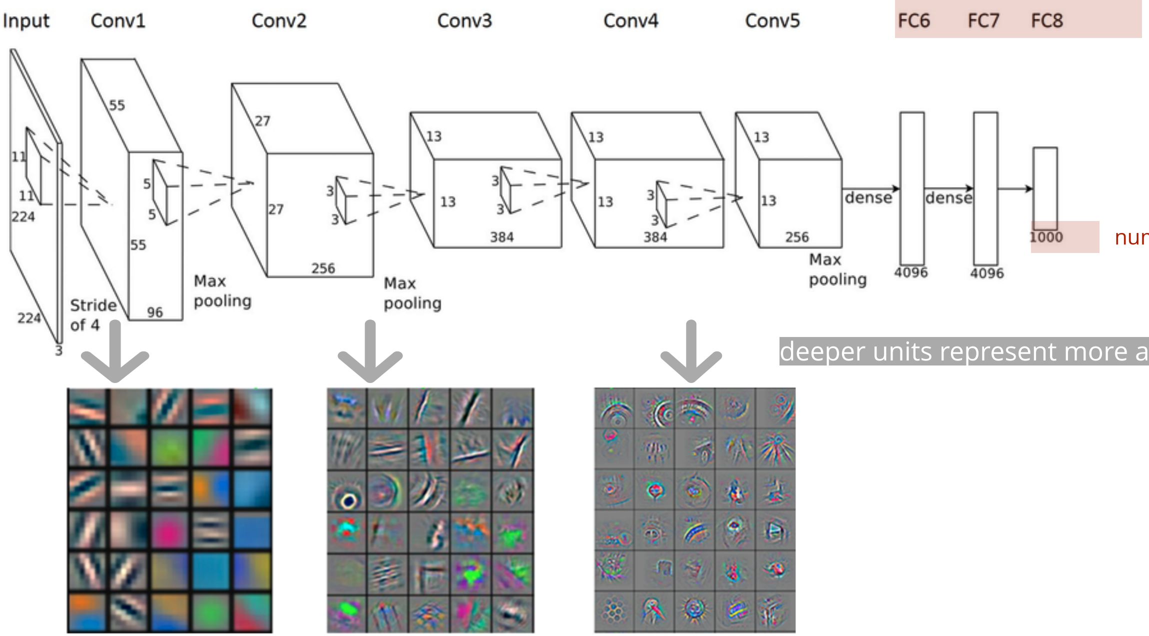


Convolutional Neural Network (CNN)

CNN or convnet is a neural network with convolutional layers (so it's a special type of MLP)

it could be applied to 1D sequence, 2D image or 3D volumetric data

example: conv-net architecture (derived from AlexNet) for image classification



fully connected layers

number of classes

deeper units represent more abstract features

Application: image classification

Convnets have achieved super-human performance in image classification

image credit: He et al'15, https://semiengineering.com/new-vision-technologies-for-real-world-applications/

Application: image classification

Convnets have achieved super-human performance in image classification

ImageNet challenge: > 1M images, 1000 classes

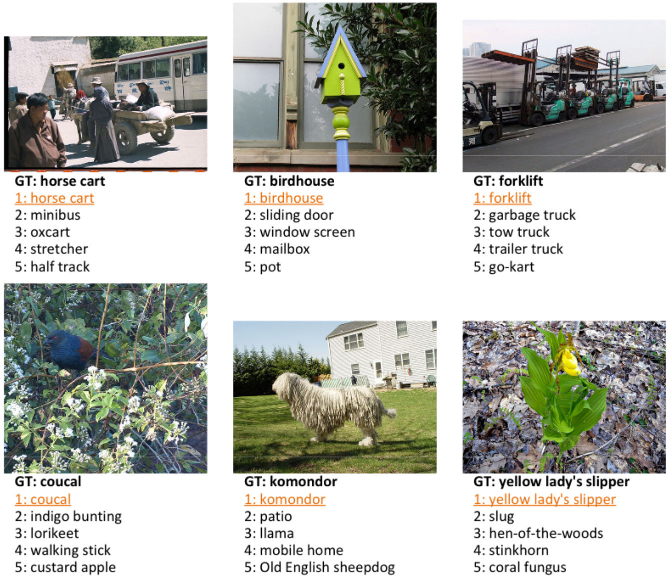
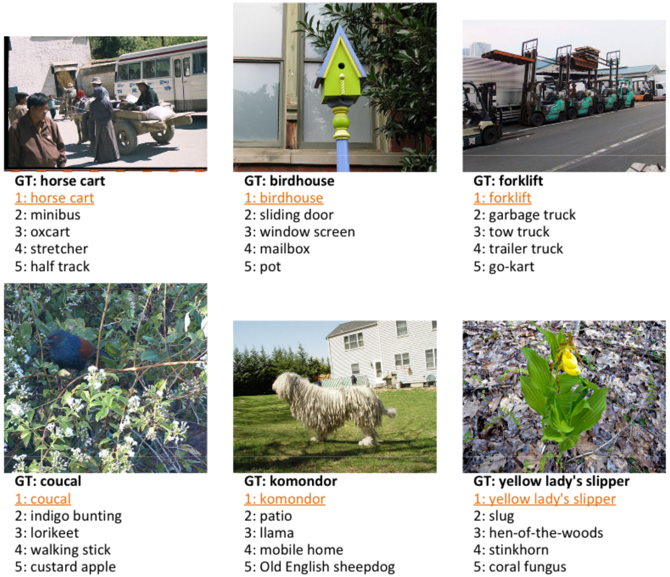


image credit: He et al'15, https://semiengineering.com/new-vision-technologies-for-real-world-applications/

Application: image classification

Convnets have achieved super-human performance in image classification

ImageNet challenge: > 1M images, 1000 classes



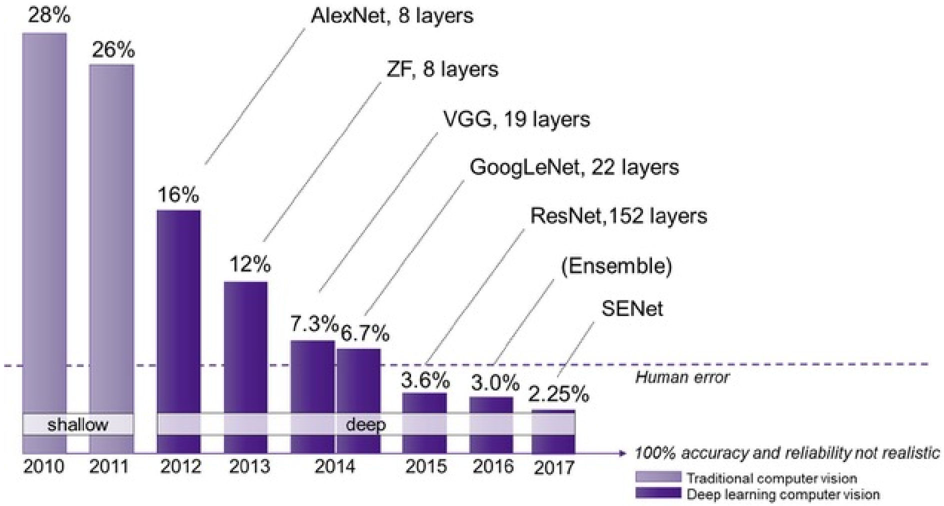


image credit: He et al'15, https://semiengineering.com/new-vision-technologies-for-real-world-applications/

Application: image classification

variety of increasingly deeper architectures have been proposed

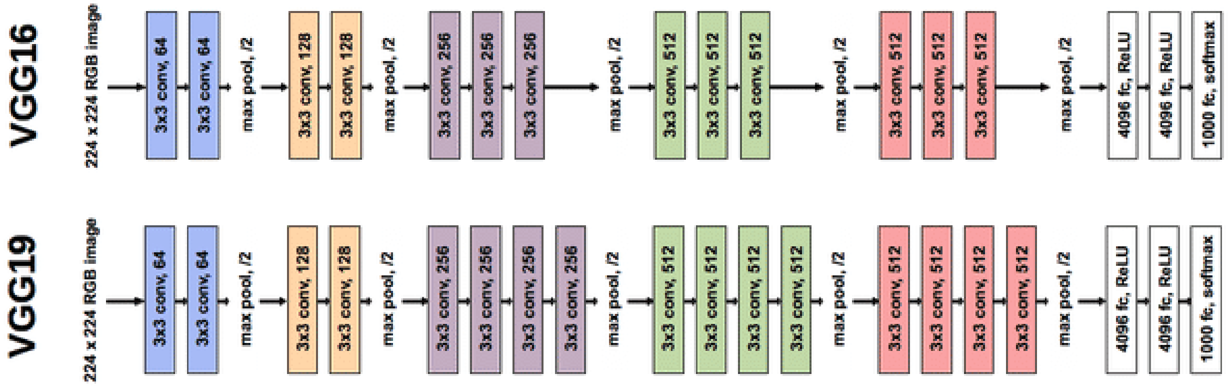


image credit: He et al'15, https://semiengineering.com/new-vision-technologies-for-real-world-applications/

Application: image classification

variety of increasingly deeper architectures have been proposed

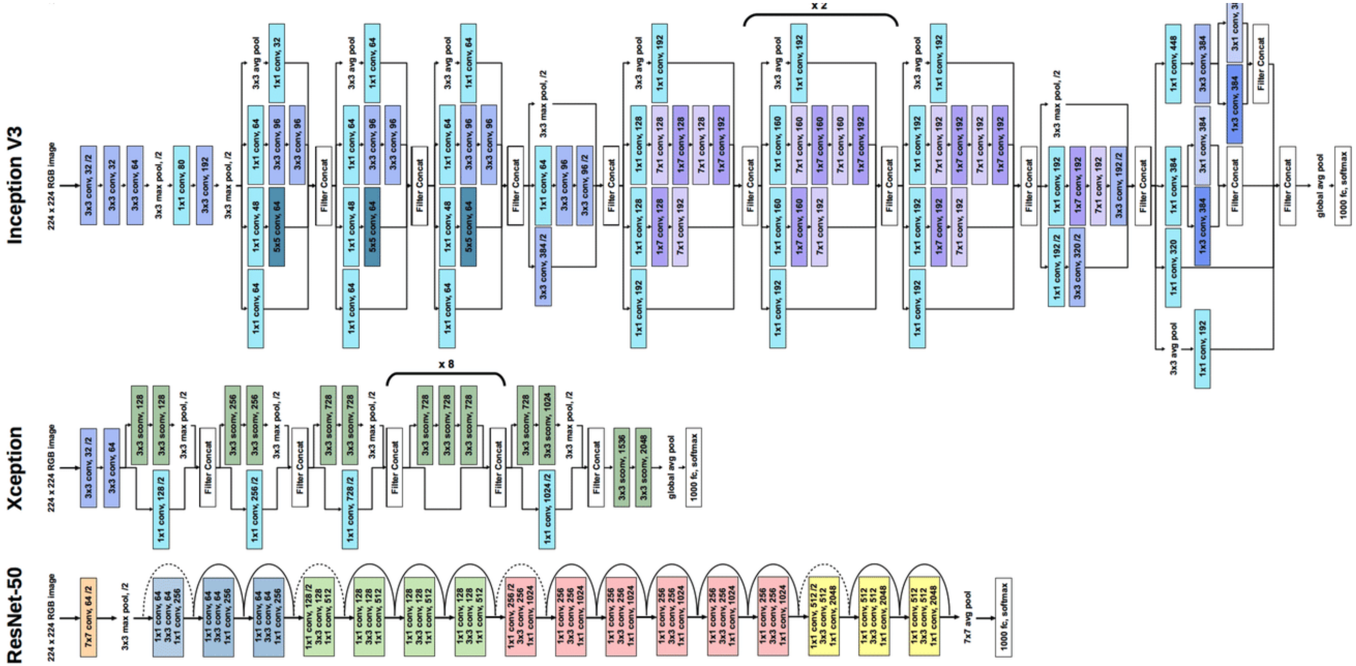


image credit: He et al'15, https://semiengineering.com/new-vision-technologies-for-real-world-applications/

Training: backpropagation through convolution

consider the strided 1D convolution op. $y_{m',d} = \sum_m \sum_k w_{m,m',k} x_{m,p(d-1)+k}$

output channel index

input channel index

filter index

stride

$$\frac{\partial J}{\partial y_{m',d'}}$$

Training: backpropagation through convolution

consider the strided 1D convolution op. $y_{m',d} = \sum_m \sum_k w_{m,m',k} x_{m,p(d-1)+k}$

output channel index   input channel index   filter index   stride

using backprop. we have $\frac{\partial J}{\partial y_{m',d'}}$ so far and we need

Training: backpropagation through convolution

consider the strided 1D convolution op. $y_{m',d} = \sum_m \sum_k w_{m,m',k} x_{m,p(d-1)+k}$

output channel index input channel index filter index stride

using backprop. we have $\frac{\partial J}{\partial y_{m',d'}}$ so far and we need

1) $\frac{\partial y_{m',d'}}{\partial w_{m,m',k}}$ so as to get the gradients

Training: backpropagation through convolution

consider the strided 1D convolution op. $y_{m',d} = \sum_m \sum_k w_{m,m',k} x_{m,p(d-1)+k}$

output channel index   input channel index   filter index   stride

using backprop. we have $\frac{\partial J}{\partial y_{m',d'}}$ so far and we need

1) $\frac{\partial y_{m',d'}}{\partial w_{m,m',k}}$ so as to get the gradients

2) $\frac{\partial y_{m',d'}}{\partial x_{m,d}}$ to backpropagate to previous layer

Training: backpropagation through convolution

consider the strided 1D convolution op. $y_{m',d} = \sum_m \sum_k w_{m,m',k} x_{m,p(d-1)+k}$

output channel index input channel index filter index stride

using backprop. we have $\frac{\partial J}{\partial y_{m',d'}}$ so far and we need

1) $\frac{\partial y_{m',d'}}{\partial w_{m,m',k}}$ so as to get the gradients $\frac{\partial J}{\partial w_{m,m',k}} = \sum_{d'} \frac{\partial J}{\partial y_{m',d'}} \frac{\partial y_{m',d'}}{\partial w_{m,m',k}}$

2) $\frac{\partial y_{m',d'}}{\partial x_{m,d}}$ to backpropagate to previous layer

Training: backpropagation through convolution

consider the strided 1D convolution op. $y_{m',d} = \sum_m \sum_k w_{m,m',k} x_{m,p(d-1)+k}$

output channel index input channel index filter index stride

using backprop. we have $\frac{\partial J}{\partial y_{m',d'}}$ so far and we need

1) $\frac{\partial y_{m',d'}}{\partial w_{m,m',k}}$ so as to get the gradients $\frac{\partial J}{\partial w_{m,m',k}} = \sum_{d'} \frac{\partial J}{\partial y_{m',d'}} \overset{x_{m,p(d'-1)+k}}{\boxed{\frac{\partial y_{m',d'}}{\partial w_{m,m',k}}}}$

2) $\frac{\partial y_{m',d'}}{\partial x_{m,d}}$ to backpropagate to previous layer

Training: backpropagation through convolution

consider the strided 1D convolution op. $y_{m',d} = \sum_m \sum_k w_{m,m',k} x_{m,p(d-1)+k}$

output channel index input channel index filter index stride

using backprop. we have $\frac{\partial J}{\partial y_{m',d'}}$ so far and we need

1) $\frac{\partial y_{m',d'}}{\partial w_{m,m',k}}$ so as to get the gradients $\frac{\partial J}{\partial w_{m,m',k}} = \sum_{d'} \frac{\partial J}{\partial y_{m',d'}} \overset{x_{m,p(d'-1)+k}}{\boxed{\frac{\partial y_{m',d'}}{\partial w_{m,m',k}}}}$

2) $\frac{\partial y_{m',d'}}{\partial x_{m,d}}$ to backpropagate to previous layer $\frac{\partial J}{\partial x_{d,m}} = \sum_{d',m'} \frac{\partial J}{\partial y_{m',d'}} \frac{\partial y_{m',d'}}{\partial x_{d,m}}$

Training: backpropagation through convolution

consider the strided 1D convolution op. $y_{m',d} = \sum_m \sum_k w_{m,m',k} \, x_{m,p(d-1)+k}$

output channel index input channel index filter index stride

using backprop. we have $\frac{\partial J}{\partial y_{m',d'}}$ so far and we need

1) $\frac{\partial y_{m',d'}}{\partial w_{m,m',k}}$ so as to get the gradients $\frac{\partial J}{\partial w_{m,m',k}} = \sum_{d'} \frac{\partial J}{\partial y_{m',d'}} \boxed{\frac{\partial y_{m',d'}}{\partial w_{m,m',k}}}$

$x_{m,p(d'-1)+k}$

$\sum_k w_{m,m',k}$

such that

$p(d'-1)+k=d$

2) $\frac{\partial y_{m',d'}}{\partial x_{m,d}}$ to backpropagate to previous layer $\frac{\partial J}{\partial x_{d,m}} = \sum_{d',m'} \frac{\partial J}{\partial y_{m',d}} \boxed{\frac{\partial y_{m',d'}}{\partial x_{d,m}}}$

Training: backpropagation through convolution

consider the strided 1D convolution op. $y_{m',d} = \sum_m \sum_k w_{m,m',k} x_{m,p(d-1)+k}$

output channel index

input channel index

filter index

stride

using backprop. we have $\frac{\partial J}{\partial y_{m',d'}}$ so far and we need

$x_{m,p(d'-1)+k}$

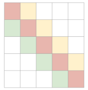
1) $\frac{\partial y_{m',d'}}{\partial w_{m,m',k}}$ so as to get the gradients $\frac{\partial J}{\partial w_{m,m',k}} = \sum_{d'} \frac{\partial J}{\partial y_{m',d'}} \boxed{\frac{\partial y_{m',d'}}{\partial w_{m,m',k}}}$

$\sum_k w_{m,m',k}$

such that

$p(d'-1)+k=d$

2) $\frac{\partial y_{m',d'}}{\partial x_{m,d}}$ to backpropagate to previous layer $\frac{\partial J}{\partial x_{d,m}} = \sum_{d',m'} \frac{\partial J}{\partial y_{m',d}} \boxed{\frac{\partial y_{m',d'}}{\partial x_{d,m}}}$



this operation is similar to multiplication by transpose of the parameter-sharing matrix **(transposed convolution)**

Naive implementation

consider the strided 1D convolution op. with stride 1. and single input-output channels

$$y_{d} = \sum_{k} w_{k} x_{d+k-1}$$

Naive implementation

consider the strided 1D convolution op. with stride 1. and single input-output channels

$$y_d = \sum_k w_k x_{d+k-1}$$

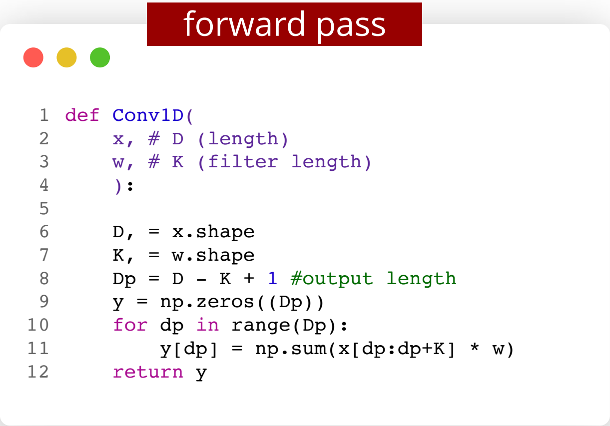
in practice most efficient implementation depends on the filter size (using FFT for large filters)

Naive implementation

consider the strided 1D convolution op. with stride 1. and single input-output channels

$$y_d = \sum_k w_k x_{d+k-1}$$

in practice most efficient implementation depends on the filter size (using FFT for large filters)



forward pass

```
def Conv1D(
    x, # D (length)
    w, # K (filter length)
    ):

    D, = x.shape
    K, = w.shape
    Dp = D - K + 1 #output length
    y = np.zeros((Dp))
    for dp in range(Dp):
        y[dp] = np.sum(x[dp:dp+K] * w)
    return y
```

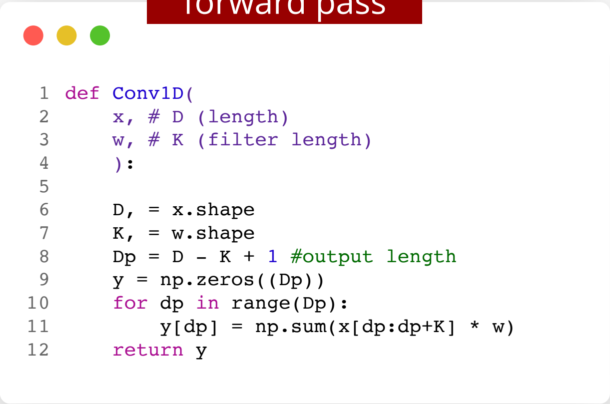
Naive implementation

consider the strided 1D convolution op. with stride 1. and single input-output channels

$$y_d = \sum_k w_k x_{d+k-1}$$

in practice most efficient implementation depends on the filter size (using FFT for large filters)

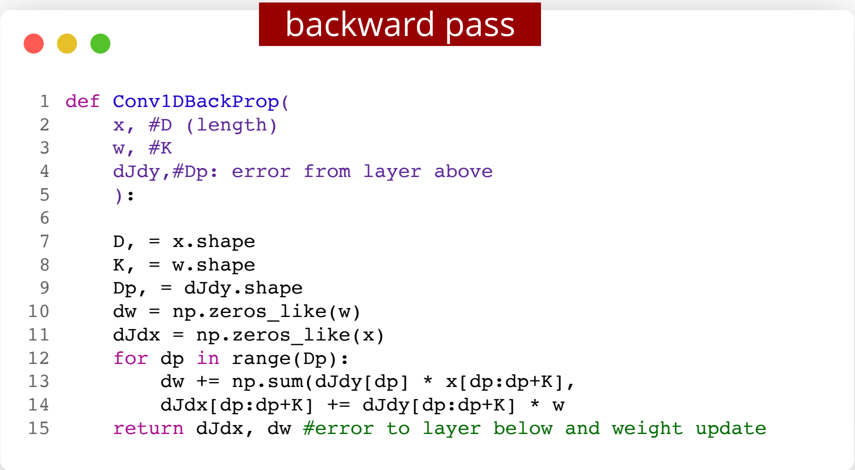
forward pass



```
def Conv1D(
    x, # D (length)
    w, # K (filter length)
    ):

    D, = x.shape
    K, = w.shape
    Dp = D - K + 1 #output length
    y = np.zeros((Dp))
    for dp in range(Dp):
        y[dp] = np.sum(x[dp:dp+K] * w)
    return y
```

backward pass



```
def Conv1DBackProp(
    x, #D (length)
    w, #K
    dJdy,#Dp: error from layer above
    ):

    D, = x.shape
    K, = w.shape
    Dp, = dJdy.shape
    dw = np.zeros_like(w)
    dJdx = np.zeros_like(x)
    for dp in range(Dp):
        dw += np.sum(dJdy[dp] * x[dp:dp+K],
        dJdx[dp:dp+K] += dJdy[dp:dp+K] * w
    return dJdx, dw #error to layer below and weight update
```

Transposed Convolution

Transposed convolution (aka deconvolution) recovers the shape of the original input

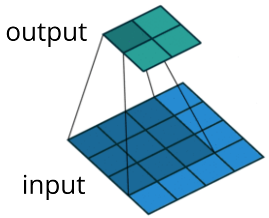
image: Dumoulin & Visin'16

Transposed Convolution

Transposed convolution (aka deconvolution) recovers the shape of the original input

Convolution with **no stride** and its transpose

no padding of the original convolution corresponds to full padding of in transposed version



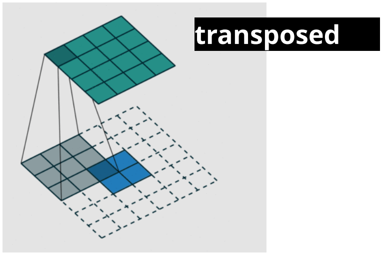


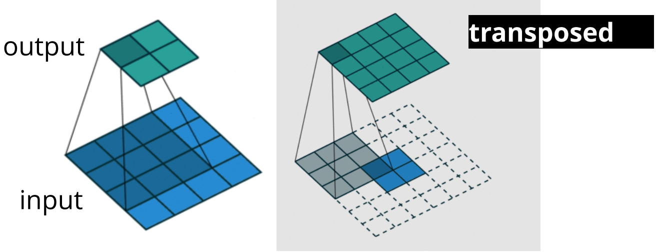
image: Dumoulin & Visin'16

Transposed Convolution

Transposed convolution (aka deconvolution) recovers the shape of the original input

Convolution with **no stride** and its transpose

no padding of the original convolution corresponds to full padding of in transposed version



full padding of the original convolution corresponds to no paddingof in transposed version

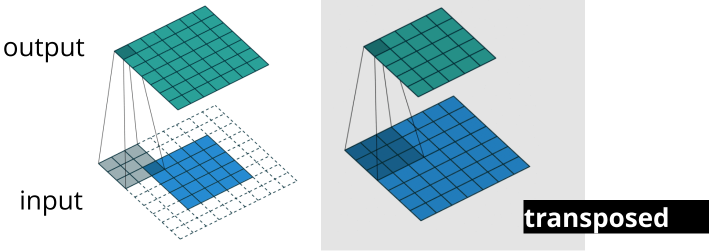


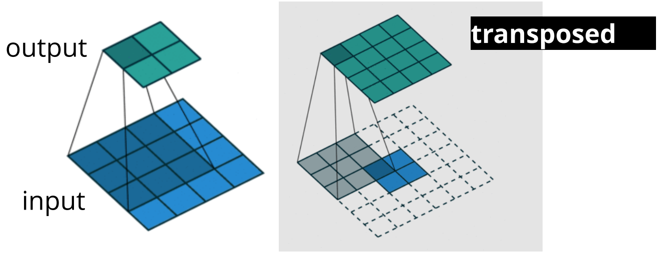
image: Dumoulin & Visin'16

Transposed Convolution

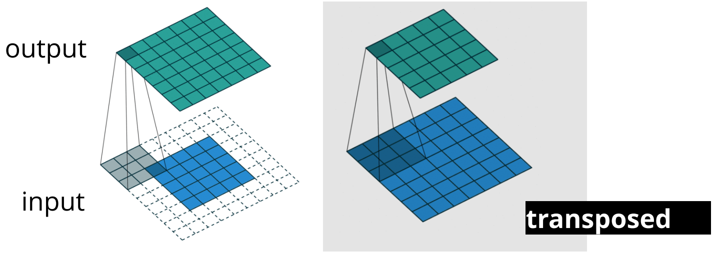
Transposed convolution (aka deconvolution) recovers the shape of the original input

Convolution with **no stride** and its transpose

no padding of the original convolution corresponds to full padding of in transposed version



full padding of the original convolution corresponds to no paddingof in transposed version



Convolution **with stride** and its transpose

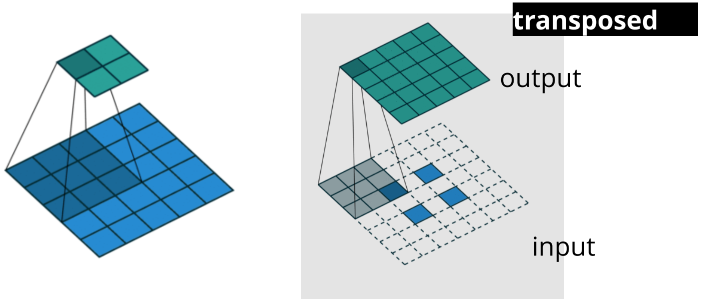


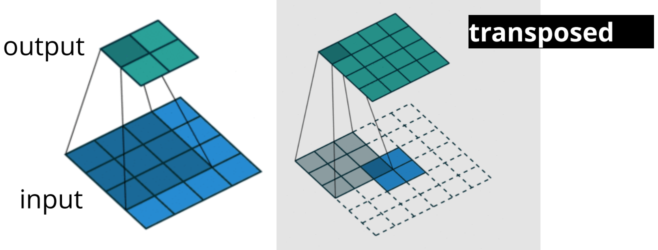
image: Dumoulin & Visin'16

Transposed Convolution

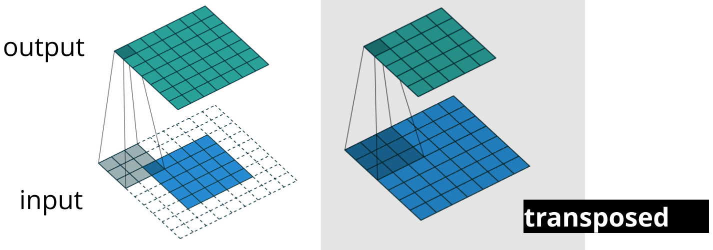
Transposed convolution (aka deconvolution) recovers the shape of the original input

Convolution with **no stride** and its transpose

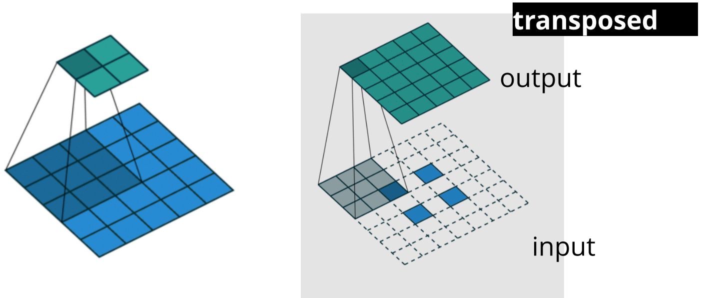
no padding of the original convolution corresponds to full padding of in transposed version



full padding of the original convolution corresponds to no paddingof in transposed version



Convolution **with stride** and its transpose



this can be used for up-sampling (opposite of stride/pooling)

as expected the transpose of a transposed convolution is the original convolution

image: Dumoulin & Visin'16

Dilated Convolution

Dilated (aka atrous) convolution

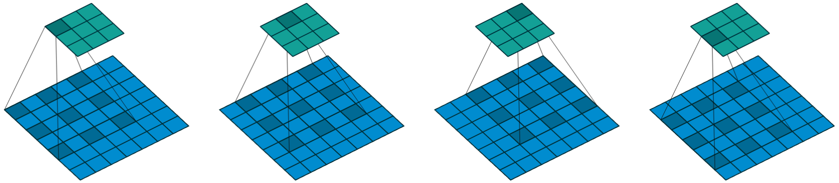
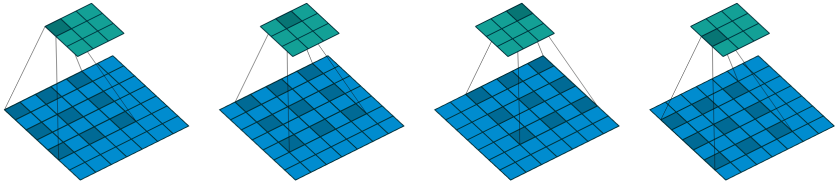


image credits: Kalchbrenner et al'17, Dumoulin & Visin'16

Dilated Convolution

Dilated (aka atrous) convolution



this can be used to create exponentially large receptive field in few layers

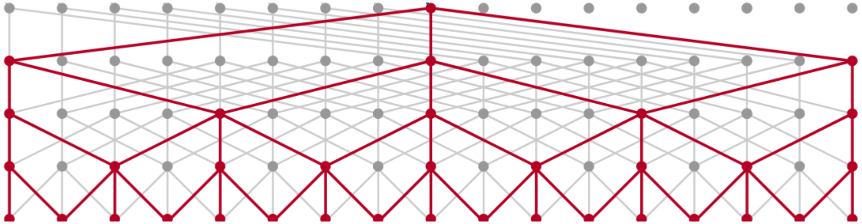
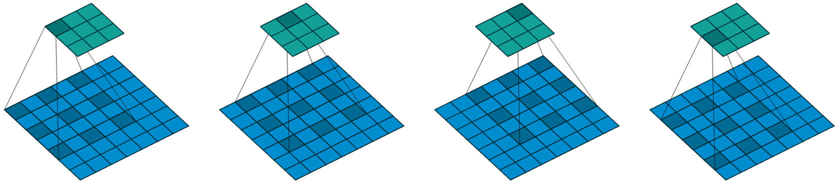


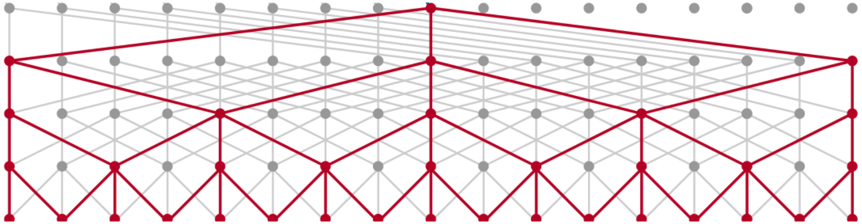
image credits: Kalchbrenner et al'17, Dumoulin & Visin'16

Dilated Convolution

Dilated (aka atrous) convolution



this can be used to create exponentially large receptive field in few layers

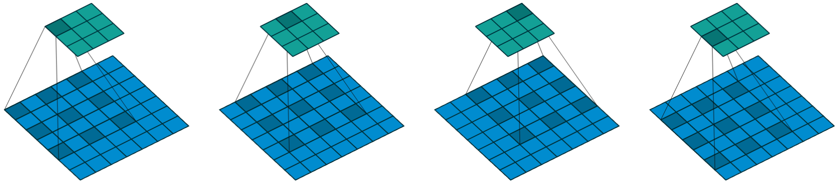


dilation = 1 (i.e., no dilation), size of receptive field = 3

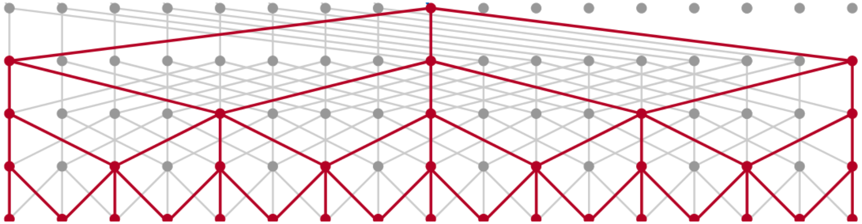
image credits: Kalchbrenner et al'17, Dumoulin & Visin'16

Dilated Convolution

Dilated (aka atrous) convolution



this can be used to create exponentially large receptive field in few layers



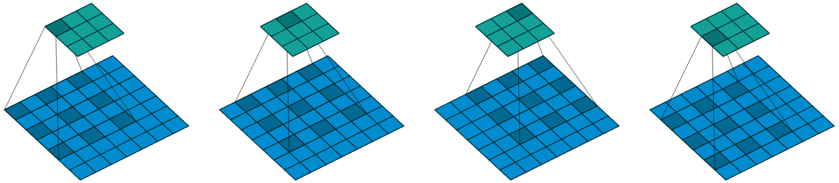
dilation = 2, size of receptive field = 7

dilation = 1 (i.e., no dilation), size of receptive field = 3

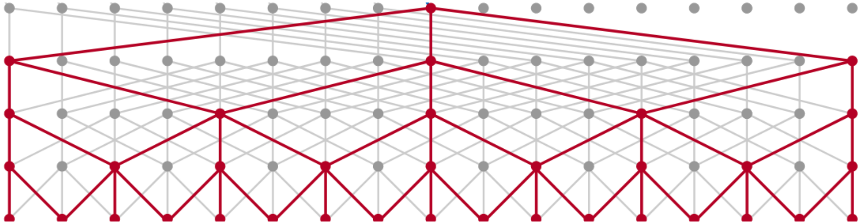
image credits: Kalchbrenner et al'17, Dumoulin & Visin'16

Dilated Convolution

Dilated (aka atrous) convolution



this can be used to create exponentially large receptive field in few layers



dilation = 4, size of receptive field = 15

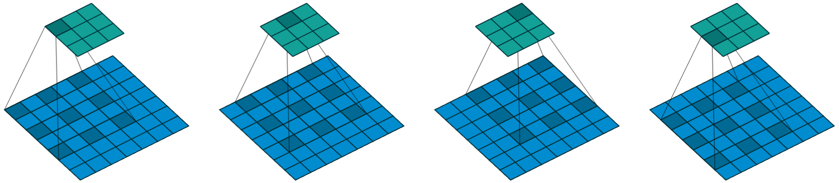
dilation = 2, size of receptive field = 7

dilation = 1 (i.e., no dilation), size of receptive field = 3

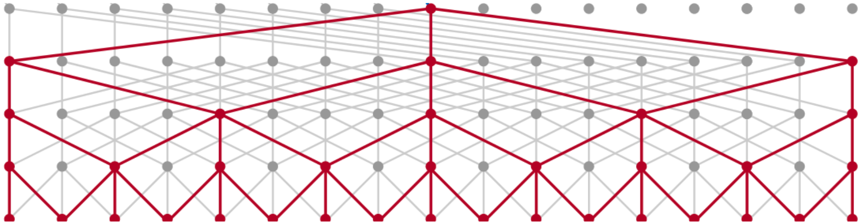
image credits: Kalchbrenner et al'17, Dumoulin & Visin'16

Dilated Convolution

Dilated (aka atrous) convolution



this can be used to create exponentially large receptive field in few layers



dilation = 8, size of receptive field = 31

dilation = 4, size of receptive field = 15

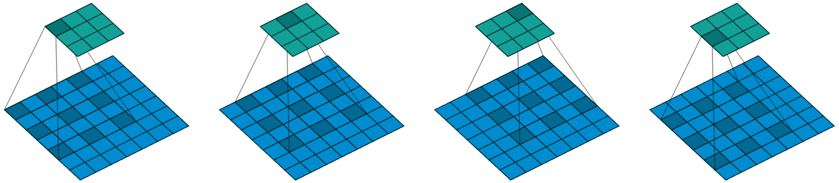
dilation = 2, size of receptive field = 7

dilation = 1 (i.e., no dilation), size of receptive field = 3

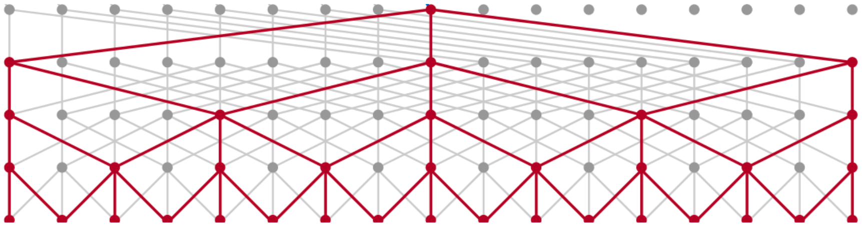
image credits: Kalchbrenner et al'17, Dumoulin & Visin'16

Dilated Convolution

Dilated (aka atrous) convolution



this can be used to create exponentially large receptive field in few layers



in contrast to stride, dilation does not lose resolution

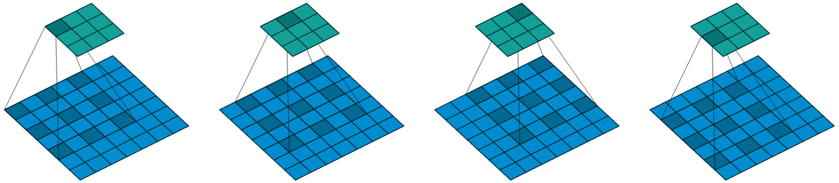
output length (for one dimension)

$$\left\lfloor \frac{D+\text{padding}-\text{dilation}\times(K-1)-1}{\text{stride}} + 1 \right\rfloor$$

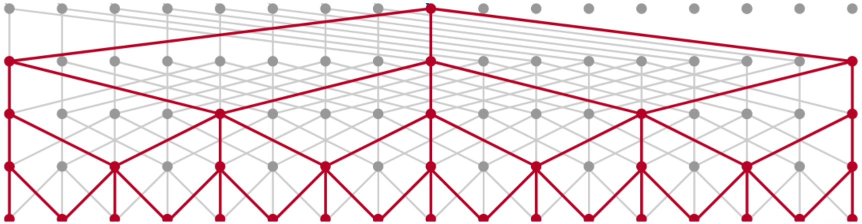
image credits: Kalchbrenner et al'17, Dumoulin & Visin'16

Dilated Convolution

Dilated (aka atrous) convolution



this can be used to create exponentially large receptive field in few layers



dilation = 8, size of receptive field = 31

dilation = 4, size of receptive field = 15

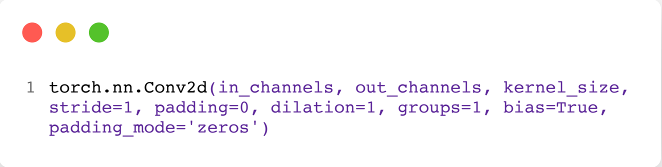
dilation = 2, size of receptive field = 7

dilation = 1 (i.e., no dilation), size of receptive field = 3

in contrast to stride, dilation does not lose resolution

output length (for one dimension)

$$\left\lfloor \frac{D+\text{padding}-\text{dilation}\times(K-1)-1}{\text{stride}}+1\right\rfloor$$



```
torch.nn.Conv2d(in_channels, out_channels, kernel_size,
stride=1, padding=0, dilation=1, groups=1, bias=True,
padding_mode='zeros')
```

image credits: Kalchbrenner et al'17, Dumoulin & Visin'16

Structured Prediction

the output itself may have (image) structure (e.g., predicting text, audio, image)

image:https://sthalles.github.io/deep_segmentation_network/

Structured Prediction

the output itself may have (image) structure (e.g., predicting text, audio, image)

example in (semantic) segmentation, we classify each pixel

loss is the sum of cross-entropy loss across the whole image

image:https://sthalles.github.io/deep_segmentation_network/

Structured Prediction

the output itself may have (image) structure (e.g., predicting text, audio, image)

example in (semantic) segmentation, we classify each pixel

loss is the sum of cross-entropy loss across the whole image

variety of architectures… one that performs well is **U-Net** ⟶

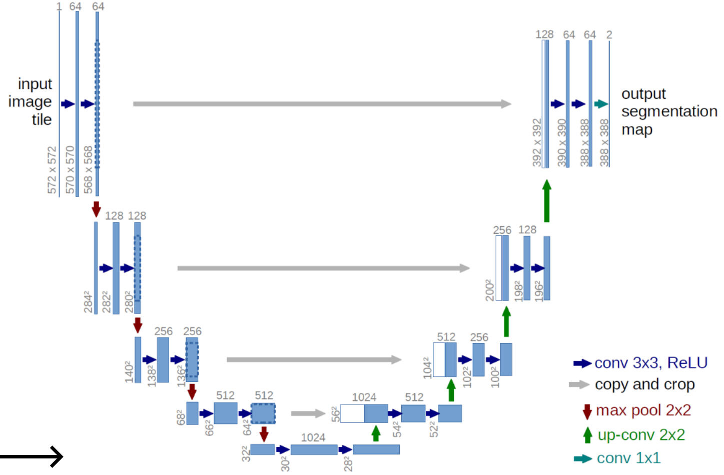


image:https://sthalles.github.io/deep_segmentation_network/

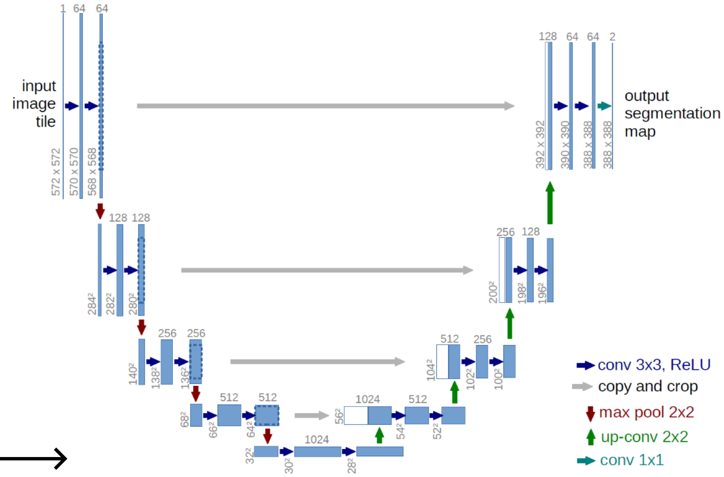
Structured Prediction

the output itself may have (image) structure (e.g., predicting text, audio, image)

example in (semantic) segmentation, we classify each pixel

loss is the sum of cross-entropy loss across the whole image

variety of architectures… one that performs well is **U-Net** ⟶



transposed convolution (upconv), concatenation, and skip connection are common in architecture design

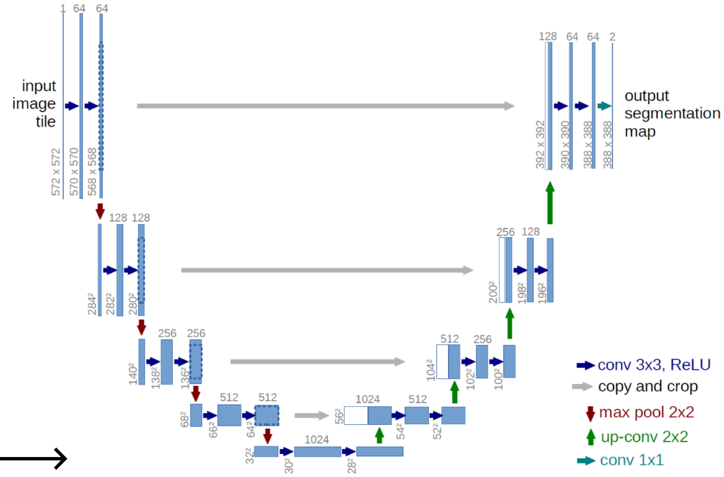
image:https://sthalles.github.io/deep_segmentation_network/

Structured Prediction

the output itself may have (image) structure (e.g., predicting text, audio, image)

example in (semantic) segmentation, we classify each pixel

loss is the sum of cross-entropy loss across the whole image



variety of architectures… one that performs well is **U-Net** ⟶

transposed convolution (upconv), concatenation, and skip connection are common in architecture design

architecture search (i.e., combinatorial hyper-parameter search) is an expensive process and an active research area

image:https://sthalles.github.io/deep_segmentation_network/

Summary

convolution layer introduces an **inductive bias** to MLP

equivariance as an inductive bias:

- translation of the same model is applied to produce different outputs (pixels)
- the layer is equivariant to **translation**
- achieved through **parameter-sharing**

Summary

convolution layer introduces an **inductive bias** to MLP

equivariance as an inductive bias:

- translation of the same model is applied to produce different outputs (pixels)
- the layer is equivariant to **translation**
- achieved through **parameter-sharing**

conv-nets use combinations of

- convolution layers
- ReLU (or similar) activations
- pooling and/or stride for down-sampling
- skip-connection and/or batch-norm to help with optimization / regularization
- potentially fully connected layers in the end

Summary

convolution layer introduces an **inductive bias** to MLP

equivariance as an inductive bias:

- translation of the same model is applied to produce different outputs (pixels)
- the layer is equivariant to **translation**
- achieved through **parameter-sharing**

conv-nets use combinations of

- convolution layers
- ReLU (or similar) activations
- pooling and/or stride for down-sampling
- skip-connection and/or batch-norm to help with optimization / regularization
- potentially fully connected layers in the end

training

- backpropagation (similar to MLP)
- SGD or its improved variations with adaptive learning rate
- monitor the validation error for early stopping