Convolutional Neural Networks

Petra Galuscakova
petra.galuscakova@uis.no

University of Stavanger

Information Access & Artificial Intelligence

Department of Electrical Engineering and Computer Science
University of Stavanger

January 8, 2026

Convolutional Neural Networks

- There isn't anything in the basic DNN that takes into account the spatial structure of the input images $\rightarrow$ the performance is quite poor.
- To improve the performance, we can use a convolutional layers.

Convolution

3x3 portion of an image				filter				
0.6	0.2	0.6		1	1	1		
0.1	-0.2	-0.3	*	0	0	0	=	2.3
-0.5	-0.1	-0.3		-1	-1	-1		

-0.6	-0.2	-0.6		1	1	1		
-0.1	0.2	0.3	*	0	0	0	=	-2.3
0.5	0.1	0.3		-1	-1	-1		

- The convolution is performed by multiplying the filter pixelwise with the portion of the image, and summming the result.
- The output is more positive when the portion of the image closely matches the filter and more negative when the portion of the image is the inverse of the filter.

Convolutional Layers

- If we move the filter across the entire image, from left to right and top to bottom, recording the convolutional output as we go, we obtain a new array that picks out a particular feature of the input, depending on the values in the filter.
- This is exactly what a convolutional layer is designed to do, but with multiple filters rather than just one.

Convolutional Layers

input layer
1 x 64 x 64 x 1
batch_size x height x
width x number of filters

1	1	1
0	0	0
-1	-1	-1

-1	0	1
-1	0	1
-1	0	1

2 filters
(each filter is 3 x 3 x 1)

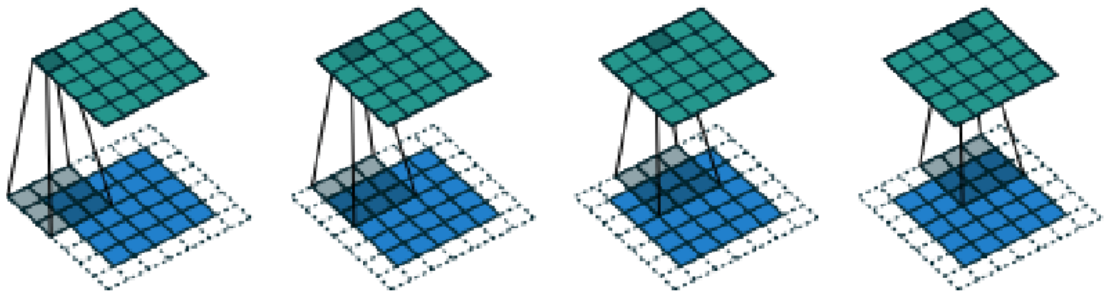
output
1 x 64 x 64 x 2
batch_size x height x
width x number of filters

Building CNN

- https://github.com/dat560-2026/info → Deep learning recap → CNN → Notebook

Padding

- The padding = "same" input parameter pads the input data with zeros so that the output size from the layer is exactly the same as the input size when strides = 1
- Setting padding = "same" is a good way to keep track of the size of the tensor as it passes through many convolutional layers.

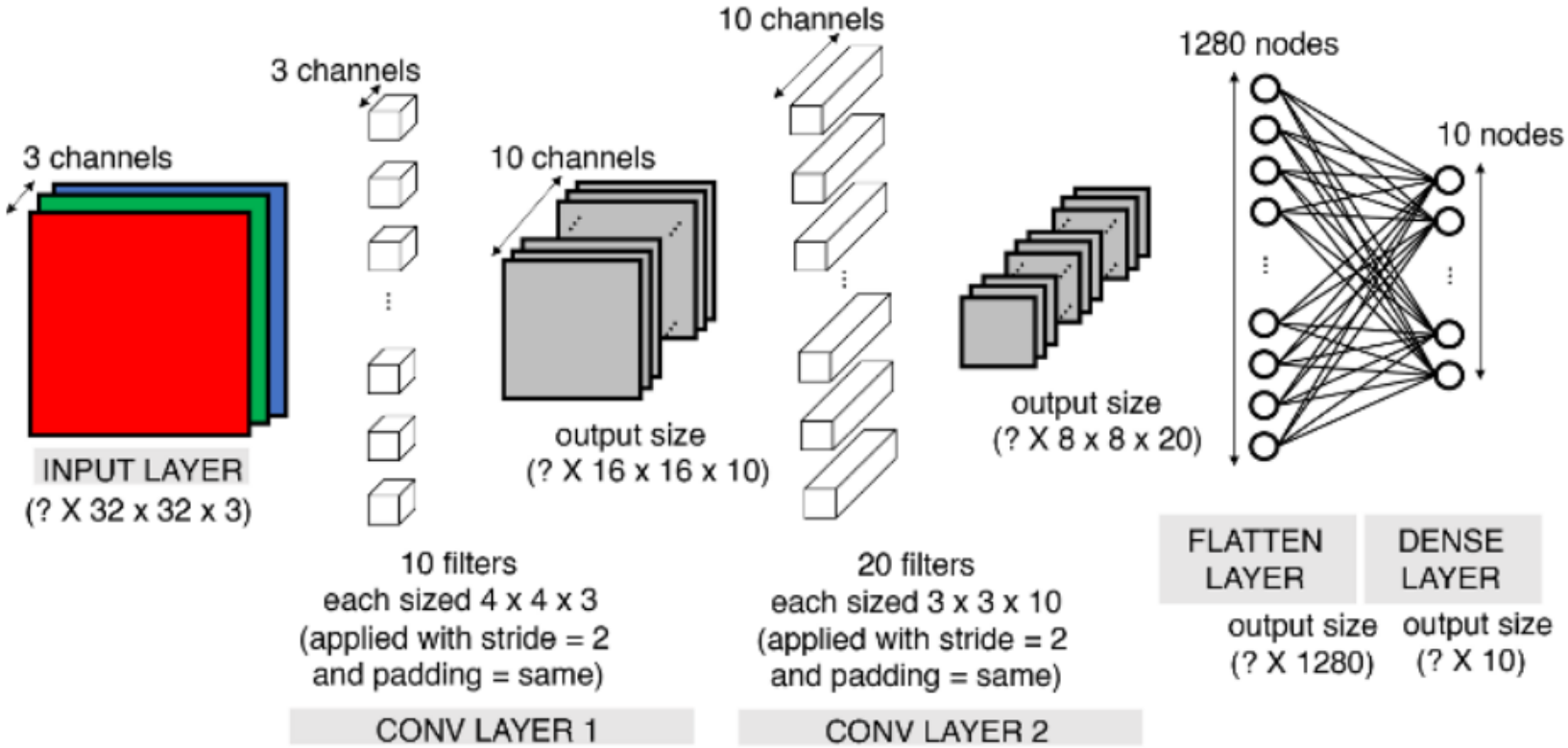


A $3 \times 3 \times 1$ kernel (gray) being passed over a $5 \times 5 \times 1$ input image (blue), with padding="same" and strides $= 1$, to generate the $5 \times 5 \times 1$ output (green) Examples:
https://github.com/jerpint/cnn-cheatsheet

Demo

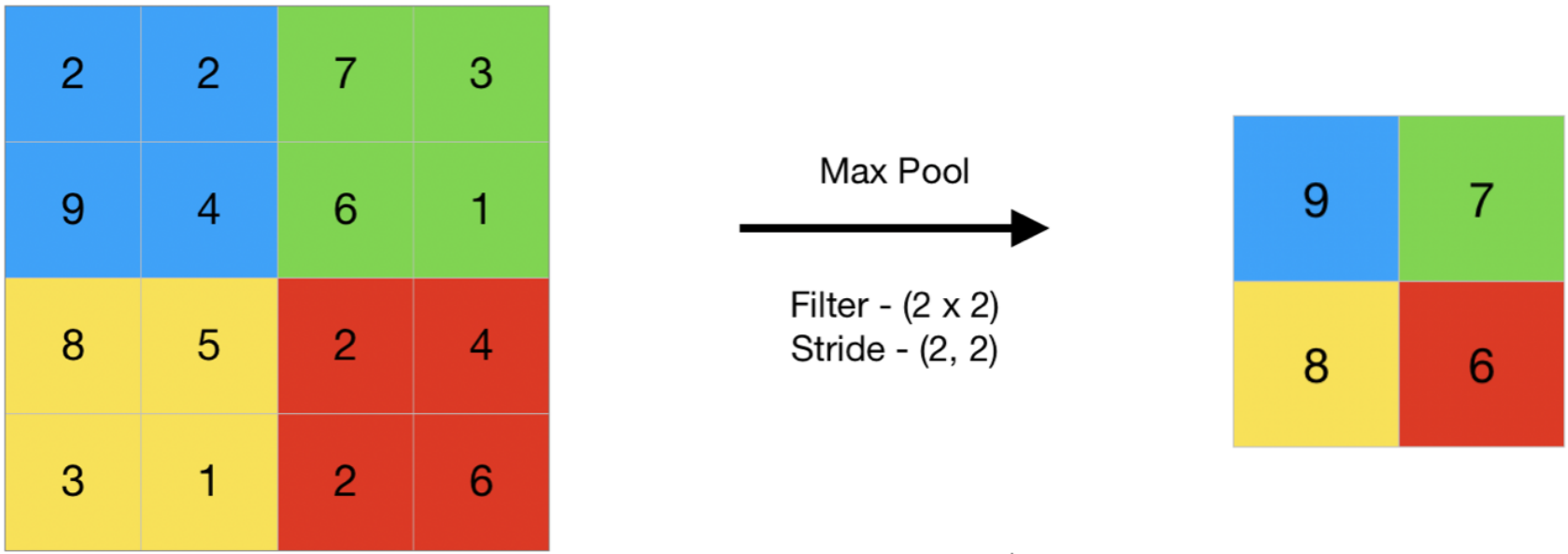
- The values stored in the filters are the weights that are learned by the neural network through training. Initially these are random, but gradually the filters adapt their weights to start picking out interesting features such as edges or particular color combinations.
- https://playground.tensorflow.org

CNN Diagram


3 channels
INPUT LAYER
(? X 32 x 32 x 3)
3 channels
10 filters
each sized 4 x 4 x 3
(applied with stride = 2
and padding = same)
CONV LAYER 1
10 channels
output size
(? X 16 x 16 x 10)
10 channels
20 filters
each sized 3 x 3 x 10
(applied with stride = 2
and padding = same)
CONV LAYER 2
output size
(? X 8 x 8 x 20)
1280 nodes
FLATTEN LAYER
output size
(? X 1280)
10 nodes
DENSE LAYER
output size
(? X 10)

Pooling

- Pooling layer is used in CNNs to reduce the dimensions of the input while retaining the most important information.
- It involves sliding a two-dimensional filter over each channel of a feature map and summarizing (e.g. using maximumg or average) the features within the region covered by the filter.



Pooling (cont.)

Pooling follows convolution to make the model more robust to small changes in an object's position (translation invariance). This ensures that if a detected feature moves by a few pixels, the pooling output remains relatively stable.

Overflow

- The common problem when training a deep neural network is ensuring that the weights of the network remain within a reasonable range of values.
- If they start to become too large, this is a sign that the network is suffering from the exploding gradient problem.
- If the loss function starts to return NaN, chances are that the weights have grown large enough to cause an overflow error.
- The pixel values are first normalized between -1 and 1, but as the network trains and the weights move further away from their random initial values, activations in the future layers break: **covariate shift**.

Covariate Shift

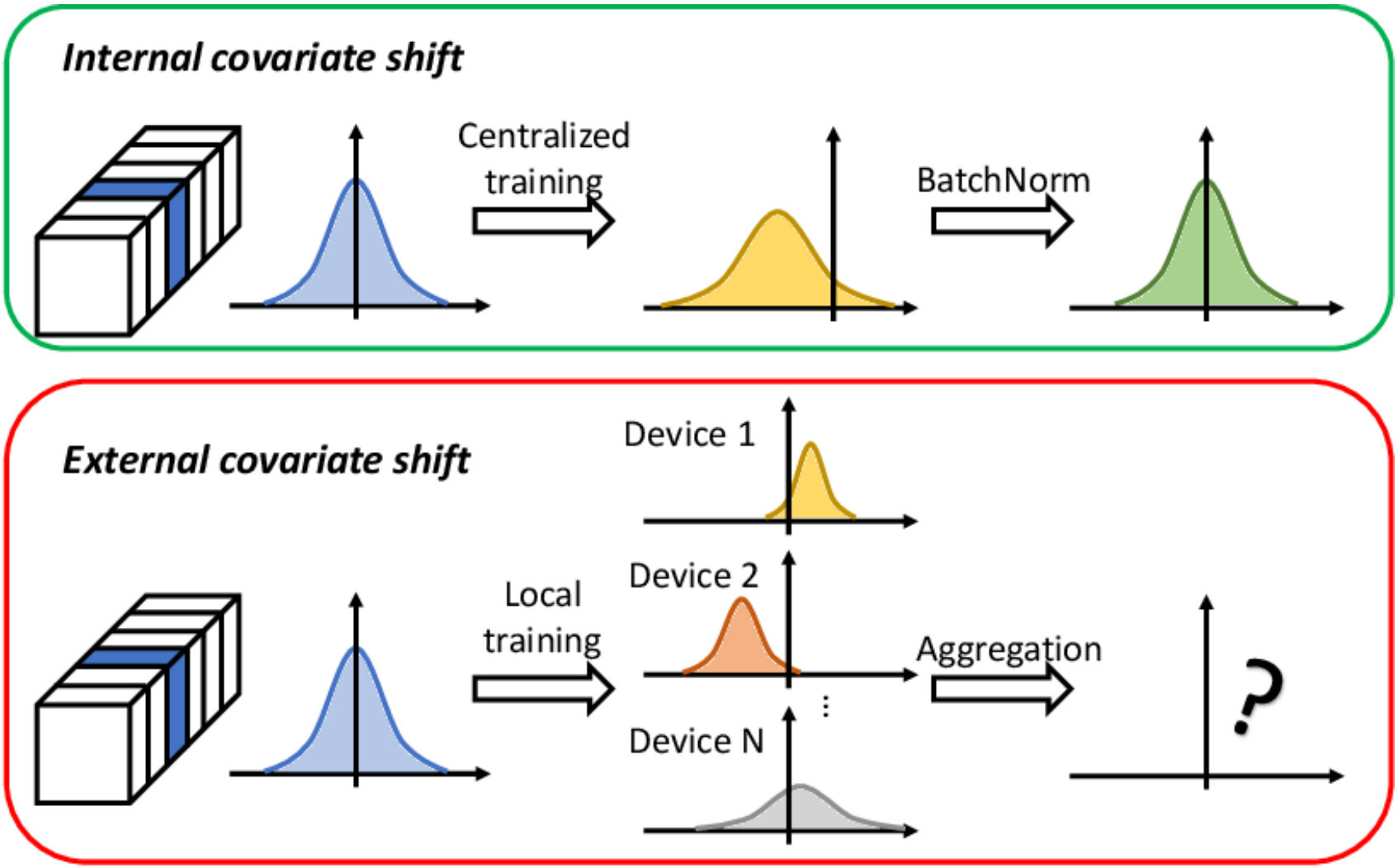


Figure 1: Internal and external covariate shift

Covariate Shift (cont.)

- To remain stable, when the network updates the weights, each layer implicitly assumes that the distribution of its input from the layer beneath is approximately consistent across iterations.

Batch Normalization

A batch normalization layer calculates the mean and standard deviation of each of its input channels across the batch and normalizes by subtracting the mean and dividing by the standard deviation.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

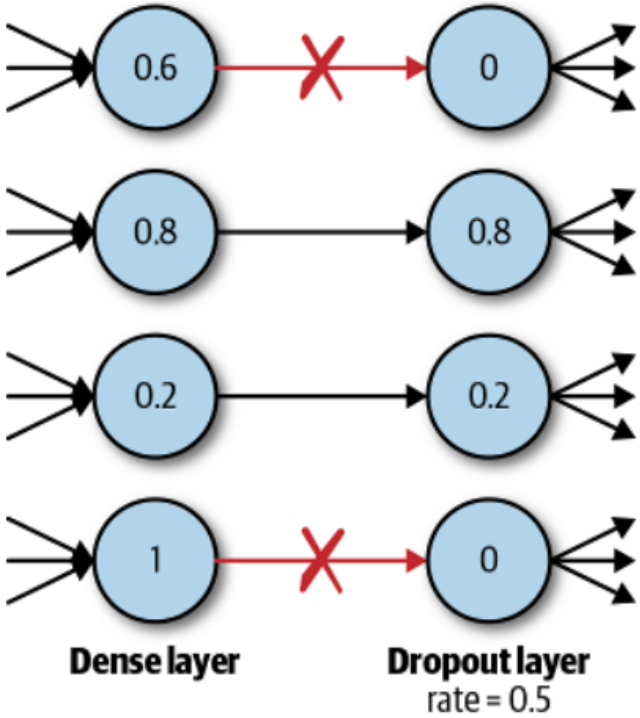
Batch Normalization (cont.)

- There are then two learned parameters for each channel, the scale (gamma) and shift (beta).
- The output is simply the normalized input, scaled by gamma and shifted by beta.
- We can place batch normalization layers after dense or convolutional layers to normalize the output from those layers.
- During test time: we may only want to predict a single observation, so there is no batch over which to take averages $\rightarrow$ during training a batch normalization layer also calculates the moving average of the mean and standard deviation of each channel and stores this value as part of the layer to use at test time.

Dropout

- **Overfitting**: Occurs if an algorithm performs well on the training dataset, but not the test dataset.
- To counteract this problem, we use **regularization** techniques, which ensure that the model is penalized if it starts to overfit.
- Commonly is done by using dropout layers: During training, each dropout layer chooses a random set of units from the preceding layer and sets their output to zero.

Dropout (cont.)



- Network doesn't become overdependent on certain units or groups of units and therefore knowledge is more evenly spread across the whole network.
- At test time, the dropout layer doesn't drop any units, so that the full network is used to make predictions.

Dropout (cont.)

- Dropout layers are used most commonly after Dense layers since these are most prone to overfitting due to the higher number of weights.

Bibliography

David Foster: Generative Deep Learning: Teaching Machines To Paint, Write, Compose, and Play, 2nd Edition, O'Reilly Media, **Chapter 2 (Deep Learning)**