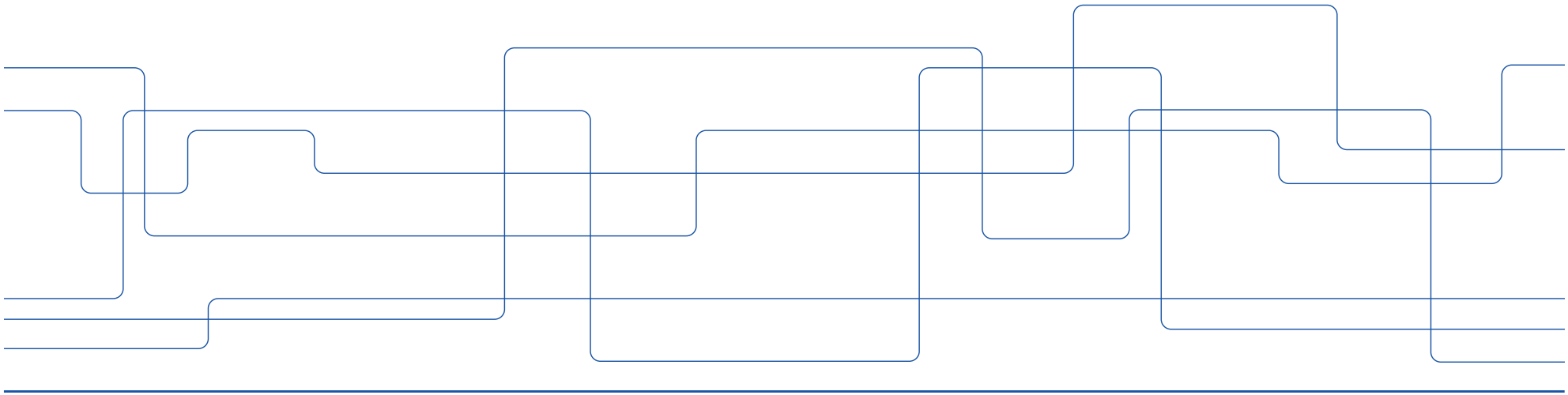
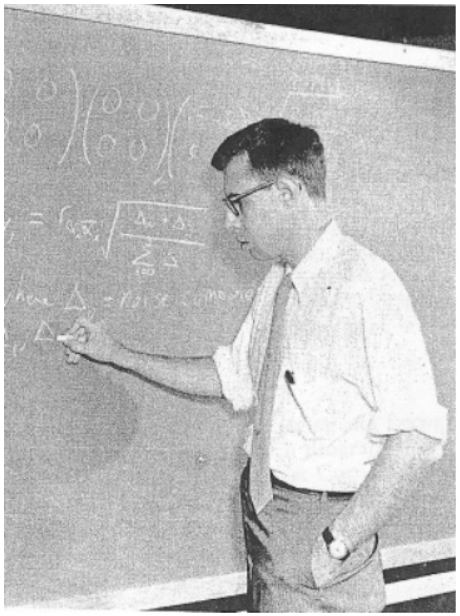
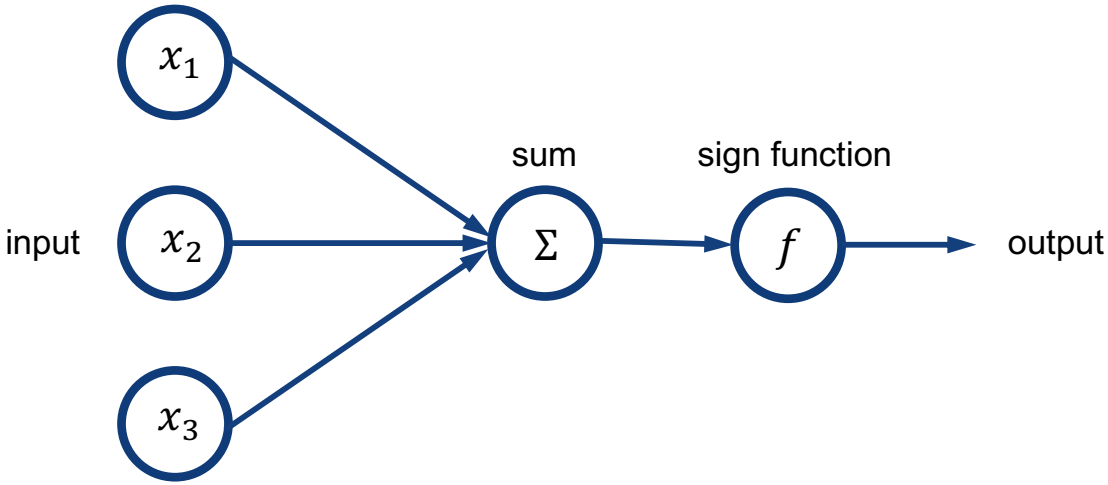
Deep networks for computer vision

Mårten Björkman



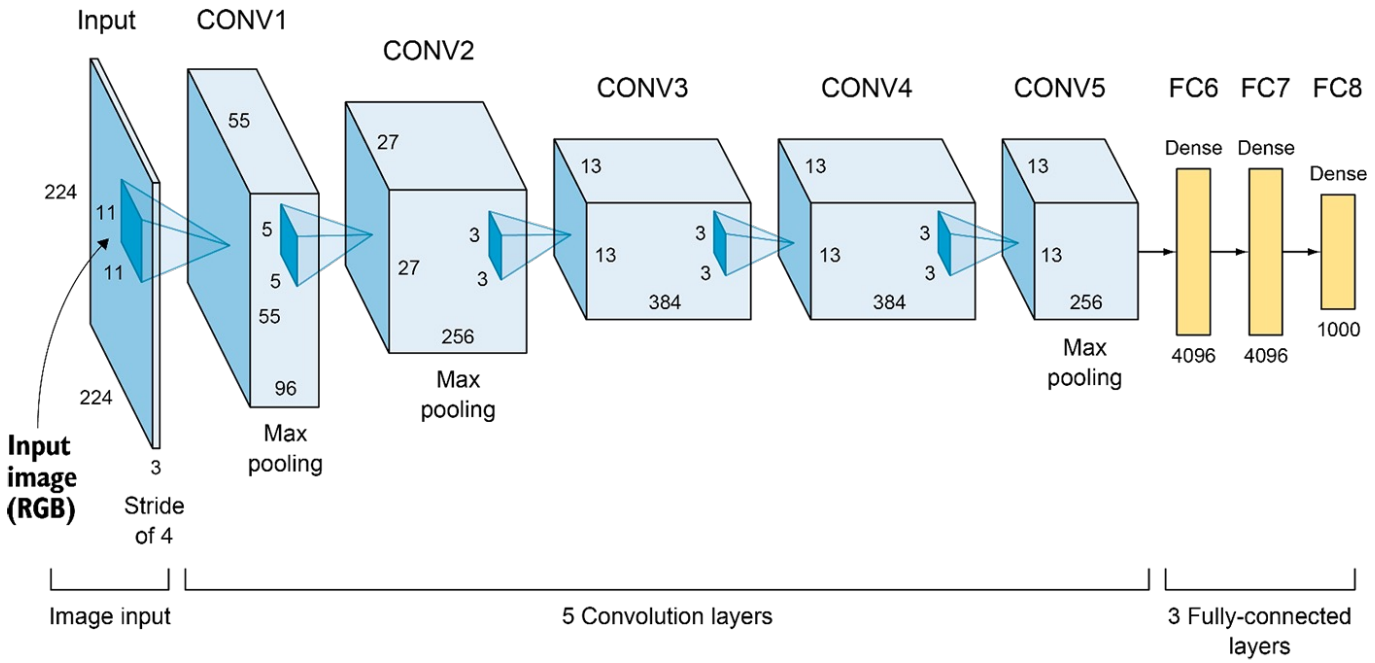
The Perceptron (Rosenblatt 1958)





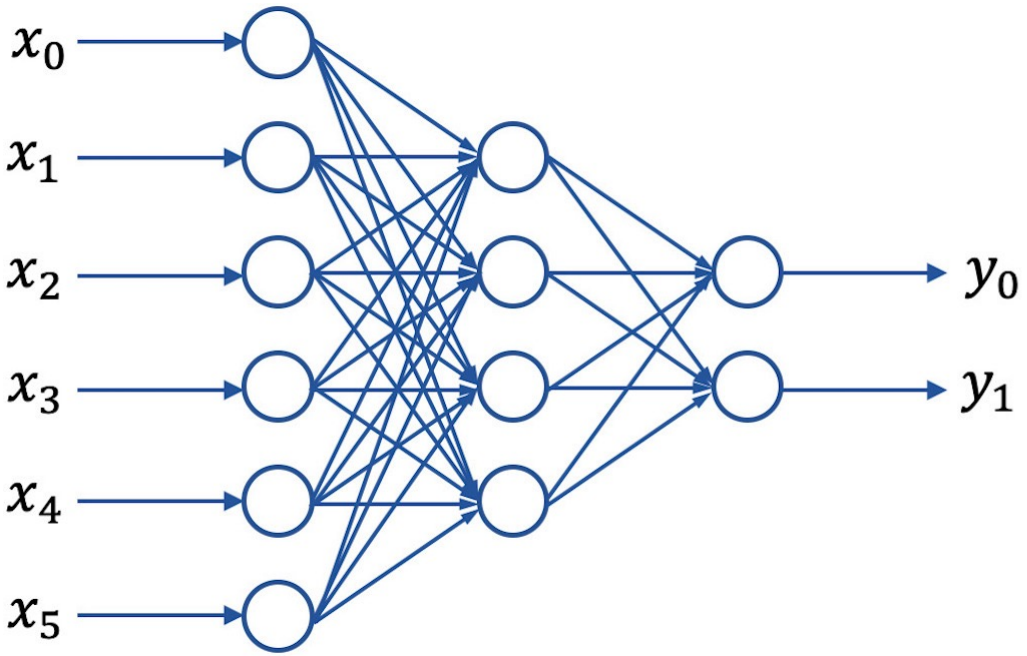
- Soon shown to be very limited in number of functions that can be implemented.
- Revival in 1986 due to multi-layer perceptron (MLP) and backpropagation.
- In 2000 more or less dead. Other areas of machine learning developed quickly.

AlexNet (Krizhevsky et al, 2012)



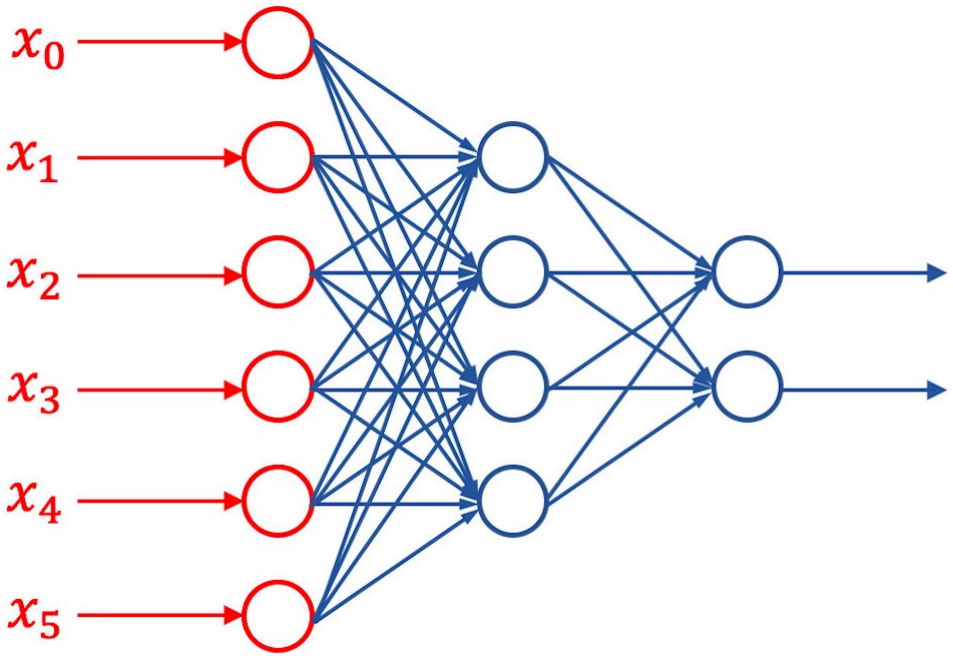
- Major breakthrough: 15.3% Top-5 error on ILSVRC2012 (Second best: 25.7%)
- Restarted the interest in neural networks, which has not stopped since then.

Fully-connected neural networks (FCN)



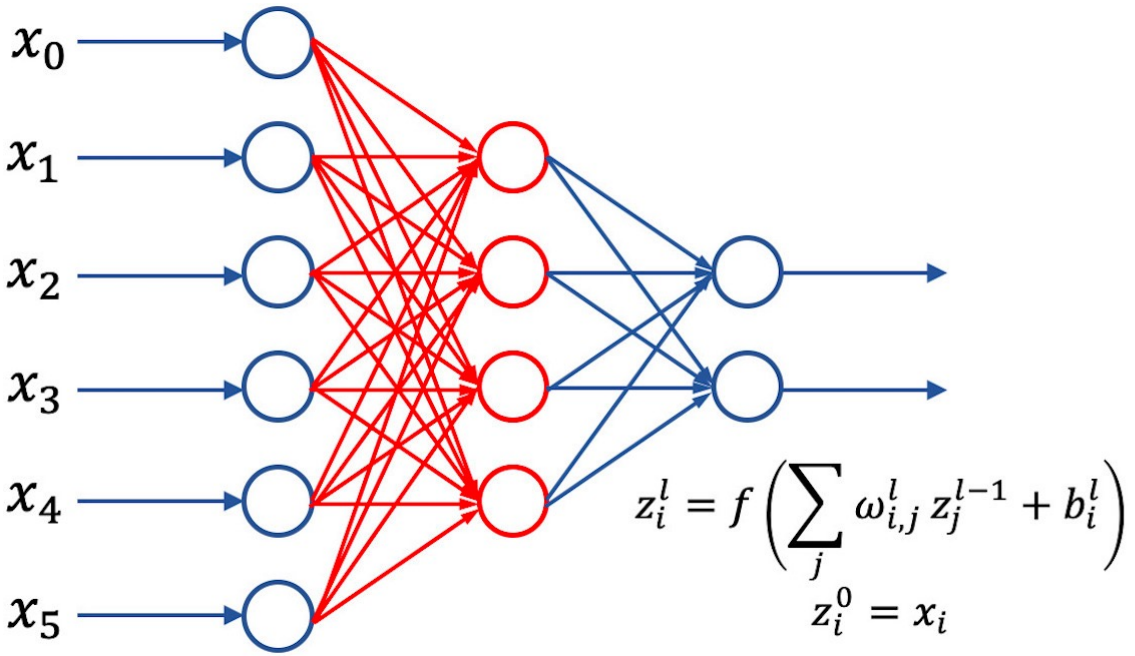
- Assume given: Many sets of training samples with matching inputs $\{x_0, \dots, x_5\}$ and expected outputs $\{y_0, y_1\}$.

FCN training (forward pass)



- Take the input values of a particular training sample and set the input neurons x_i to these values.

FCN training (forward pass)

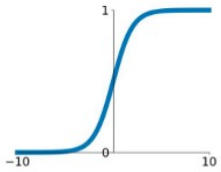


- Compute a weighted sum of input neurons $z_i^0 = x_i$, add a bias b_i^0 and apply a non-linear activation function f.

Activation functions

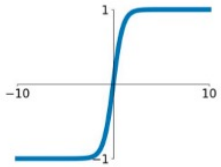
Sigmoid

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



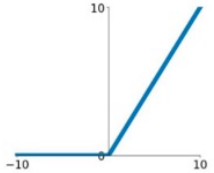
tanh

$$\tanh(x)$$



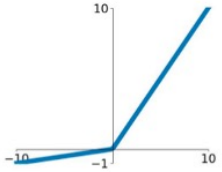
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

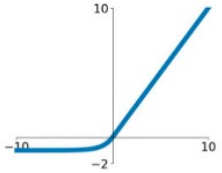


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

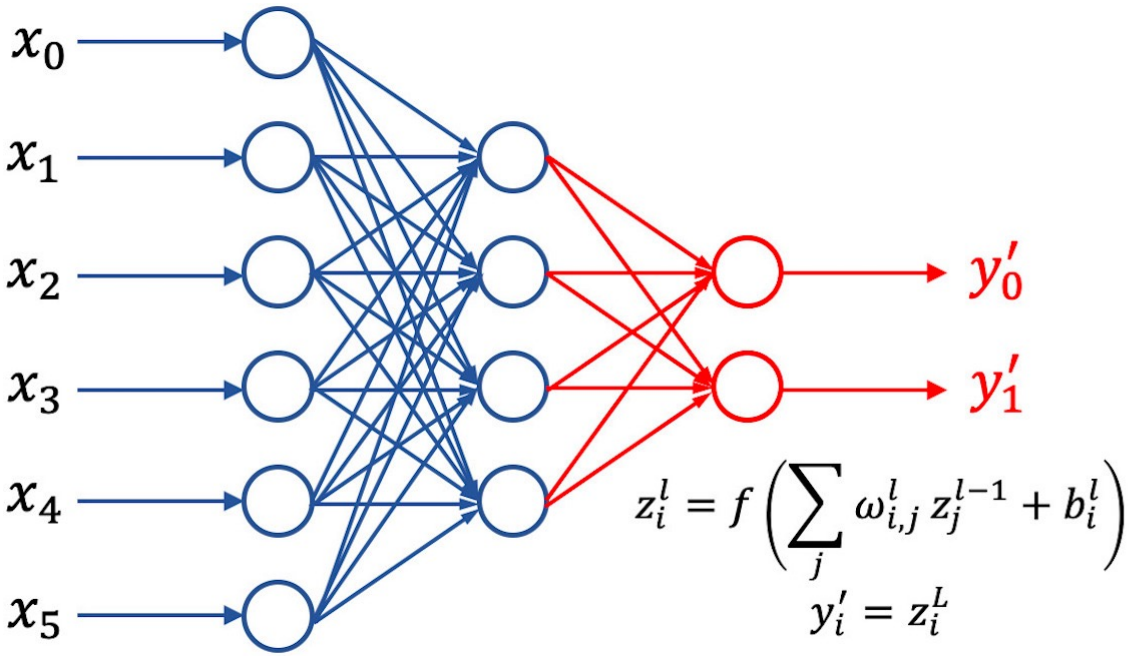
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



- ReLU is the most popular today, while Sigmoid and tanh dominated before.
- Activation functions have to be non-linear. Otherwise, multiple layers will just collapse into one layer.

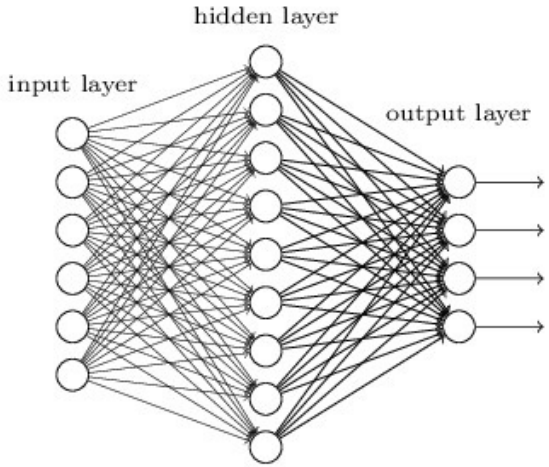
FCN training (forward pass)



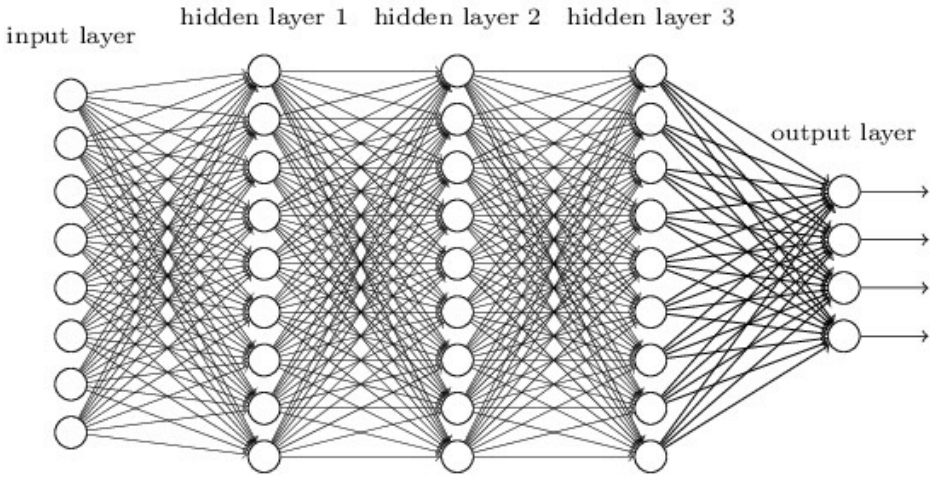
- Compute the activation of the next layer neurons, which results in a predicted output $y'_i = z_i^L$.

Fully-connected neural networks (FCN)

"Non-deep" feedforward neural network



Deep neural network

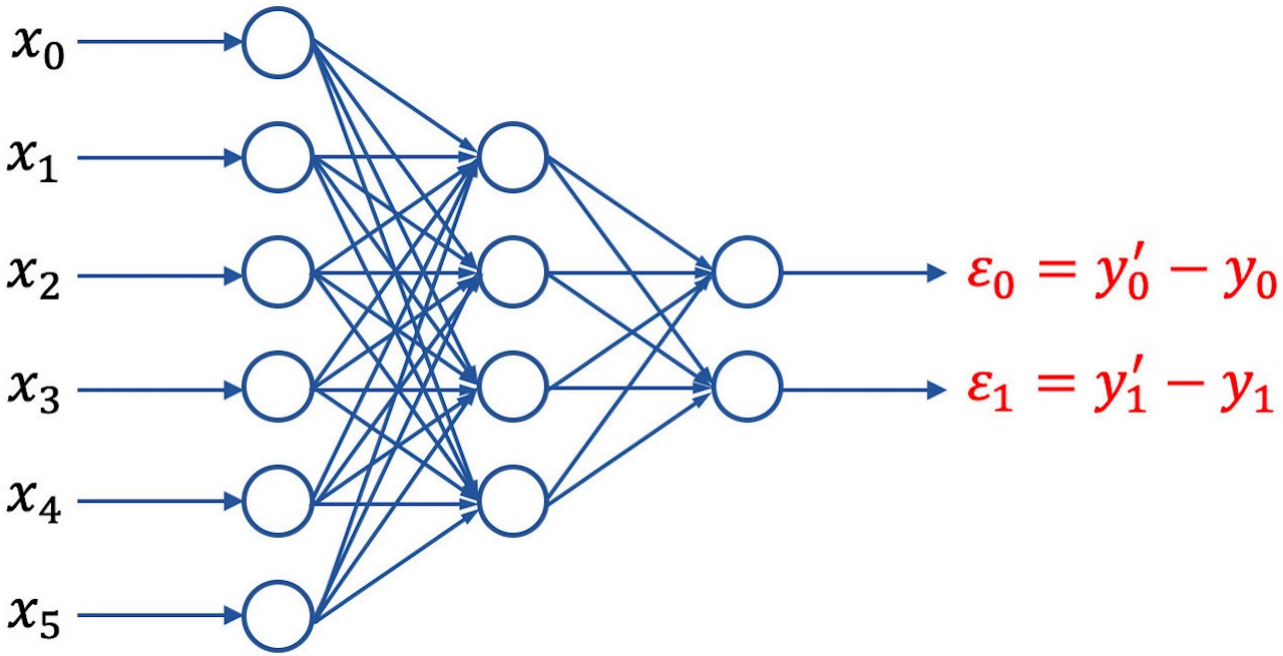


- Neurons on one layer depends on neurons from layer before

$$z^l = f(W^l z^{l-1} + b^l)$$

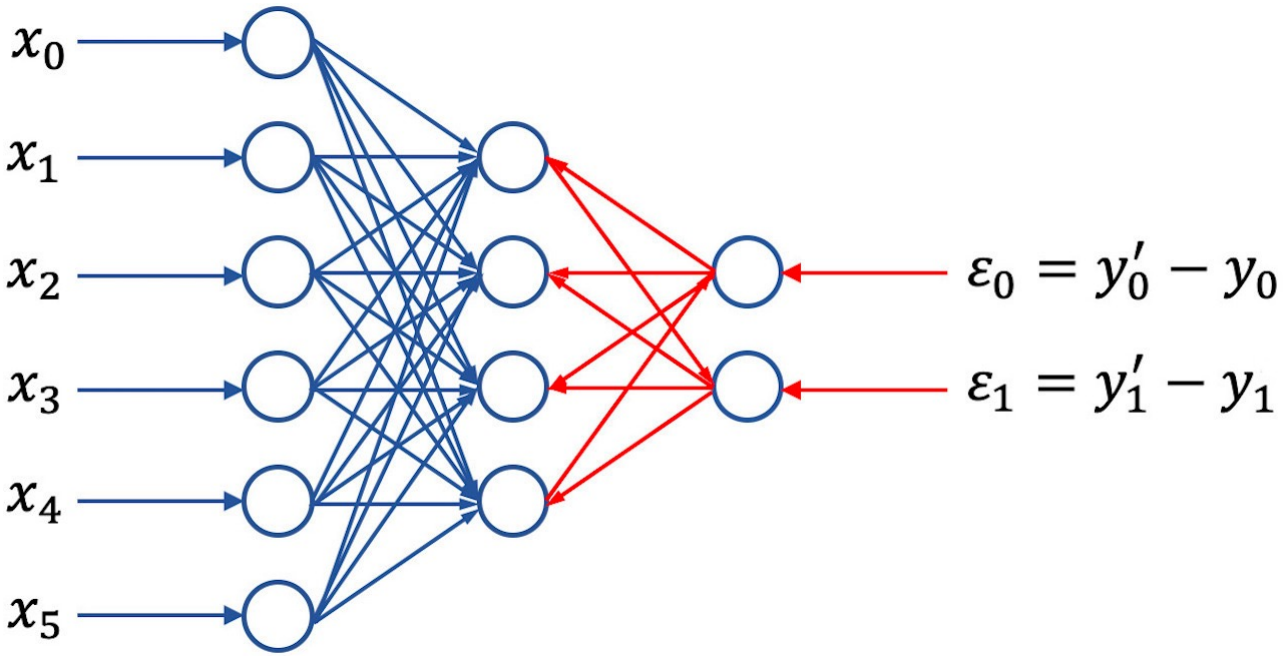
with hidden neurons $z^l, 0 < l < L$, input neurons $x = z^0$, output neurons $y = z^L$, weight matrix W^l , bias vector b^l, and activation function f.

FCN training (backward pass)



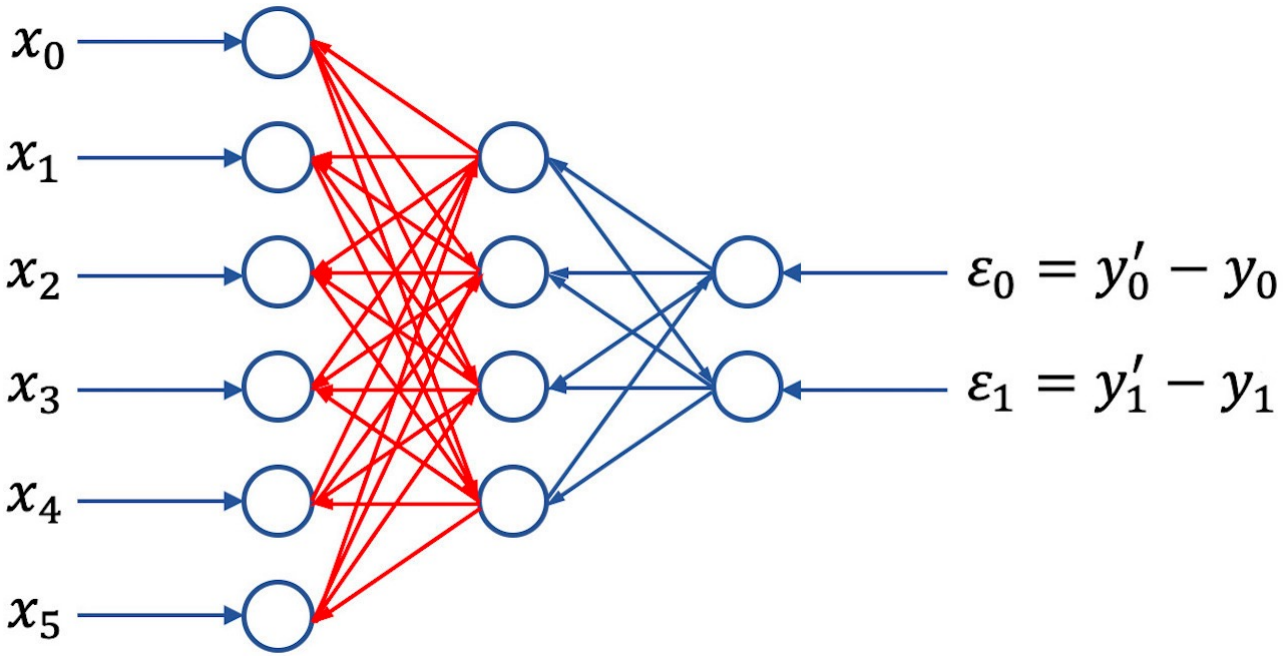
- Typically, there is an error ε, the difference between predicted y' and expected output y. This error is propagated backwards to updates the weights W^l.

FCN training (backward pass)



- Propagate the error ε backwards and adjust the weights W^l and biases b^l along the way.

FCN training (backward pass)



- With gradient descent weights and biases are adjusted so that average errors will gradually decrease.

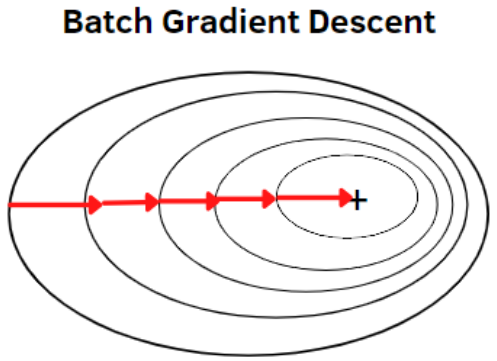
Gradient descent

- Trainable parameters θ are all weights and biases, $\theta = \{W^1, b^1, W^2, b^2, \dots\}$.
- The error is measured in terms of a loss function, e.g. $\mathcal{L}(y', y) = \frac{1}{2}\sum_i (y'_i - y_i)^2$
- The gradient $\frac{\delta \mathcal{L}}{\delta \theta}$ with respect to the parameters θ, is the direction of steepest ascent. If you step in the opposite direction, with a small step size η (learning rate), the loss will probably decrease and θ moves towards a local minimum.

- This leads to a gradient descent step:

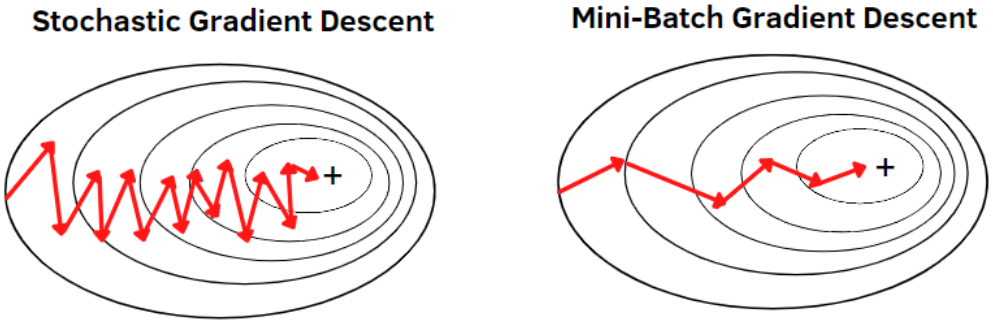
$$\theta \leftarrow \theta - \eta \frac{\delta \mathcal{L}}{\delta \theta}$$

- However,
  - If η is too large, we may miss the minimum.
  - If η is too small, convergence might be very slow.



Stochastic gradient descent

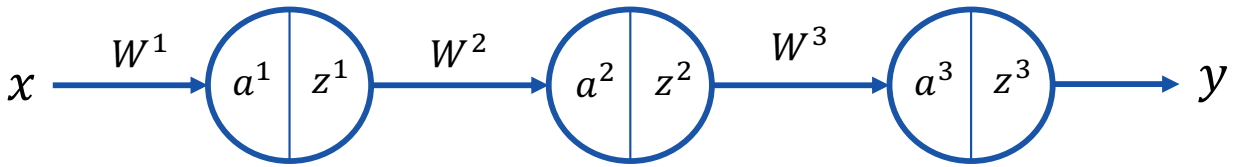
- To get the nice behaviour on previous slide, the update is used using the sum of losses for all training images. This is unnecessarily slow.
- The other extreme is to pick random images and then update once for each such image, but then the gradients will be very stochastic.



- A better solution is to use mini-batches, sum up losses from e.g. 32 random images and then update once.
- Stochasticity is good, since you can jump from a poor local minimum to a nearby better one.

Backpropagation

For each layer we have $z^l = f(a^l)$, where $a^l = W^l z^{l-1} + b^l$ is the pre-activation.



Using the chain rule, the derivative with respect to the weights of the third layer becomes

$$\frac{d\mathcal{L}}{dW^3} = \frac{da^3}{dW^3}\frac{dz^3}{da^3}\frac{d\mathcal{L}}{dy}$$

The derivative with respect to the weights of the second layer becomes

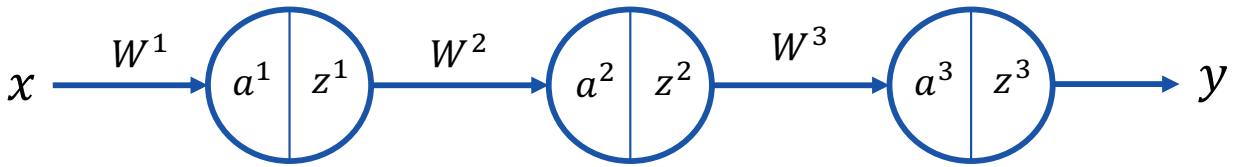
$$\frac{d\mathcal{L}}{dW^2} = \frac{da^2}{dW^2}\frac{dz^2}{da^2}\frac{da^3}{dz^2}\frac{dz^3}{da^3}\frac{d\mathcal{L}}{dy}$$

The derivative with respect to the weights of the first layer becomes

$$\frac{d\mathcal{L}}{dW^1} = \frac{da^1}{dW^1}\frac{dz^1}{da^1}\frac{da^2}{dz^1}\frac{dz^2}{da^2}\frac{da^3}{dz^2}\frac{dz^3}{da^3}\frac{d\mathcal{L}}{dy}$$

Backpropagation

For each layer we have $z^l = f(a^l)$, where $a^l = W^l z^{l-1} + b^l$ is the pre-activation.



Using the chain rule, the derivative with respect to the weights of the third layer becomes

$$\frac{d\mathcal{L}}{dW^3} = \frac{da^3}{dW^3}\frac{dz^3}{da^3}\frac{d\mathcal{L}}{dy} = z^2 f'(a^3)\,(y' - y)$$

The derivative with respect to the weights of the second layer becomes

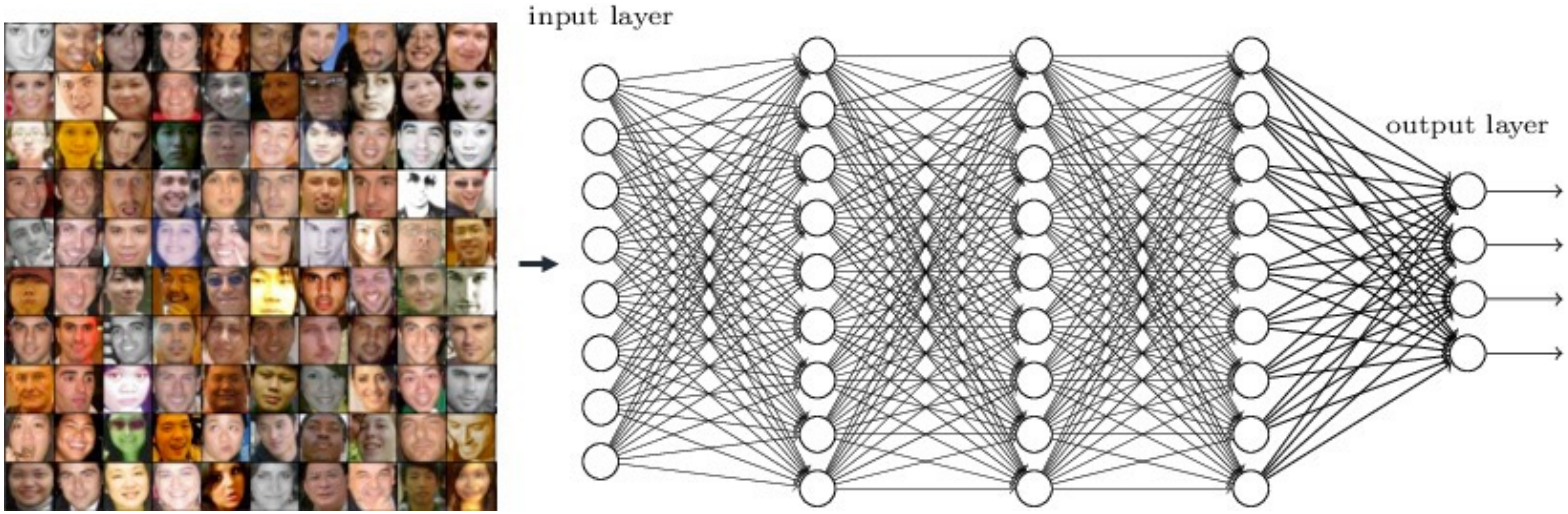
$$\frac{d\mathcal{L}}{dW^2} = \frac{da^2}{dW^2}\frac{dz^2}{da^2}\frac{da^3}{dz^2}\frac{dz^3}{da^3}\frac{d\mathcal{L}}{dy} = z^1 f'(a^2)(W^3)^T f'(a^3)(y' - y)$$

The derivative with respect to the weights of the first layer becomes

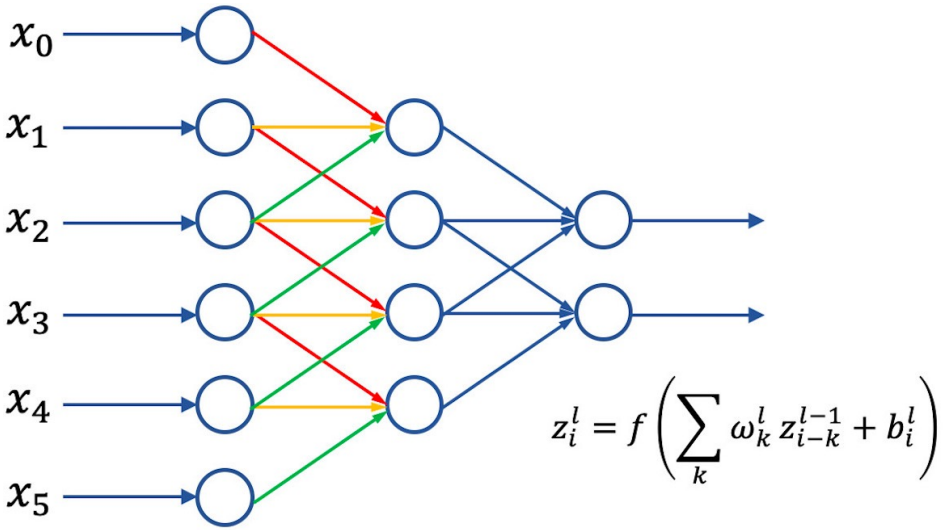
$$\frac{d\mathcal{L}}{dW^1} = \frac{da^1}{dW^1}\frac{dz^1}{da^1}\frac{da^2}{dz^1}\frac{dz^2}{da^2}\frac{da^3}{dz^2}\frac{dz^3}{da^3}\frac{d\mathcal{L}}{dy} = x f'(a^1)(W^2)^T f'(a^2)(W^3)^T f'(a^3)(y' - y)$$

Fully-connected neural networks (FCN)

- Problem: If you have one input neuron per pixel, the number of connections with unknown weights will be extremely large.
- Thus, fully-connected networks are only used for small images, e.g. characters.

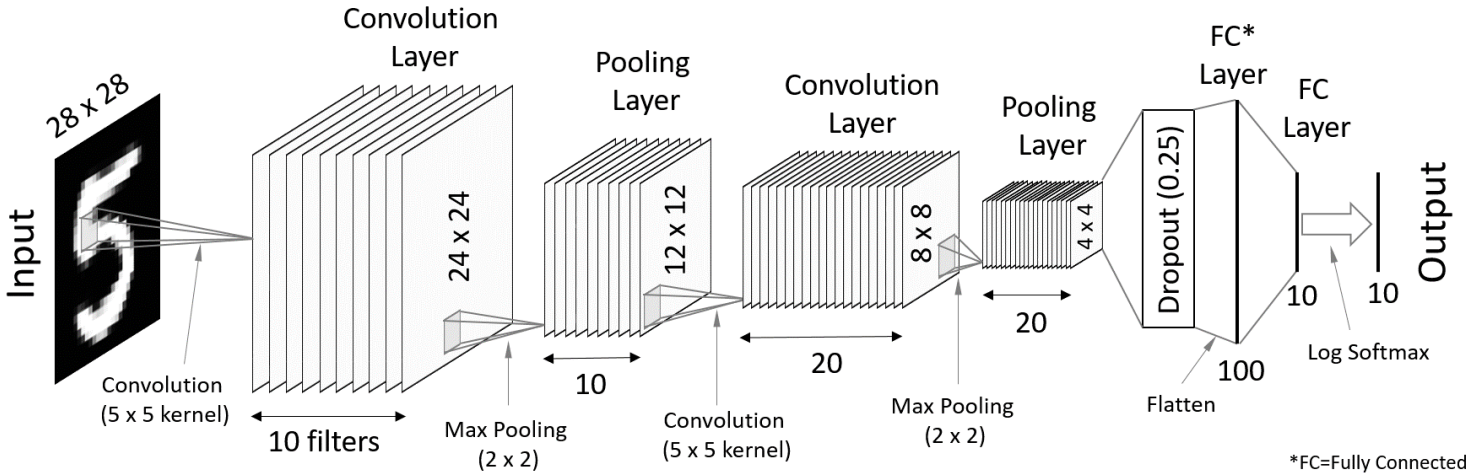


Convolutional neural networks (CNN)



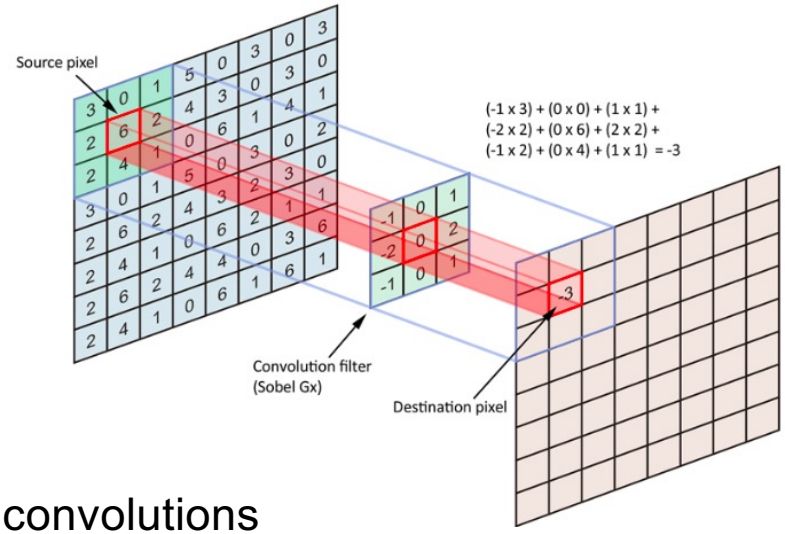
- Two important modifications:
  - Weight sharing: use the same weight for all links of similar colour.
  - Only connect to neurons in a local neighbourhood, not all neurons.
- In this case: $3 + 1 = 4$ unknown parameters, instead of $6 \times 4 + 4 = 28$.

Convolutional neural networks (CNN)



- Instead of a large weight matrix, apply multiple small local filters.
  Fewer parameters to learn $\Longrightarrow$ easier to train for images.
  ex. FCN: $28^4 = 614'656$, CNN: $5{\times}5{\times}10{\times}20 = 5'000$ parameters
- Pooling: reduce size by maximizing (or averaging) in small local windows.
- Finish with a couple of fully-connected (FC) layers.

CNN: Convolutional layers



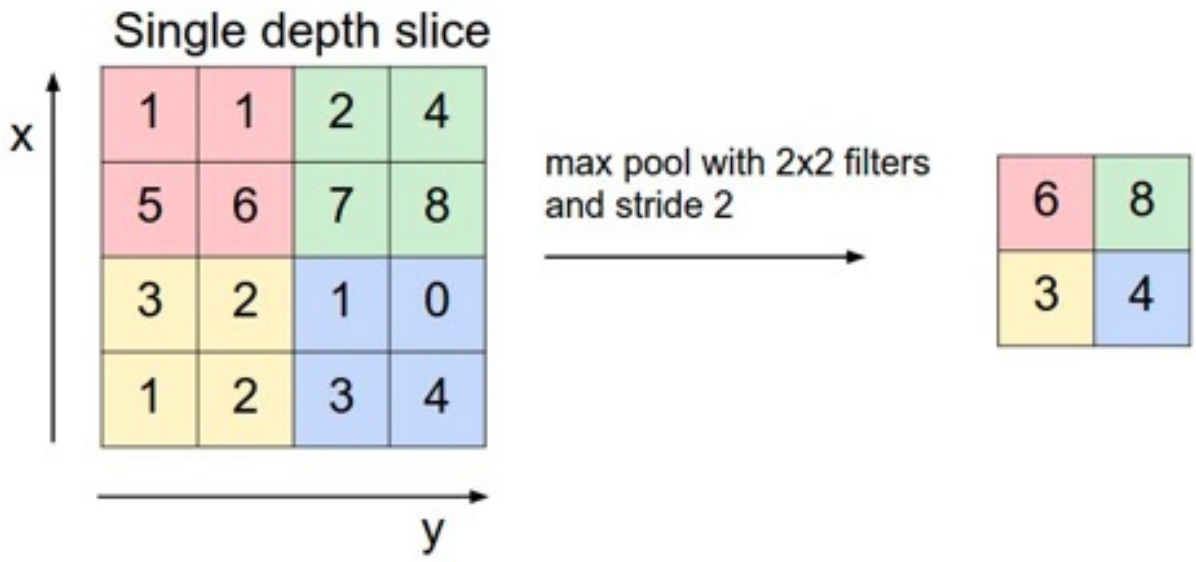
- Layers based on convolutions

$$z_{c'}^{l} = f\left(\sum_{c} W_{c,c'}^{l} * z_{c}^{l-1} + b_{c'}^{l}\right)$$

with filter kernels $W_{c,c'}^{l}$ and neurons z_{c}^{l} organized in channels c.

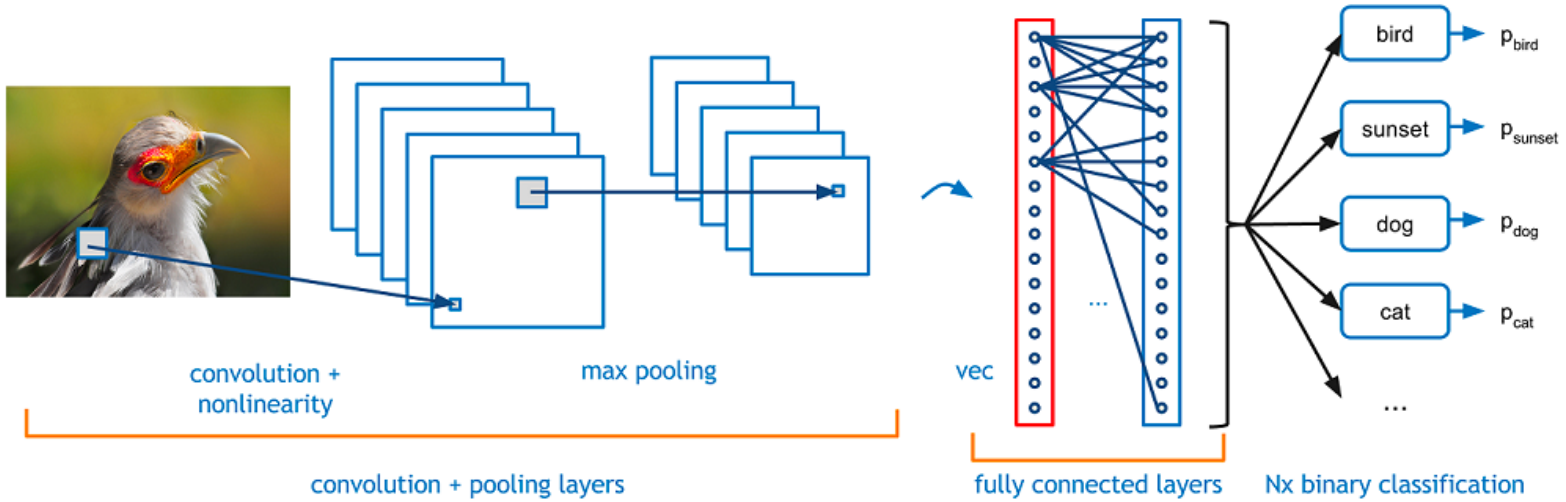
- Similar to the linear shift-invariant filters that we have seen before, but these are learned from large amounts of data, and summed up over channels.

CNN: Pooling layers



- Pooling reduces image sizes by local maximisation (or averaging).
- With an increasing number of channels for each new layer, the number of neurons would otherwise be too many.
- Gradually makes the results more and more invariant to translations.

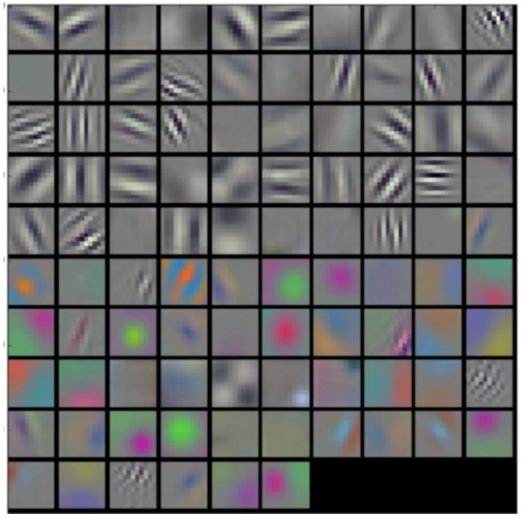
CNN: Fully-connected layers



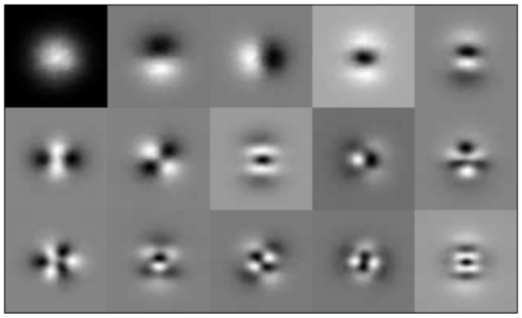
- After flattening the data into one long vector, the last layers are fully connected.
- It is in these layers the actual classification is done.
- Feature learning: Train for classification, but use results just before the fully-connected layers for something else.

Receptive fields obtained by learning

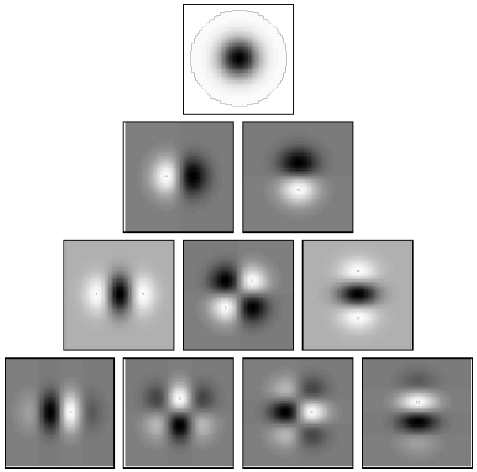
First layer filters of AlexNet



PCA of natural images



Gaussian derivatives



If you look at the learned filters of the first layers of neural networks, many of them look qualitatively similar to Gaussian derivatives.

Figures from Krizhevsky, Sutskever and Hinton, “ImageNet classification with deep convolutional neural networks”, NIPS 2021 and Hancock, Baddeley, and Smith, “The principal components of natural images”, Network: Computation in Neural Systems 3(1): 61–70, 1992.

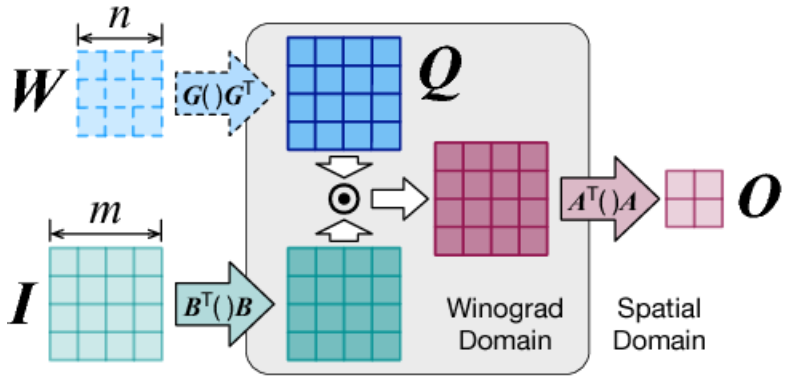
Speeding up CNN with Fourier Transform

- Convolution layers are based on convolutions

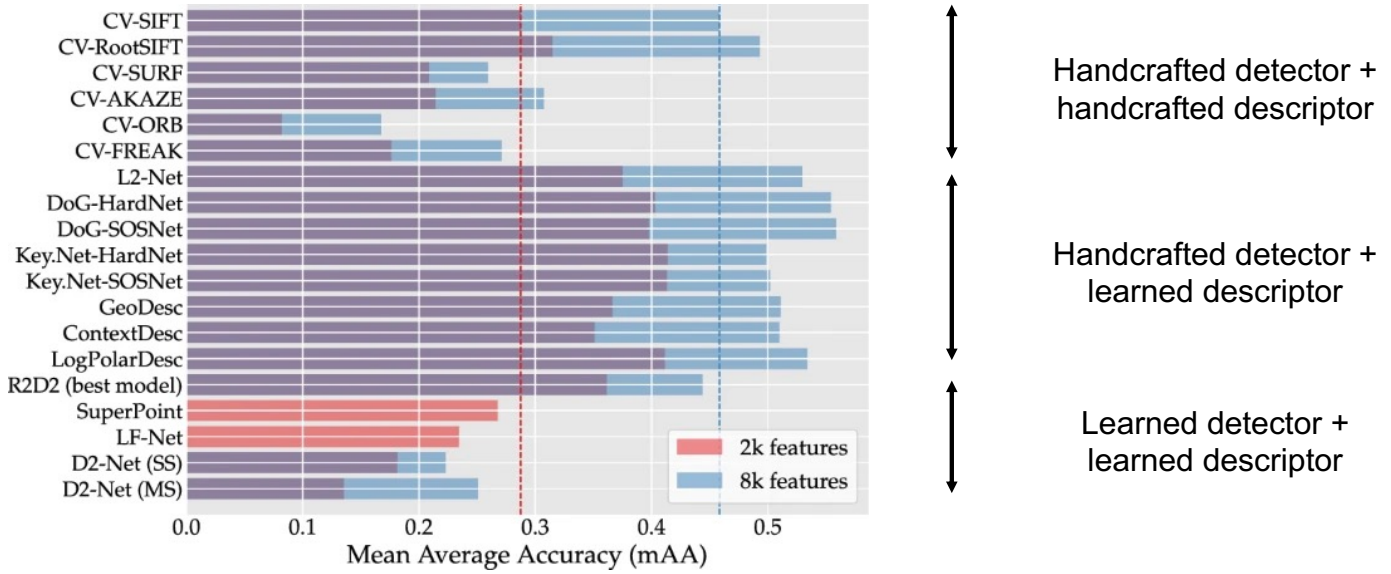
$$z_{c'}^{l} = f\left(\sum_{c} W_{c,c'}^{l} * z_{c}^{l-1} + b_{c'}^{l}\right)$$

- Computational cost is $\mathcal{O}(C'CN^2m^2)$ with image size $N \times N$, filter size $m \times m$, number of input channels C and output channels C'.
- This can be speeded up with convolution using Fourier transform (or similar transform), so that complexity becomes $\mathcal{O}(C'CN^2 log N)$.

Winograd transform
2.25× speedup



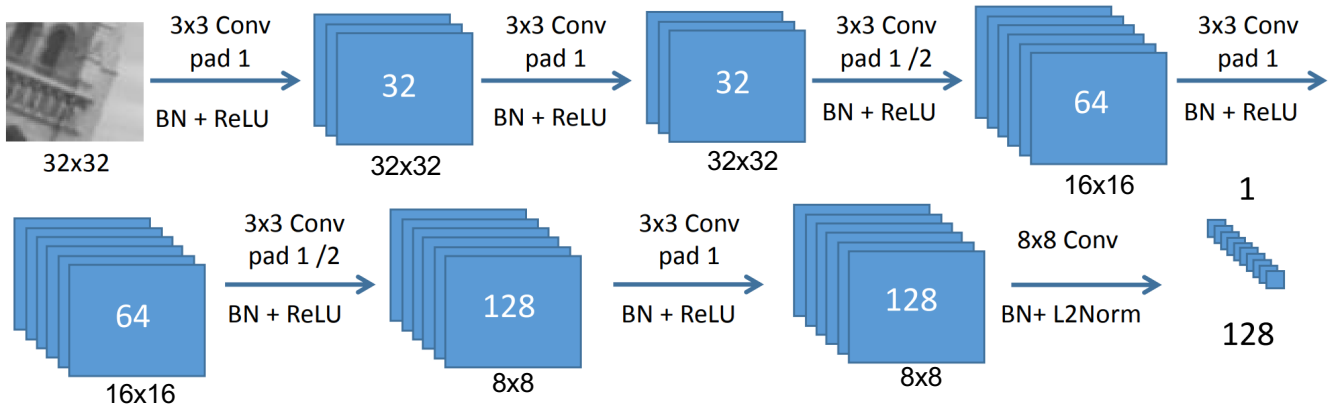
Handcrafted vs Learned image features



- Mean Average Accuracy refers to errors in estimated relative rotation and translation between two images.
- The handcrafted features are much faster than the learned ones.
- The best methods are learned, but SIFT is still much competitive.

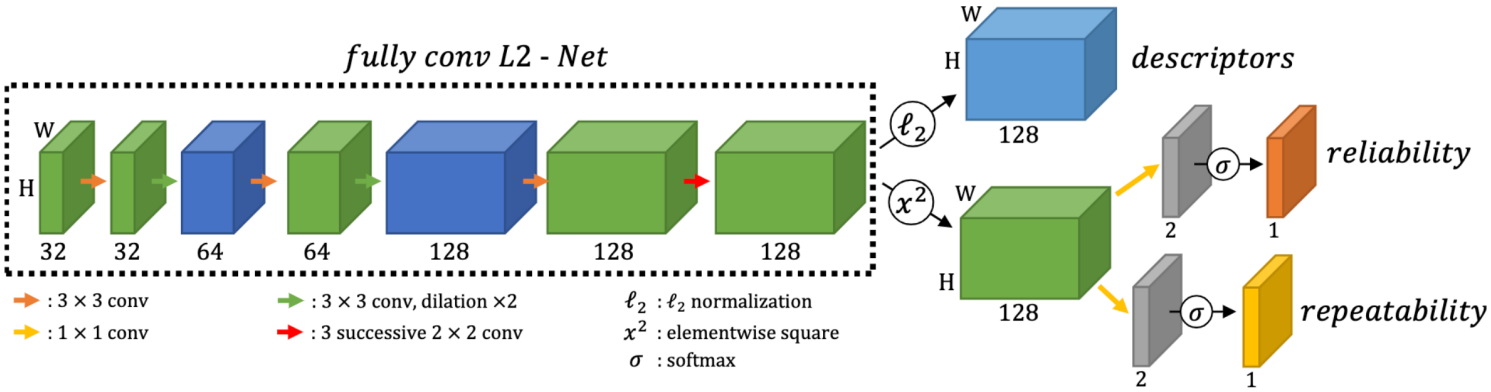
Jin et al, ”Image Matching Across Wide Baselines: From Paper to Practice”, IJCV 2020.

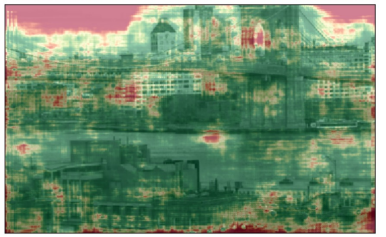
HardNet feature descriptor (Mishchuk et al, 2017)

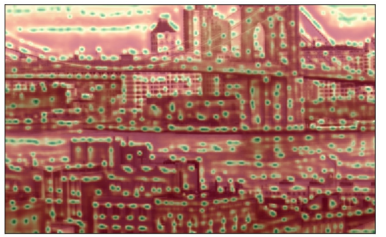


- Uses Difference-of-Gaussian (DoG) to detection feature points, just like SIFT.
- Uses a standard CNN, but subsamples twice, instead of using pooling layers.
- The network trained using sets of earlier matched patches, with a loss that
  - Minimises the difference between a patch and its correct match, while
  - Maximises the difference between a patch and its hardest incorrect match.

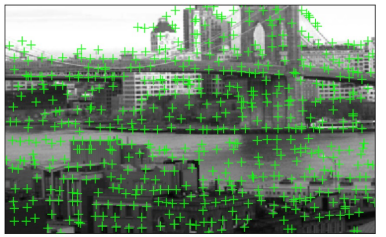
R2D2 image features (Revaud et al, 2019)



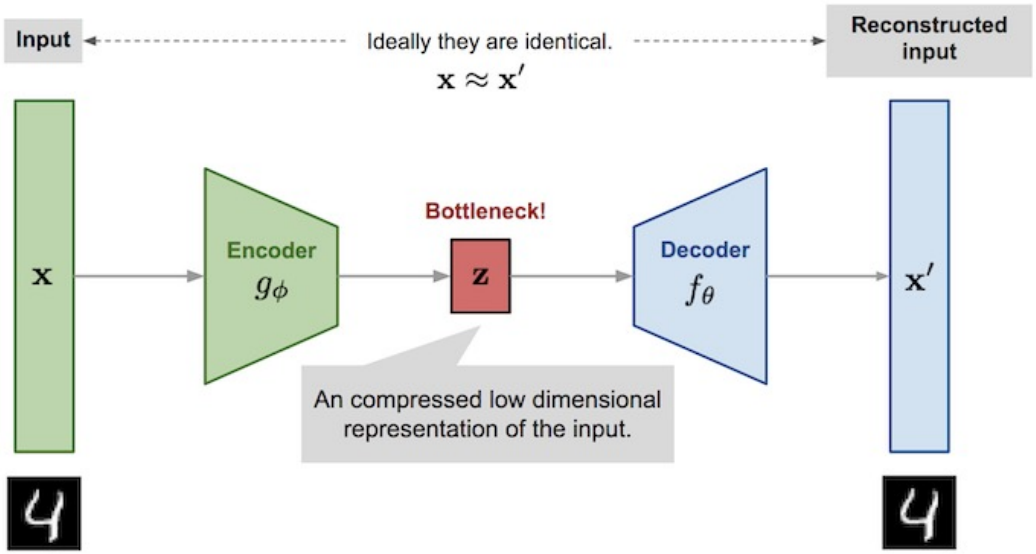




- Computes descriptors and detection scores for every image point.
- Detection scores are given by $reliability \times repeatability$
  - $reliability$ = how discriminant each local area is
  - $repeatability$ = how well local maxima overlap between images
- The training data is based on already matched images.

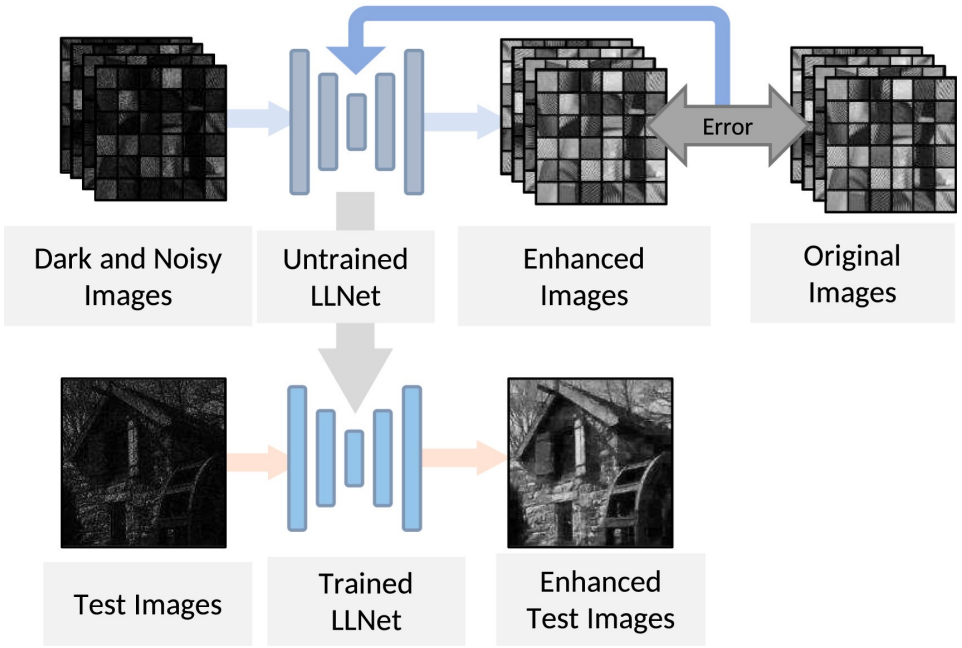


Autoencoder for representation learning



- Alternative to PCA for low-dimensional representation of images.
  - Encoder: Gradually reduce number of neurons to a small latent space.
  - Decoder: Gradually increase size to that of original image.
- Train network by using the input image as the target output, no annotation.
- Use the bottleneck neurons as a low-dimensional image representation.

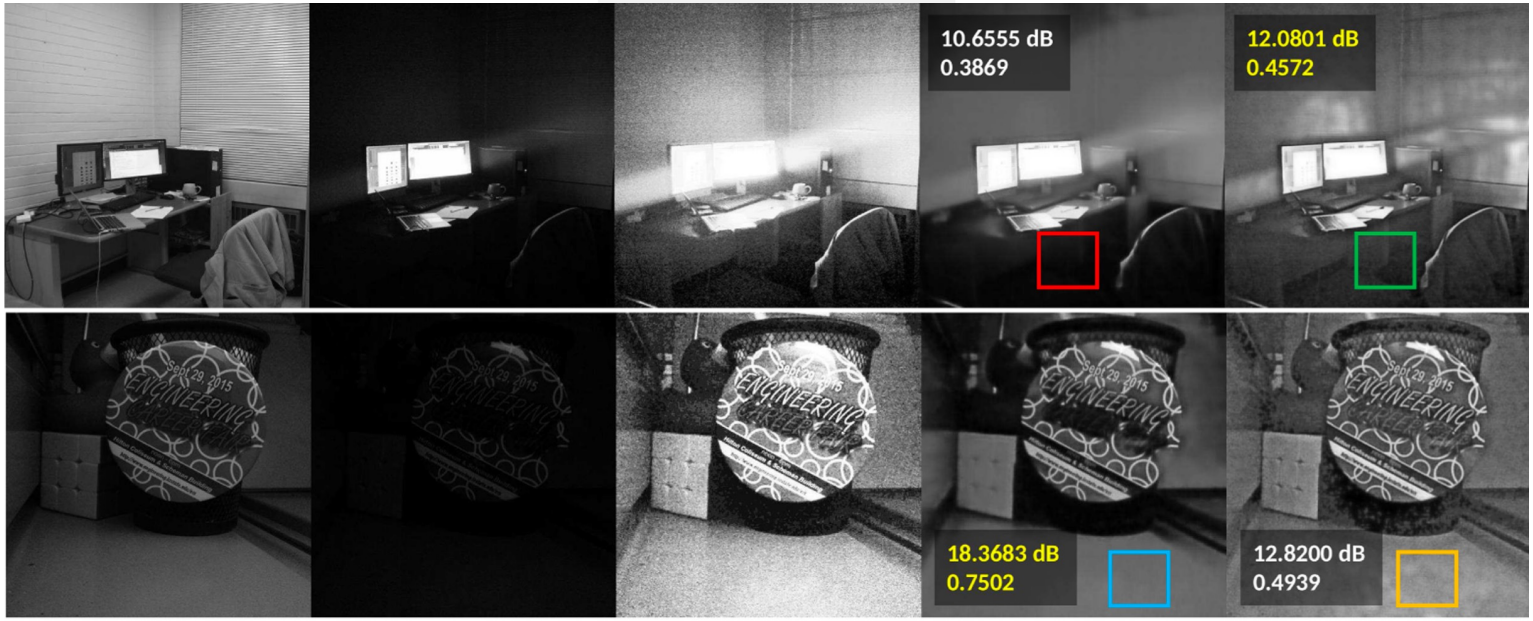
LLNet: Image enhancement with autoencoder



- Instead of constructing the noisy input image, reconstruct a noise-free version.
- Uses 17×17 pixel patches for training and full images once trained.

Lore et al, “LLNet: A deep autoencoder approach to natural low-light image enhancement”, 2017.

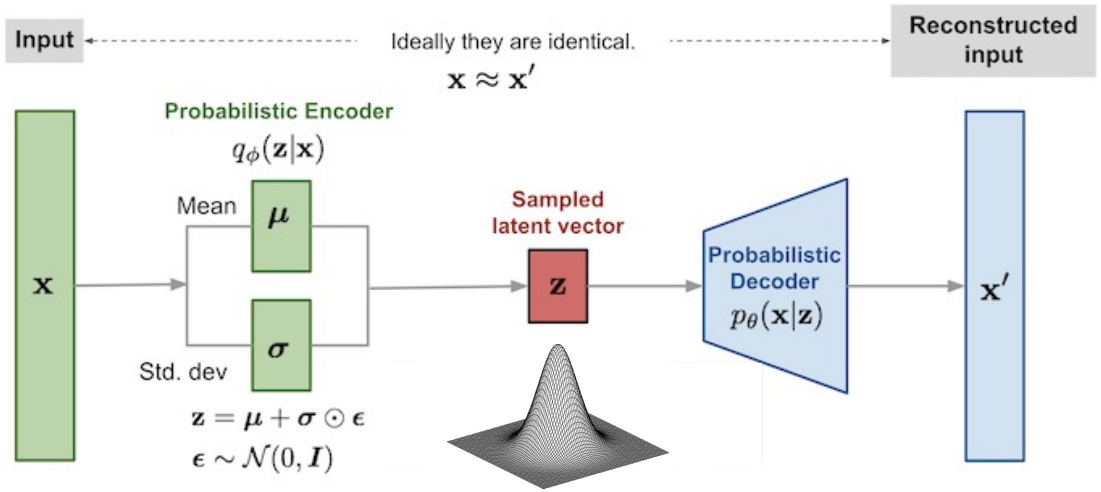
LLNet: Image enhancement with autoencoder



(i) Natural Bright | (ii) Natural Dark | (iii) Noise Structure from HE | (iv) LLNet - Gaussian | (v) LLNet - Poisson

- Results compared to histogram equalization (HE) with Gaussian or Poisson noise for generating pairs of training images.

Variational autoencoder (VAE)

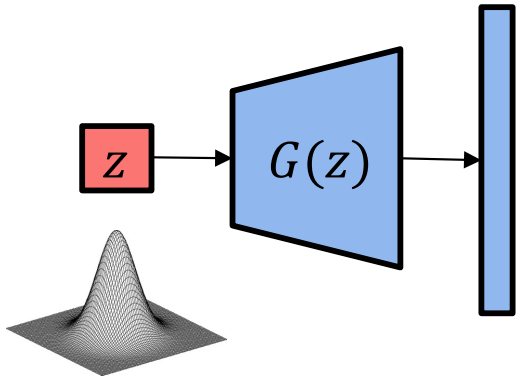


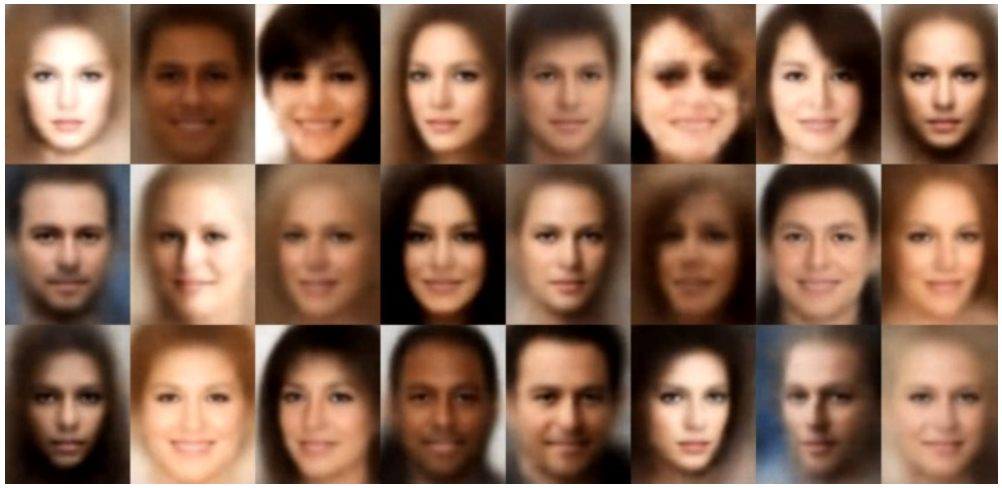
- The output of the encoder is modelled as a distribution, not just a single vector.
- During training two losses are used.
  - Reconstruction loss: forces the make the encoder reconstruct the input image.
  - Regularisation loss: forces the latent space to a given distribution, e.g. Gaussian.
- Once trained new images can be generated by disconnecting the decoder and use it as a generator by adding noise as input.

Variational autoencoder (VAE)

- Move around in the latent space and generate new images with the decoder.

VAE for image generation

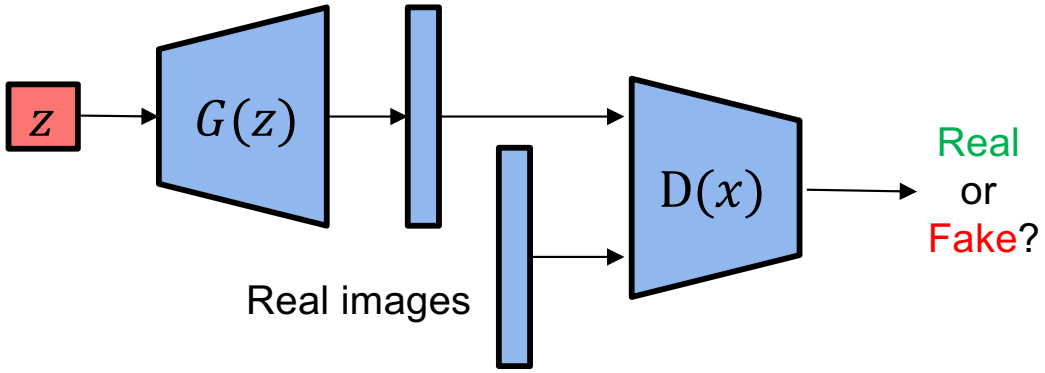




Kingma & Welling. "Auto-encoding variational bayes.", 2013.

- If you train a VAE with a large database, you can later reuse the decoder part as a Generative network, e.g. to generate images of faces from random noise.
- However, there is no guarantee that these images will look natural. Typically, from the face image it is clear the edges are quite diffuse.

Generative adversarial network (GAN)



- Use a large set of images and train an additional Discriminator network to tell the real and generated images apart.
- GANs can be trained with the following loss function:

$$\mathcal{L} = E_x\left[\log(D(x))\right] + \mathrm{E_z}\left[\log(1 - D\big(G(z)\big)\right]$$

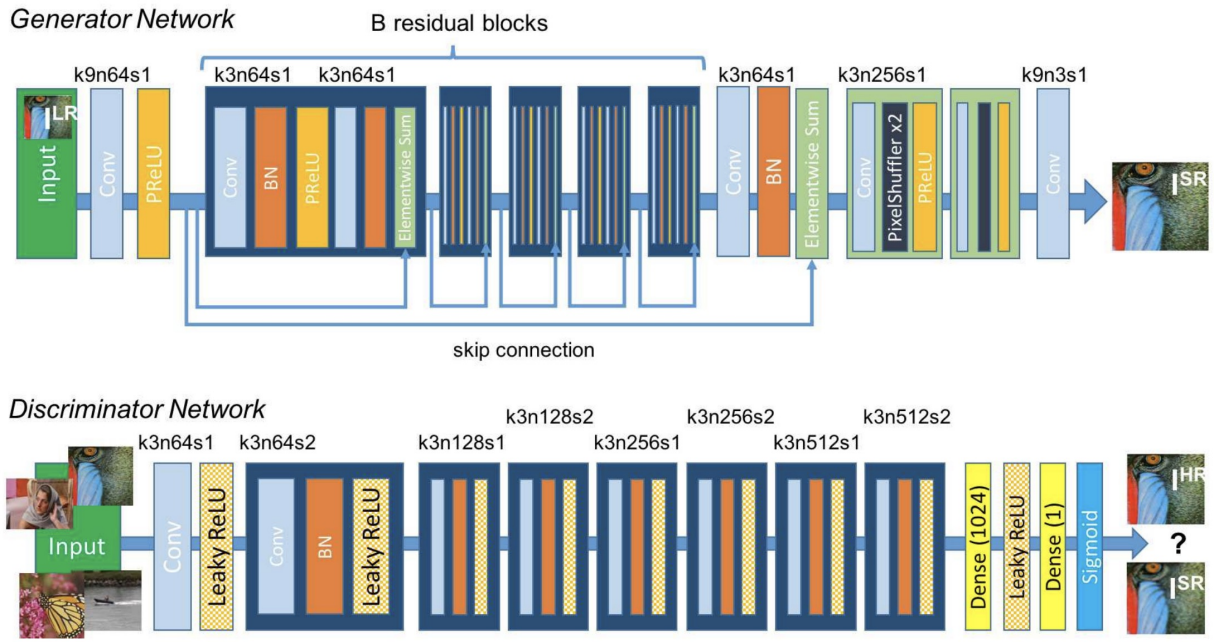
- When training has completed the generator $G(z)$ has become so good that the discriminator $D(x)$ cannot tell the difference between real or fake.

StyleGAN-XL (Sauer at al, 2022)

- One of the best GAN-based image generation networks of today.
- Training time is ”only” 400 days on a Nvidia V100.
- The images show interpolations between two different images.

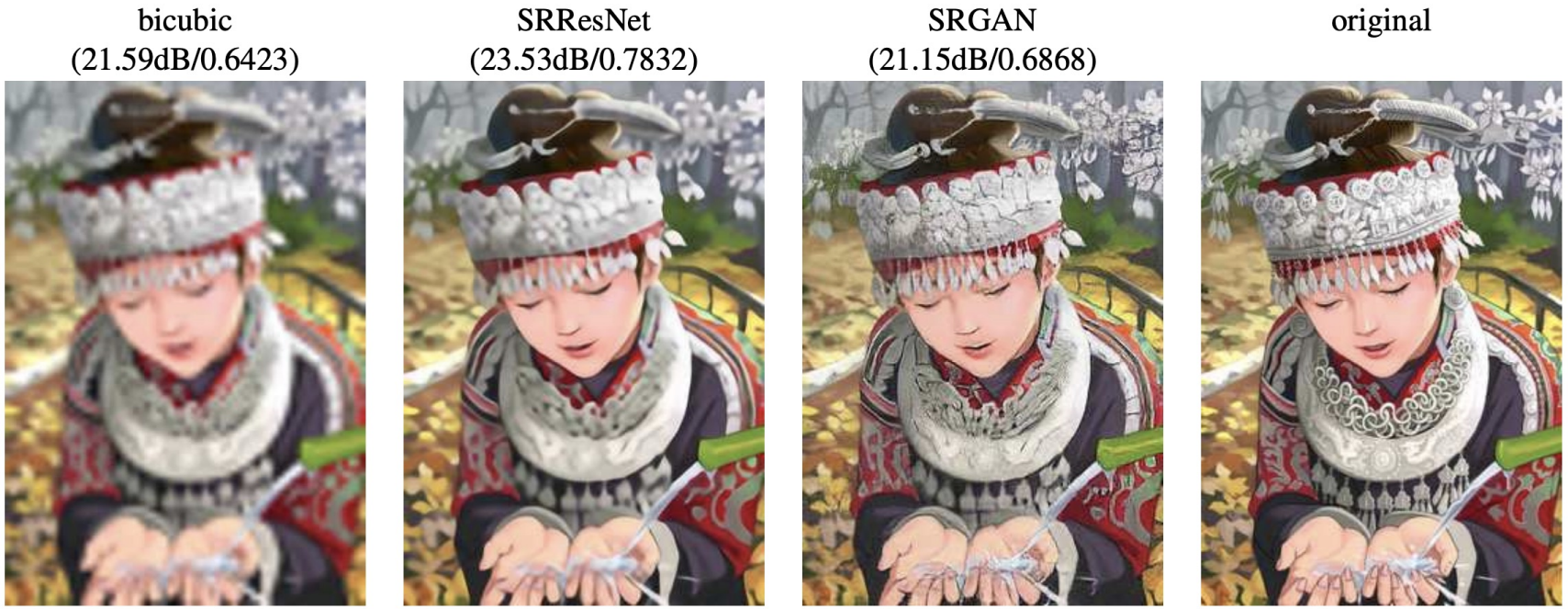
Sauer, Schwarz, and Geiger, ”StyleGAN-XL: Scaling StyleGAN to Large Diverse Datasets”, SIGGRAPH, 2022.

Super-resolution with GAN



- Generator: is trained to upscale and increase the resolution of an an image.
- Discriminator: is trained to classify whether an image is an upscaled image or an real image already in high resolution.

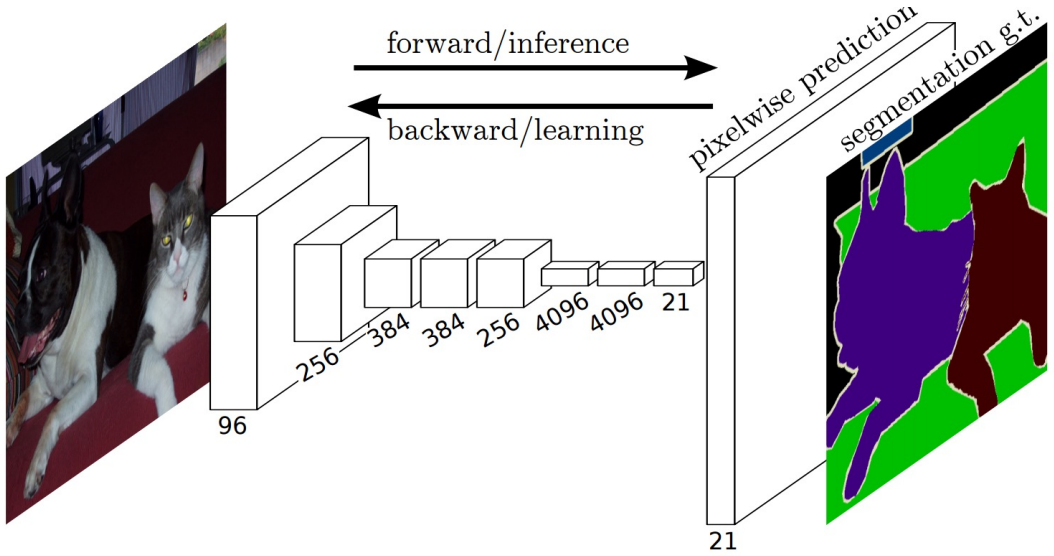
Super-resolution with GAN



- A simple bicubic filter for upsampling 4x times (left), two variants based on GANs, and the original high-resolution image (right).

  Ledig et al., ”Photo-realistic single image super-resolution using a generative adversarial network”, 2017.

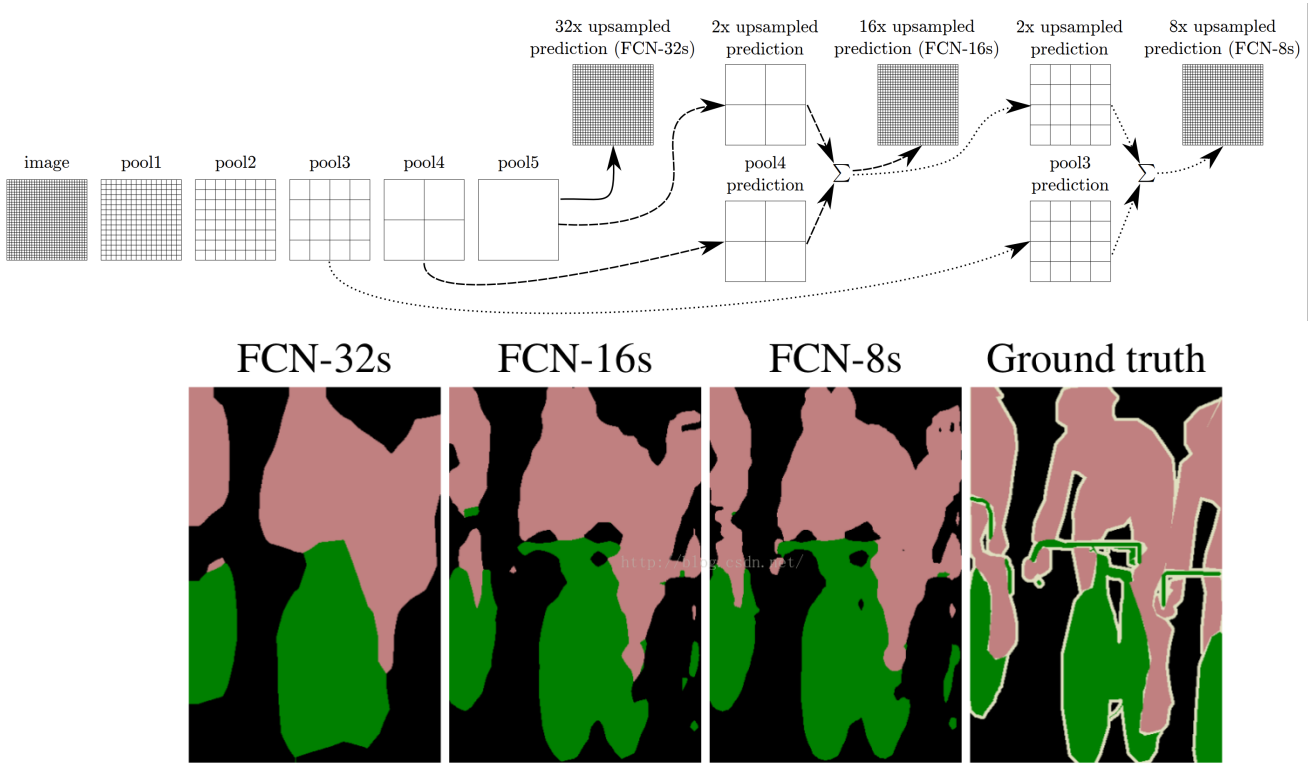
Semantic segmentation with FCN



- Among the first attempts to use deep networks for semantic segmentation.
- Train a network for classification with a set of semantic classes.
- Apply the network to many different overlapping windows in the image.
- Segment the image based on most likely class in each window.

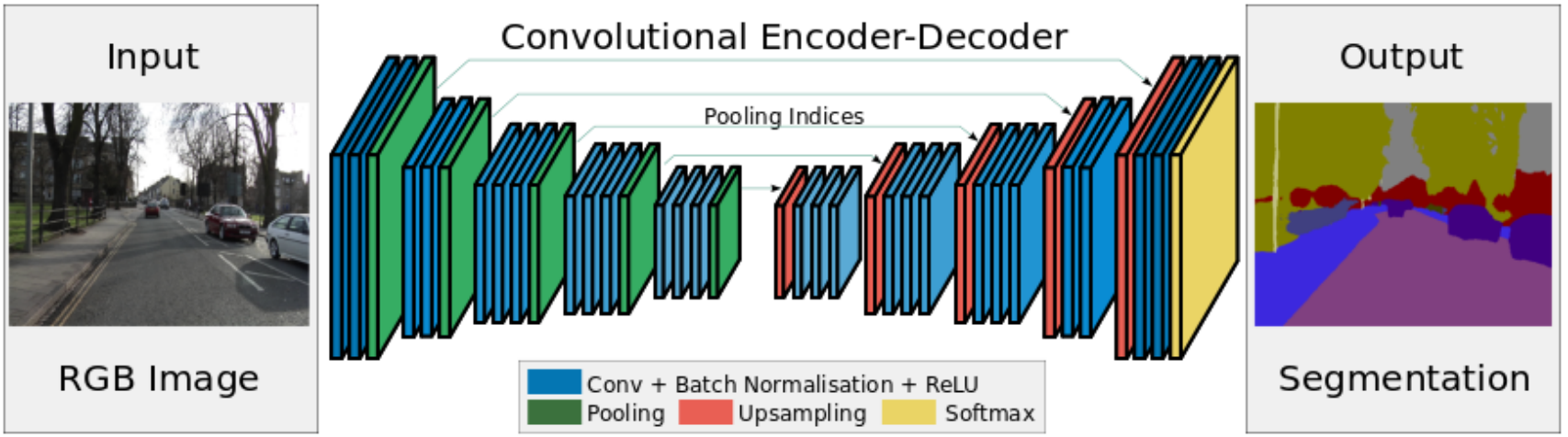
Long et al., “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015.

Semantic segmentation with FCN



For better segmentation, combine the output from multiple layers of the network.

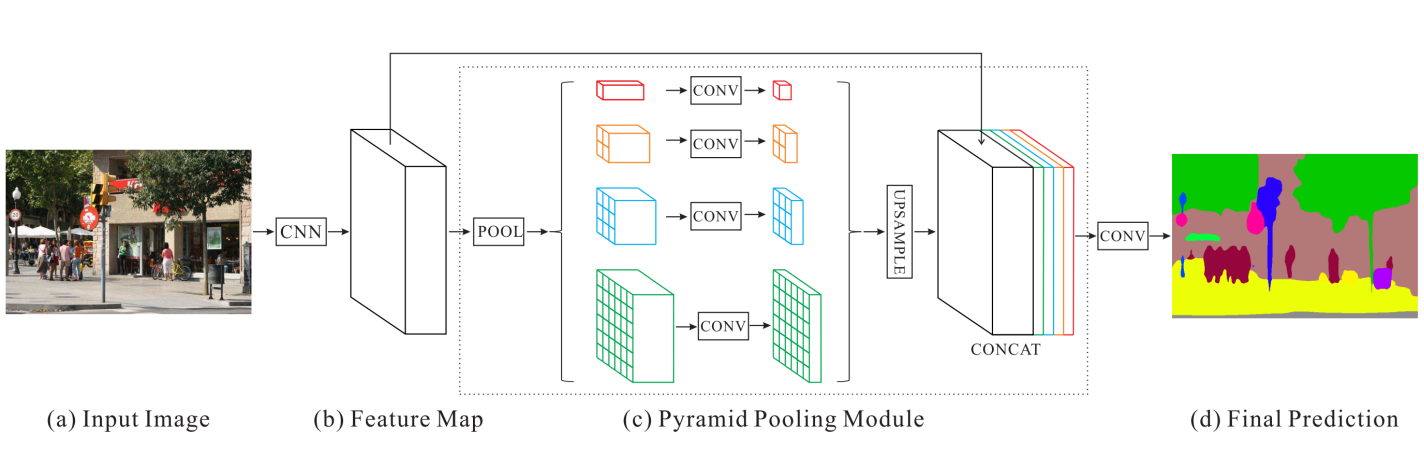
SegNet: Segmentation with encoder-decoder

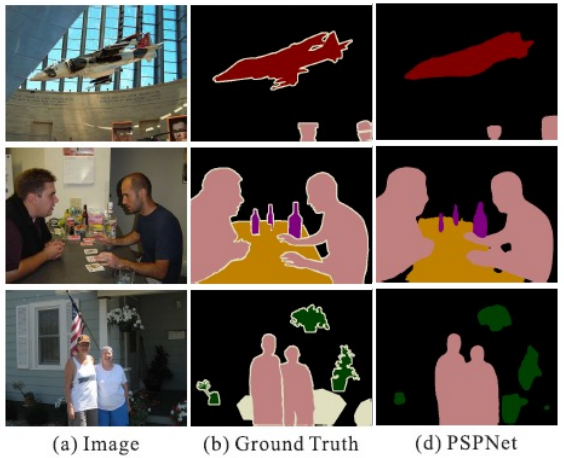


- FCN applied to many overlapping windows can be very slow.
- A faster alternative is to use an encoder-decoder network.
- In SegNet, images upsampled by propagating pooling indices.

Badrinarayanan et al., ”SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation.”, 2015.

PSPNet: Pyramid scene parsing





- This solution tries to use context around each region to improve segmentation.
  - A Pyramid Pooling Module works at different scales of a common feature map.
  - Results are upscaled and combined with the feature map to produce a segmentation.
- A lot of state-of-the-art methods today are based on the same principle.

Zhao, Hengshuang, et al. "Pyramid scene parsing network.", CVPR, 2017.

PIDNet-S (Xu et al, 2022)

Currently one of the best performing methods on the real-time Cityscapes benchmark.

Summary of good questions

- How does a perceptron work?
- Why do you need an activation function?
- What is a loss function?
- How does backpropagation work in principle?
- Why does a CNN have so many fewer parameters than a FCN has?
- What are the typical layers of a CNN?
- How do learned image descriptors perform compared to handcrafted ones?
- What is the structure of an autoencoder?
- What is the difference between a VAE and a GAN?
- What can GANs be used for?
- Mention an innovation that has affected modern networks for segmentation.

Recommended reading

- Szeliski: Chapters 5.3, 5.4.1, 5.5.4