Deep Learning; U-Nets; GANs

Today

- Deep Learning: History of significant ideas and advances
- More recent problems/architectures
  - Semantic image segmentation
  - Generative adversarial networks

Large Neural Networks

- CNNs have been around since the 1990s, autoencoders and RNNs since the 1980s,
- For many years, neural networks were not considered state of the art for many prediction tasks
- Larger neural networks needed for greater accuracy, but they can be difficult to train
  - Vanishing gradients problem (large gradients at output layer are reduced to small gradients at early layers)
  - More (bad) local minima, backprop gets stuck
  - Not enough data to prevent overfitting
  - Insufficient computational resources
  - Missing "know-how"
- Then neural networks experience a renaissance that began around 2006

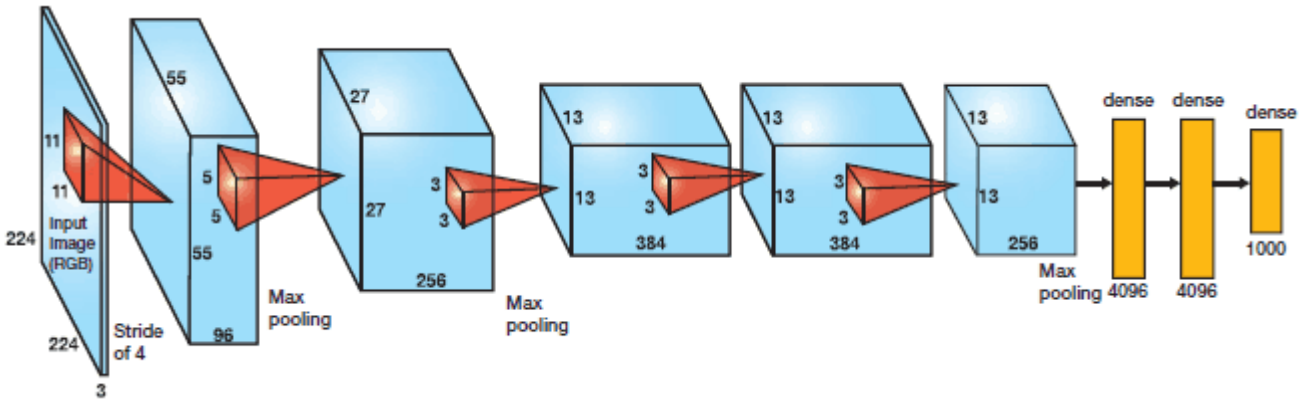
Deep Learning

Enabled by

- Graphics processing units (GPUs): parallel floating-point calculators with 100s-1000s of cores
- Large, public databases such as ImageNet (Fei Fei Li, 2009) which has over 14 million *labeled* examples and 20 thousand classes of objects
- New training strategies
  - Dropout
  - Modifications of SGD (e.g., Adam)
- New architectural elements
  - Residual connections
  - Layer normalization
  - Batch normalization
- Rectified linear units and other activation functions (helps with vanishing gradient problem)

AlexNet

- The big breakthrough
- Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton (2012)
- Reduced error rate on ImageNet from 26% to 16%
- Used GPUs, dropout, ReLU, which have since become standard

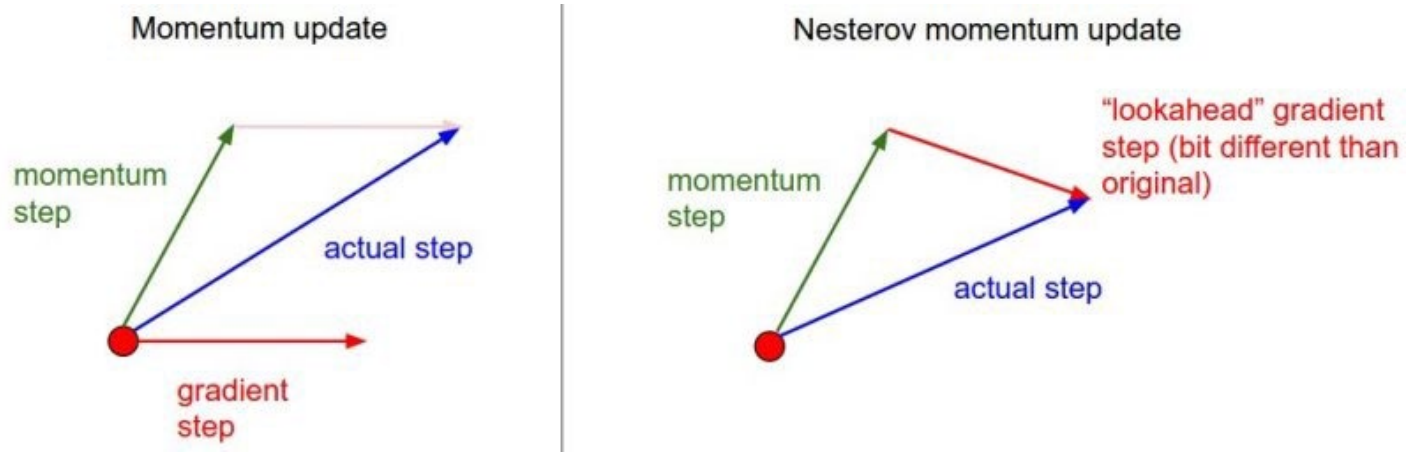


SGD with Momentum

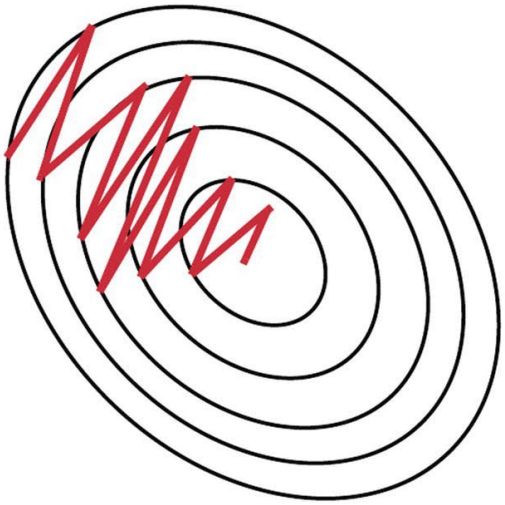
- Improves upon mini-batch SGD
- Basic idea: gradient descent with momentum

$$\text{update direction} = \beta \times (\text{current gradient}) + (1 - \beta) \times (\text{previous gradient})$$

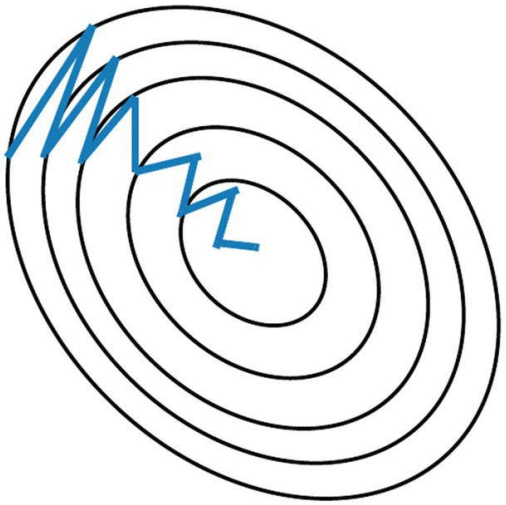
- Several possible implementations (e.g., Nesterov acceleration)



SGD with Momentum



SGD **without** momentum



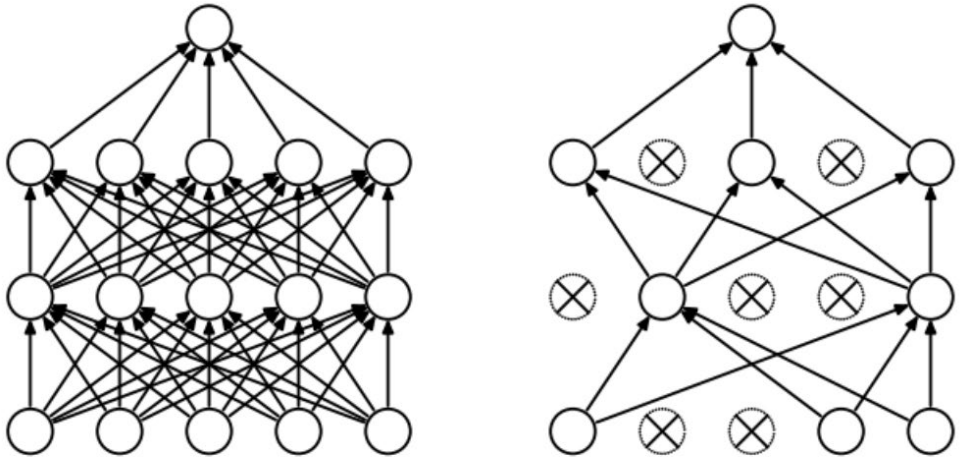
SGD **with** momentum

Adam Optimizer

- Improves upon gradient descent/SGD
- First ingredient: SGD with momentum
- Second ingredient: different step size for each weight depending on variance of gradient
- Many subsequent improvements and variants
- Kingma and Ba, "Adam: A Method for Stochastic Optimization," ICLR 2015

Dropout

- For each minibatch, every node in the network is omitted with some probability. The weights connecting to those nodes are not updated.
- To evaluate trained network, use the full network. If a node is retained with probability p, the weights emanating from that node are multiplied by p.
- See Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," JMLR 2014.



(a) Standard Neural Net

(b) After applying dropout.

Poll

Which of the following are ways to regularize neural network training:

(A) Dropout

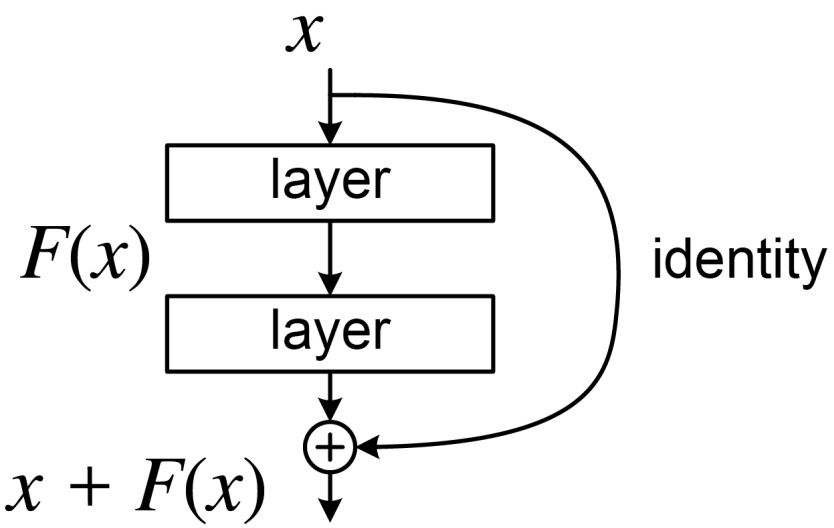
(B) Early stopping (stop iterating before convergence)

(C) Weight decay (add squared Euclidean norm of weight vector to empirical risk)

(D) All of the above

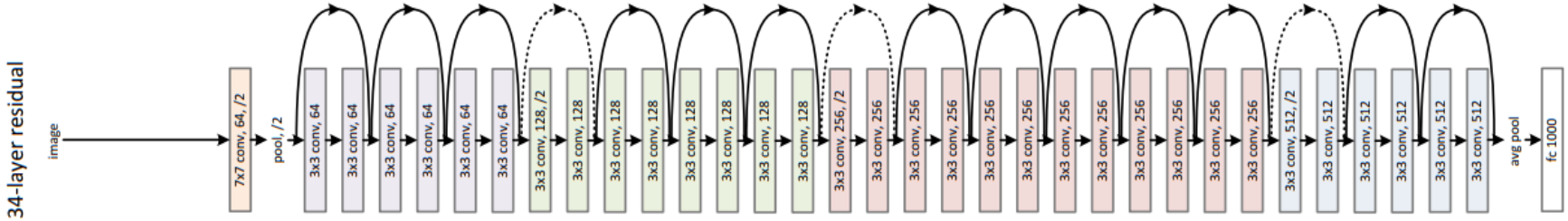
Residual Networks

- "Residual connections"
- Helps with vanishing gradients; gradient propagates directly back to earlier layers



- Addition is performed before applying activation function

ResNet


34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000

Batch Normalization

- Standardizing the inputs to a NN improves convergence
- Why not further standardize the inputs to each layer?
- A *batch normalization* layer is a layer with two learnable weights for every node from the previous layer

$w_1, b_1, \ldots, w_D, b_D$

- Consider a mini-batch of training data $\{\boldsymbol{x}_i\}_{i \in \mathcal{B}}$
- Suppose that the corresponding values at a hidden layer (after applying the activation function) are $\{\boldsymbol{a}_i\}_{i \in \mathcal{B}}$, where $\boldsymbol{a}_i \in \mathbb{R}^D$.
- Let $\{\tilde{\boldsymbol{a}}_i\}_{i \in \mathcal{B}}$ be the values after standardization: for each coordinate, subtract off mean and divide by standard deviation
- Output of batch normalization layer is

$$\boldsymbol{z}_i = \begin{bmatrix} w_1 & & 0 \\ & \ddots & \\ 0 & & w_D \end{bmatrix} \tilde{\boldsymbol{a}}_i + \begin{bmatrix} b_1 \\ \vdots \\ b_D \end{bmatrix}$$

Layer Normalization

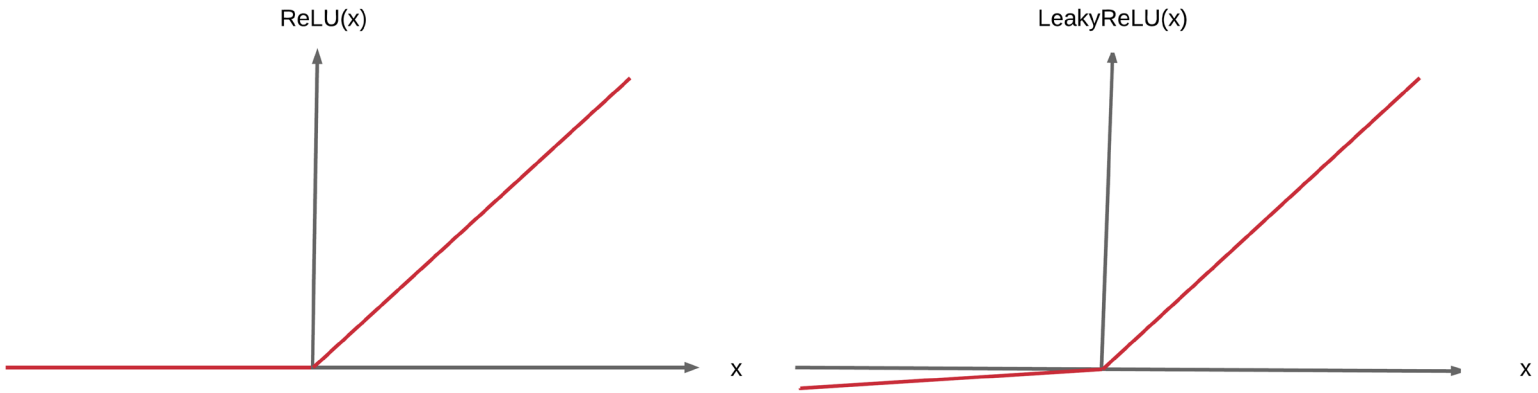
- Alternative to batch normalization
- Has the same form as batch normalization, except that $\{\tilde{\boldsymbol{a}}_i\}_{i \in \mathcal{B}}$ is obtained by standardizing each $\boldsymbol{a}_i$ itself, that is, for each $\boldsymbol{a}_i$ subtract off the mean of its coordinates, and divide by their standard deviation, to get $\tilde{\boldsymbol{a}}_i$

Batch vs Layer Normalization

- BN popular in CNNs; residual connections help to avoid exploding gradient
- LN popular in RNNs and transformers, works with small batch sizes

Activation Functions

- ReLU: $\sigma(t) = \max(0, t)$
- Leaky ReLU: $\sigma(t) = \max(\alpha t, t)$, $\alpha > 0$ (common choice: $\alpha = 0.01$)
  - Parameters continue to change after input becomes negative



Activation Functions

- Exponential linear unit (ELU)

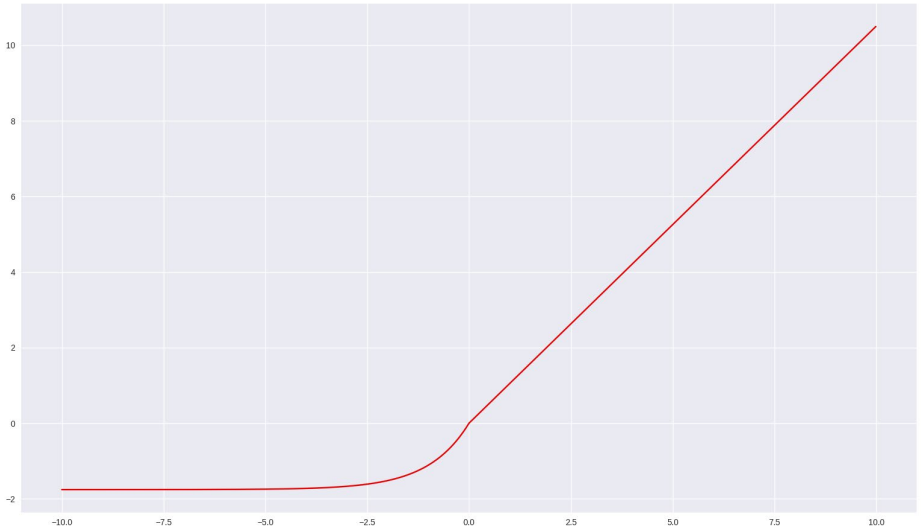
$$\sigma(t) = \begin{cases} \alpha(e^t - 1) & t \leq 0 \\ t & t > 0 \end{cases}$$

- Scaled exponential linear unit (SELU)

$$\sigma(t) = \lambda \begin{cases} \alpha(e^t - 1) & t \leq 0 \\ t & t > 0 \end{cases}$$

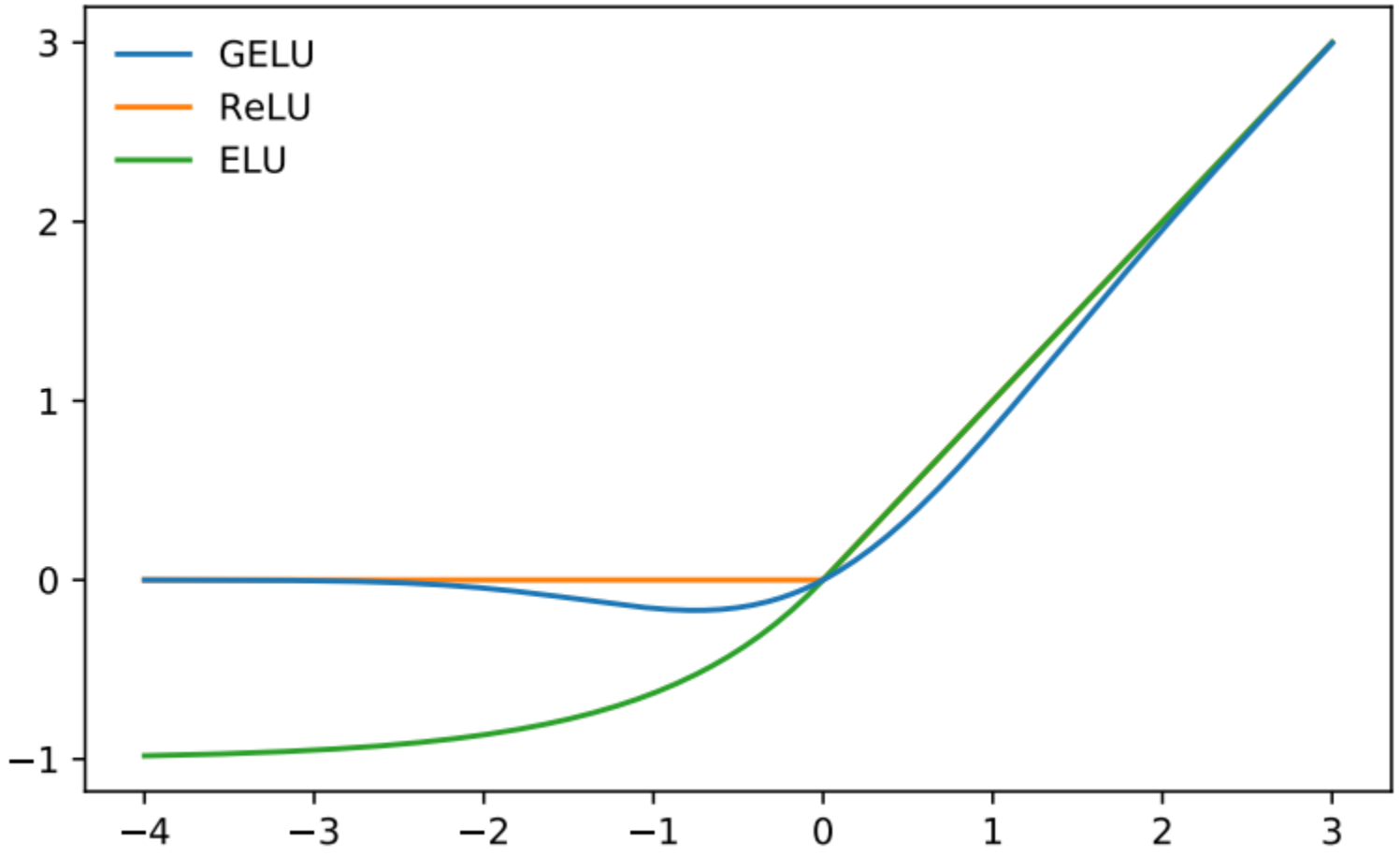
where $\alpha \approx 1.6733$ and $\lambda \approx 1.0507$.

  - Proposed in "Self-normalizing neural networks," Klambauer et al., NeurIPS 2017



Activation Functions

- Gaussian Error Linear Unit



Deep Learning in Practice

- (Deep) neural networks are most successful when the architecture is tailored to structure inherent in the data, e.g., spatial structure (CNNs), temporal structure (RNNs).
- Neural networks have been extended to a growing list of non-Euclidean data structures. Some keywords:
  - Graph neural networks
  - Graph convolution
  - Geometric deep learning
  - Meshed surface
  - Point clouds

Novel Deep Models

- Semantic Image Segmentation
  - U-Net
- Deep Generative Models
  - Variational autoencoders
  - Generative adversarial networks
  - Normalizing flows
  - Diffusion models
- Transformer networks (next lecture)

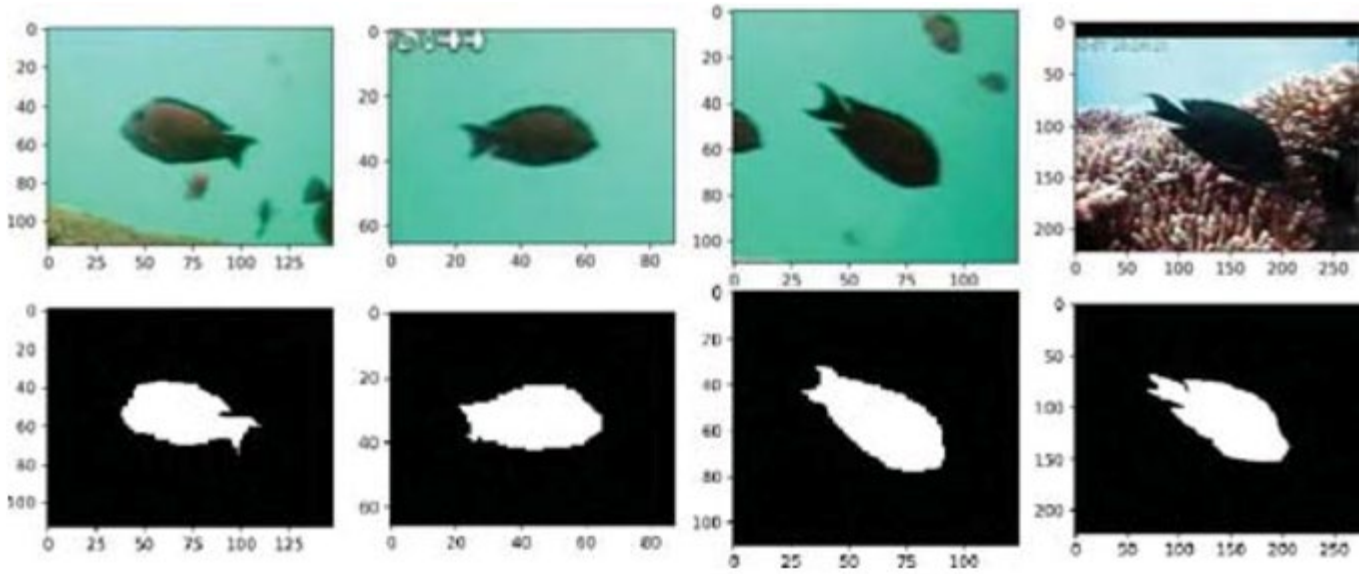
Final lecture:

(A) Diffusion models

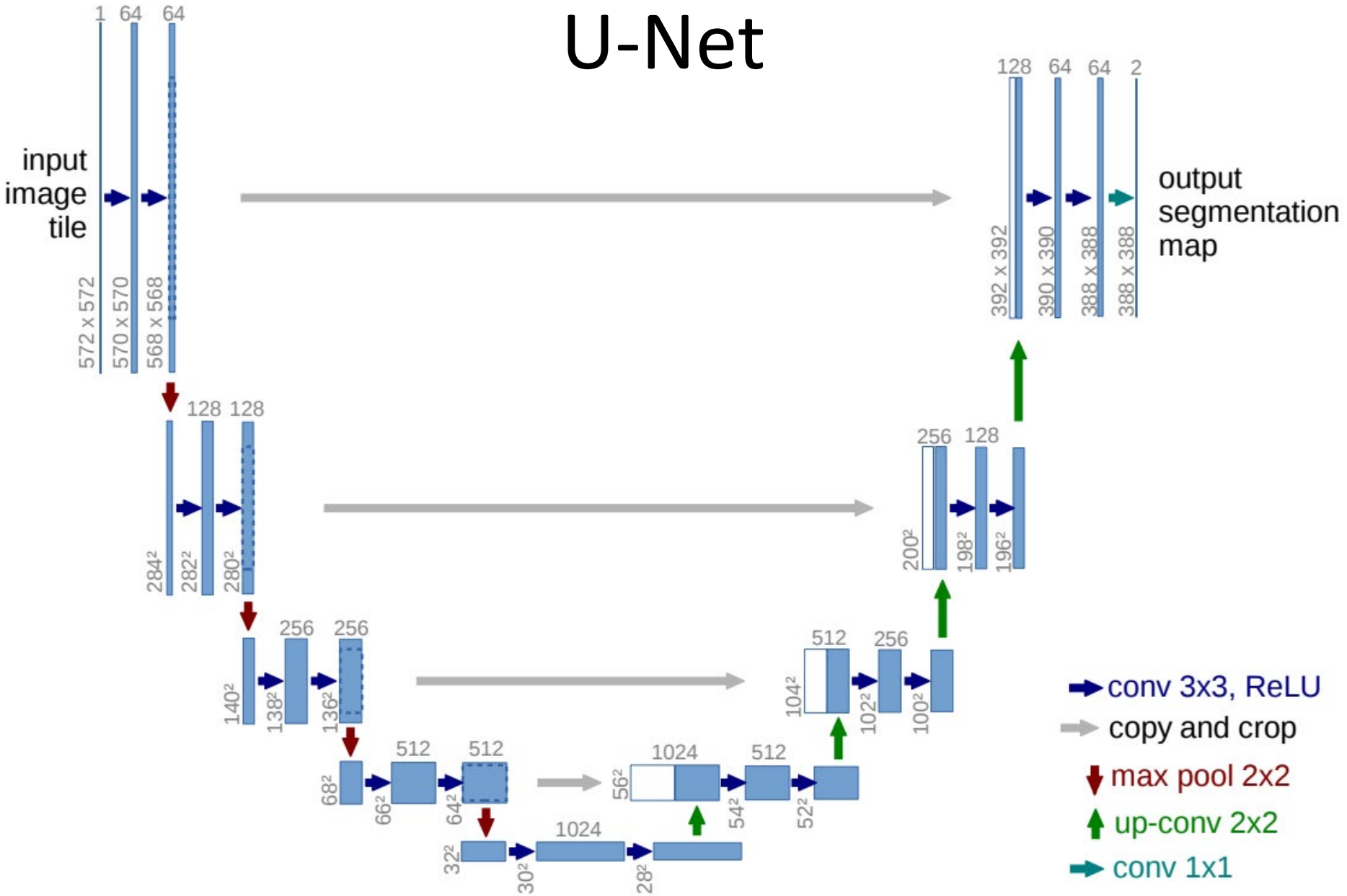
(B) Socially responsible machine learning

Semantic Image Segmentation

- Given: training images together with their segmentation maps
- Goal: learn to correctly classify pixels of test images



U-Net


input image tile
572 x 572
570 x 570
568 x 568
1 64 64
128 128
284²
282²
280²
256 256
140²
138²
136²
512 512
68²
66²
64²
1024
32²
30²
28²
1024
56²
512
54²
512
52²
512 256
104²
102²
100²
256 128
200²
198²
196²
128 64 64 2
392 x 392
390 x 390
388 x 388
388 x 388
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

U-Net Details

- Left half is basically a standard CNN
- In the figure, there is no zero padding for the convolutions
- Up-Conv 2×2:
  - Upsample each spatial dimension by factor of two
  - Followed by 2×2 convolution that halves number of channels (evidently this convolution uses 0 padding)
- Overall the U-Net is like an autoencoder where
  - the decoder is the mirror image of the encoder
  - after each up-conv 2×2 step, the corresponding values from the encoder are copied, cropped (if necessary), and concatenated
    - this fills in additional detail necessary for pixel-wise classification
    - defining feature of U-Net
- Output layers (2): Real-valued output of 1×1 convolution, followed by a sigmoid activation at final layer to produce grayscale outputs
- Training objective $\approx$ sum of cross-entropy losses across all pixels

```
1            1, 2
2  Input = (3, 4)
3
4            1, 1, 2, 2
5  Output = (1, 1, 2, 2)
6            3, 3, 4, 4
7            3, 3, 4, 4
```

U-Net Details

- To segment an arbitrarily sized input image, apply U-Net multiple times on different 572×572 patches, such that the central 388×388 portions of those patches cover the entire image.

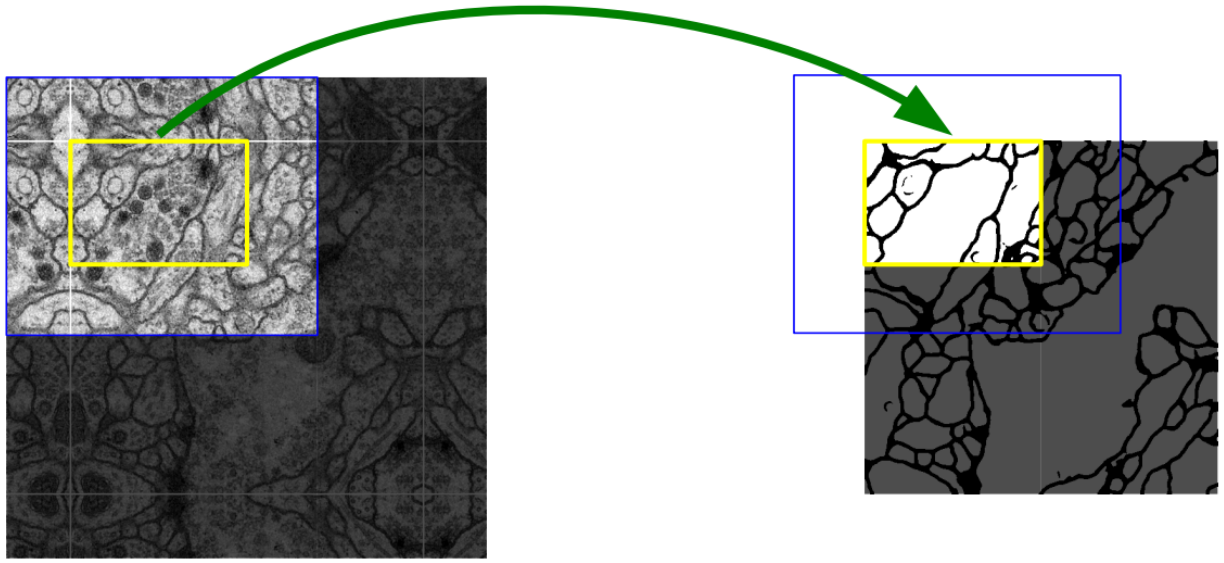
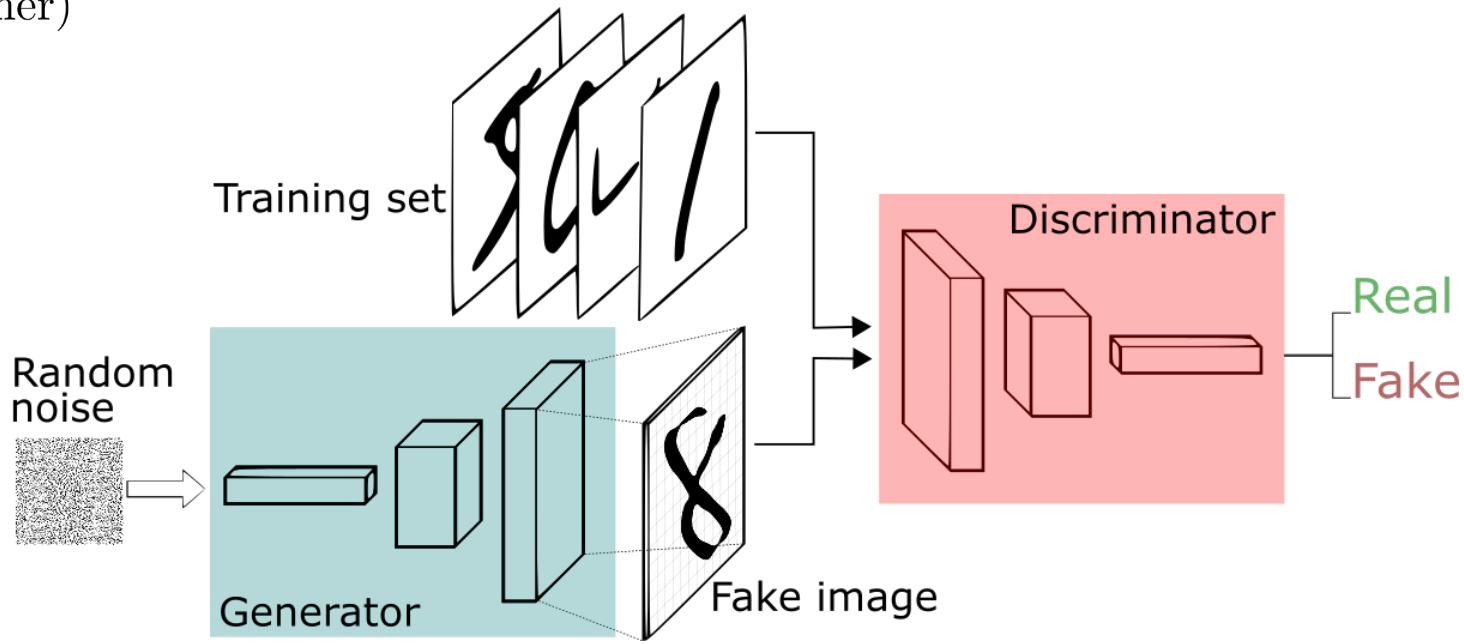


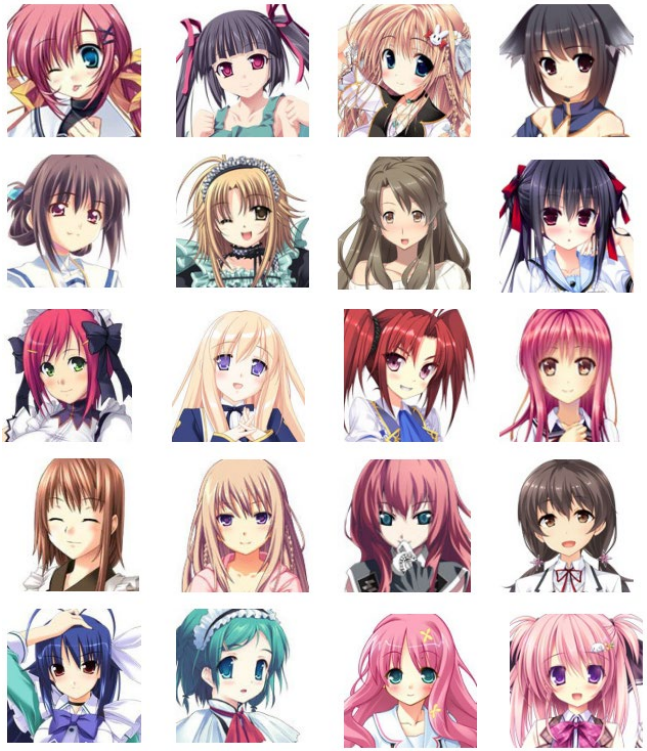
Fig. 2. Overlap-tile strategy for seamless segmentation of arbitrary large images (here segmentation of neuronal structures in EM stacks). Prediction of the segmentation in the yellow area, requires image data within the blue area as input. Missing input data is extrapolated by mirroring

Generative Adversarial Networks

- Used to simulate new realizations of a class of objects
- Generator network is similar to decoder in U-Net, but without the copy-(crop)-concatenate operation
- Discriminator network and generator network compete against each other
- Train by alternating optimization (fix one network while optimizing the other)



GAN for Anime Character Generation



Real training images

Generated images