Natural Computation Methods for Machine Learning – 2022

Convolutional Neural Networks and Deep Learning

Damian Matuszewski
damian.matuszewski@it.uu.se

Centre for Image Analysis
Uppsala University

Image analysis fundamental steps

image acquisition

preprocessing,
enhancement

segmentation

Representation, description,
feature extraction

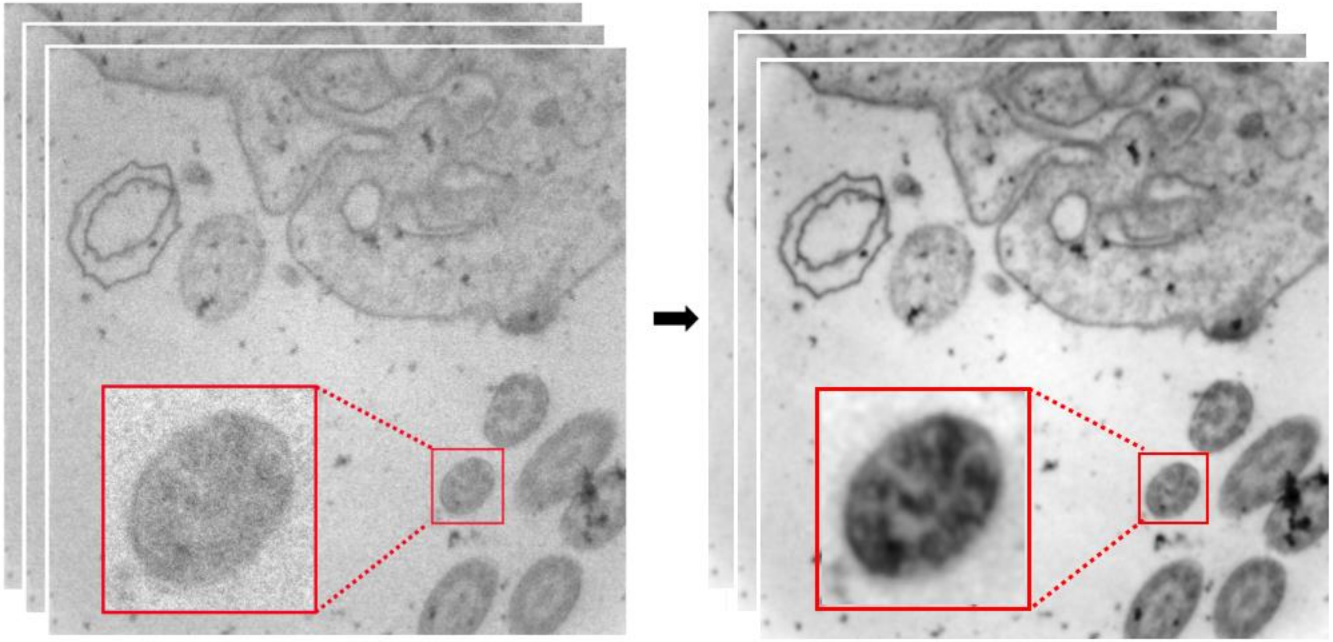
Classification,
interpretation, recognition

result

DL can be used for any step
or a subset of steps, including the
“complete” application: i.e. from a raw
image to its classification / recognition.

Applications in Pre-processing

- Denoising,
- Super-resolution, etc.

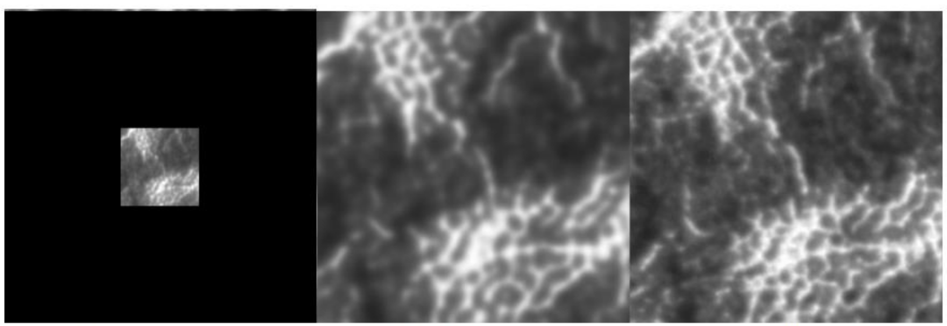


An example of full size denoised transmission electron microscopy image using deep learning.

B. Bajic*, A. Suveer*, A.Gupta*, I.Pepic, J. Lindblad, N. Sladoje, and I.M. Sintorn, “Denoising of Short Exposure Transmission Electron Microscopy Images for Ultrastructural Enhancement”, 15th IEEE International Symposium on Biomedcial Imaging (ISBI), 2018

Applications in Pre-processing

- Denoising,
- Super-resolution, etc.

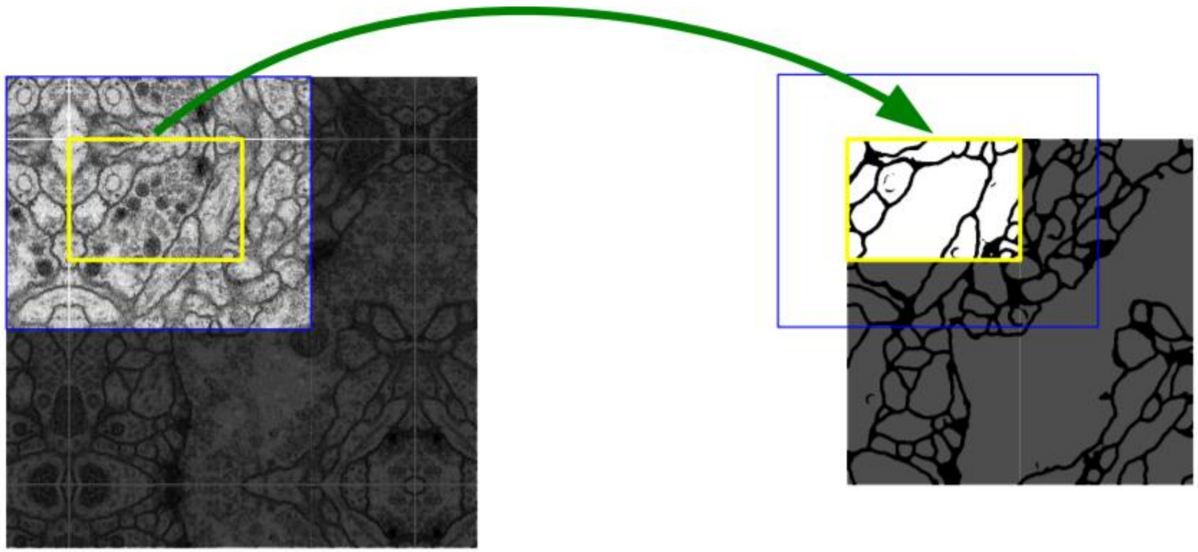


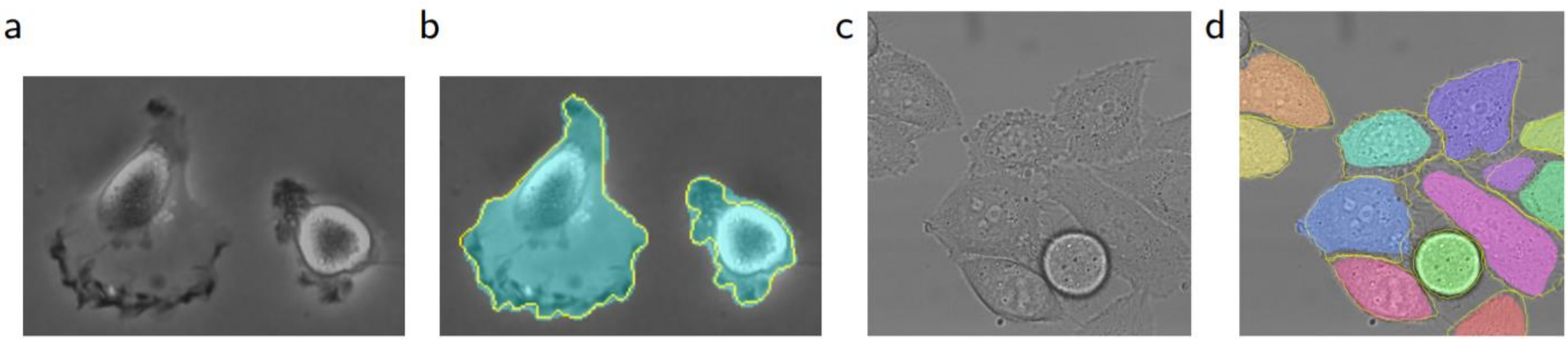
SR reconstruction of TEM images

SR reconstruction of generic images

Dahl R, Norouzi M, Shlens J. Pixel recursive super resolution. arXiv preprint arXiv:1702.00783. 2017.

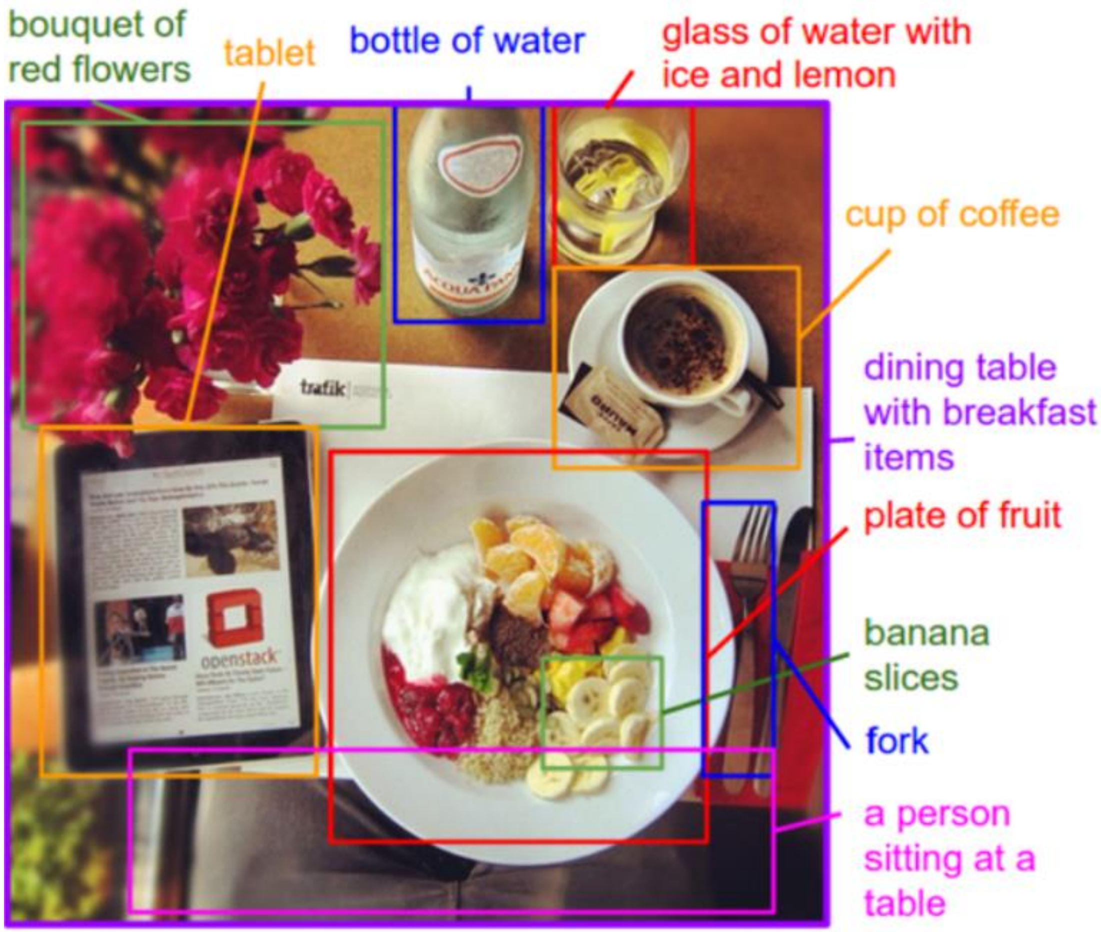
Applications in Segmentation

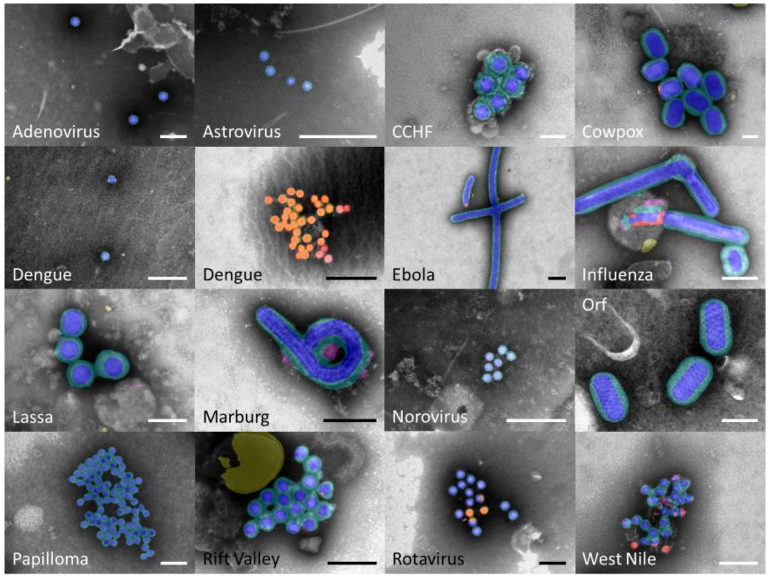




Olaf Ronneberger, et al. ,U-Net: Convolutional Networks for Biomedical Image Segmentation, MICCAI 2015

Applications in Classification



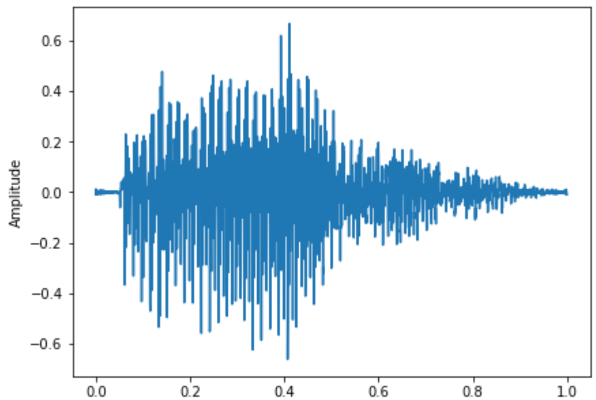


Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3128-3137. 2015.

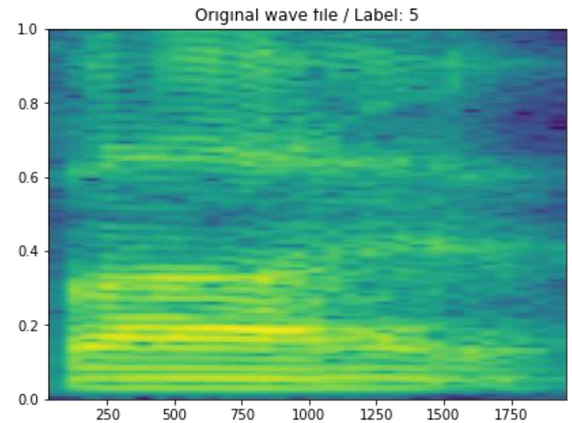
Matuszewski, Damian J., and Ida-Maria Sintorn. "Reducing the u-net size for practical scenarios: Virus recognition in electron microscopy images." *Computer methods and programs in biomedicine* 178 (2019): 31-39.

(Not) only images

- Audio



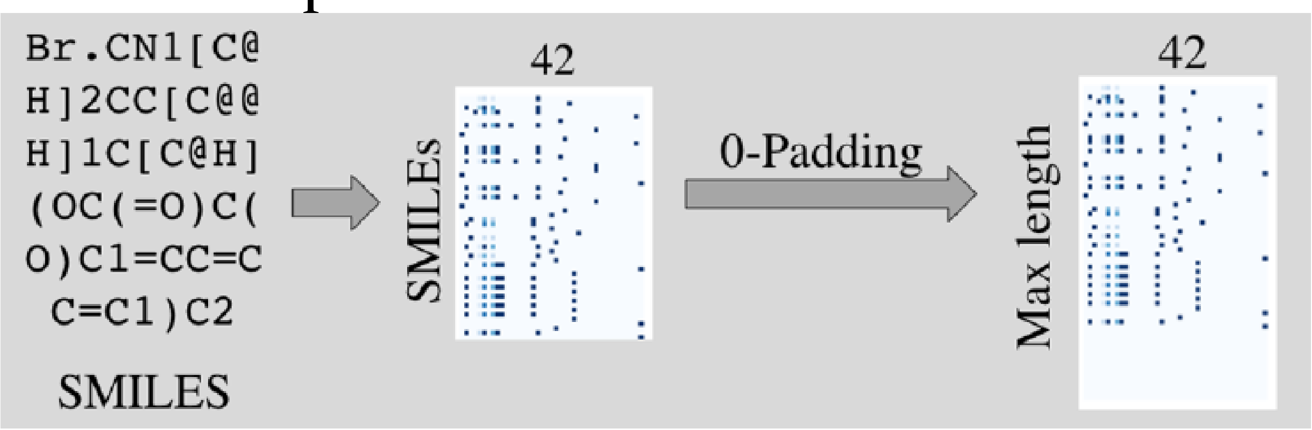
Amplitude



Spectogram

https://medium.com/@keur.plkar/audio-data-augmentation-in-python-a91600613e47

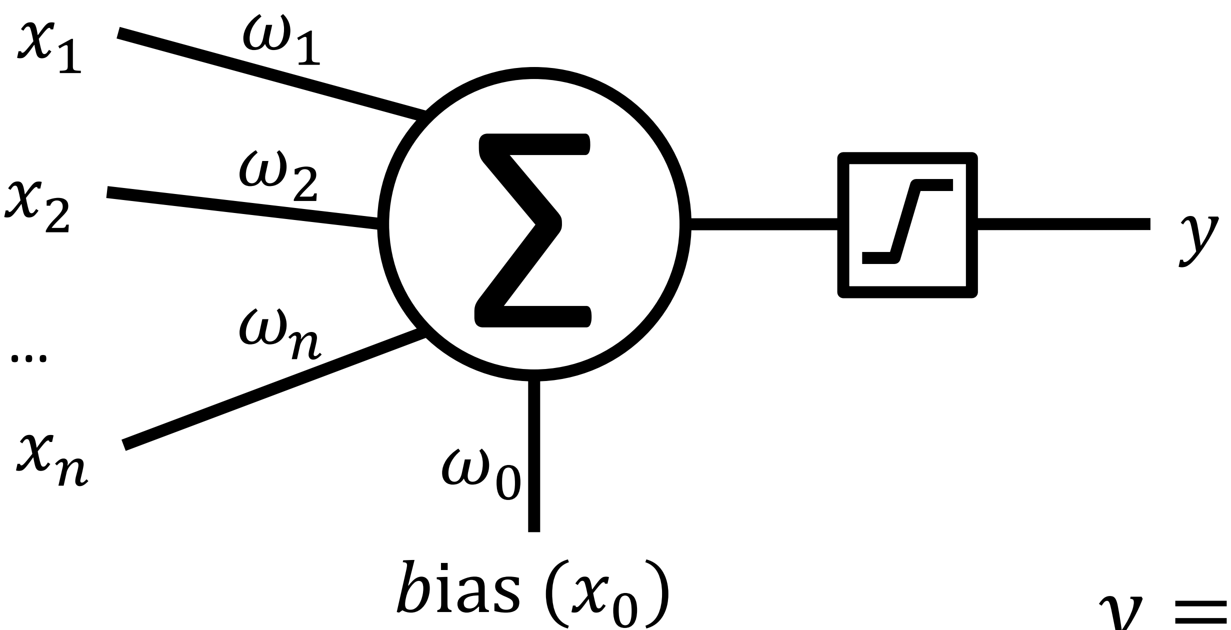
- Chemical compounds



Hirohara. M, Saito. Y, Koda. Y, et al. *"Convolutional neural network based on SMILES representation of compounds for detecting chemical motif"*. BMC Bioinformatics 19, 526 (2018). https://doi.org/10.1186/s12859-018-2523-5

Neural Network

Perceptron – the smallest unit (also called a node or a neuron) in a Neural Network



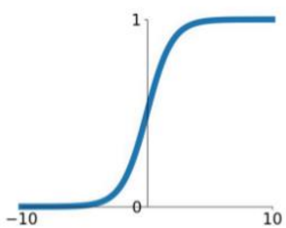
$$y = f\left(\sum_{i=0}^{n} x_i \omega_i\right)$$

Activation functions

- Normalize layer / node output
- Introduce nonlinearity

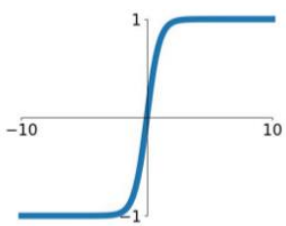
Sigmoid

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



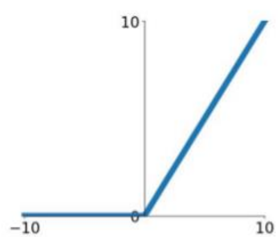
tanh

$\tanh(x)$



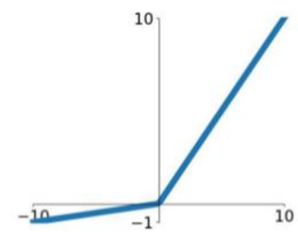
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

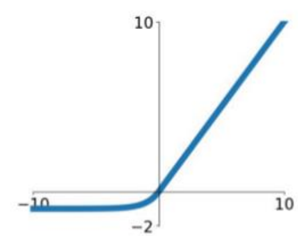


Maxout

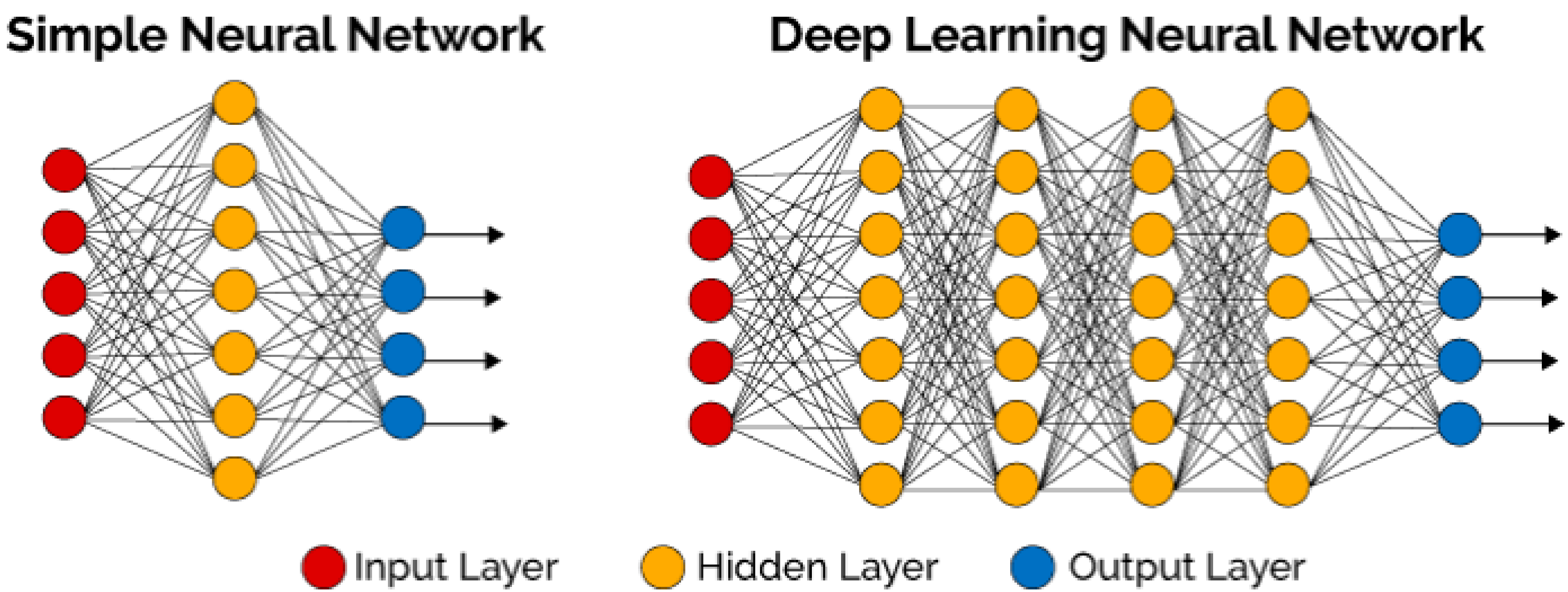
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

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



Deep Neural Networks



More than 2 hidden layers -> Deep Learning

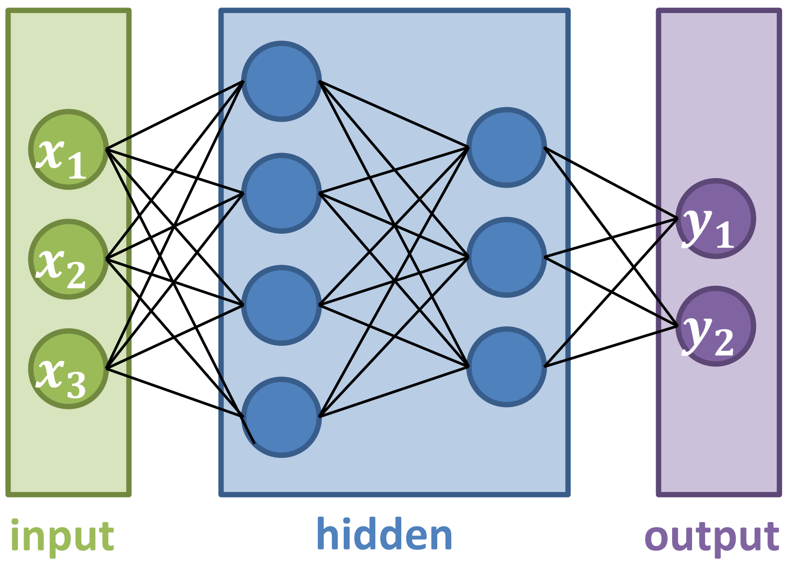
Src. https://www.kdnuggets.com/2017/12/deep-learning-made-easy-deep-cognition.html

Layers

- The most common types of layers
  - Trainable:
    - Fully connected
    - Convolutional
    - LSTM
    - …
  - Auxiliary
    - Pooling
    - Drop out
    - Flattening
    - Normalization
    - …

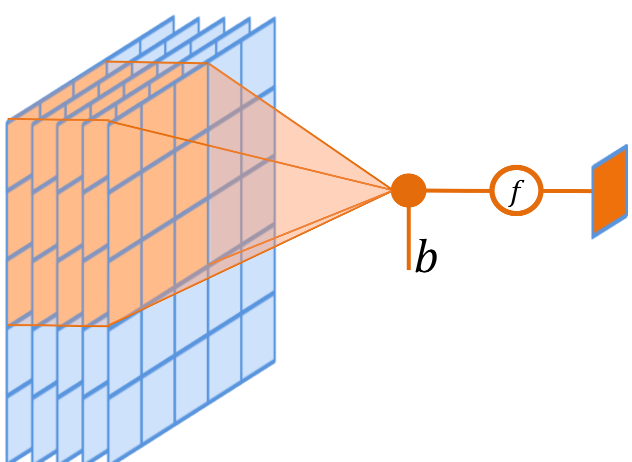
Layers – Fully connected

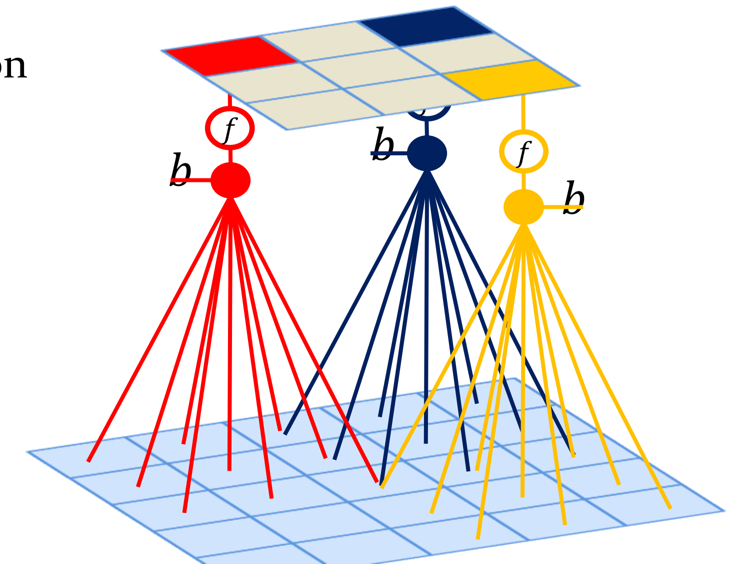
Fully connected – a node in a given layer takes all the output from the previous layer as its input



Convolutional Layers

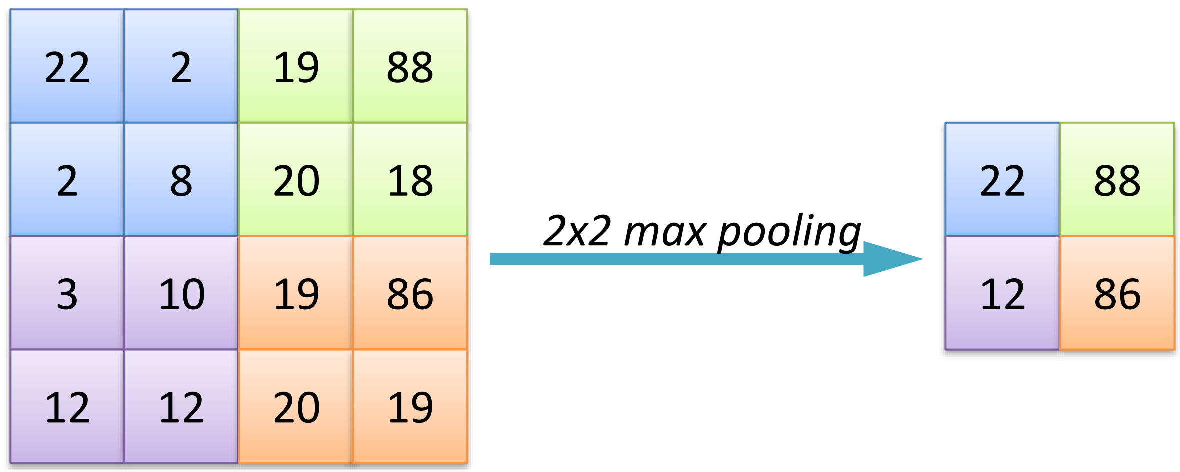
- Convolutional Neural Networks – NNs with convolutional layers
- They use spatial connections resembling those in the sliding window of convolution filters. That's perfect for images!
  - Size of the window / receptive field (typically 3 x 3)
  - Stride – speed of sliding over input (typically 1)
  - Padding: valid (none) or same (zeros)
- Nodes in the same feature map of a given layer have the same weights so the pattern is detected regardless of its position in the input
- Many feature maps, each detecting a specific pattern, compose a layer
- Nodes take all input channels (feature maps) in the previous layer under their respective spatial coverage
  - 2D convolutional is in fact 3D, 3D is in fact 4D and so on





Pooling layers

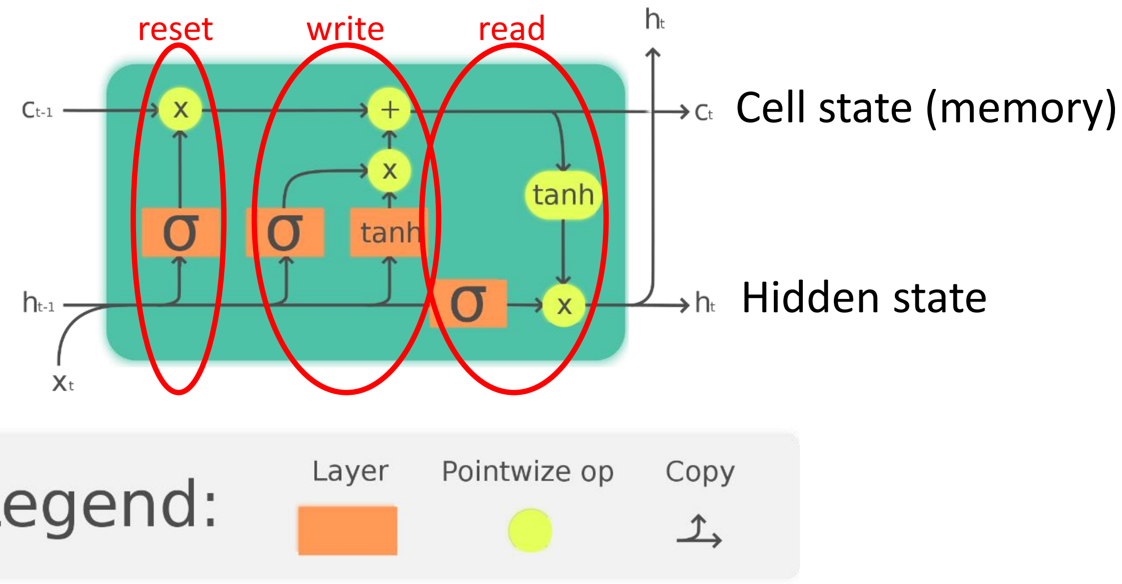
- The most common is 2 x 2 max pooling with stride 2



- They take one feature map at a time in contrary to convolutional layers that take all the feature maps in the same time
- They reduce the size of feature maps (but not their number!) in the following layers
- This increases the receptive field (spatial coverage w.r.t. the original input image) which helps detecting larger patterns
- It also reduces the computational complexity of the network (less to compute) and may reduce the number of trainable weights and thus help prevent overfitting

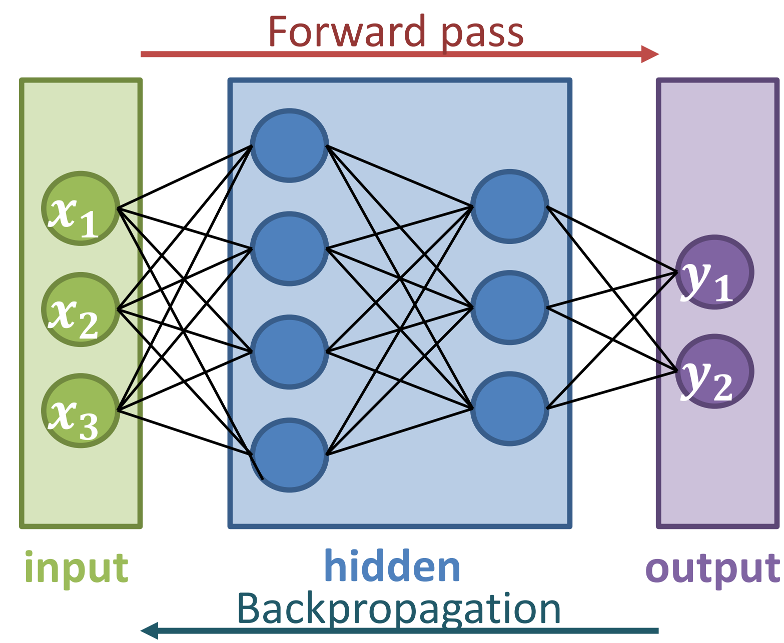
LSTM – for sequential data

- Long Short-Term Memory (LSTM)
  - a type of recurrent neural network
    - *A network that has an internal state that can represent context information*
  - can process data sequentially and keep its hidden state through time
  - can learn order dependence in **sequence prediction** problems



Training

- Data preparation
  - Split into train, validation and test sets
- Data preprocessing
  - Normalize the images by subtracting the mean and dividing by the standard deviation (make sure to convert them to floating point first!)
- Architecture design / choice
  - What is the application?
- Weight Initialization
  - Small random numbers centered at 0
- Optimization & Backpropagation
  - Forward pass
  - Loss calculation, $f(pred, true)$
  - Gradient estimation (optimization)
  - Weight update (backpropagation)
- Iterative process
  - Train on training set, evaluate and upgrade on validation set, measure the final performance on the test set

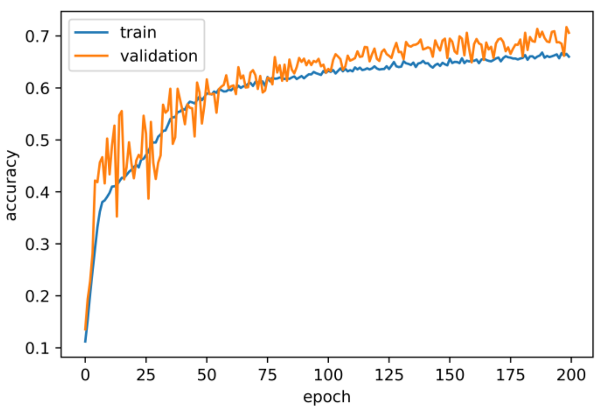


Overfitting and underfitting

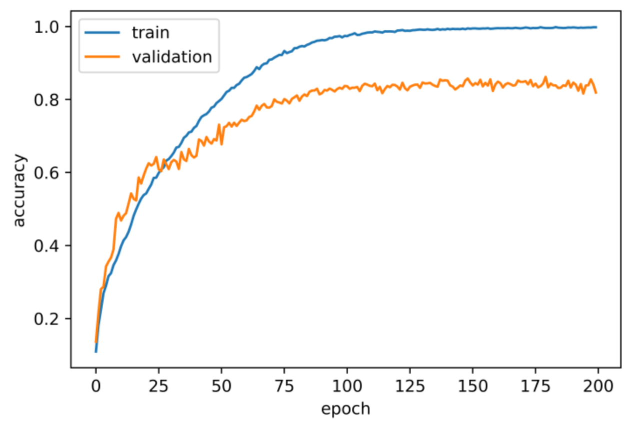
- It is very important to monitor the training / learning process of the network
  - Plot the loss functions (for training and validation) during the training
  - You can also plot other metrics (i.e. accuracy)
- Three scenarios are possible:
  - Underfitting
  - Good fit
  - Overfitting

Accuracy plots – train vs validation

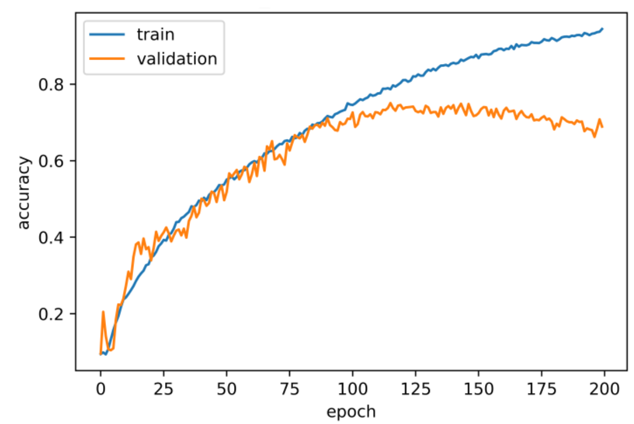
Underfitting



Good fit

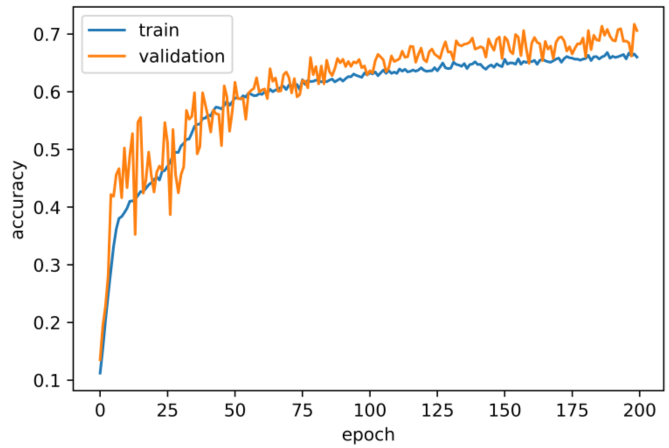


Overfitting



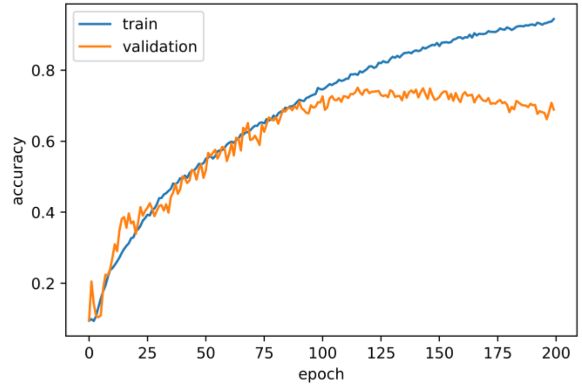
Underfitting

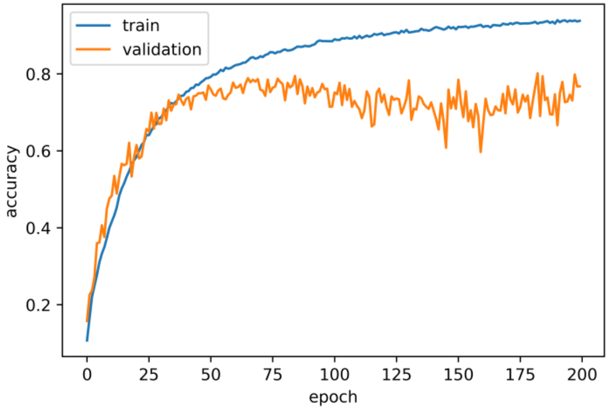
- Underfitting – the network is struggling with learning the patterns in the data
- Characterized by
  - Better performance on the validation set than on the training set
- Diagnosis
  - The network is too simple to model the patterns or
  - It didn't train for long enough to learn the patterns or
  - The validation set is too small or too simple w.r.t. the training set
- Solution
  - Increase the network complexity
  - Train longer
  - Change learning rate
  - Make sure that all your data sets (training, validation and test) are representative



Overfitting

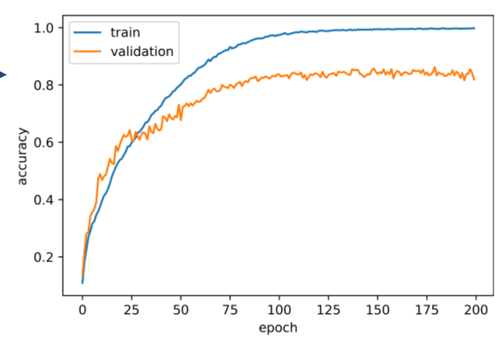
- Overfitting – the network memorizes the training samples instead of learning the patterns
- Characterized by
  - High performance on training set
  - Low performance on validation set
  - Sometimes there is the characteristic decrease of validation performance over epochs / updates
- It is also a **poor generalization** – ability to correctly analyze previously unseen samples
- It will **always** happen in deep learning
  - We can only minimize / reduce this effect
- The worst scenario:
  - Too complex network
  - trained for too long
  - on too little data

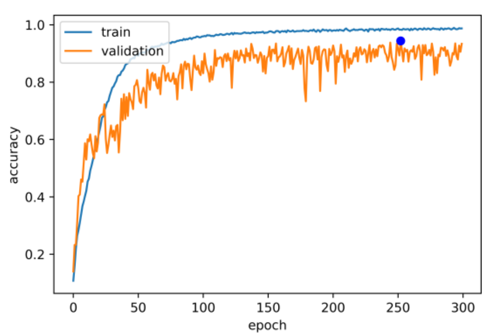




Generalization, Overfitting and Performance

- N trainable weights require typically 2N – N^2 training samples
  - That's difficult to satisfy even for shallow Neural Networks
  - DL can easily reach millions of trainable weights
- Improve generalization / reduce overfitting / improve the performance
  - Increase the quality of training data (make it representative)
    - Split the data wisely
    - Preprocess the images
  - Increase the amount of training data
    - Acquire and annotate new samples
    - Use data augmentation
      - Real time / on-the-fly random augmentations
  - Early stopping
    - Use the validation set / cross-validation
    - Stop training when the performance on the validation set drops
  - Reduce or modify the network complexity / size / architecture
  - Drop out layers
  - Regularization
  - Batch normalization
  - Pretrain / transfer learning



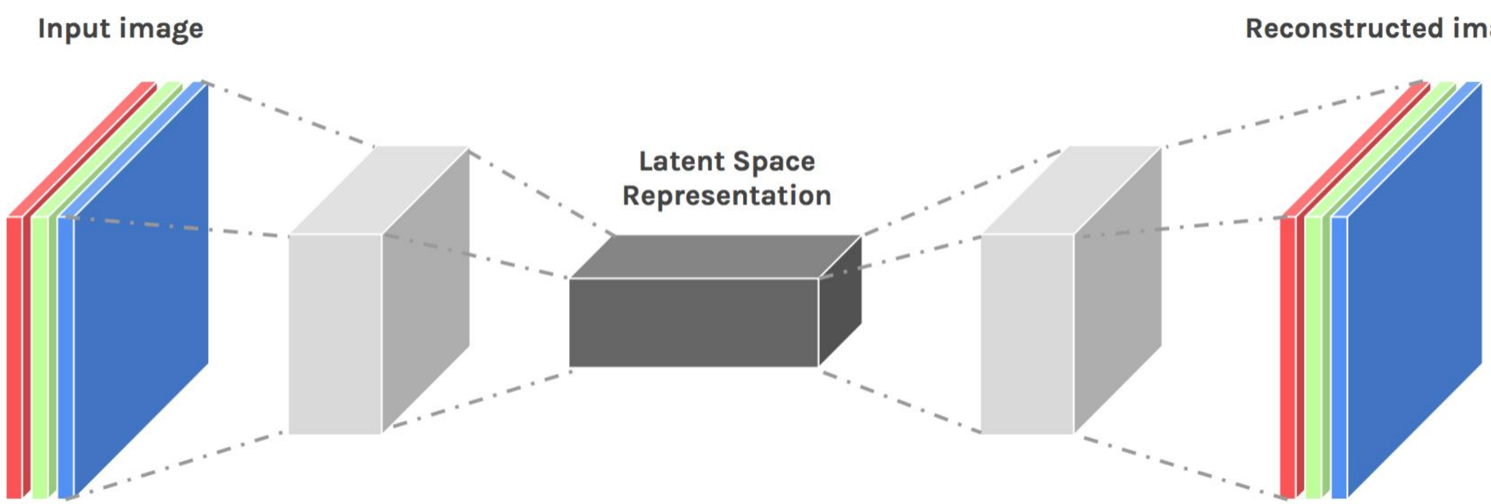


Transfer Learning

- Train a network on a similar task (simpler or with more annotated data available) and then fine tune the network for your target application
  - You can also train an autoencoder on your target data
- You can also use already existing networks trained on millions of images
- Typical fine tuning in transfer learning
  - Take or train the base network
  - Remove the last (often fully connected) layers
  - Replace them with new ones, designed for your application
  - Train only the newly added layers on your target data
  - Sometimes you can also train a couple of the last layers from the base network

Autoencoders

- Unsupervised feature learning
- Used in dimensionality reduction, feature extraction and transfer learning / pretraining
- Try to recreate input after mapping to lower dimension (or latent space)
- You can stack multiple autoencoders to create a deeper network



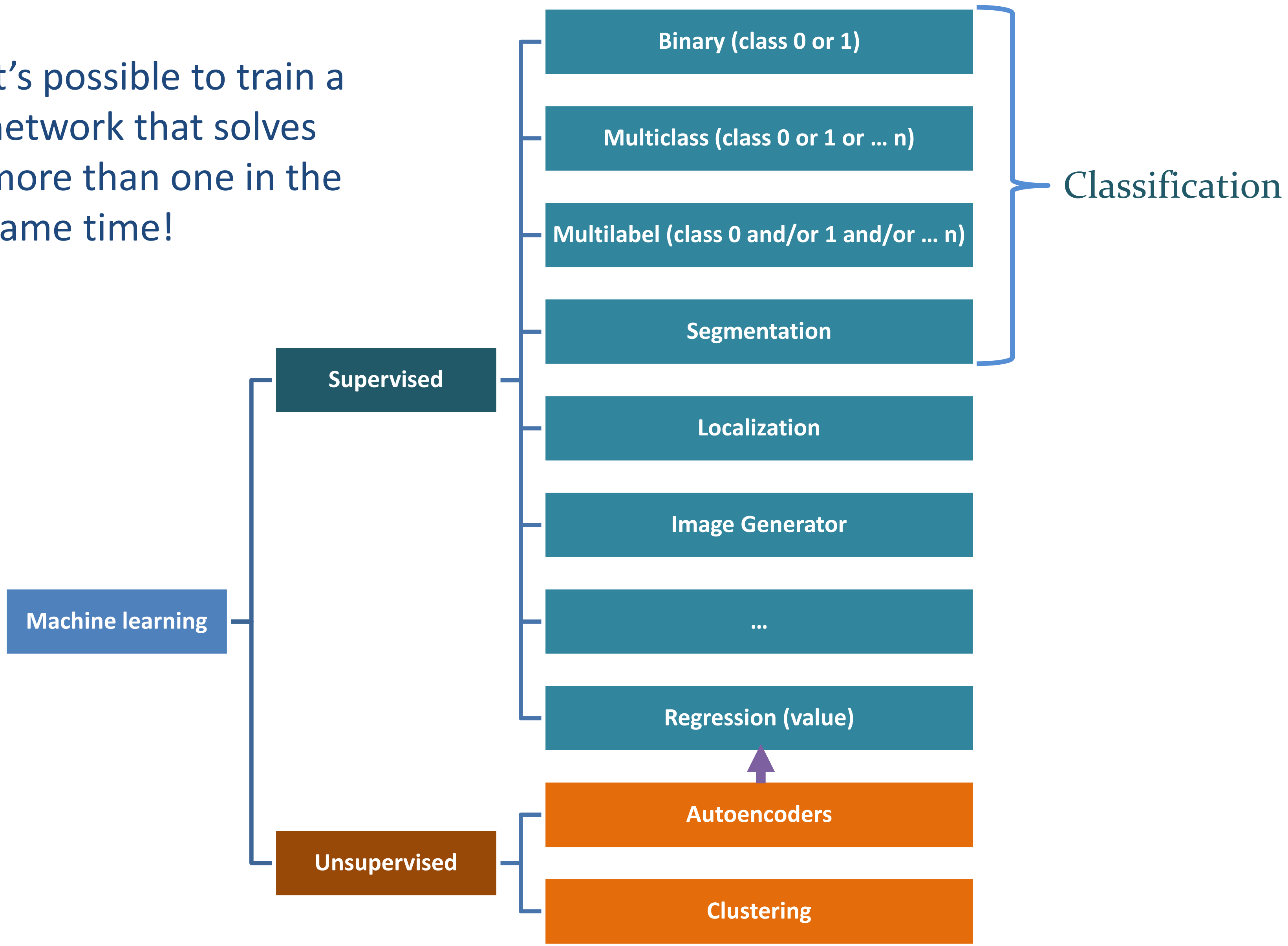
Src: https://becominghuman.ai/understanding-autoencoders-unsupervised-learning-technique-82fb3fbaec2

Hyperparameters

- Parameters
  - Architecture:
    - Number of layers
    - Connections between layers
      - Feed forward / skip connections
    - Types of layers (neural connections or auxiliary layers)
      - Number of nodes / feature maps
      - Convolutional
        - Size
        - Stride
        - Padding
      - Initialization method
      - Regularization
      - Activation functions
      - …
    - …
  - Training:
    - Number of epochs / updates / stopping condition
    - Batch size
    - Learning rate
    - Optimizer
    - Loss function
    - …

Types of machine learning tasks possible with DL

It's possible to train a network that solves more than one in the same time!



Typical loss functions and output layer activation functions

Classification type	Labels	Activation function	Loss function
Binary	1) [0,1] or [1,0] 2) [0 or 1]	1) Softmax 2) Sigmoid	Binary cross-entropy
Multiclass (MC)	• [0, 1, 0, 0, …, 0] • 0-N integer	Softmax	Categorical cross-entropy
Multilabel (ML)	[1, 1, 0, 0, …, 1]	Sigmoid	Binary cross-entropy
Segmentation	Binary image for each possible label including background	Softmax	• Categorical cross-entropy (MC) • Binary cross-entropy (ML) • Log Dice loss (MC / ML)
Regression	1) Raw value(s) 2) Normalized to 0-1	1) None 2) Sigmoid	• Mean / Sum of Absolute Errors • Mean / Sum of Square Errors
Autoencoders	Input image normalized to 0-1	• None • Sigmoid	• Mean / Sum of Absolute Errors • Mean / Sum of Square Errors
…	…	…	…

- You can always design your own loss functions!
  - Make sure they minimize on positive values (required for convergence) i.e. the better the performance the smaller the value but never smaller than zero.
  - Add logarithm for smoothing

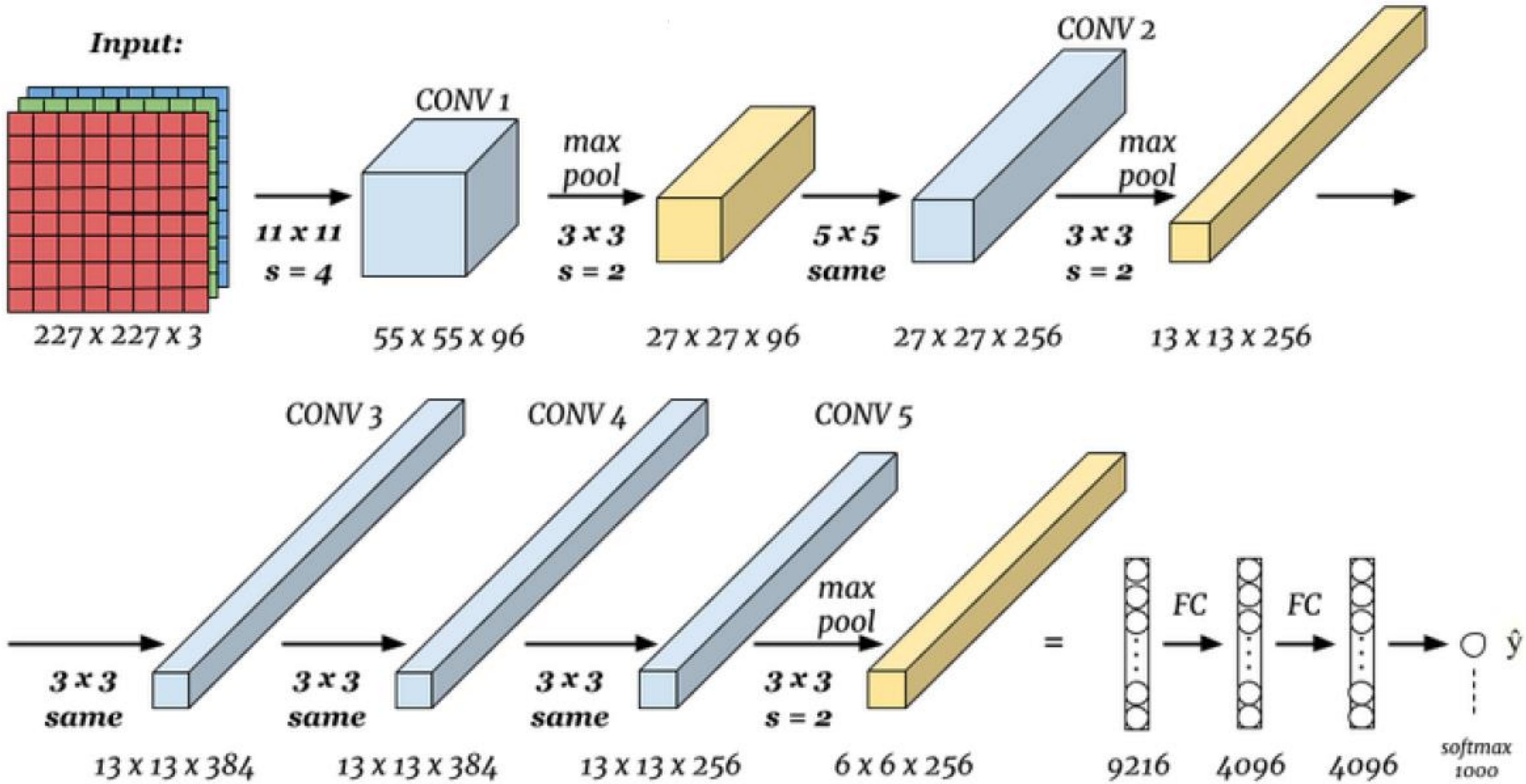
Softmax

- In a multiclass classifier, each node in the output layer is dedicated to one class
- The labels are one-hot coded (one 1, rest 0)
- Then most of the time, the activation function in the last layer is a global softmax

$$\sigma(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_{k=1}^{N} e^{x_k}}$$

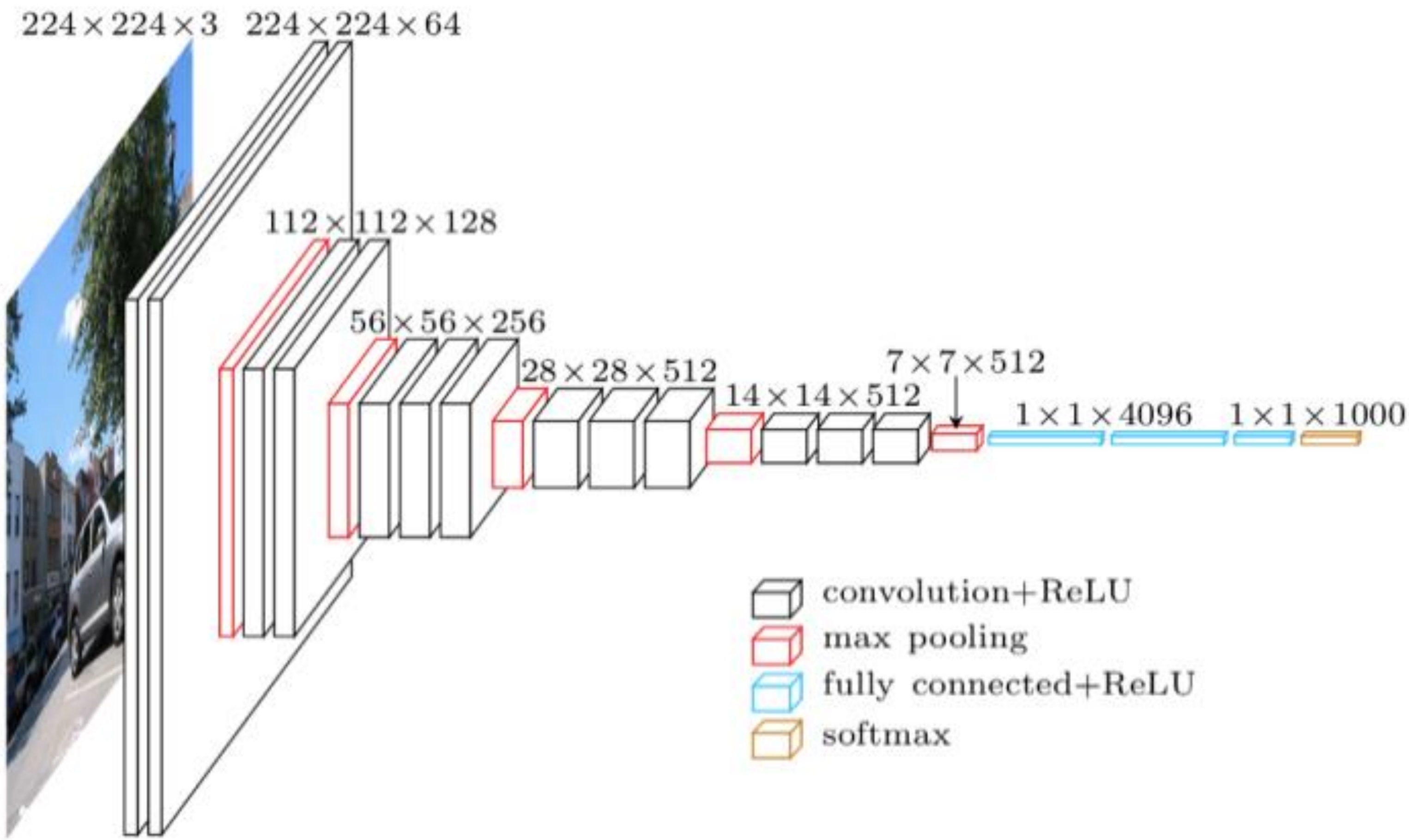
- It normalizes the output so it sums up to 1
- The highest raw value is still the highest softmax output and is often interpreted as the "class probability"

Notable Networks: AlexNet



Src: https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

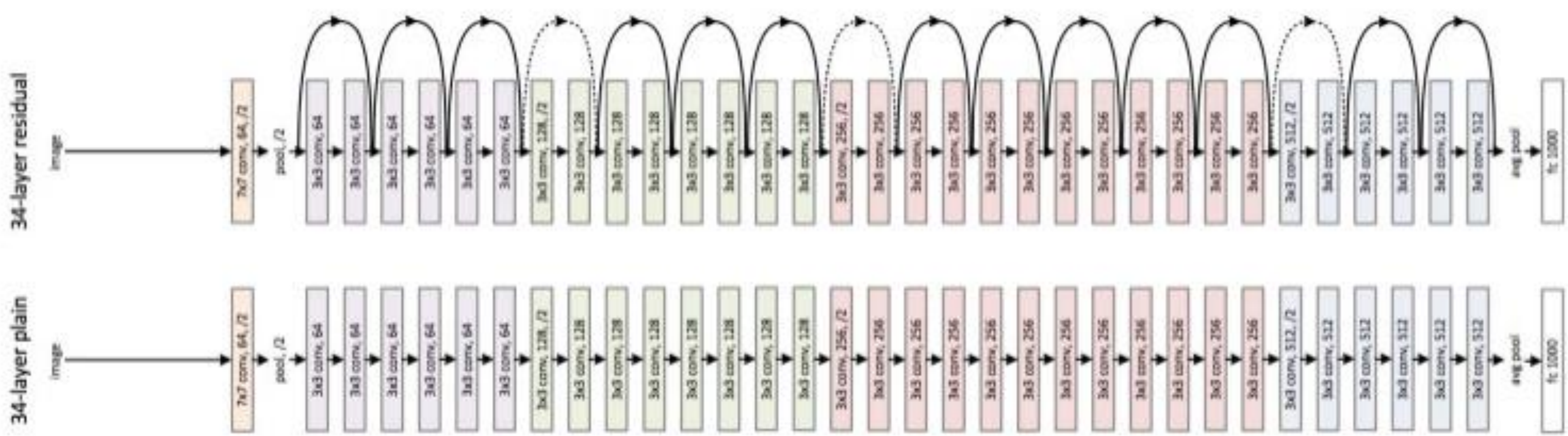
Notable Networks: VGG



Src: https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

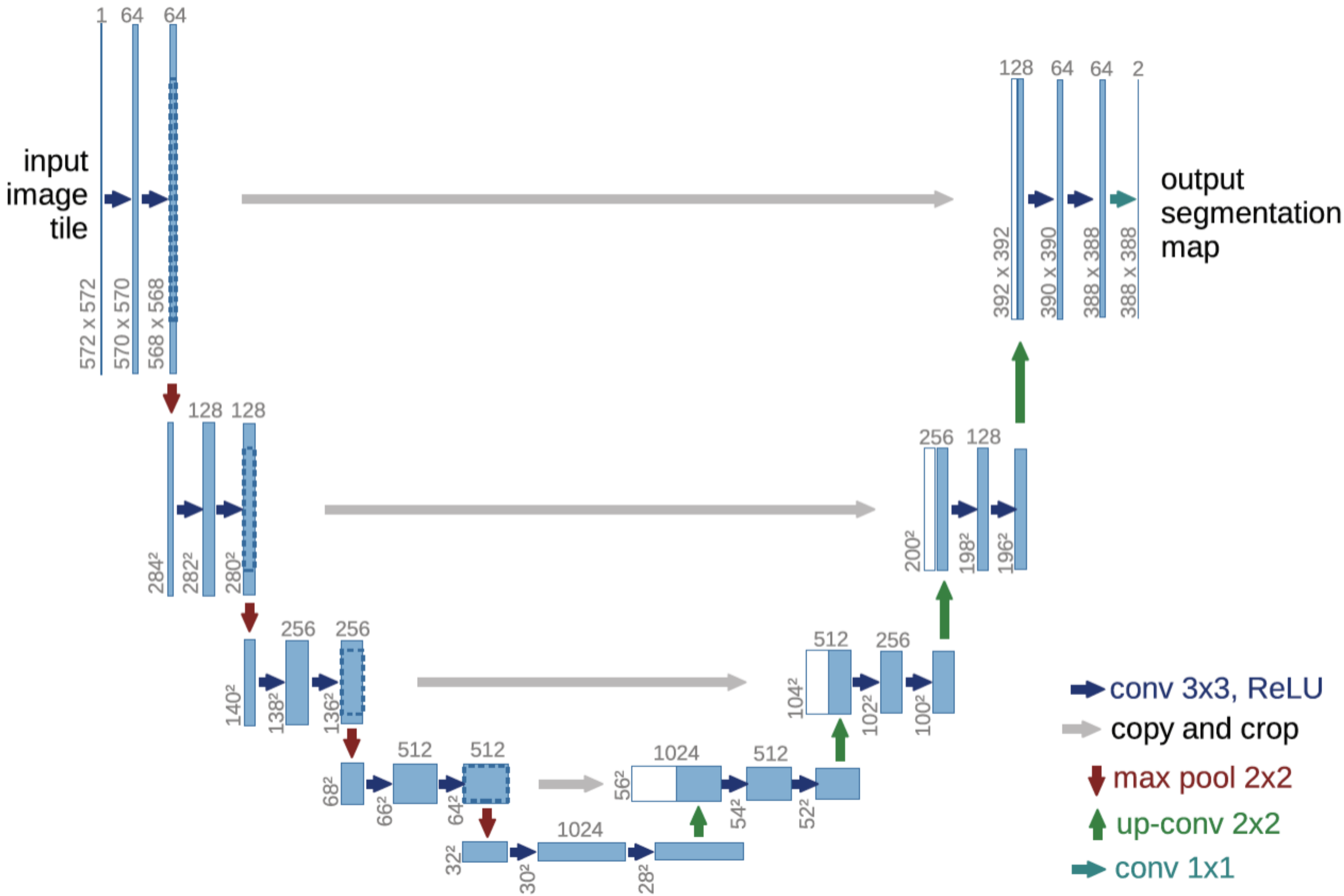
Notable Networks: ResNet

- Residual (skip) connection combines the input of one layer with at least one skipped layer
- It helps against vanishing gradient and often improves the performance



Src: https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

Notable Networks: U-Net



Olaf Ronneberger, et al., U-Net: Convolutional Networks for Biomedical Image Segmentation, MICCAI 2015

Some of the other network types

- Graph Neural Networks
  - Data represented in graphs
- Transformers (attention models)
  - Sequential data (e.g., natural Language Processing)
  - Data represented in graphs
  - Also applied to images
- Bayesian Neural Networks
  - Improving the results transparency by adding confidence score
- Recurrent Neural Networks
  - Sequential data

DL limitations

- They require large amounts of annotated data
- They are often tricky to get started with
  - They require some programming knowledge
  - They need fine tuning that may be hard to do in the beginning
    - The number of hyperparameters and their combinations can be overwhelming
- They can easily overfit
  - Which results in poor generalization
- They "hide" the logic behind the decisions
  - We don't have good visualization / inspection tools for DL yet
  - It's difficult to trust their results
  - They can spot subtle clues (image acquisition artifacts) that should not be taken into the account
- They don't adapt well to new applications
  - They strongly rely on the training set samples and their quality
  - New applications require retraining, fine tuning or even architecture redesigning
- They are stochastic
  - There are no guaranties on convergence, good performance or repeatability
- Training may take hours and tuning the parameters – days, weeks or months
- They often require relatively expensive hardware
- The amount of DL variations and novelties may be overwhelming

Tools for DL development

- TensorFlow
  - **Keras**
- **PyTorch**
  - fast.ai
- MATLAB
- Theano
- Caffe
- Neon
- Chainer
- …

Quick start:

- Get Python (Anaconda)
- Install TensorFlow or PyTorch
- Try “hello world” code
- You can code in Spyder (included in Anaconda)
- In TensorFlow, you can investigate your network training and performance in TensorBoard

- Google for DL tutorials for more info
- You can use Google colabs (plugin allowing you to run Python notebooks in the cloud)

Extra (easy) reading

- Convolutional Neural Networks
  - A good introduction blog (read all 3 parts)
    https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/
  - A good practical introduction to Keras
    https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/
  - A bunch of great ideas on improving your CNN's performance
    https://machinelearningmastery.com/improve-deep-learning-performance/
  - Keras plug-in for deep learning image segmentation
    https://github.com/qubvel/segmentation_models
  - LSTM explained (blogs)
    https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
    https://colah.github.io/posts/2015-08-Understanding-LSTMs/
  - Damian's thesis (read section 2.2)
    https://www.diva-portal.org/smash/get/diva2:1359483/FULLTEXT01.pdf