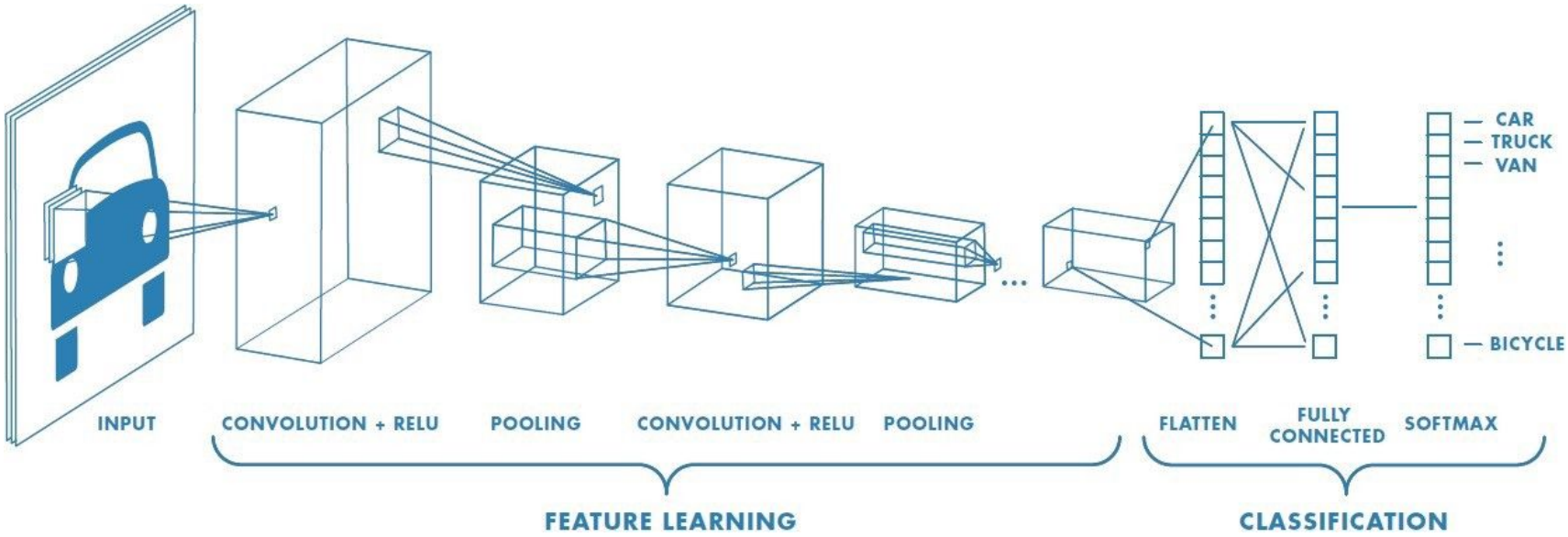
Deep Supervised Learning Architectures

University of Victoria - PHYS-555

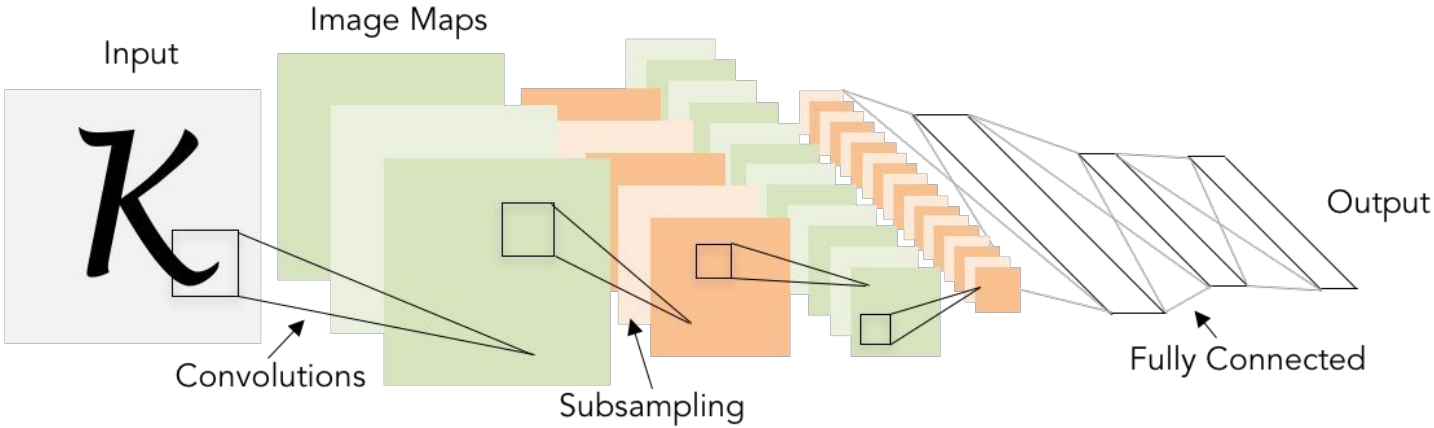
Previously: CNN building blocks



Tensors to Scalars

LeNet-5 (LeCun 1998)

- Composition of:
  - 2 convolution layers with 5x5 filters
  - 2 pool layers (subsampling) with stride of 2
  - 2 FC layers



ImageNet

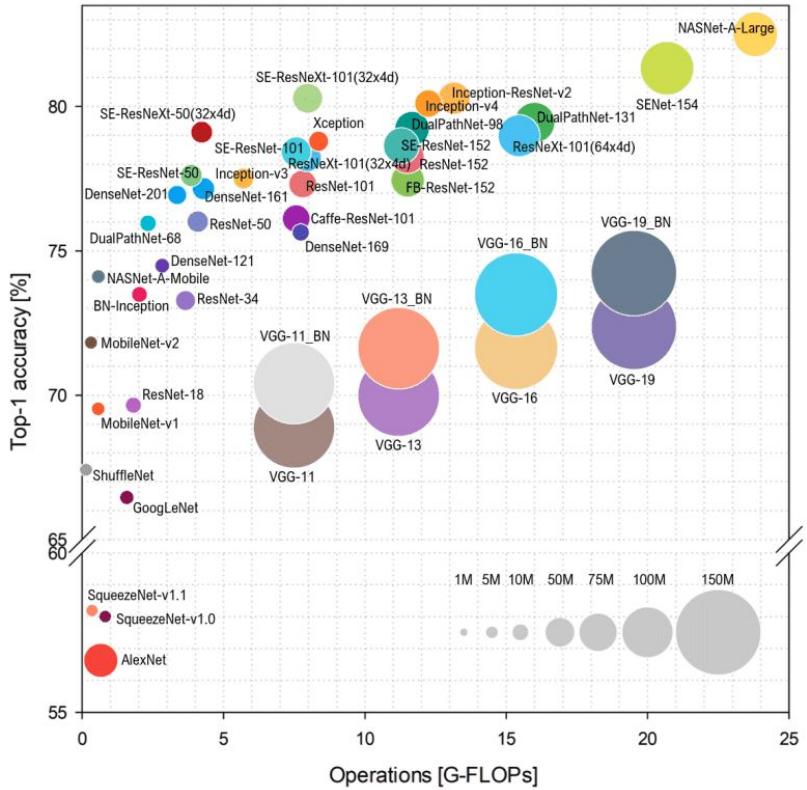
Image Classification Challenge:

- 1,000 object classes
- 1,431,167 images

  www.image-net.org

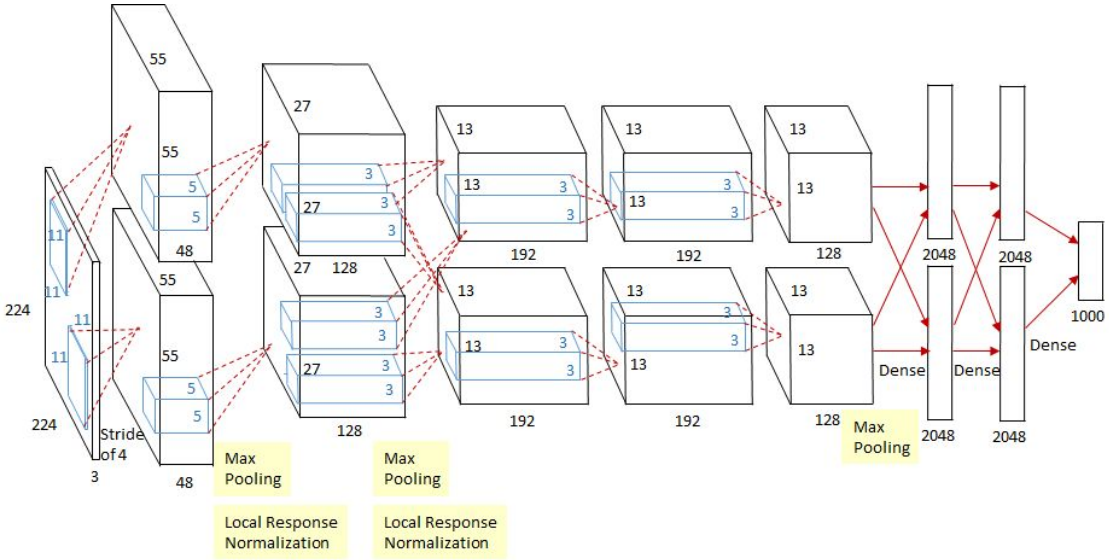
- Benchmark research on new architectures

ImageNet: A Benchmark for Architecture



AlexNet (Krizhevsky et al. 2012)

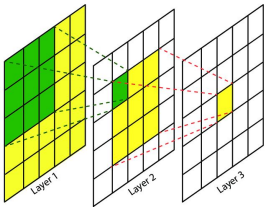
- Applied on ImageNet (224x224x3)
- Composition of:
  - 5 Convolutional layers
  - 3 FC layers
  - Use ReLU instead of tanh
  - Use dropout in FC layers
  - 60M parameters
- Ensemble of ConvNets
- Wins ImageNet challenge
- Start the DL revival

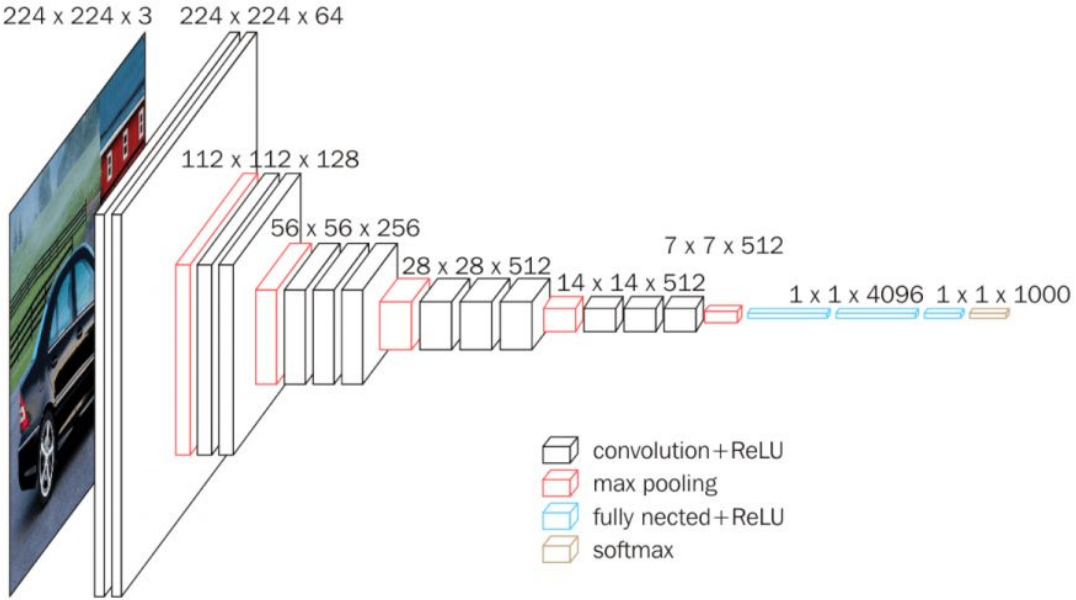


VGGNet (Simonyan & Zisserman 2014)

Deeper network with:

- 13 Convolutional layers
- 3 FC layers
- Reduce kernels to 3x3
- FC(4096) generalizes to other tasks

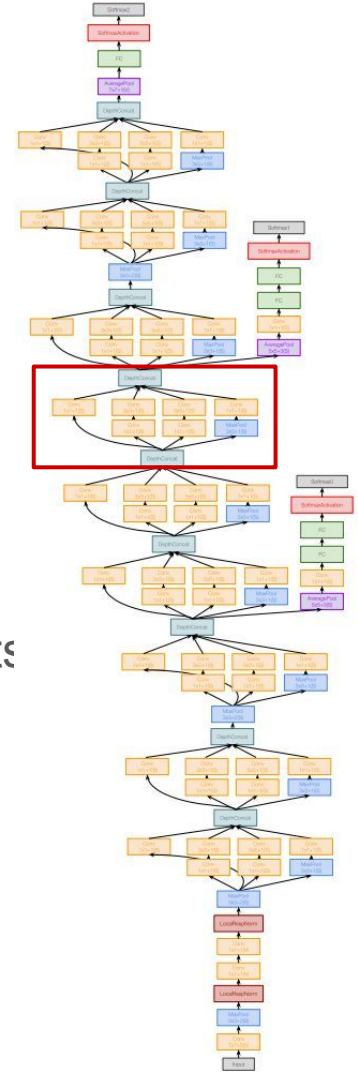




Visual Geometry Group Oxford

GoogLeNet (Szegedy et al. 2014)

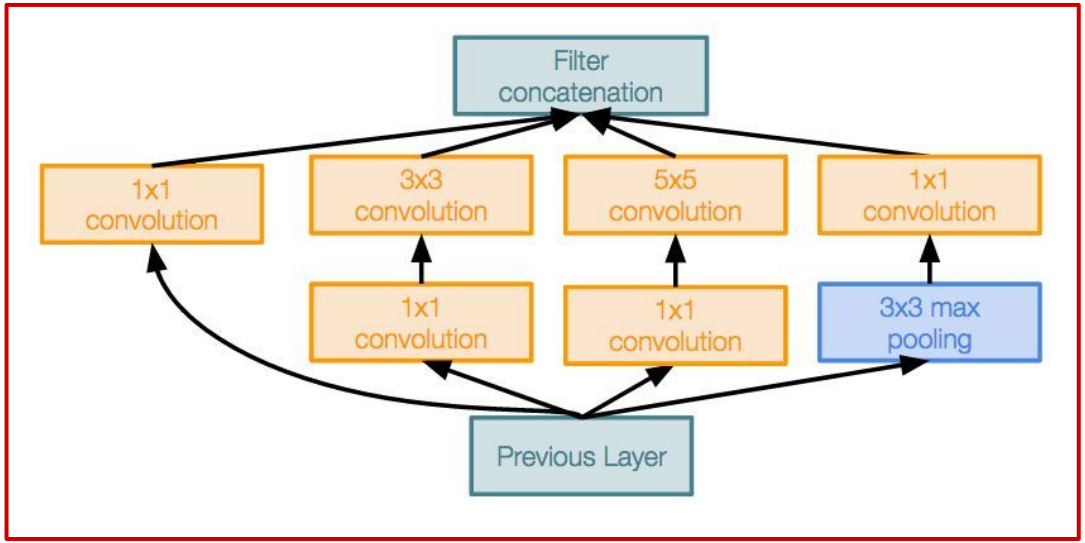
- Include 9 inception blocks
- 22 layers
- Global average pooling at the end
- Very deep -> vanishing gradient:
  - include intermediate classifiers
- Now on version v4, including tricks from other architectures
- Main benefits: include multiple filter sizes in a single layer, reducing computation while being better at recognition of objects with different sizes



Inception module

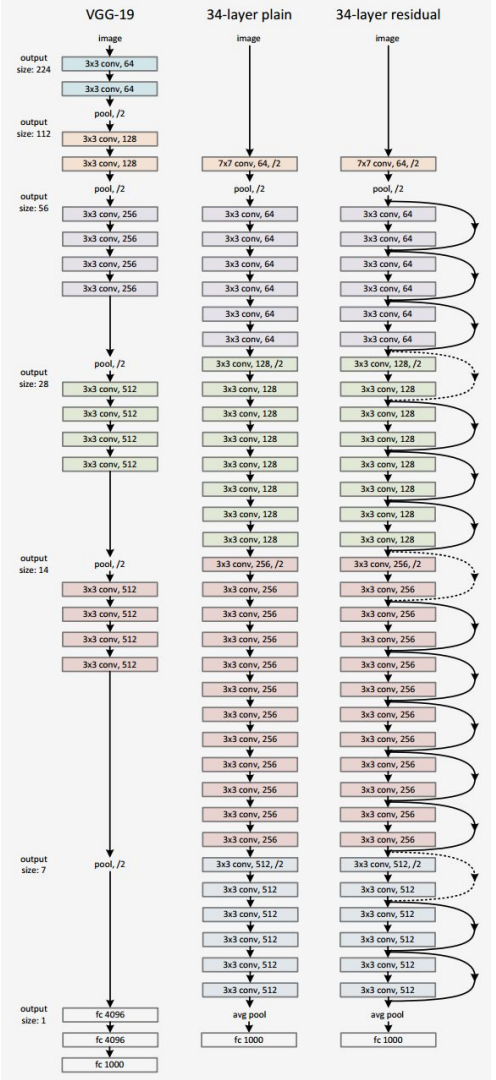
Design a good local network topology (network within a network) and then stack these modules on top of each other

Code for a inception block in PyTorch?



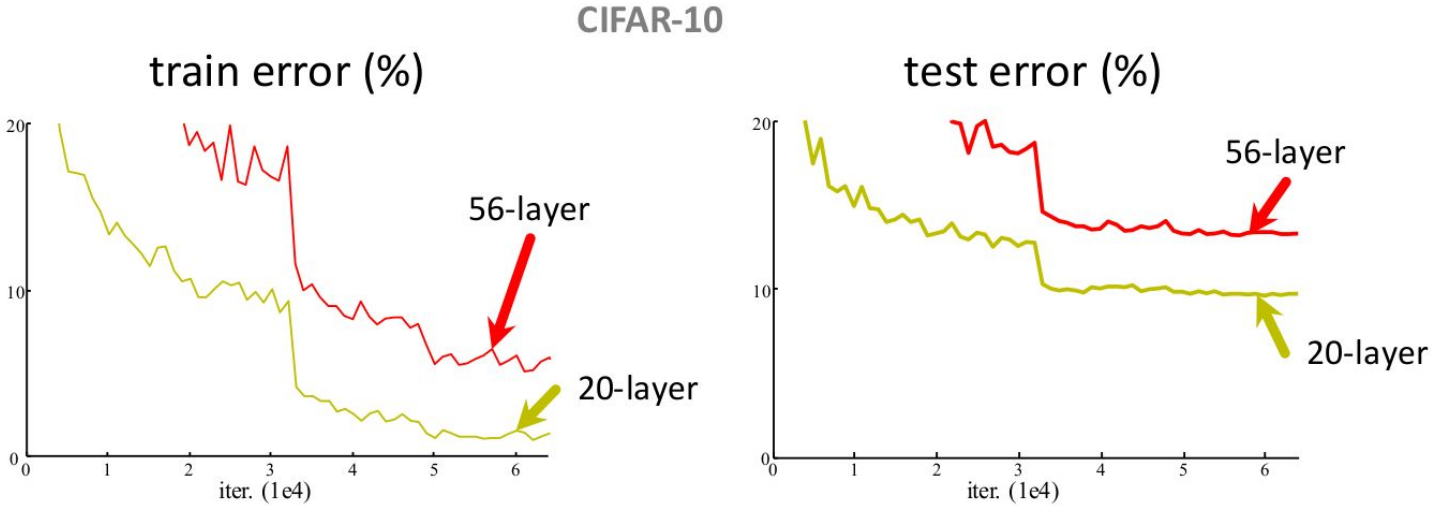
ResNet (He 2016)

- Include residual connections
- Less parameters (25M vs. for 138M VGG)
- Very deep: 34, 50, 101 and 152 layers
- Good optimization properties
- Fully convolutional
- Include residual connections and batch normalization



How can we keep going deeper?

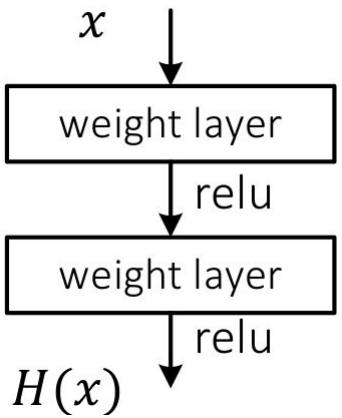
He 2016
CVPR



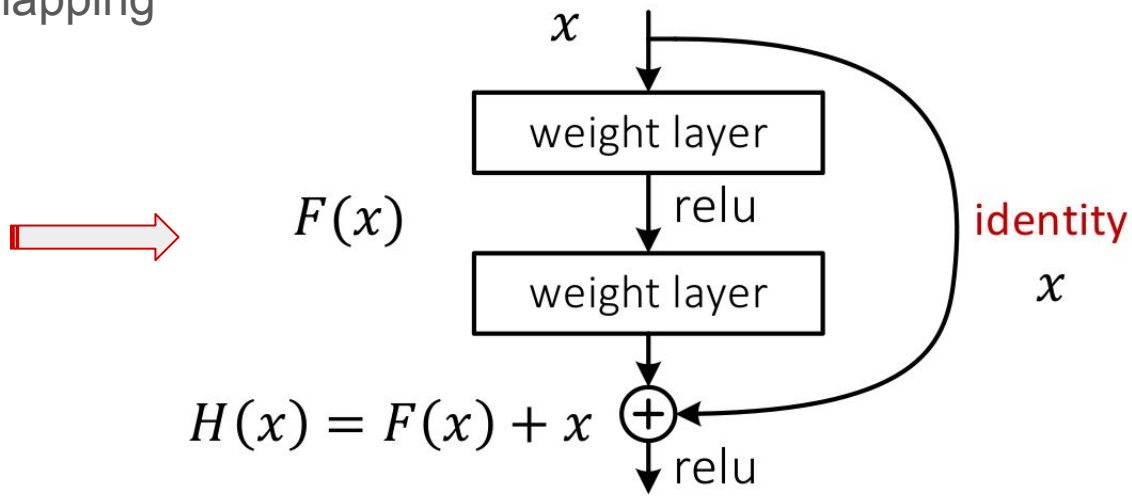
- ConvNets with 3x3 filters
- 56-layer net has higher training error than 20-layer net!
- General phenomenon observed in many datasets
- We can not stack simple layers and improve performance

Residual connection

- Optimization difficulties: solvers cannot find the solution when going deeper
- Solution: use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

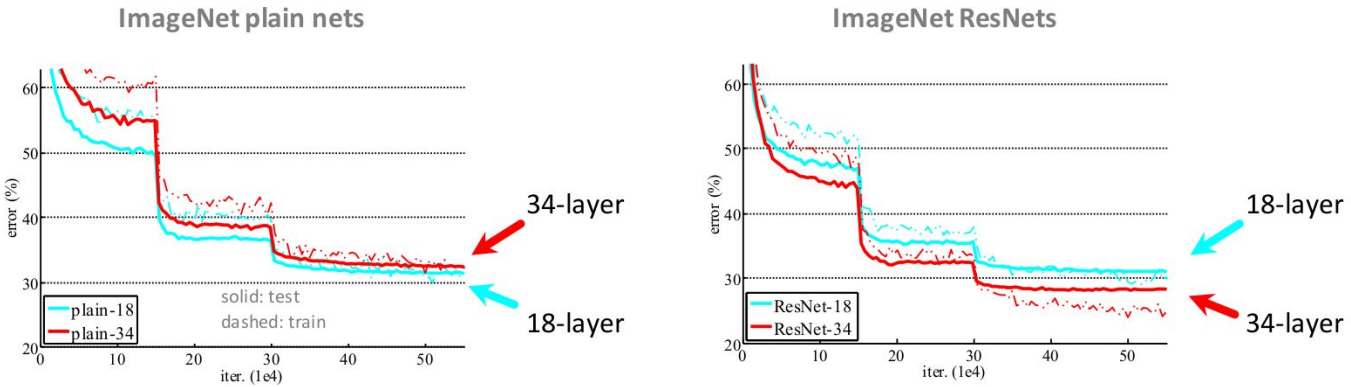


Plain Connection



Residual Connection
aka "Skip Connection"

Residual connection: effect on ImageNet



- ConvNets with residual connections can be trained with less difficulties
- Deeper ResNets have both lower training and testing errors

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

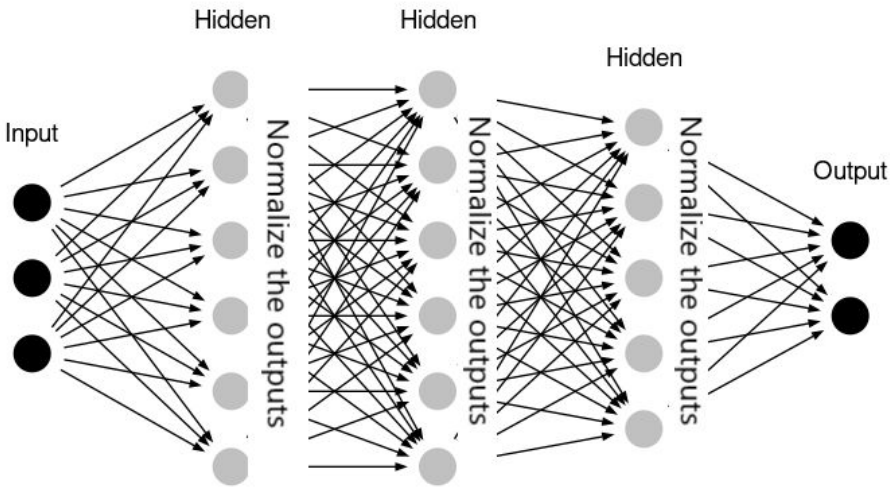
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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



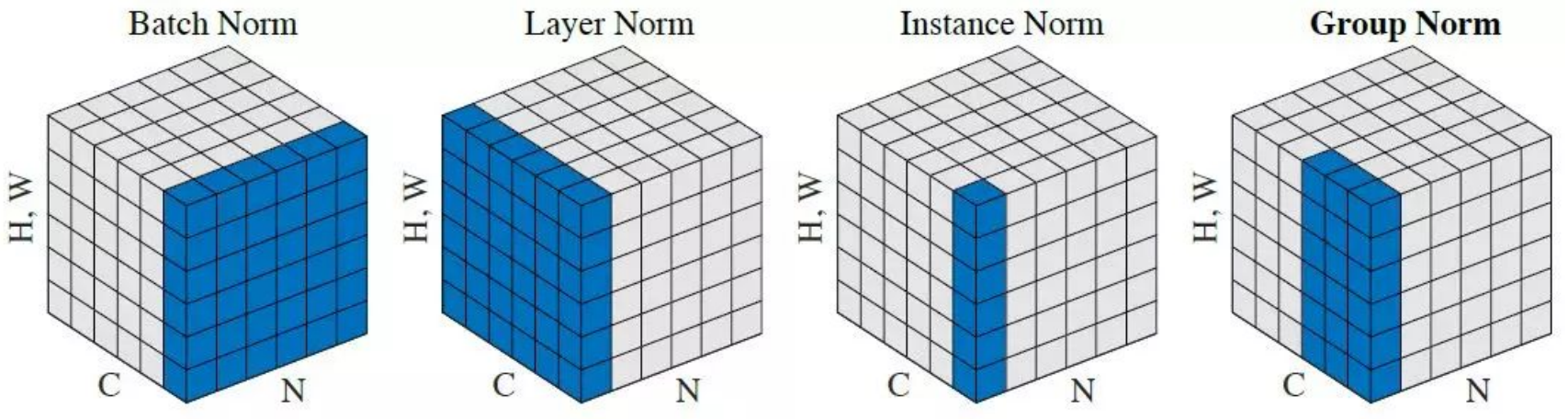
Batch Normalization, Loffe & Szegedy, arXiv:1502.03167

Batch Normalization

- Gradients are easier to flow through networks
- Optimization less sensitive to some hyperparameters
  - can use higher learning rates → faster convergence
  - less sensitive to initialization
- smoother optimization landscape (arXiv:1805.11604).
- “implicit regularizer”: affects network capacity and generalization
- Typically inserted before activation, but sometimes debated

Behaves differently at training and test time, since test time does not have "mini-batches".

Batch, Layer, Instance, and Group Normalization



There are various types of normalization layers that use mean and variance statistics along different dimensions. Only BatchNorm is batch dependent.

More details in: Group Normalization, Wu & He, arXiv:1803.08494, CS231n Spring 2019, Lecture 7

Automating Design

Hyperparameter Search

Select several values for each hyperparameter, then train for each (in parallel!)

Typically scales log-linearly, examples:

Grid Search:

- Sample weight decay from $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$
- Sample learning rate from $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$

Random Search:

- Sample weight decay from log-uniform ~ $[10^{-4}, 10^{-1}]$
- Sample learning rate from log-uniform ~ $[10^{-4}, 10^{-1}]$

Beyond 3 dimensions: look for Bayesian Optimization.

Bergstra & Bengio

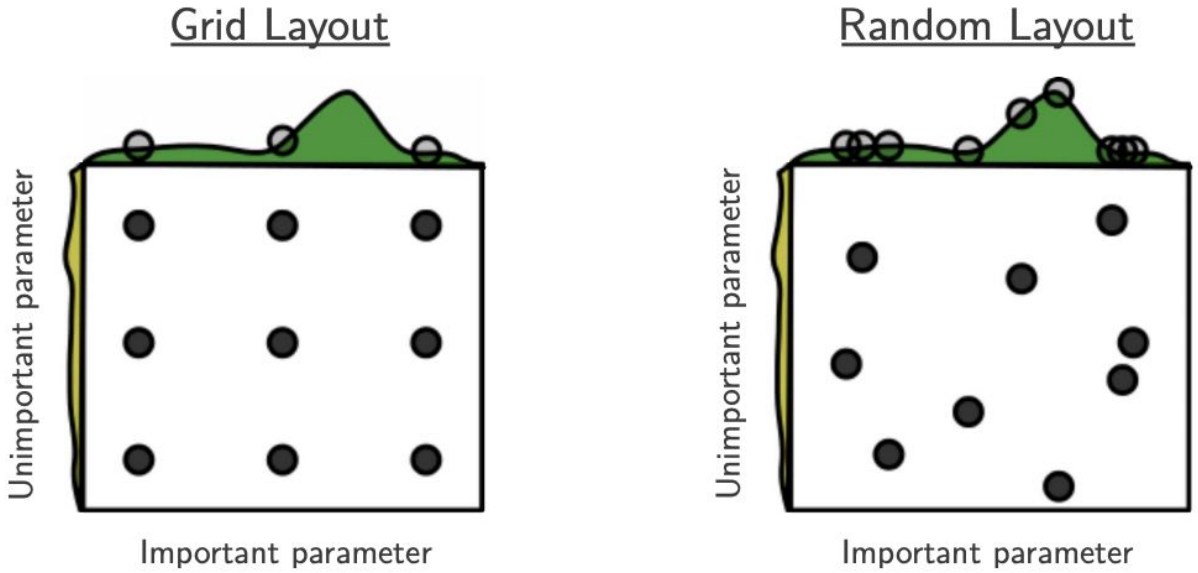


Figure 1: Grid and random search of nine trials for optimizing a function $f(x,y) = g(x) + h(y) \approx g(x)$ with low effective dimensionality. Above each square $g(x)$ is shown in green, and left of each square $h(y)$ is shown in yellow. With grid search, nine trials only test $g(x)$ in three distinct places. With random search, all nine trials explore distinct values of g. This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

Bayesian Optimization

Intuition: We want to find the peak of our true function

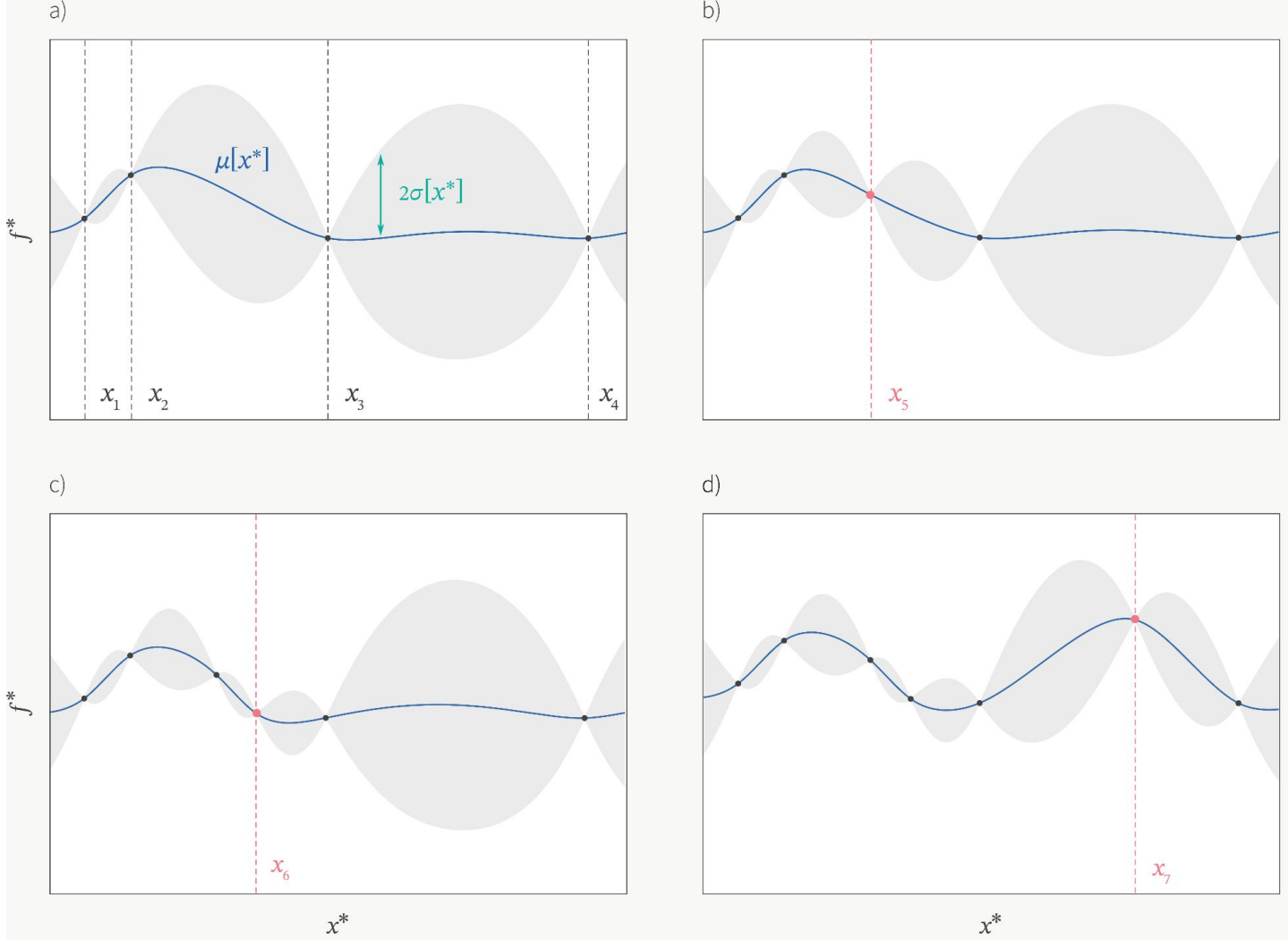
e.g. accuracy as a function of hyperparameters

To find this peak, we will fit a Gaussian Process to our observed points and pick our next best point where it “believe” the maximum will be.

This next point is determined by an acquisition function - that trades of exploration and exploitation.

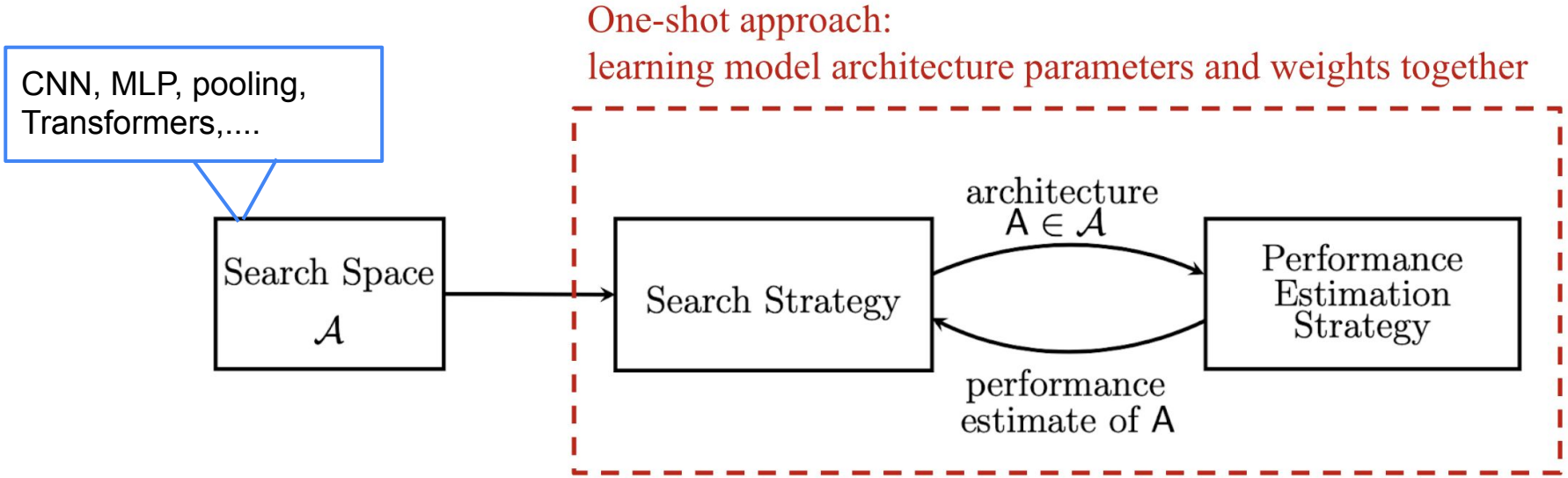
Remark: can be used for computationally expensive physics simulations

See this tutorial for example


a)
b)
c)
d)
$\mu[x^*]$
$2\sigma[x^*]$
f^*
x^*
x_1
x_2
x_3
x_4
x_5
x_6
x_7

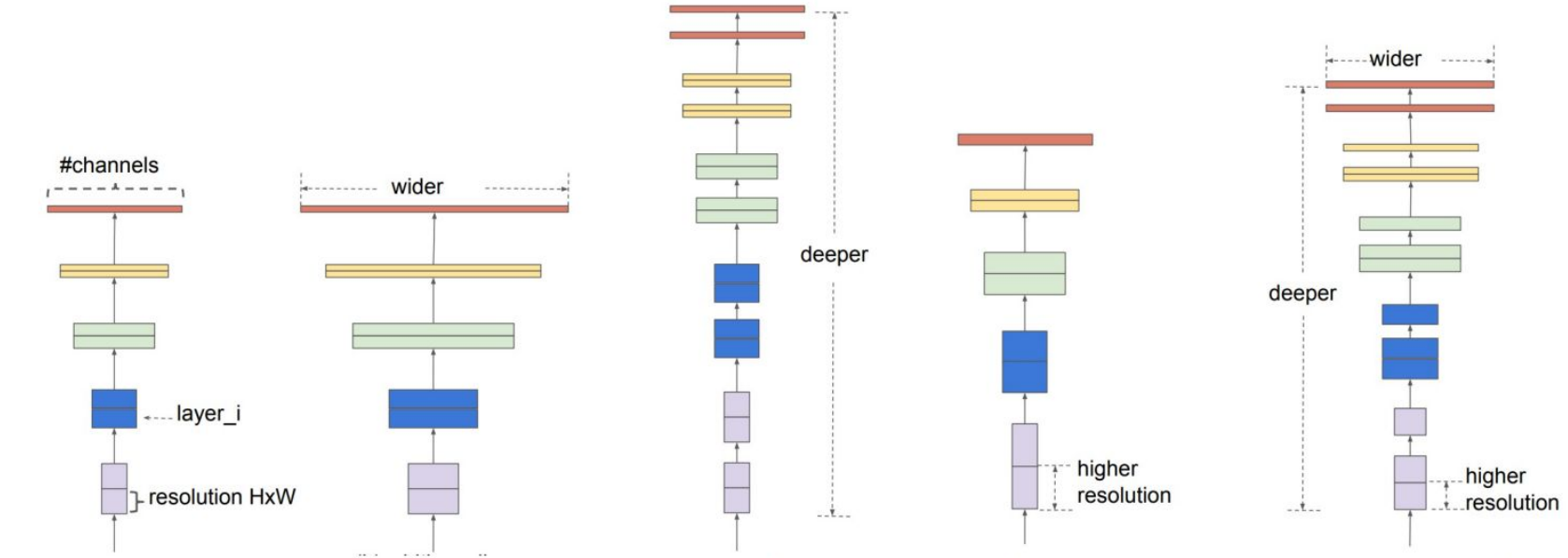
Neural Architecture Search

Hyperparameter search more extreme.



See this explanation.

How can we scale up a model to improve performance?



Baseline Architecture

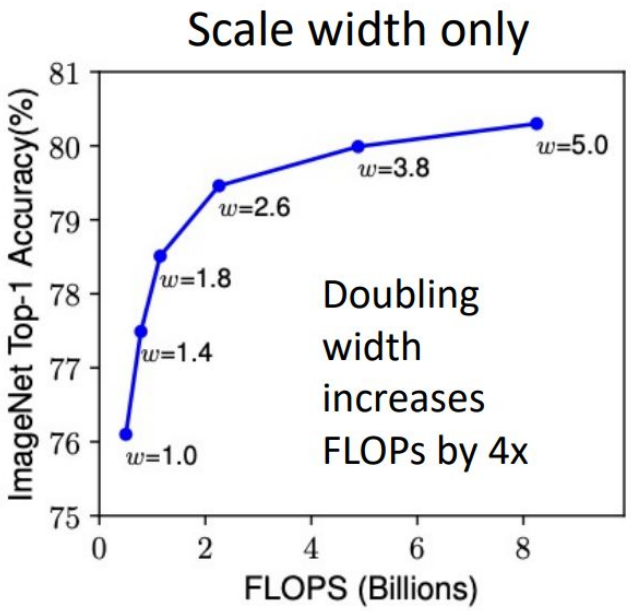
Width : Increase channels in all layers

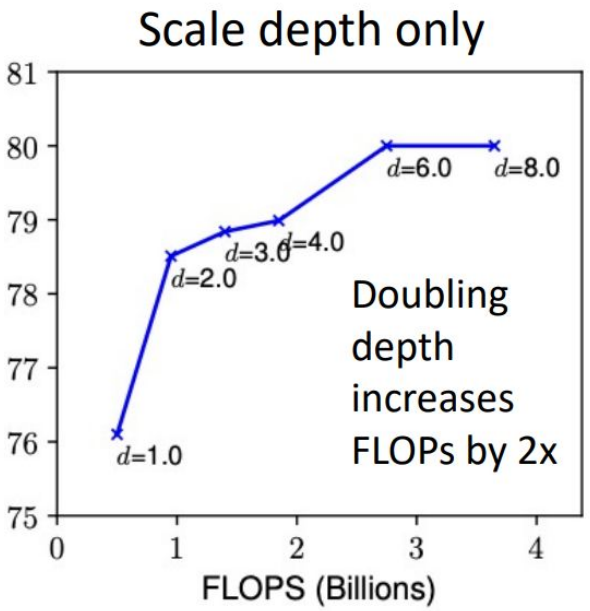
Depth: Use more layers

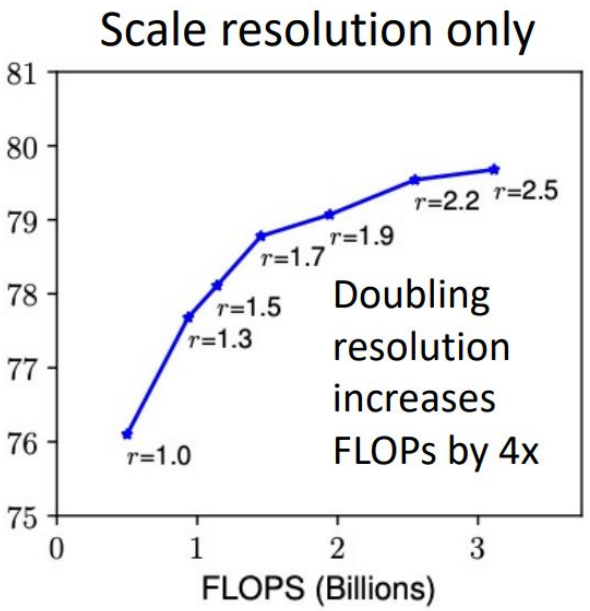
Resolution: Higher res input image

Compound: Scale all jointly

Model Scaling with EfficientNet



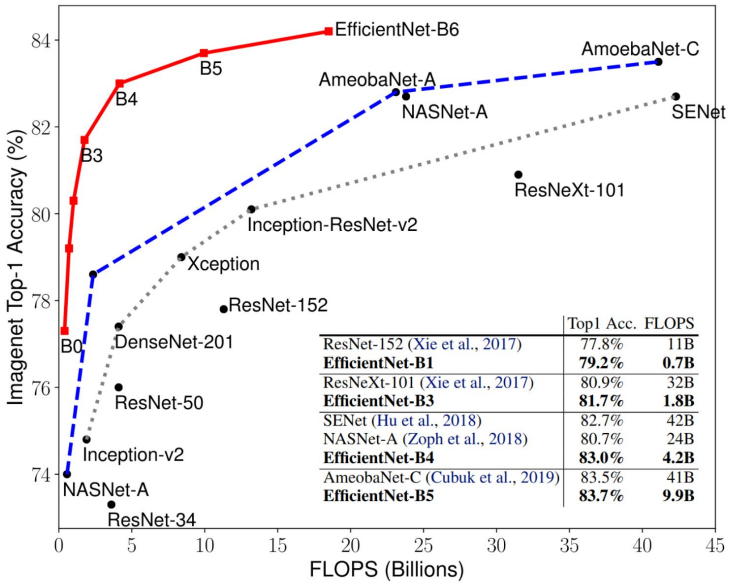




EfficientNet

- Use Neural Architecture Search
  to get initial architecture
- Find optimal scaling architecture factors
- Adjust for FLOPS and accuracy

Now on version v2 with more efficiency.



	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**79.2%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.7%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**83.0%**	**4.2B**
AmeobaNet-C (Cubuk et al., 2019)	83.5%	41B
EfficientNet-B5	**83.7%**	**9.9B**

Figure 5. **FLOPS vs. ImageNet Accuracy –** Similar to Figure 1 except it compares FLOPS rather than model size.

Recent Trends on Architecture Evolutions Vision

- Transformer-based models for vision: e.g. ViT
- MLP-like architectures with patches: e.g. MLP-Mixer
- Revisit the classic ResNet with many new tricks: e.g. ResNet-RS
- Efficient search of design space: e.g. RegNet
- New CNN inspired from transformers: ConvNext

Remember data is often a more limiting factor than architectures!

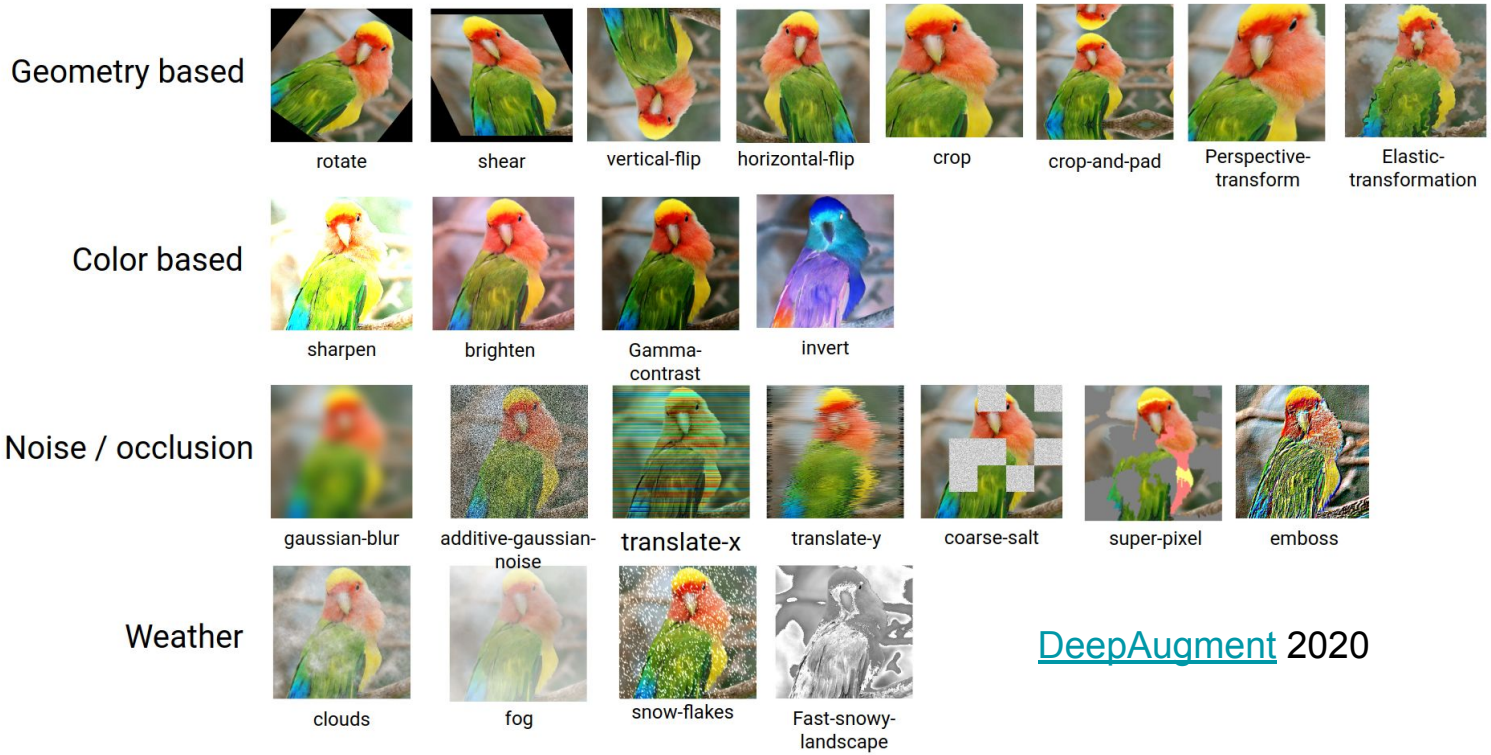
Not enough data?

What are the solutions?

- Collect more data
- Synthetic data
- Artificially increase the data set with generating new data with simple transformations

 Data Augmentation

Data augmentation

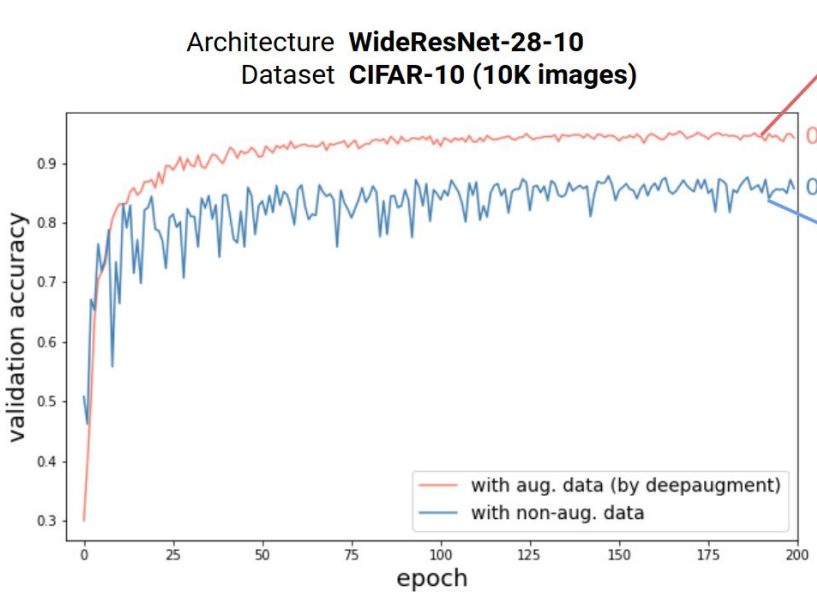


DeepAugment 2020

Data augmentation example

Data Augmentation Libraries:
- Augmentor
- AugLy

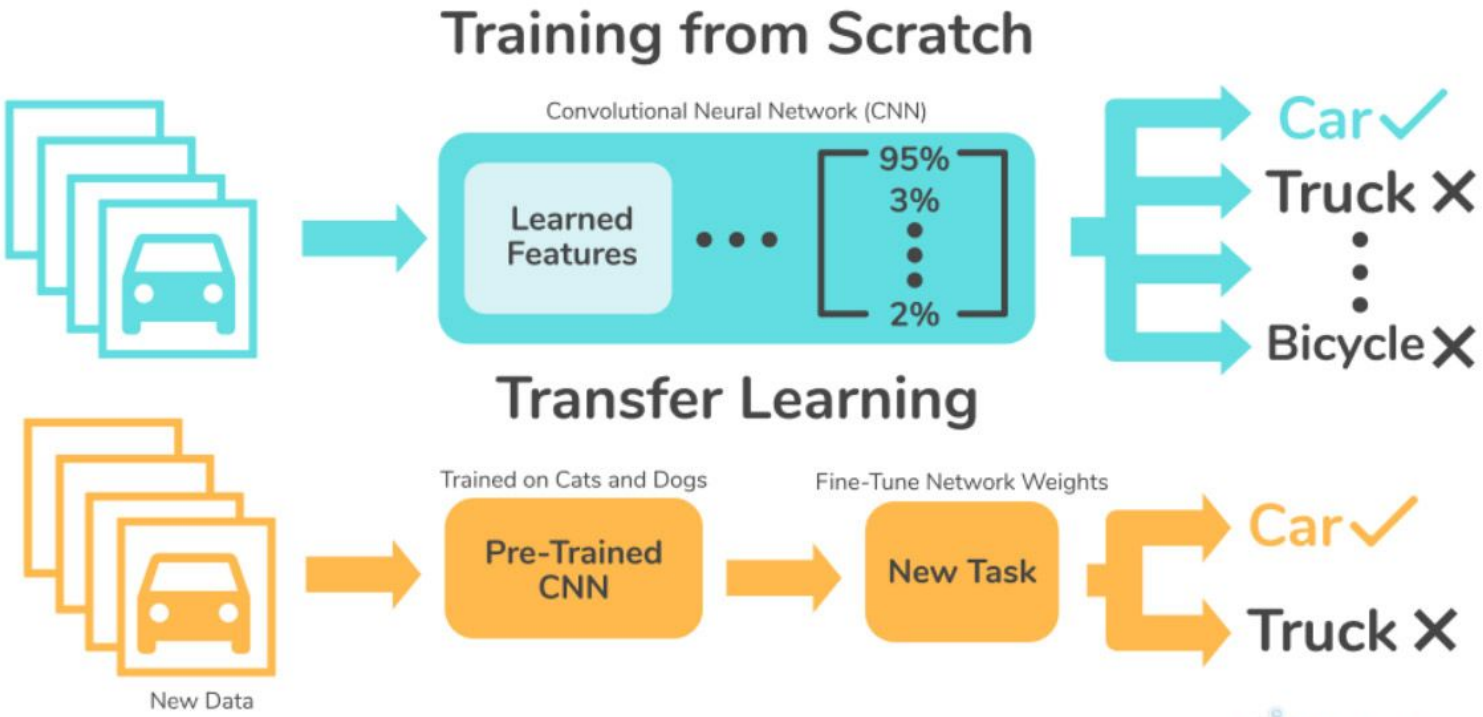
DeepAugment applied



8.5% accuracy increase
= **60%** reduction in error

DeepAugment not applied
(no augmentation)

Transfer Learning: Another solution for small data sets



Pre-trained Models

- Training large models can take weeks even with GPUs.
- Models tend to learn similar low-levels representations
- Many large models trained on large data sets such as ImageNet or NLP data sets are public, but not always.
- Once trained, how can we reuse them?
  - for initialization of our smaller models
  - for feature extraction

⇒ Transfer learning

A Simple Application of the Transfer Learning Idea

Source dataset

Huge, multi-million samples

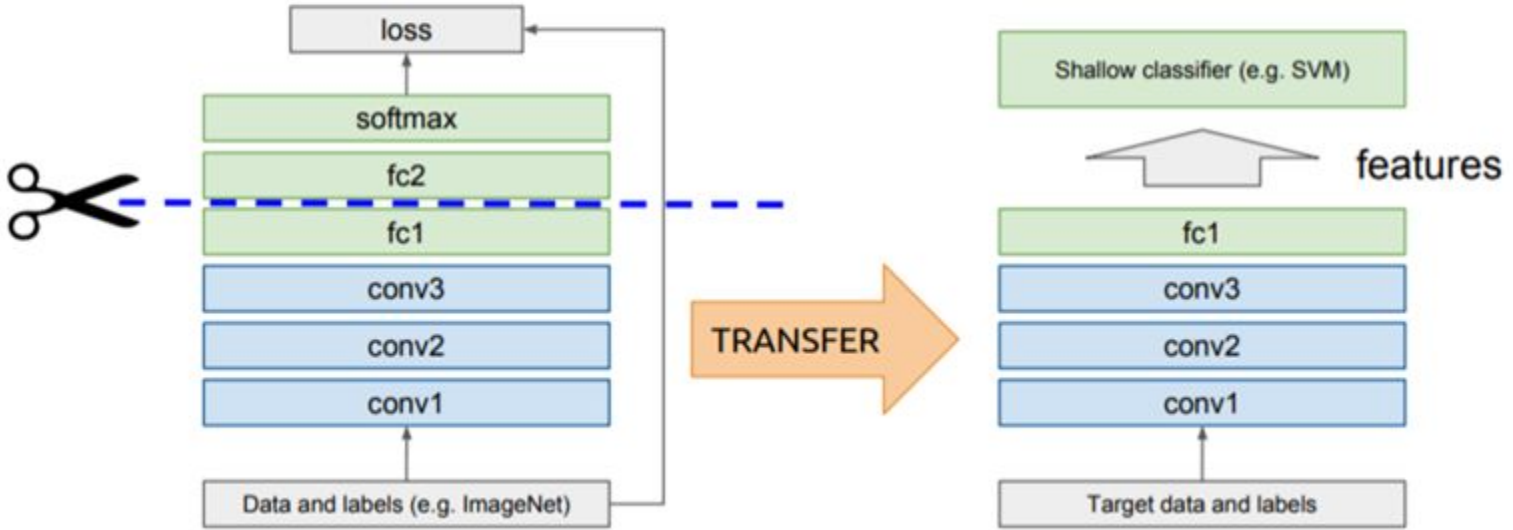
Target dataset

Smaller dataset and/or few labels.

Could we reuse the network trained on the source?

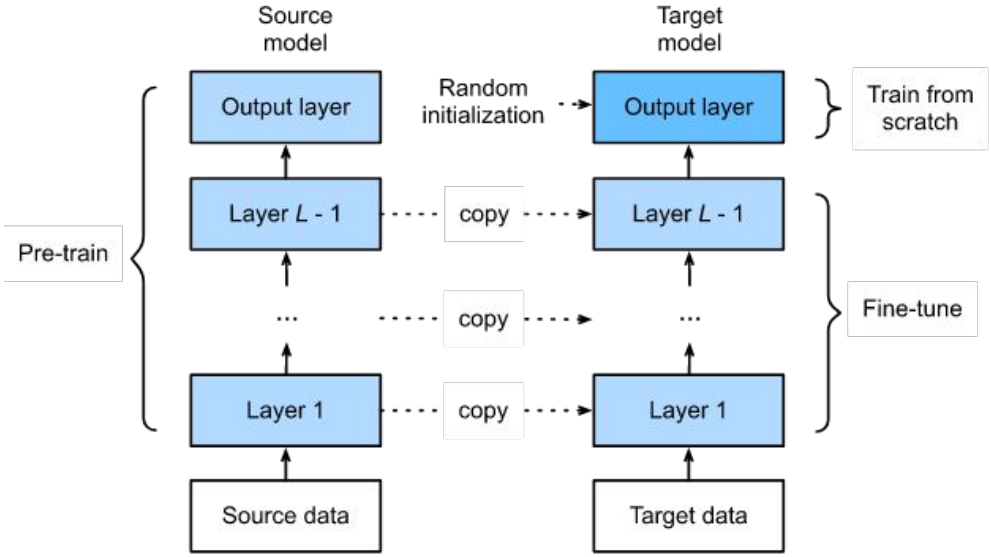
Pre-trained Feature Extraction

- Remove last layer of a pre-trained network, rest is fixed and feature extractor
- Train the model on target data with these features
- Often better than training a new model from scratch on target data.



Transfer Learning: Fine Tuning

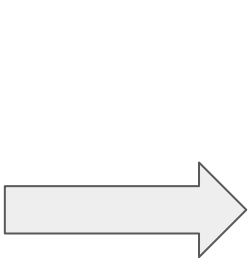
- Fine tune the weights of the pre-trained network
- Continue backprop
- Mild learning rate for pre-trained layers
- Aggressive learning rate for new layers

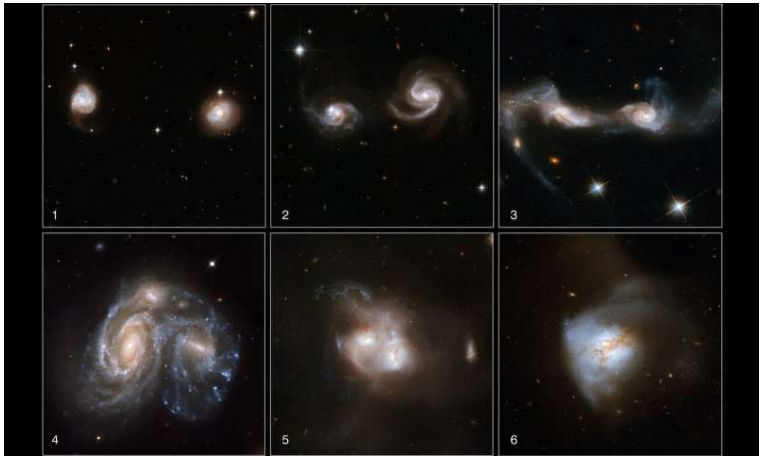


Where to Find Pretrained Models

- In PyTorch:
  - torchvision
  - torchtext
  - torchaudio
  - …
  - also check the timm python module, hard to beat.
- In Tensorflow:
  - tensorflow hub
- Model hub in huggingface is probably the most complete (as of 2023)

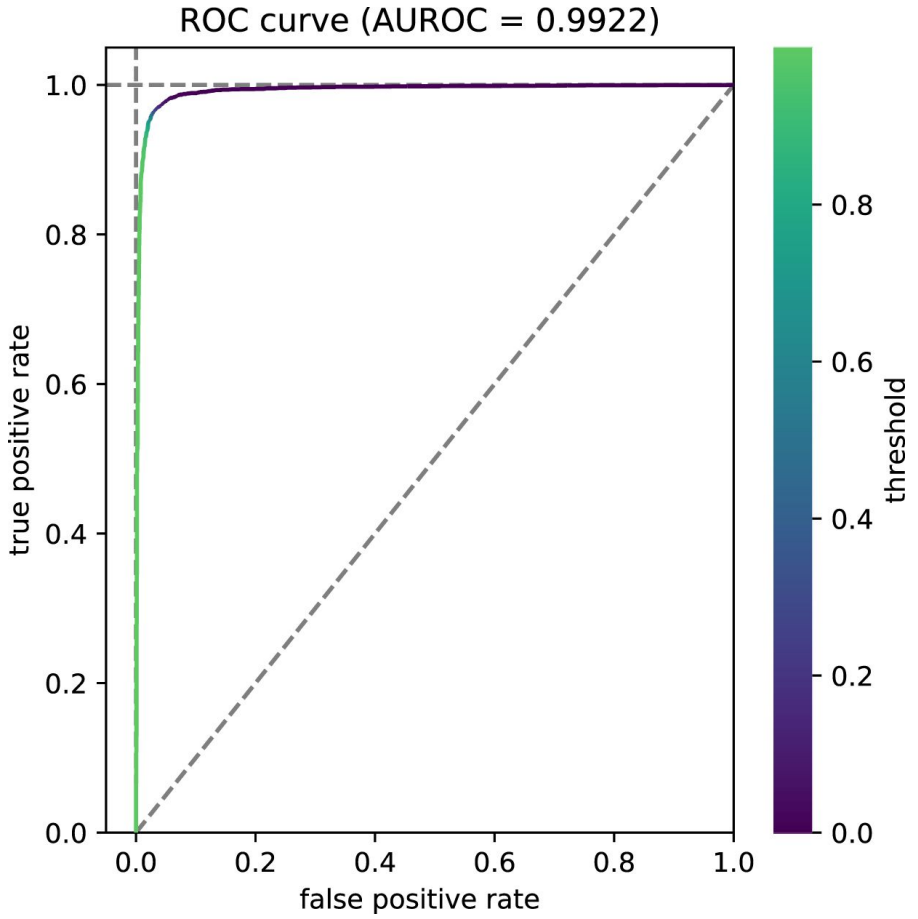
Transfer learning: does it work?

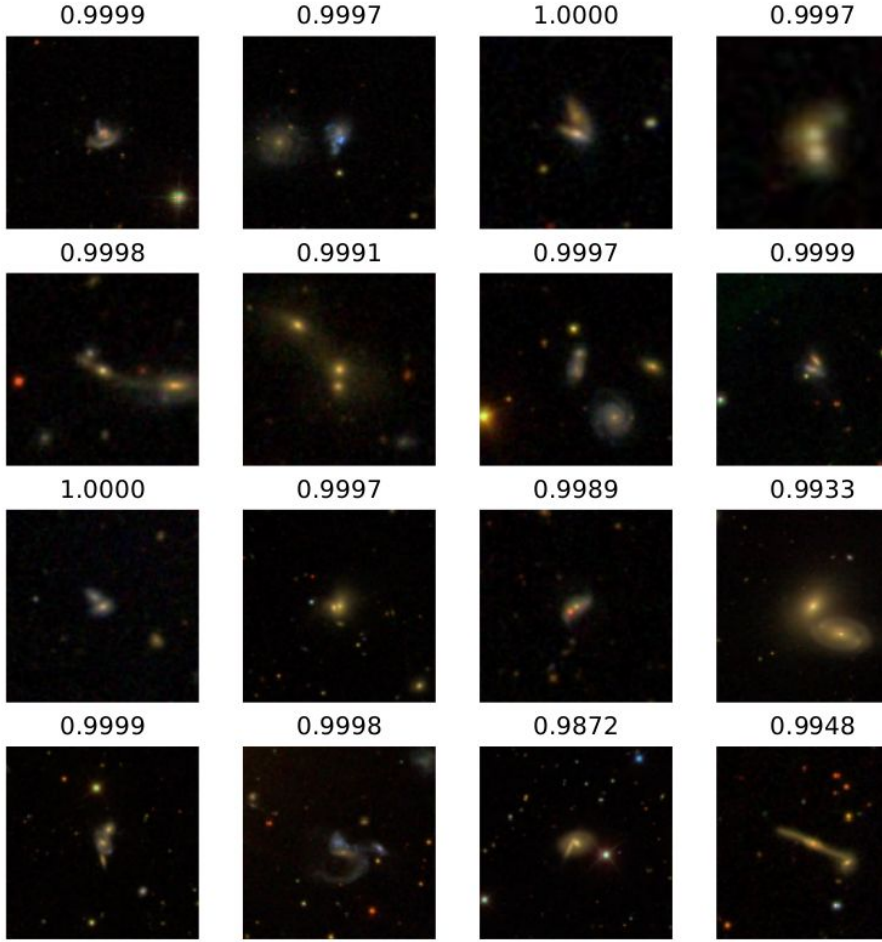




Transfer learning: e.g. galaxy merger detection

Ackerman et al. (2018)

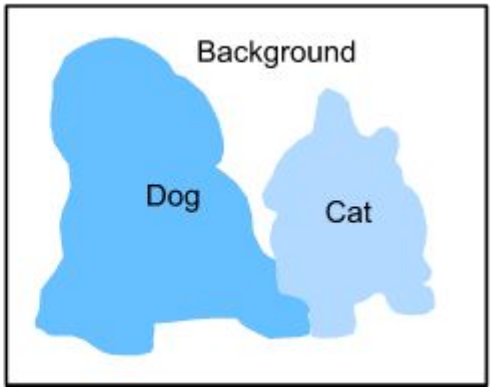

ROC curve (AUROC = 0.9922)
true positive rate
false positive rate
threshold
0.0
0.2
0.4
0.6
0.8
1.0


0.9999
0.9997
1.0000
0.9997
0.9998
0.9991
0.9997
0.9999
1.0000
0.9997
0.9989
0.9933
0.9999
0.9998
0.9872
0.9948

Tensor to Tensor

Semantic Segmentation

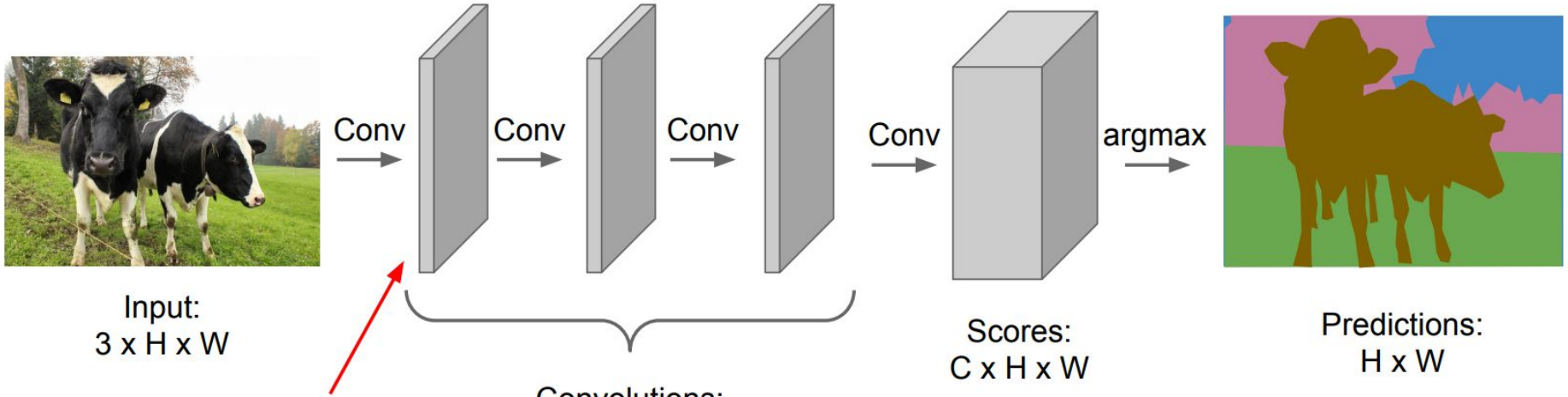
- Partition an image into semantic categories regions
- Predict object at the pixel level



- Recast segmentation into a CNNs with output a classification image

Fully Convolutional Network: Follow the Pixel

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!



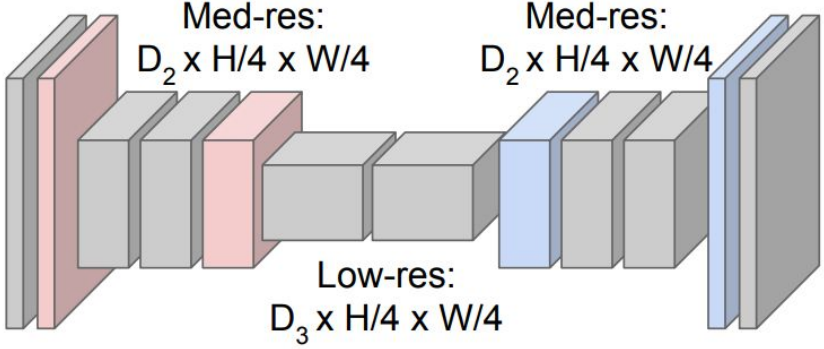
Fully Convolutional network: bottleneck it

Downsampling:
Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling:
???

Input:
3 x H x W



High-res:
D_1 x H/2 x W/2

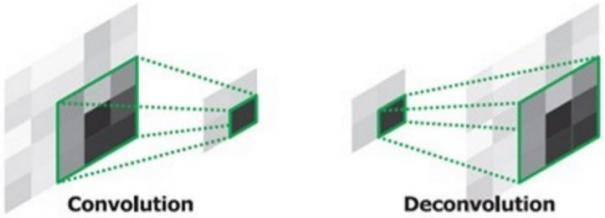
High-res:
D_1 x H/2 x W/2

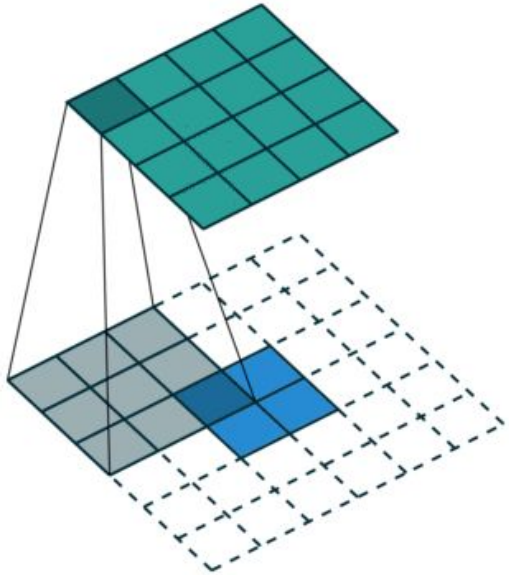
Predictions:
H x W

Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015
Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

Transposed Convolution

- Allow to upsample to the next layer
- Same as Upsample+Convolution
- Forward and backward are swapped
- Sometimes mistakenly named "deconvolution".

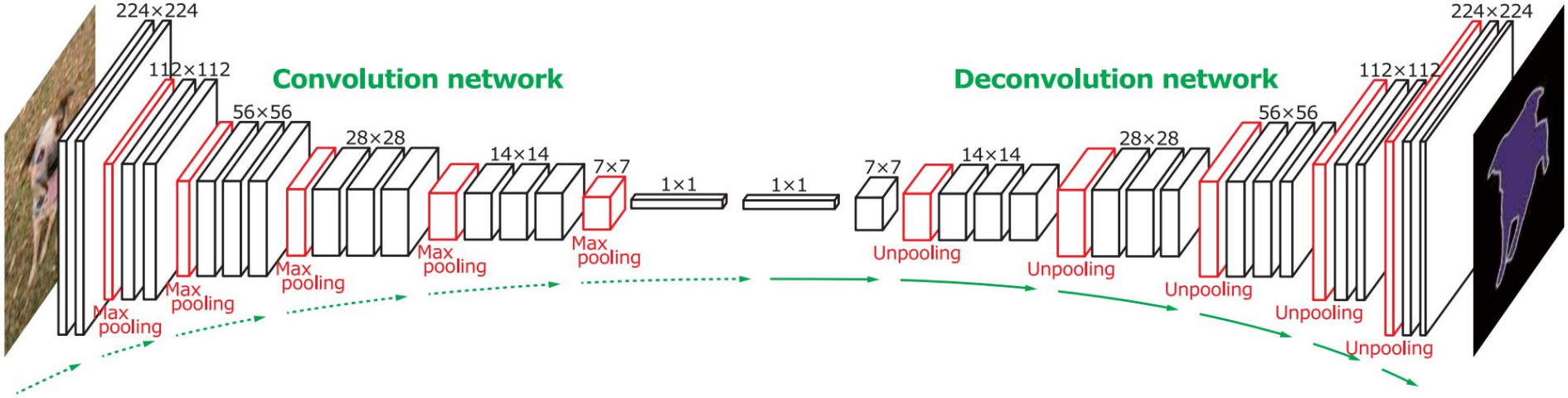




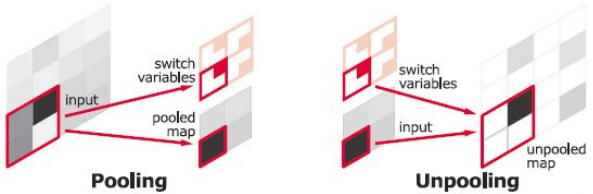
- Check for [checkerboard artifacts]

[Convolution Arithmetic - Dumoulin]

Segmentation: FCN (Noh, Hyeonwoo 2015)

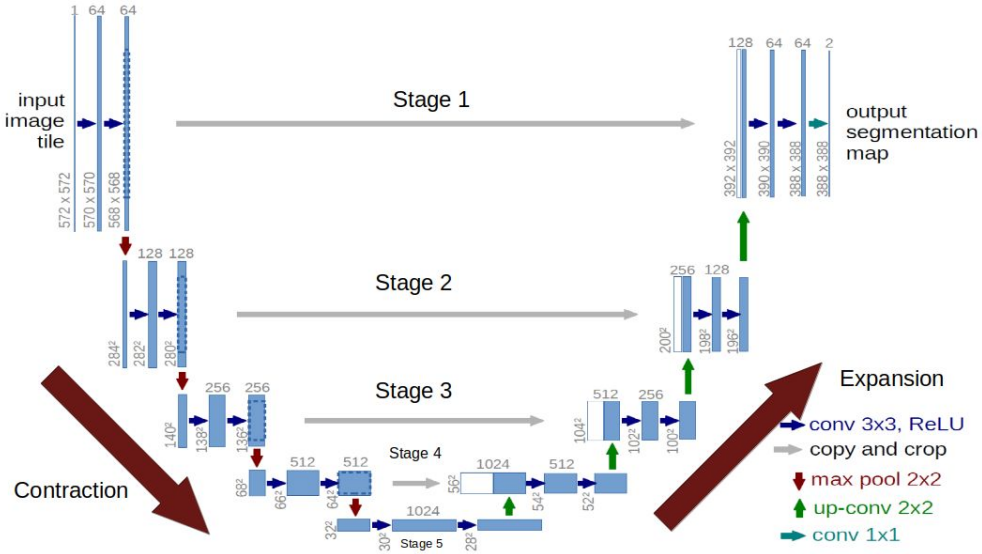


- Combine DeconvNet with Unpooling
- Number of channels: number of categories



UNet (Ronneberger et al. 2015)

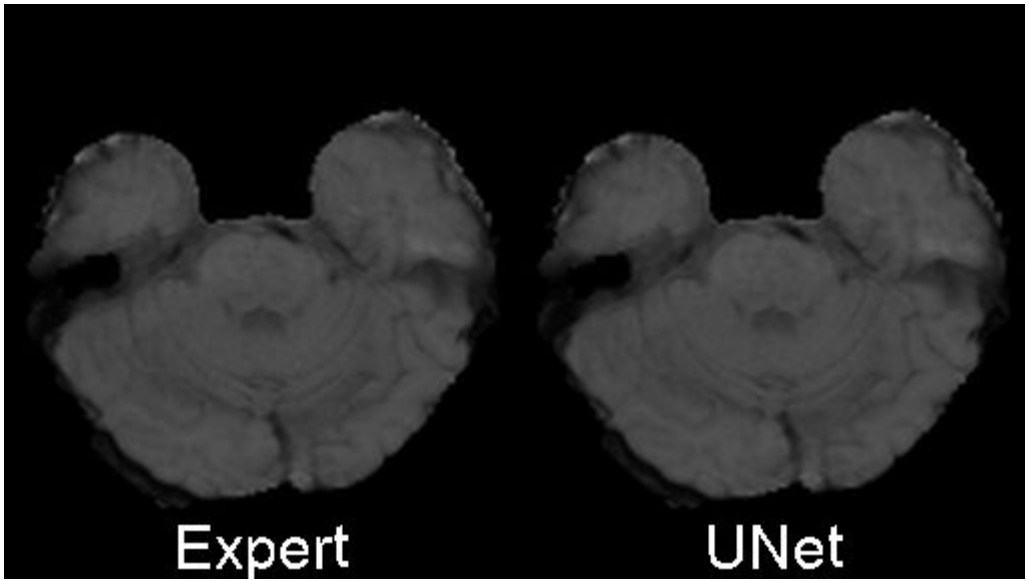
- builds on FCN architecture.
- symmetric contraction and expansion paths
- concatenation of HR features in contraction to the unsampled features in expansion
- skip connections
- allow localisation
- popular in science



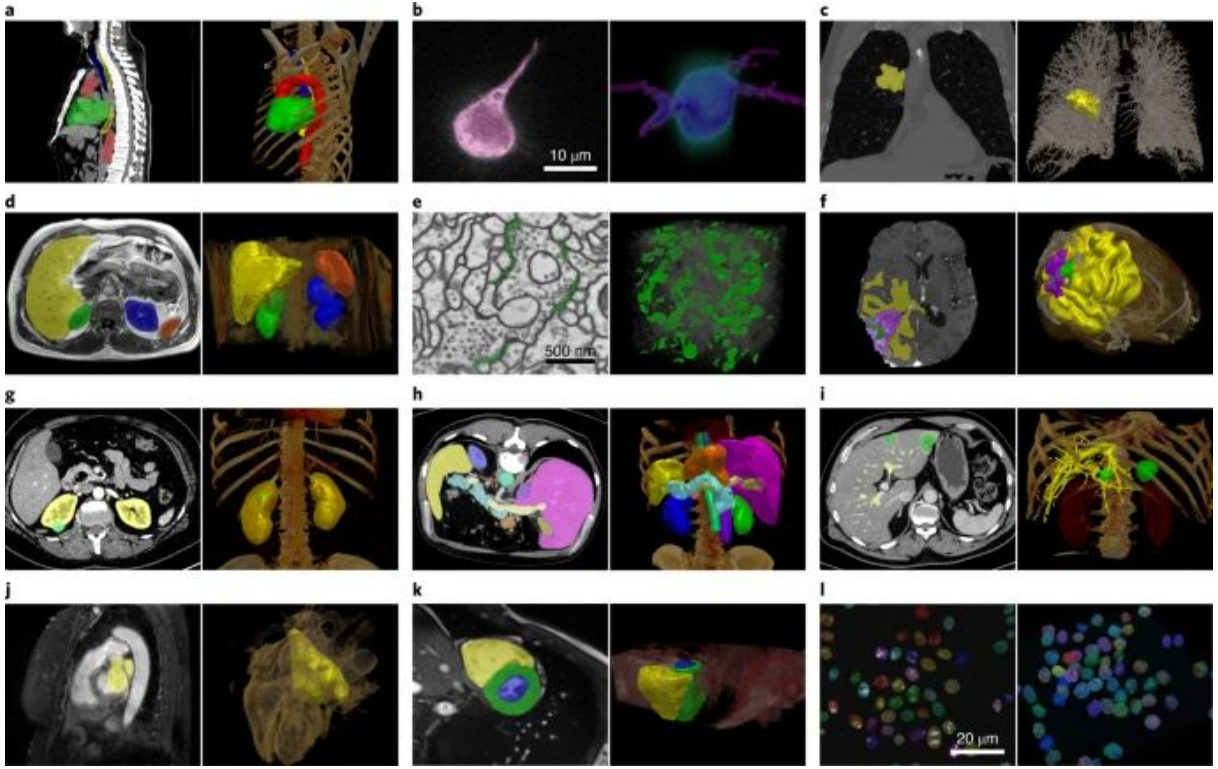
UNet: brain imaging

- Volumetric (3D) segmentation
- detection of brain tumor

Cicek et al. (2016)

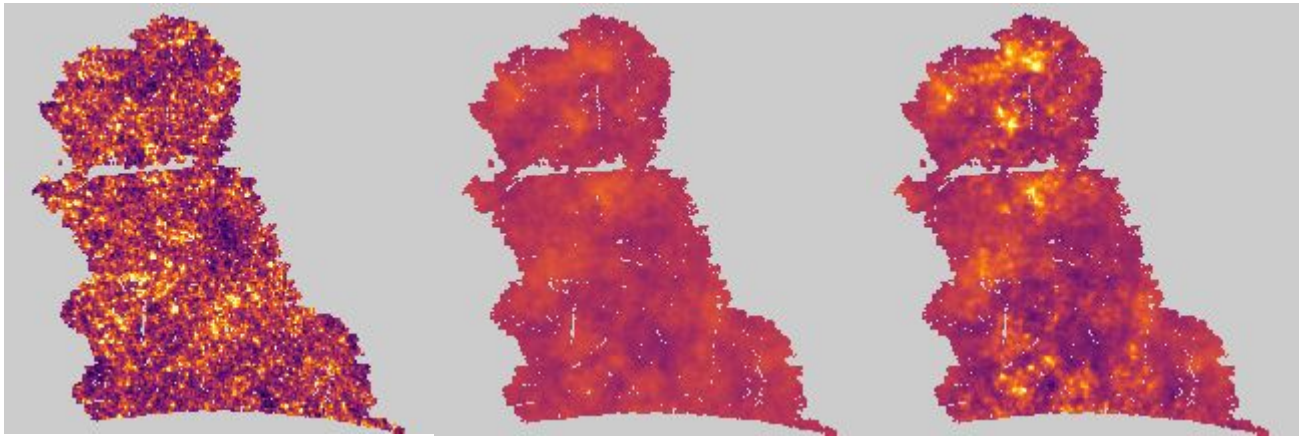


nnUNET : end-to-end flexible UNET for medical imaging



Can work with 512x512x512 voxels

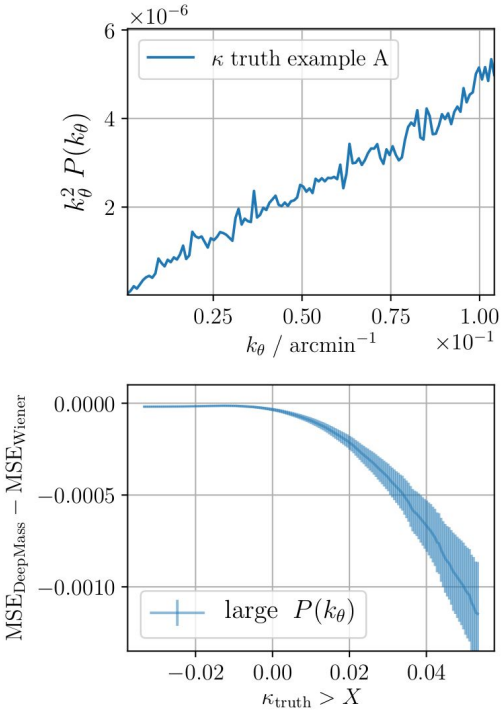
UNet: mass maps in cosmology



Ground truth

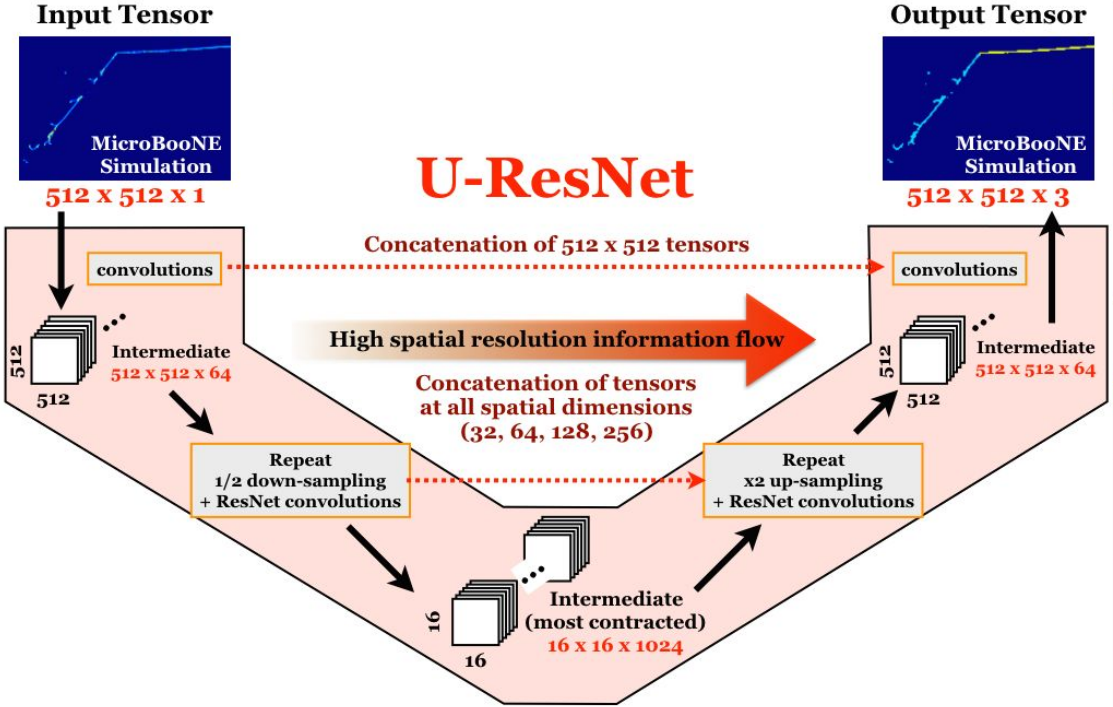
Wiener filter

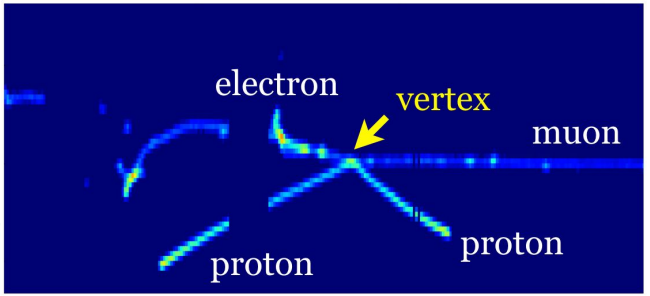
UNet

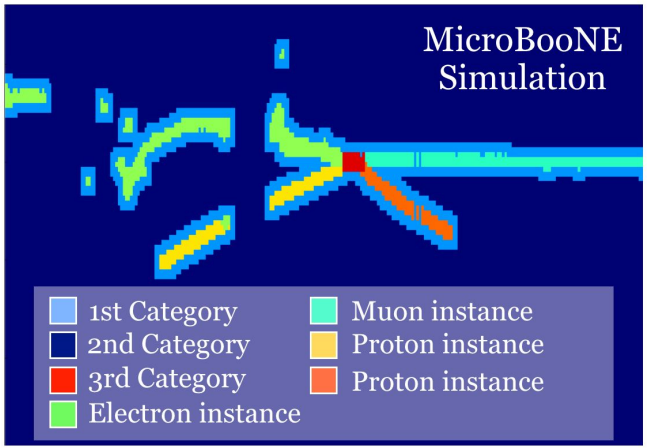


[Jeffrey et al (2019)]

UNet: particle identification







MicroBoone (2018)

Deep Learning for Tabular Data

When should we use deep learning on tabular data?

Gradient Decision Tree-based method (GBT) are already very strong:

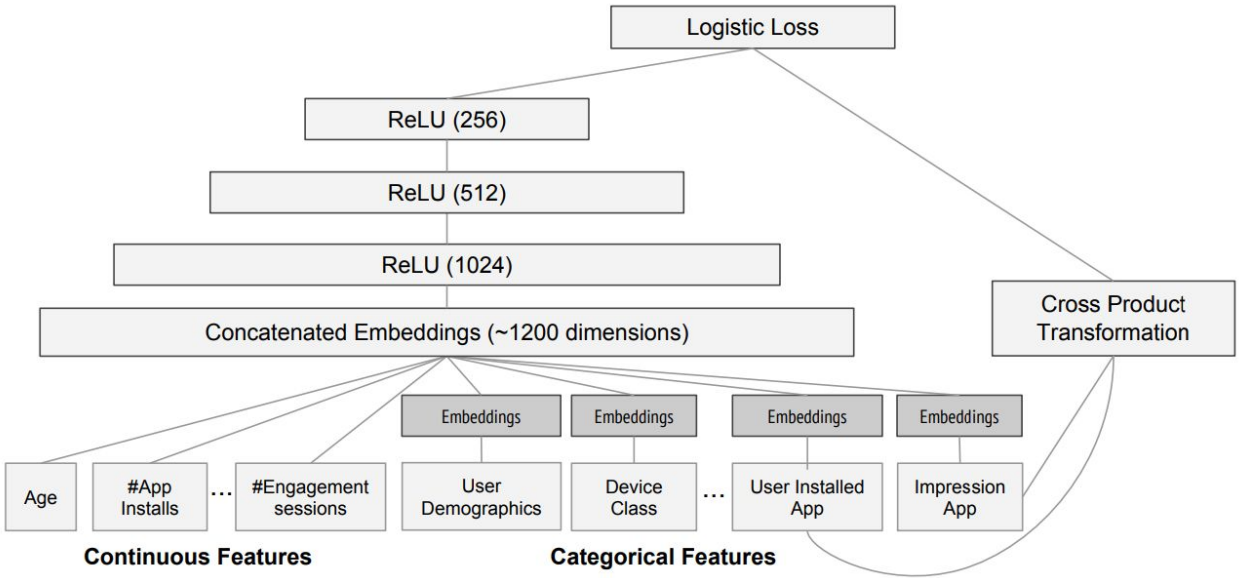
- obtain great performances
- are somewhat explainable (features importance ranking)
- easier to tune and setup than NN
- can deal with missing data naturally
- do not need data pre-processing (scaling)

Some recent architectures in DL may be worth checking if one of the case:

- you have a lot of high-dimensional categorical variables
- you have a very high number of columns and rows
- you would rather fiddle with NN architectures than feature engineering

Tabular data with MLP: mixing Wide and Deep

- Helps combining categorical and continuous features
- Great performance on large scale recommender systems



[Wide & Deep Learning for Recommender Systems]

Example: HiggsML Challenge Revisited

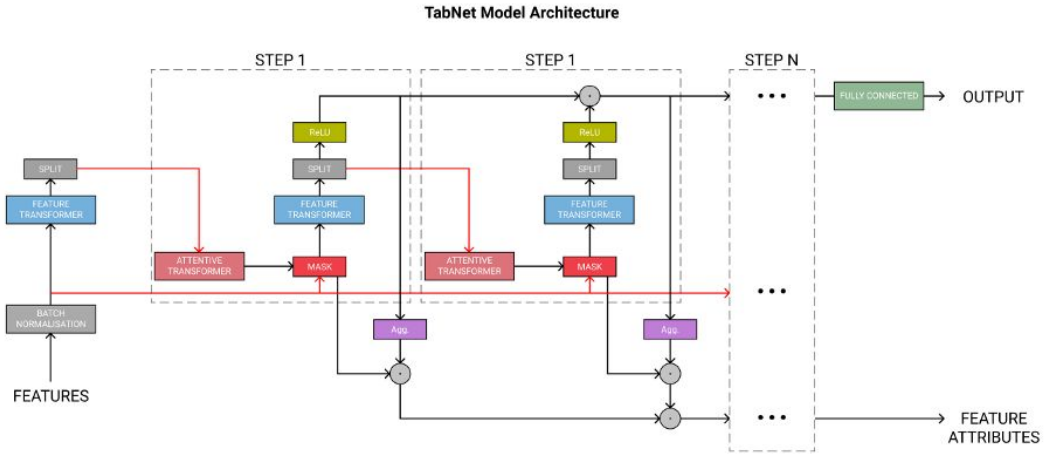
- Launched in 2014, the Higgs ML Kaggle competition was designed to help stimulate outside interest in HEP problems
- The data contains simulated LHC collision data for Higgs to di-tau and several background processes
- Participants were tasked with classifying the events in order to optimise the Approximate Median Significance
- The competition was highly successful, and helped introduce new methods to HEP, as well as produce more widely used tools, such as XGBoost

Strong (2020)

	Our solution	1st place	2nd place	3rd place
Method	10 DNNs	70 DNNs	Many BDTs	108 DNNs
Train-time (GPU)	8 min	12 h	N/A	N/A
Train-time (CPU)	14 min	35 h	48 h	3 h
Test-time (GPU)	15 s	1 h	N/A	N/A
Test-time (CPU)	3 min	???	???	20 min
Score	3.806 ± 0.005	3.80581	3.78913	3.78682

TabNet: a specific deep architecture for tabular data

- Feature Transformer to avoid pre-processing
- Feature Selector based to Attention Transformer
- More explainable than MLP
- Comparable performance as tree-based methods



See a light explanation and how to use [on this blog]

Resources

- Dive into Deep Learning Chapters 9 and 13
- Fei-Fei Li CS231n Lecture 9 and 15
- fastai library has often all the tricks built-in