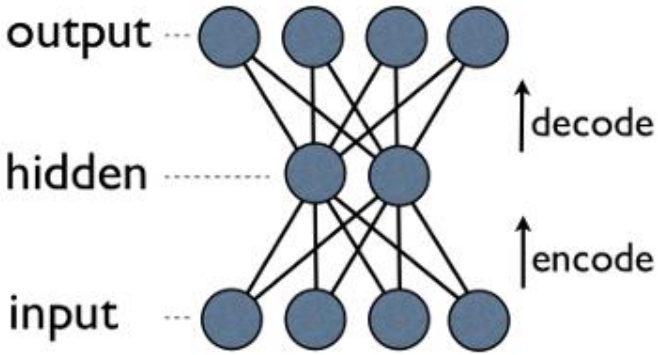
LESSON 22

Autoencoders
Fine Tuning and Transfer Knowledge
Automatic Feature Extraction
Training of Deep Learning Models




Outline

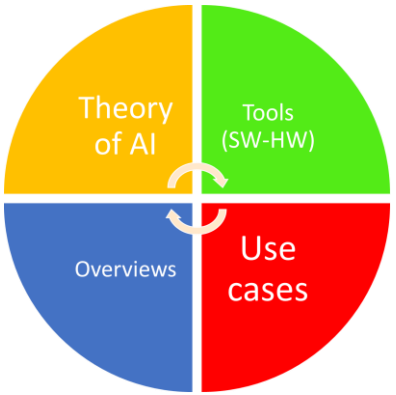
- Autoencoders
- Training and design of deep learning models
  - Fine Tuning
  - Transfer Knowledge
  - Data augmentation
- Automatic Feature Extraction
- Problems of the training of deep learning models.
- A solution: Greedy Layer-Wise Training
- Generative Adversarial Networks
  - Generation of realistic synthetic images



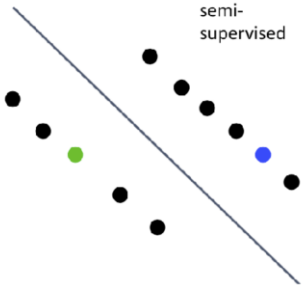
THEORY

Unsupervised deep learning models: Autoencoders

Working in training with no labels

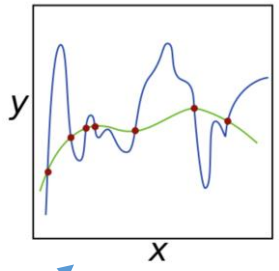


Unsupervised learning

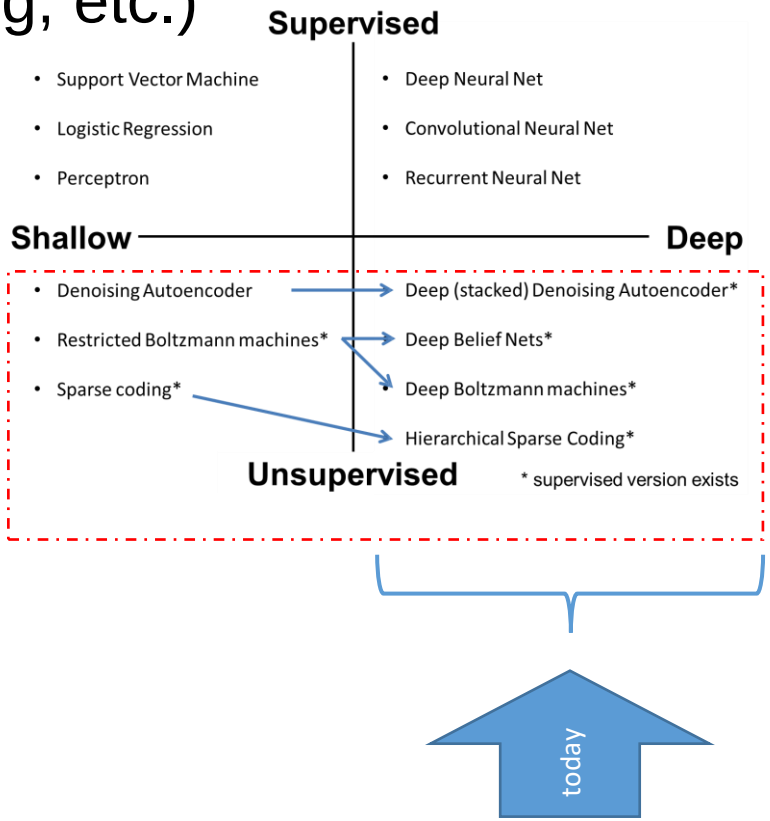


- Can use **unlabeled data** (unlimited)
- Can be refined with standard supervised techniques (e.g. backpropagation like a normal feedforward NN)
- Useful when the amount of labels is small → partially supervised dataset (semi-supervised)

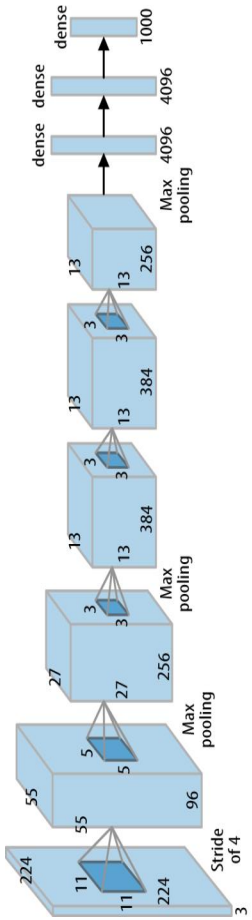
Unsupervised learning for classical and deep learning models



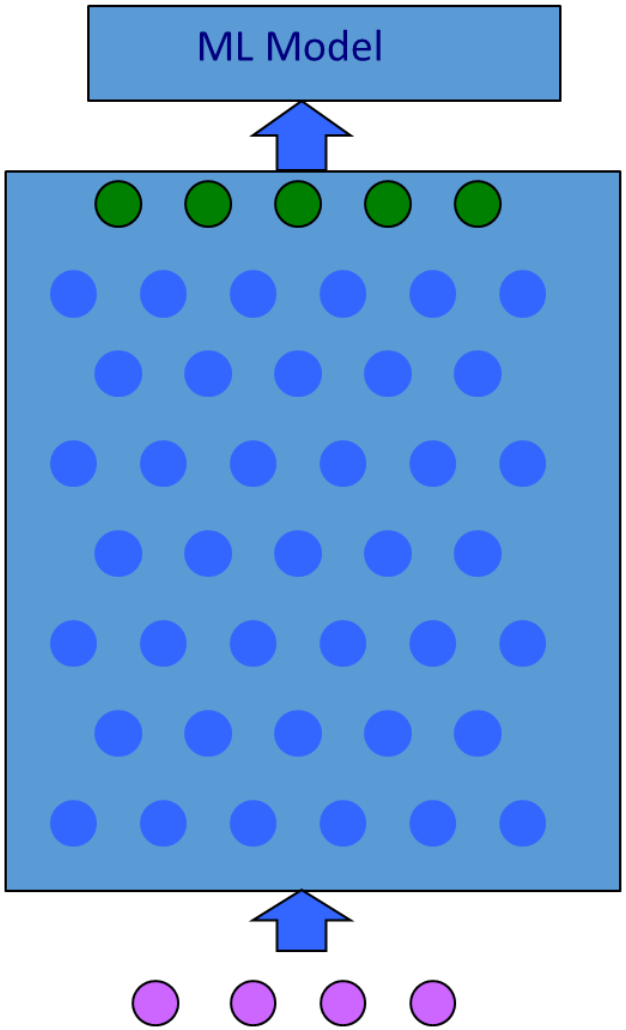
- Main idea: since we have no labels, methods try to model the distribution of input data (e.g., the clusters)
  - Reconstruction error + Regularization (sparsity, denoising, etc.)
  - Log-likelihood of data
- Models
  - Basic: PCA, KMeans
  - Independent Comp. Analysis
  - Denoising autoencoders
  - Sparse autoencoders
  - Restricted Boltzmann machines
  - Sparse coding
  - …



Deep Net with Greedy Layer Wise Training



New Feature Space



Original Inputs

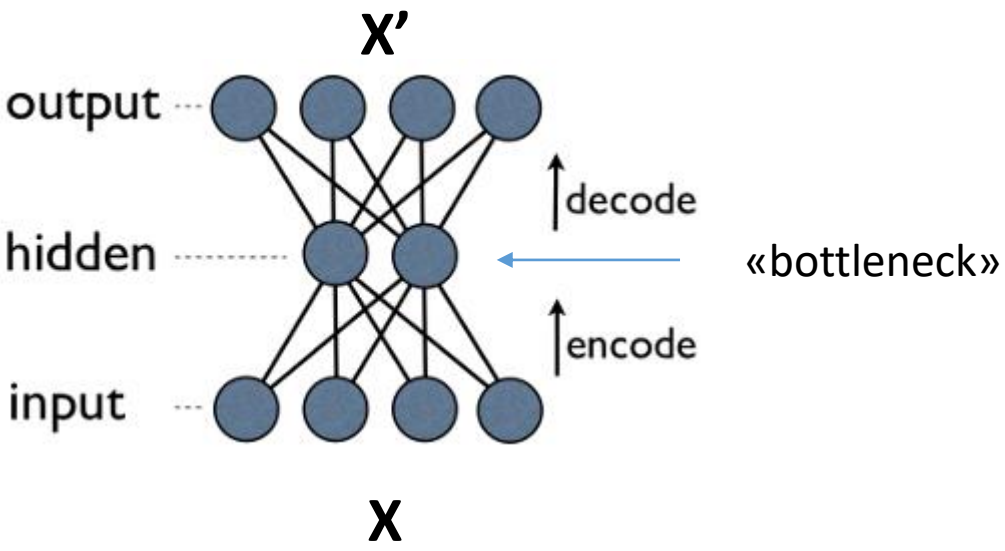
Supervised Learning

Unsupervised Learning

This hard task will be solved by using Autoencoders

Autoencoders (AEs)

- Try to create generic features from data (encoding)
  - Learn identity function (output = input) by learning important subfeatures
  - Compression, etc.
- We can just use the new features in the new training set or concatenate both.

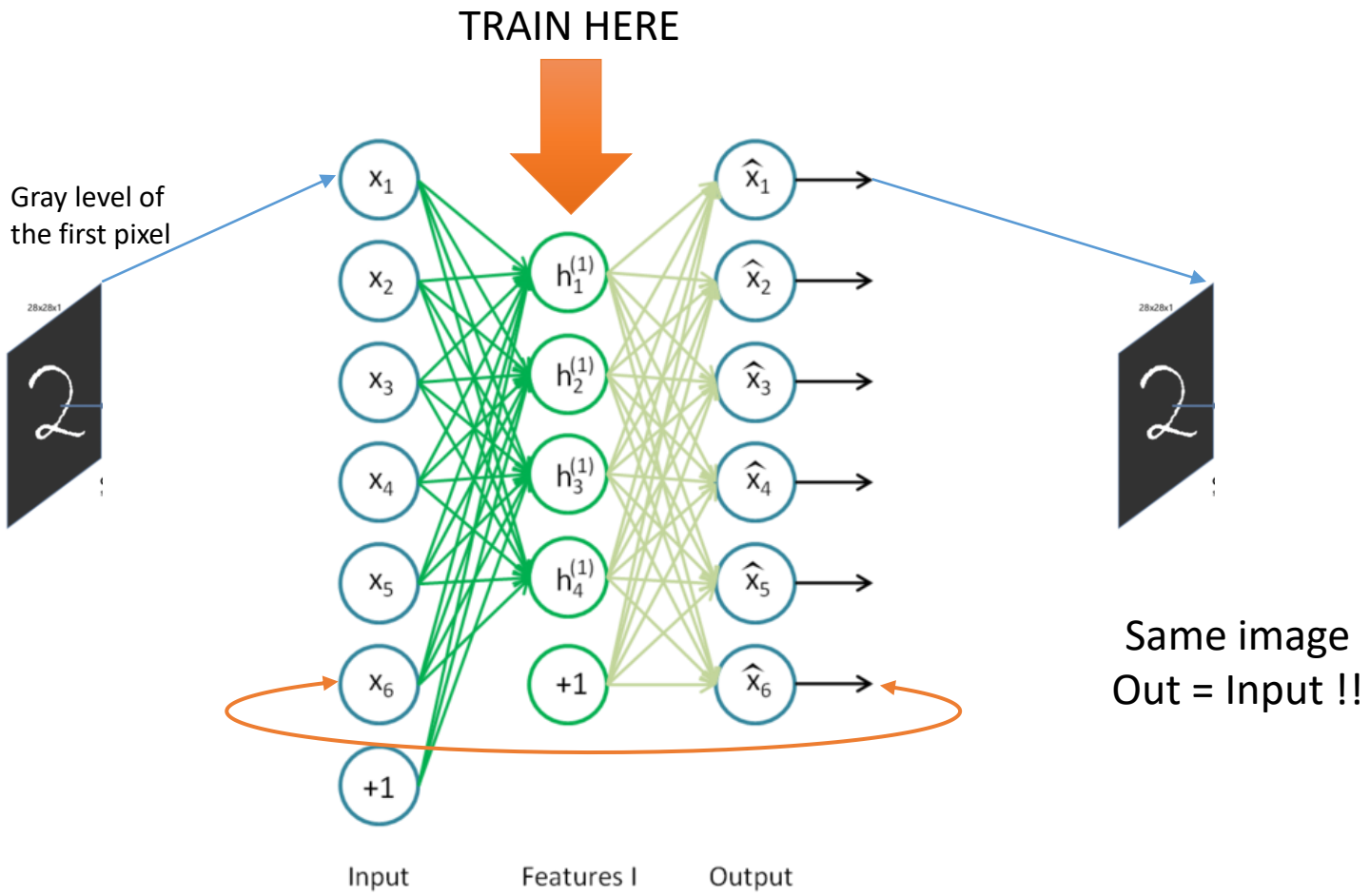


The training aims to **X=X'**

Using backpropagation like a normal feedforward NN

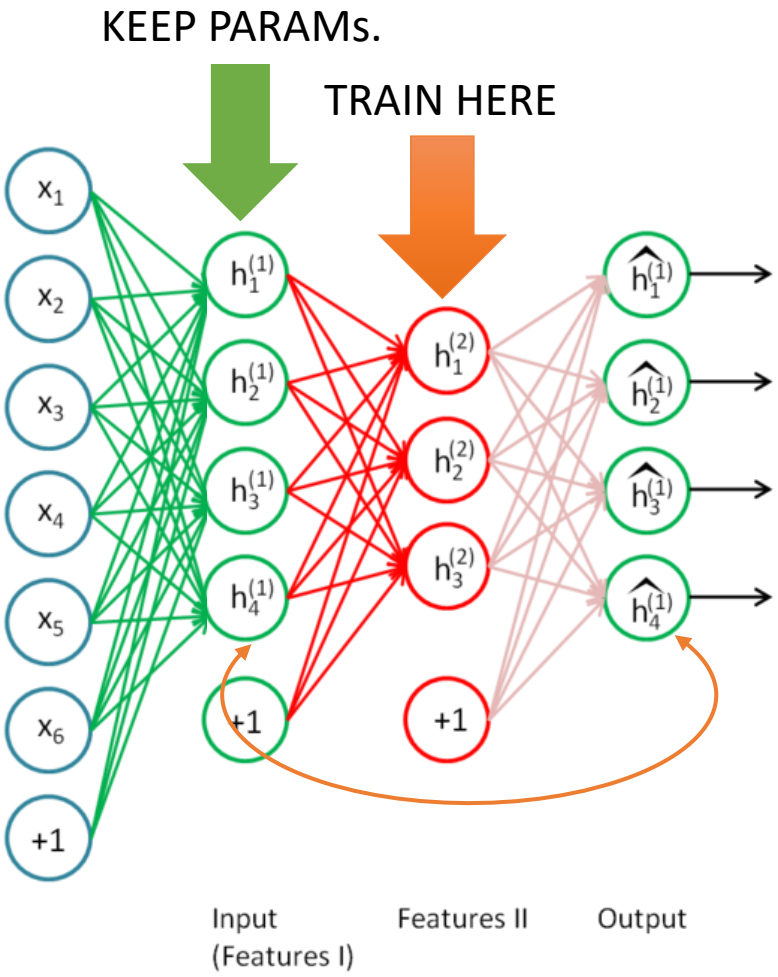
Stacked Auto-Encoders

- Stack many (sparse) auto-encoders in succession and train them using greedy layer-wise training
- Drop the decode output layer each time



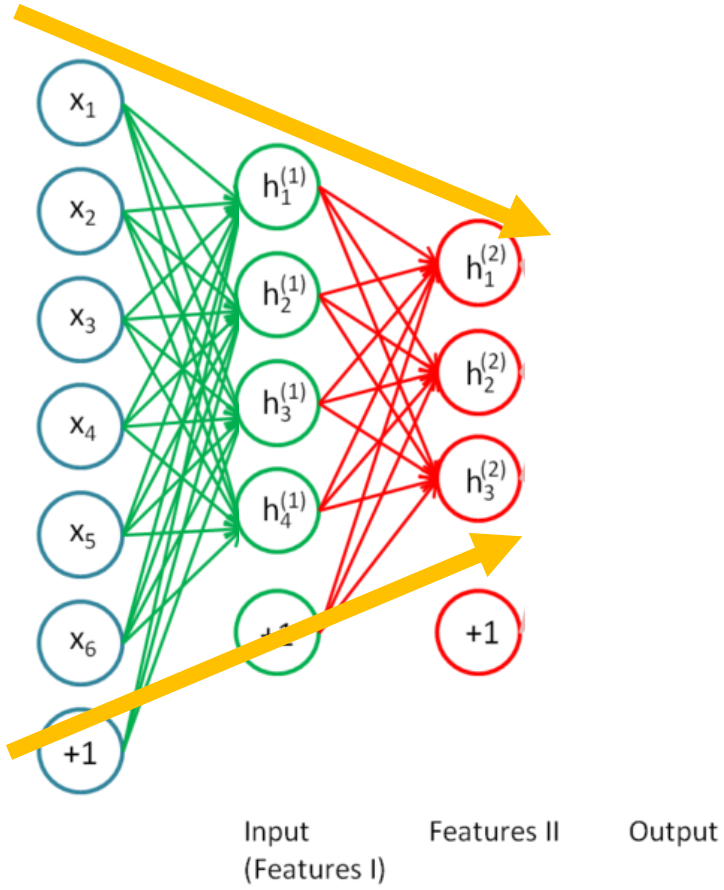
Stacked Auto-Encoders

- Stack many (sparse) auto-encoders in succession and train them using greedy layer-wise training
- Drop the decode output layer each time



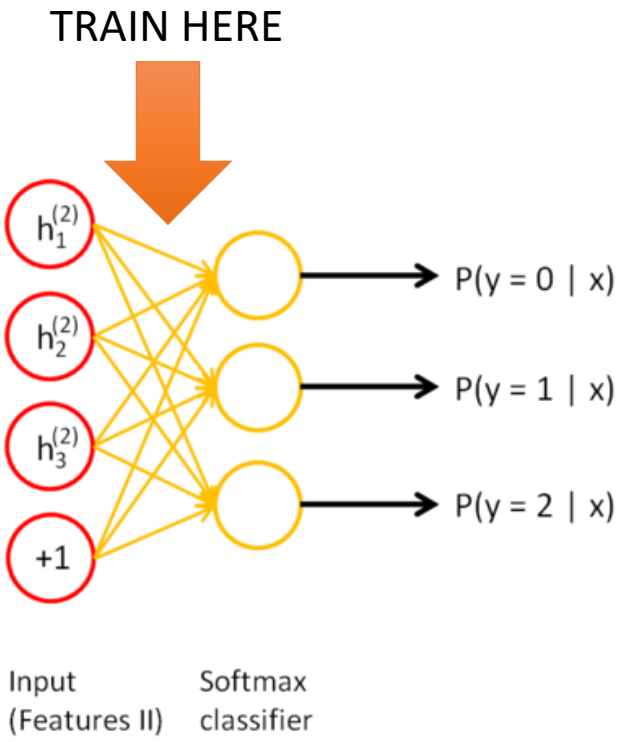
Stacked Auto-Encoders

- The golden rules is to
  reduce the number of neurons going forward to the outputs



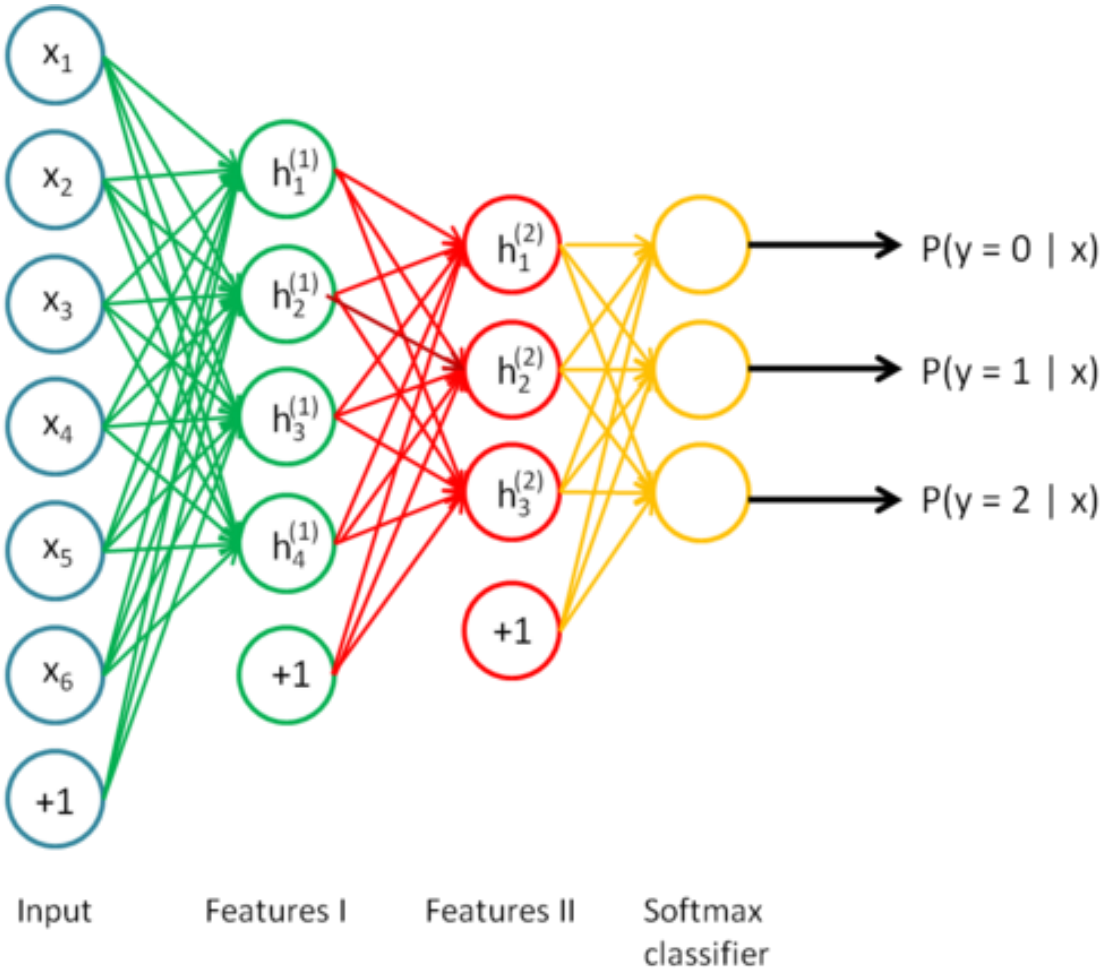
Stacked Auto-Encoders (2)

- Do **supervised training** on the last layer.
  We can do it since we have
  - The inputs X and the weights of the AEs → we can process the inputs of the final layer (h1, h2, …)
  - We have the labels in the Y output vector



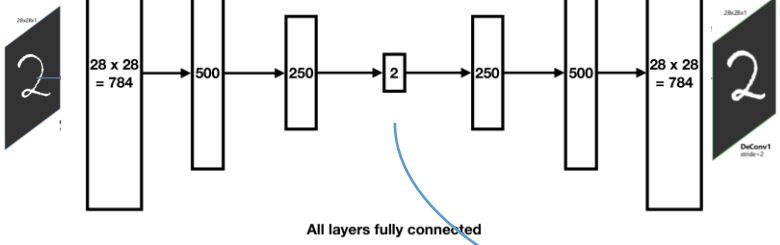
Stacked Auto-Encoders (3)

- Then do **supervised training on the entire network** to fine- tune all weights



Applications of AEs: Dimensionality reduction

Example from the MNIST dataset (28×2828×28 black and white images of single digits) from the original 784 dimensions to 2

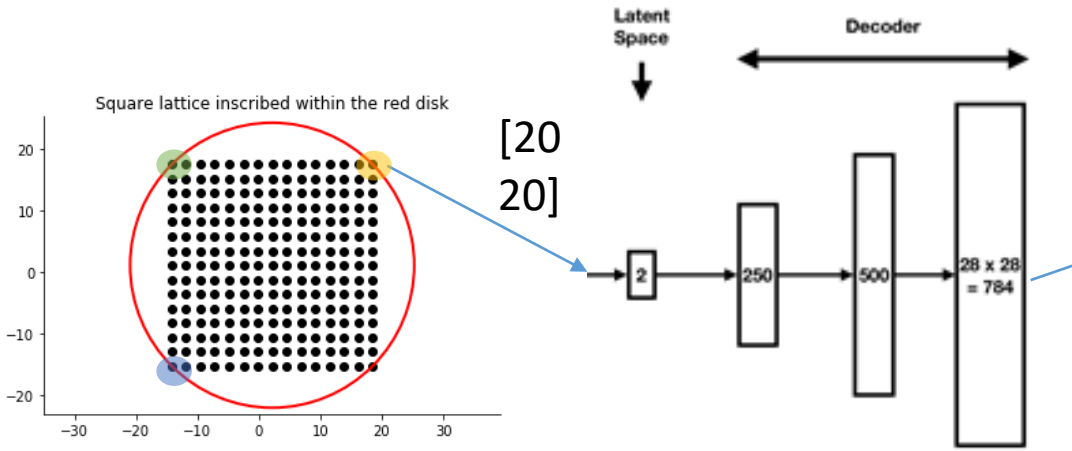


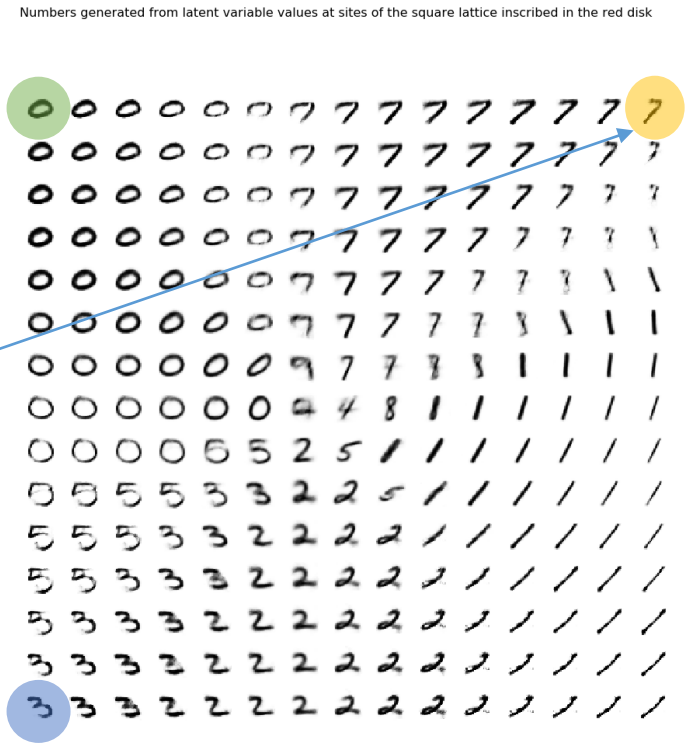
Note: PCA is an unsupervised method as AEs

Understanding AEs:

After training, we can use just the decoder

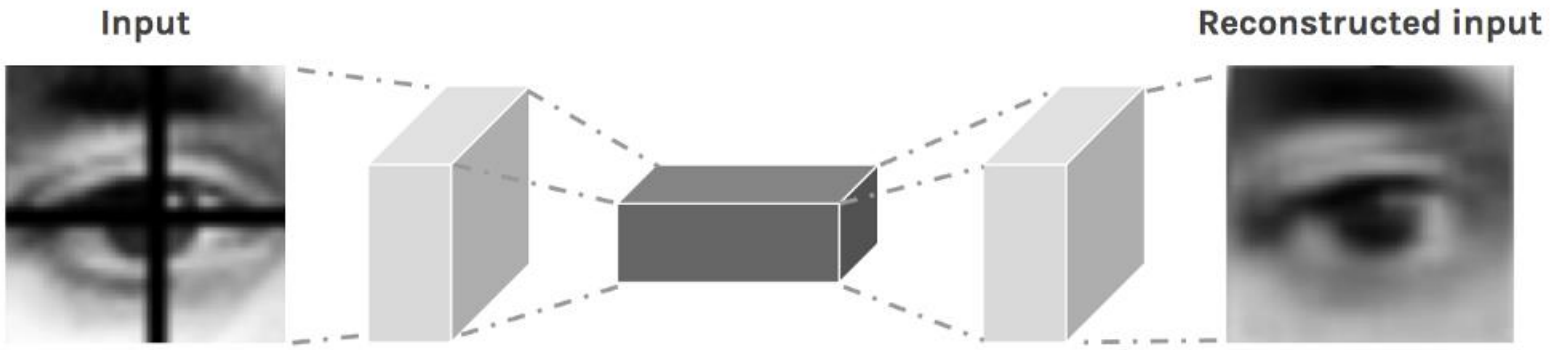
Let's try with different 2D input vectors, for example the input [20,20] will produce the output image 



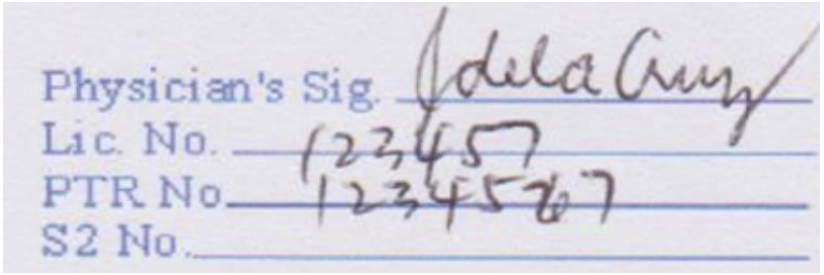


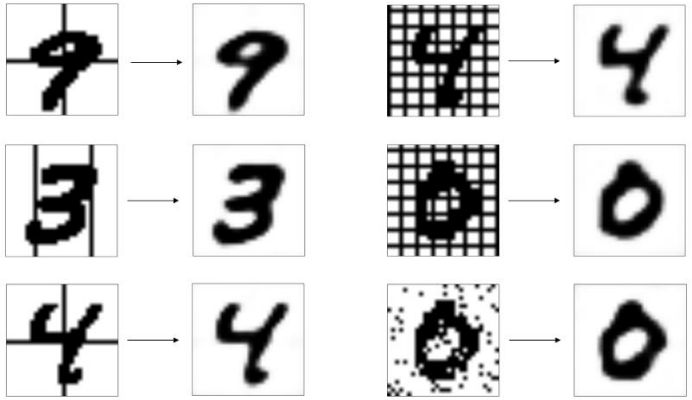
NOTE: observe the interesting continuity of the latent space!

Applications of AEs: Noise reduction

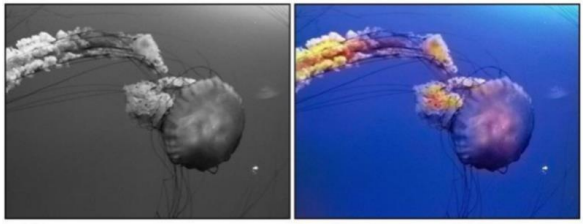


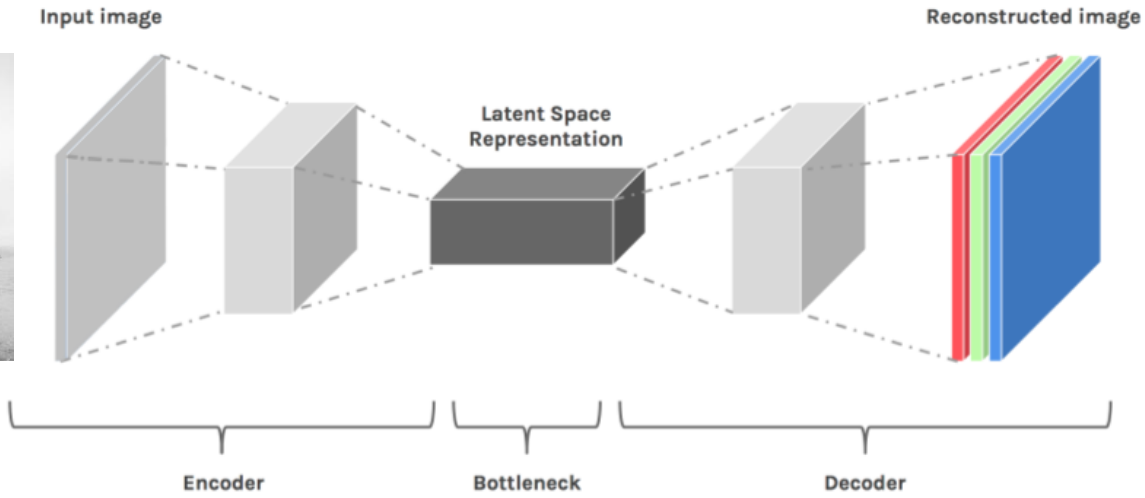
Missing pixels, noise, bars





Applications of AEs: Auto Colorization

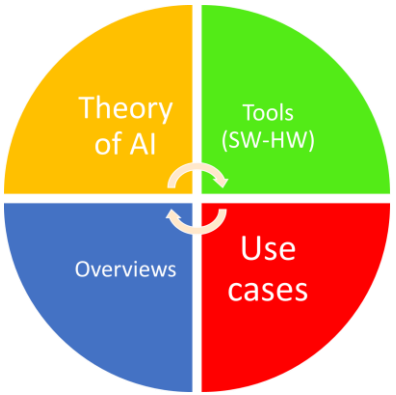




THEORY

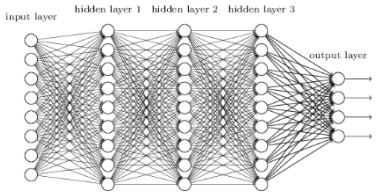
Transfer learning and fine-tuning in deep models

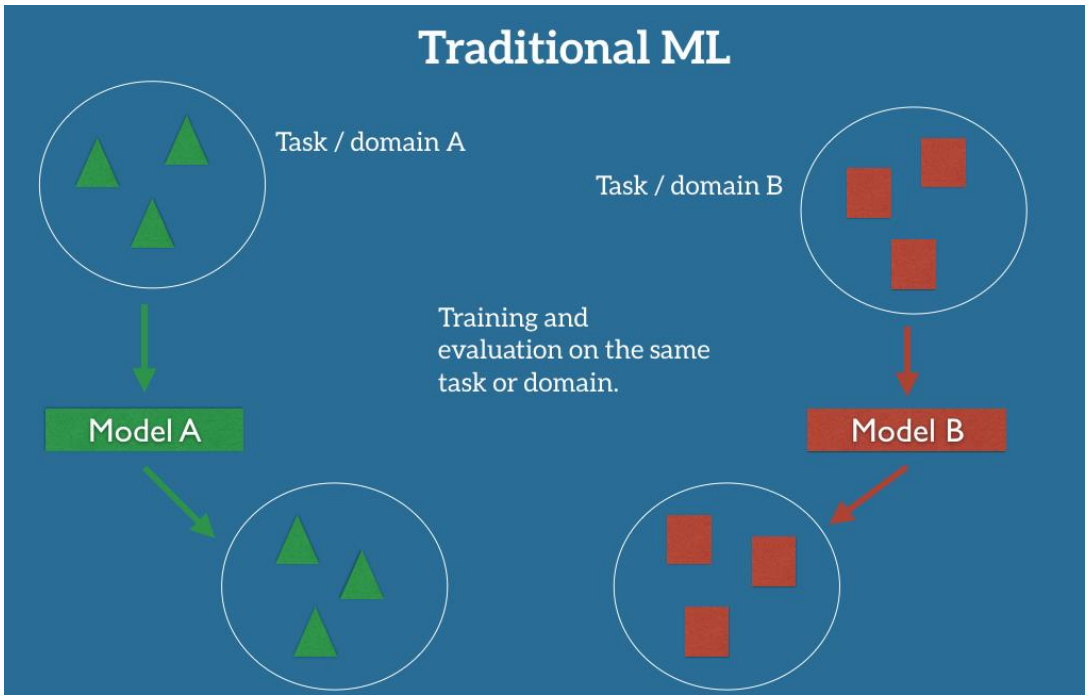
Exploit previous training and knowledge



Transfer Learning to avoid overfitting and improving learning

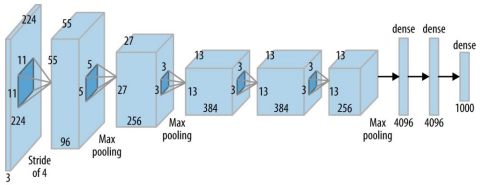
Traditional ML methods need to start over every time the learning process when you change the application domain/datasets





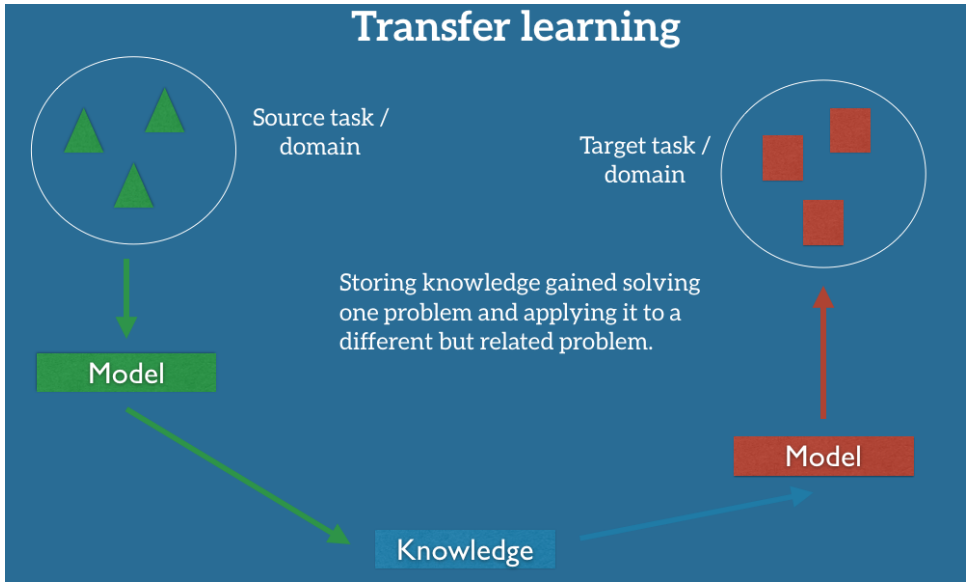
http://ruder.io/transfer-learning/

Transfer Learning to avoid overfitting and improving learning



Domain 1

Step1 no background
→ Better learning



Domain 2

Step2 more complex scenes Can be faced during learning

http://ruder.io/transfer-learning/

Pretrained models are available online

- Large CNNs take weeks to train across multiple GPUs on image dataset like ImageNet
- Many researcher release their final CCN for the benefit of others who can use the networks for fine-tuning.

For example, the Caffe library has a Model Zoo where people share their network weights.

https://github.com/BVLC/caffe/wiki/Model-Zoo


Table of Contents

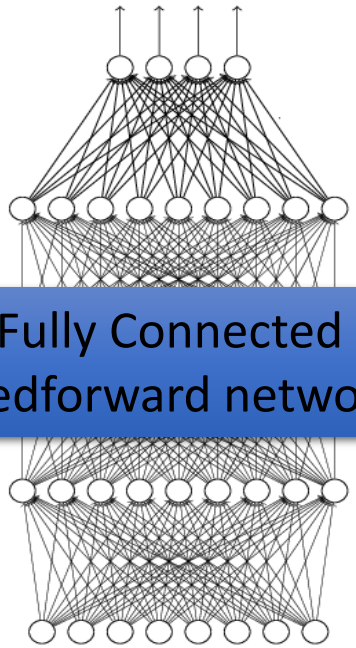
- Berkeley-trained models
- Network in Network model
- Models from the BMVC-2014 paper "Return of the Devil in the Details: Delving Deep into Convolutional Nets"
- Models used by the VGG team in ILSVRC-2014
- Places-CNN model from MIT.
- GoogLeNet GPU implementation from Princeton.
- Fully Convolutional Networks for Semantic Segmentation (FCNs)
- CaffeNet fine-tuned for Oxford flowers dataset
- CNN Models for Salient Object Subitizing.
- Deep Learning of Binary Hash Codes for Fast Image Retrieval
- Places_CNDS_models on Scene Recognition
- Models for Age and Gender Classification.
- More Models for Age and Gender Classification.
- GoogLeNet_cars on car model classification
- ParseNet: Looking wider to see better
- SegNet and Bayesian SegNet
- Conditional Random Fields as Recurrent Neural Networks
- Holistically-Nested Edge Detection
- CCNN: Constrained Convolutional Neural Networks for Weakly Supervised Segmentation
- Emotion Recognition in the Wild via Convolutional Neural Networks and Mapped Binary Patterns
- Facial Landmark Detection with Tweaked Convolutional Neural Networks
- Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks
- ResNets: Deep Residual Networks from MSRA at ImageNet and COCO 2015
- Pascal VOC 2012 Multilabel Classification Model
- SqueezeNet: AlexNet-level accuracy with 50x fewer parameters
- Mixture DCNN

FINE TUNING

starting from a public CNN

Classification PB1:
Face, body, dog, horse, …

Classification PB2:
FACE YES/NO



Fully Connected Feedforward network

Convolution

Max Pooling

Convolution

Max Pooling

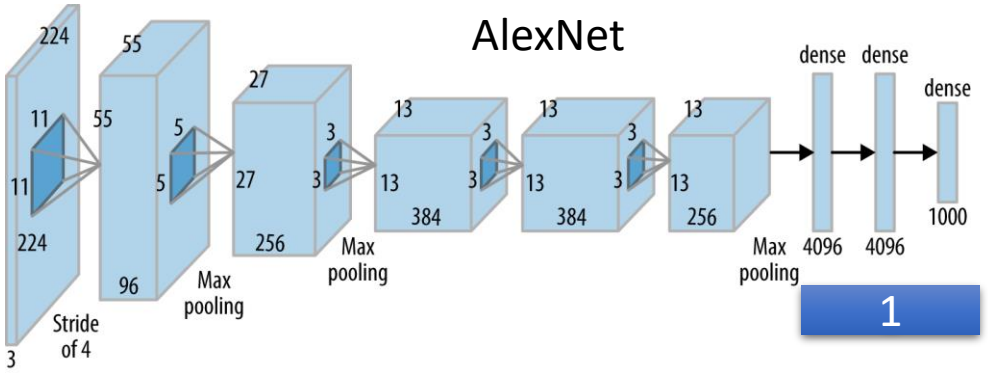
Can repeat many times

Flattened

TUNE

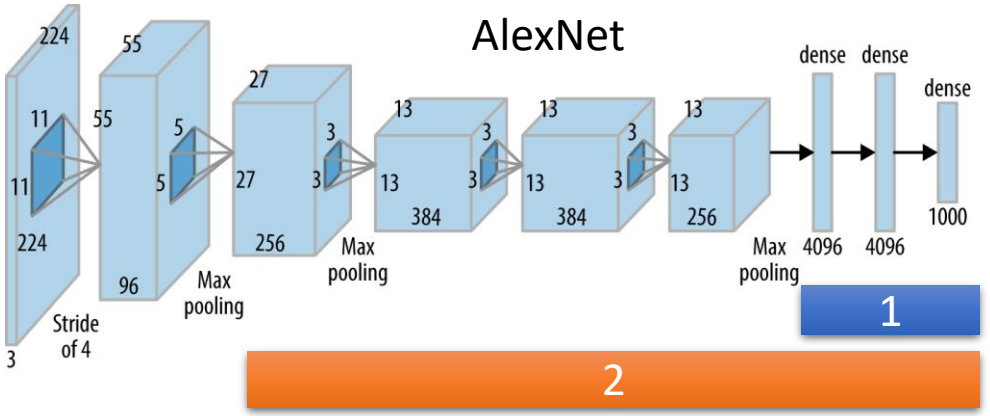
FREEZE

Fine-tuning the CCN (2)



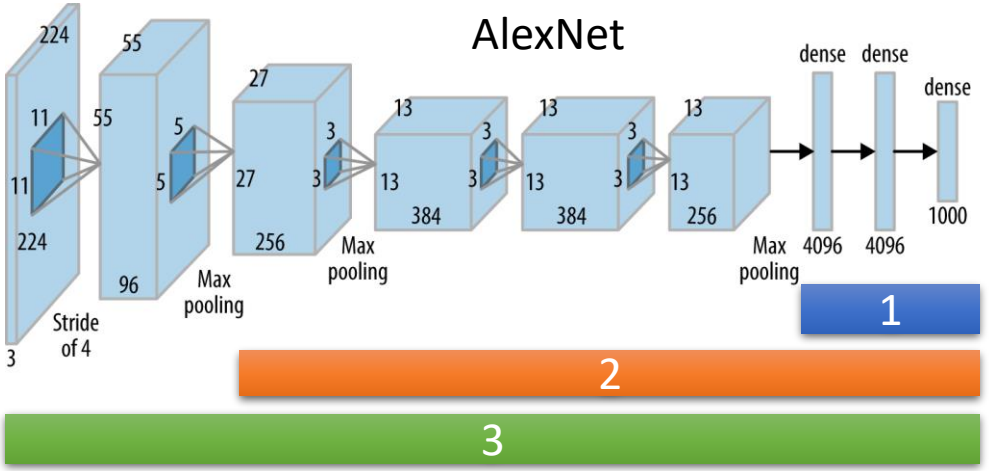
- Options
  - 1) replace and retrain the classifier on top of the ConvNet on the new dataset,

Fine-tuning the CCN (2)



- Options
  - 1) replace and retrain the classifier on top of the ConvNet on the new dataset,
  - 2) Keep some of the earlier layers fixed (due to overfitting concerns) and only fine-tune some higher-level portion of the network.
    - This is motivated by the observation that the earlier features of a ConvNet contain more generic features (e.g. edge detectors or color blob detectors) that should be useful to many tasks, but later layers of the ConvNet becomes progressively more specific to the details of the classes contained in the original dataset.

Fine-tuning the CCN (2)



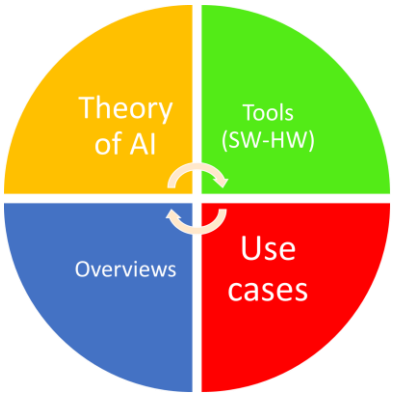
Options

- 1) replace and retrain the classifier on top of the ConvNet on the new dataset,
- 2) Keep some of the earlier layers fixed (due to overfitting concerns) and only fine-tune some higher-level portion of the network.
  - This is motivated by the observation that the earlier features of a ConvNet contain more generic features (e.g. edge detectors or color blob detectors) that should be useful to many tasks, but later layers of the ConvNet becomes progressively more specific to the details of the classes contained in the original dataset.
- 3) fine-tune all the weights of the pretrained network by continuing the backpropagation.

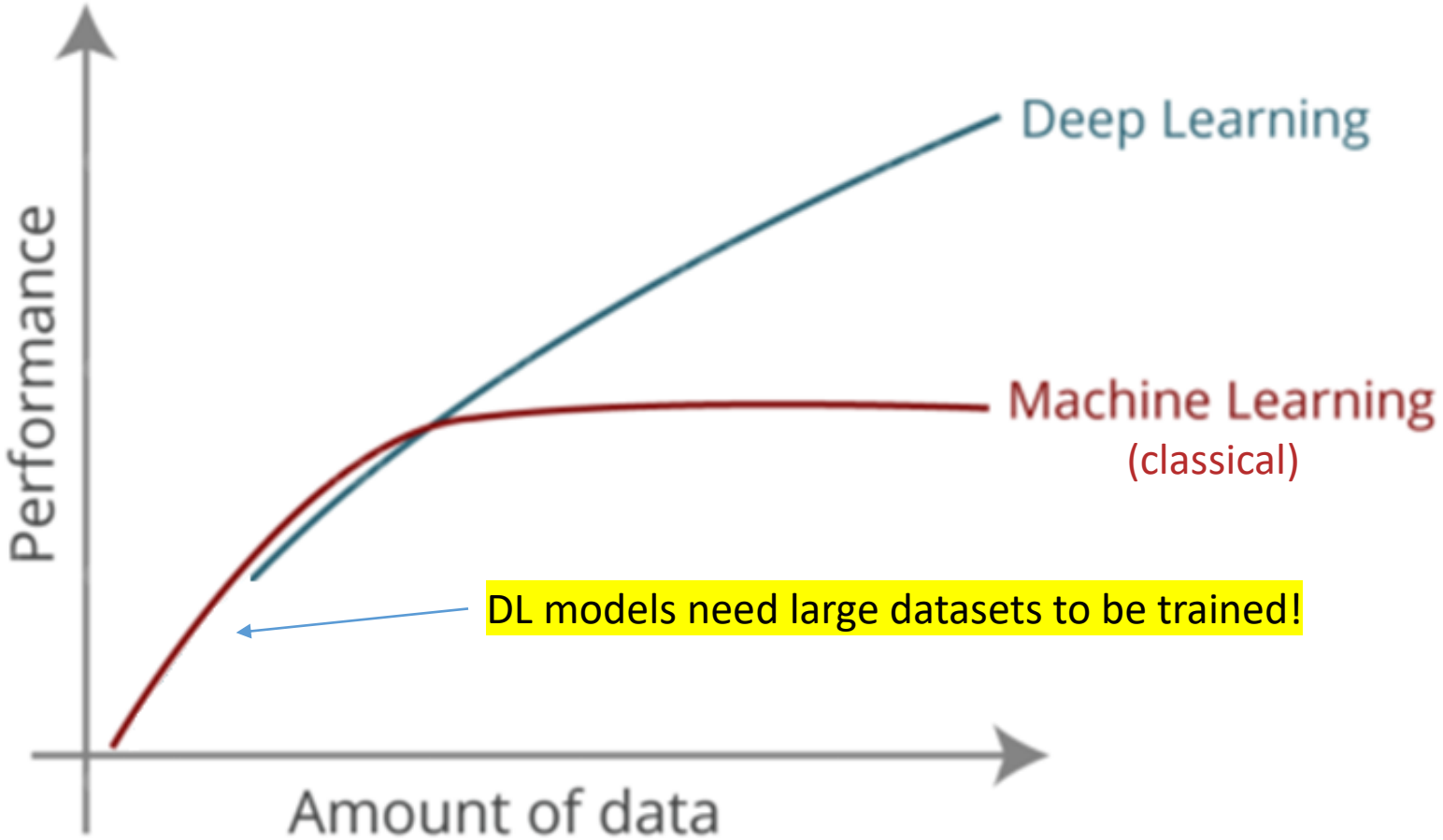
THEORY

Data augmentation

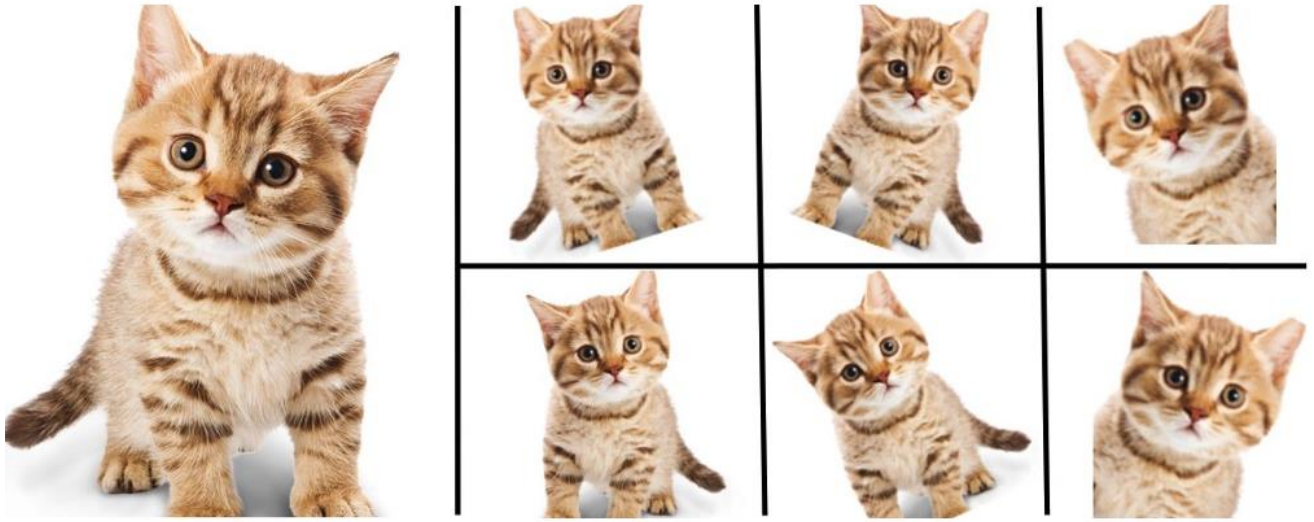
Improving/enlarging the training dataset



Classical ML vs DeepLearning



Data augmentation



- YOUR ALLAY TO OVERCOME OVERFITTING PROBLEMS!
  Please keep in mind that you can have >100M free parameters to fix/tune!

- Data augmentation will not solve the application by itself: you still need good real data

Data augmentation #1: rotations, translations, scales

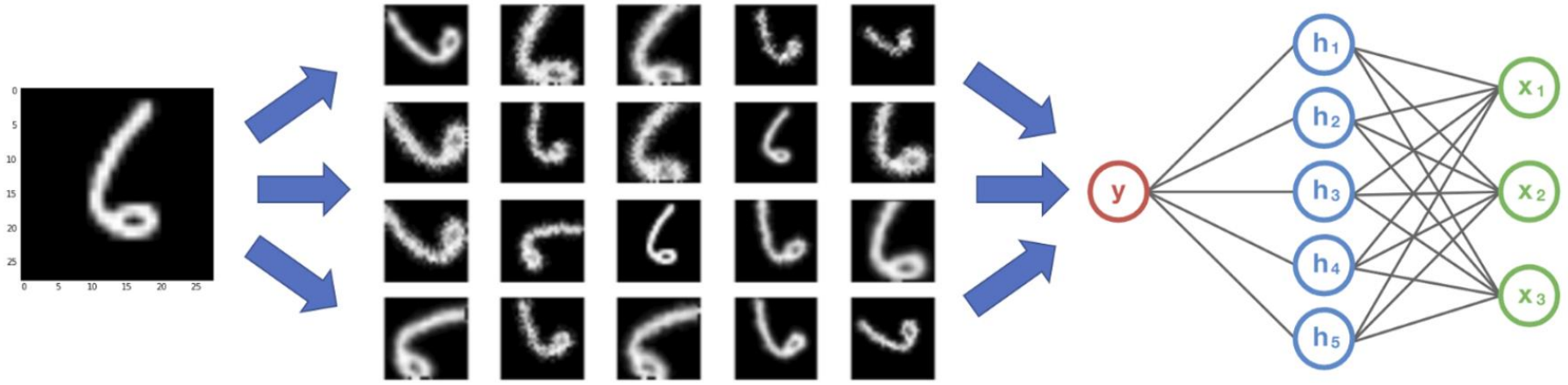
Same tennisball input image.
We let the CNN learn to detect the object in different
→ positions
→ scales
→ rotations

Example: the Matlab imageDataAugmenter is randomly flipping the training images along the vertical axis, and randomly translating them

```
pixelRange = [-30 30];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
    'DataAugmentation',imageAugmenter);
```

Advanced toolboxes have specific functions to create this kind of data augmentation.

Data augmentation #2: with noise/noises



Noise of different types (gaussian, additive, multiplicative), application conditions, sensor noise (pixel noise), resolutions, environments (indoor, outdoor, backgrounds), …

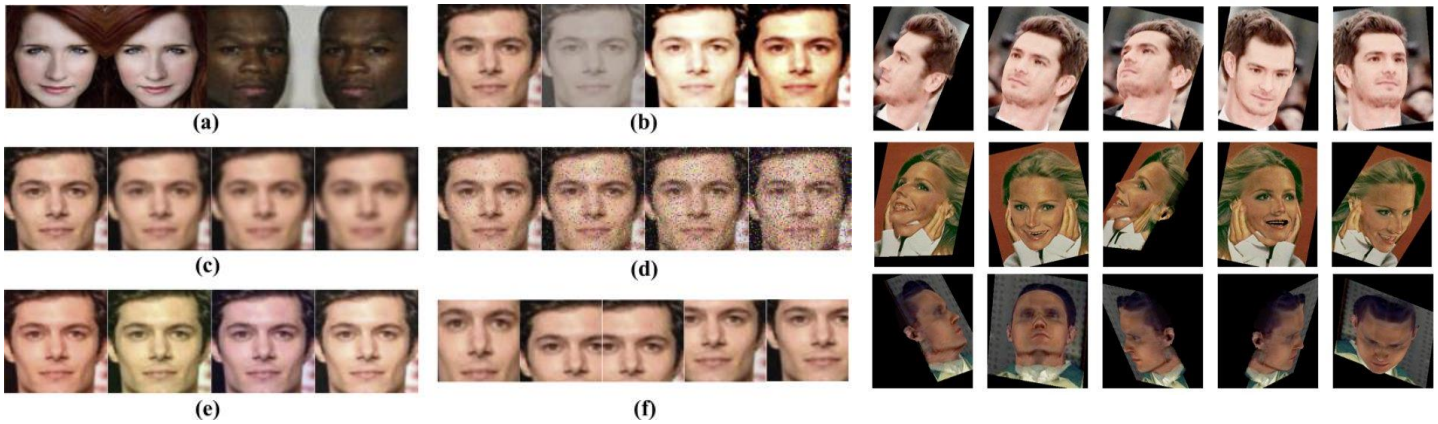
Data augmentation #3:
with different (simulated) conditions (even for big datasets)

You should consider to reproduce real conditions in training

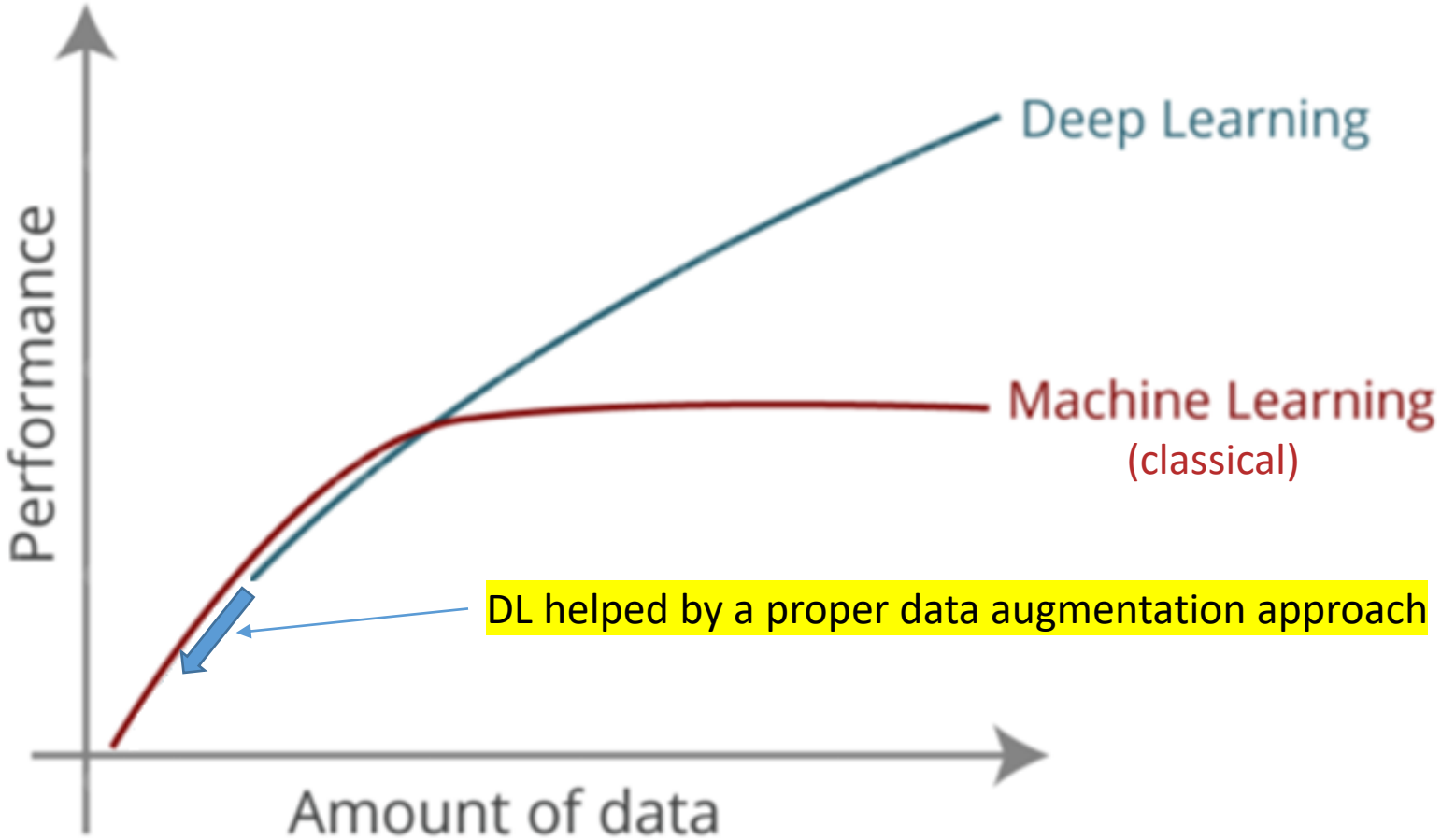
You do it using public CNNs!

Landmark perturbation for face alignment.

Flipping
clipping
color casting
blurring



DeepLearning + Data Augm.

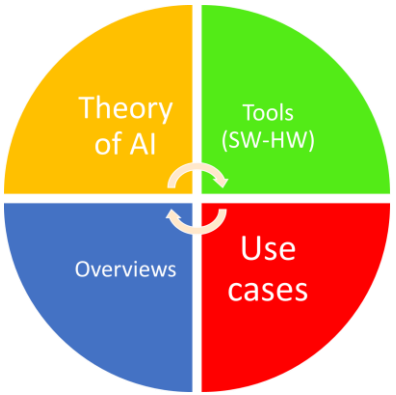




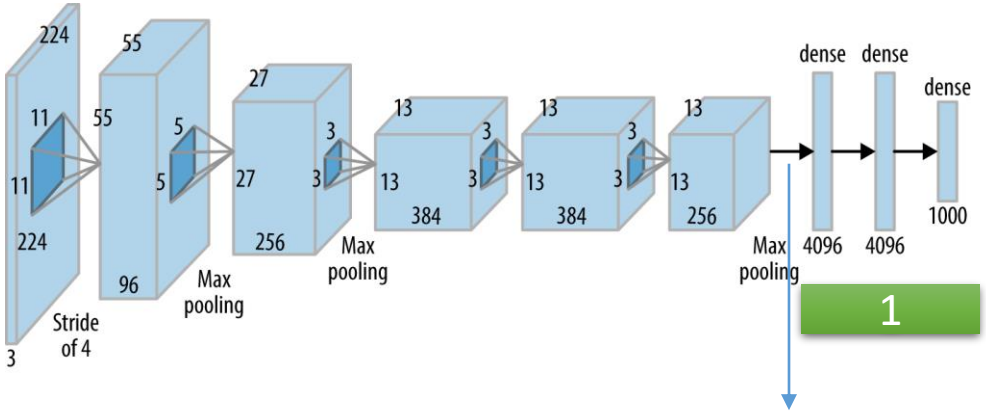

THEORY

Automatic Feature Extraction with deep learning models

Improving the PCA approach using DL



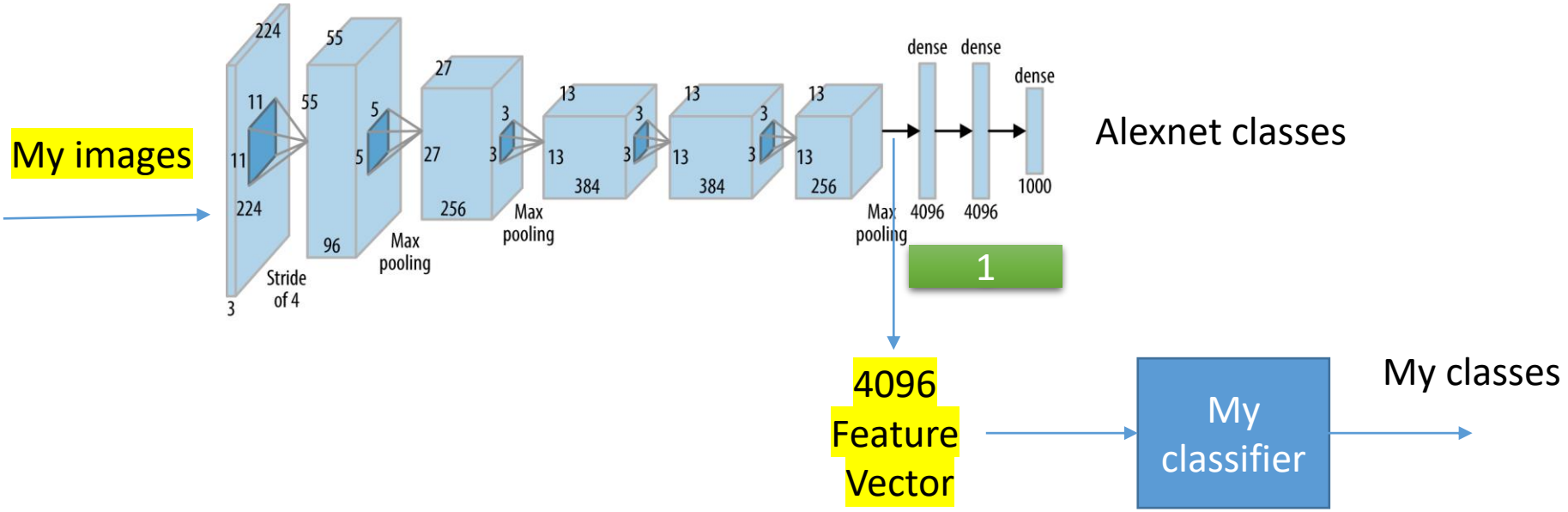
CNNs as feature extractors



4096 Feature Vector

1. In a pretrained CNN remove the last fully-connected layer [1]
   - For example, this layer's outputs are the 1000 class scores for a different task like ImageNet
2. Then treat the rest of the CNN as a fixed feature extractor for the new dataset.
   - In an AlexNet, this would compute a 4096-D vector for every image that contains the activations of the hidden layer immediately before the classifier.
   - Note: It is important to note if these features are "ReLUed" (i.e. thresholded at zero) if they were also thresholded during the training of the CNN (as is usually the case).

ConvNets as feature extractors (2)

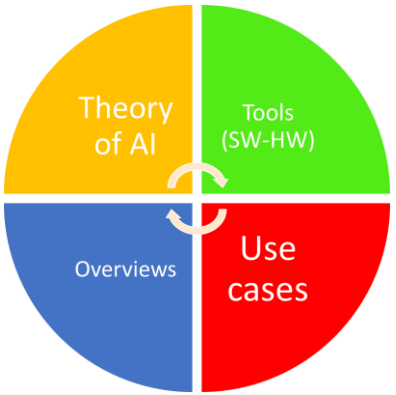


- Once you extract the 4096-D codes for all images, train a linear classifier (e.g. Linear SVM or Softmax classifier) for the new dataset (your image dataset).

THEORY

Training and designing deep learning models

Facing the complexity of >100M parameters



First fo all… DeepLearning Design

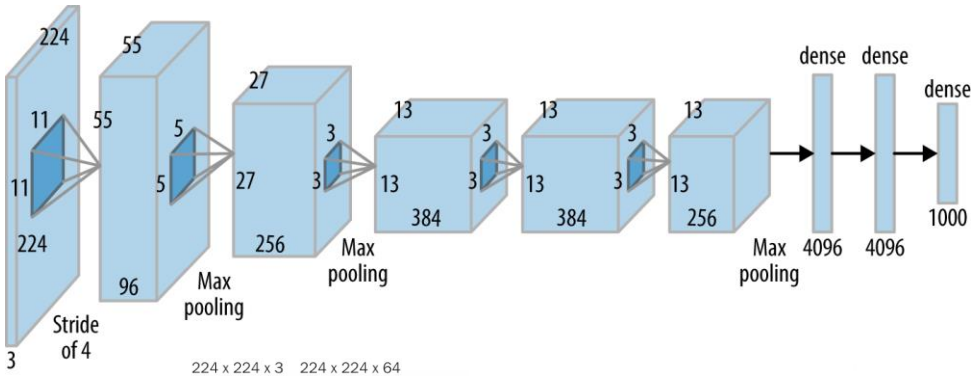
- While all **weights** are trained, the **structure (topology)** of the CNN is currently usually hand crafted with trial and error.
  - Number of total layers,
  - kernels types,
  - kernel sizes,
  - Size of sub-sampling (pooling) fields,

- Typically going along with layers from inputs → outputs
  - decrease size of kernels
  - increase number of feature maps

Deep Learning Training hints

Trained with **Back Propagartion (BP)**

- Use “weight tying” in each feature map (a sharing method for the weight matrix)
- Randomized initial weights through entire network
- Use pre-trained models



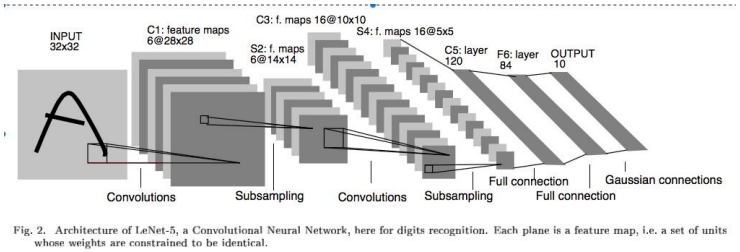
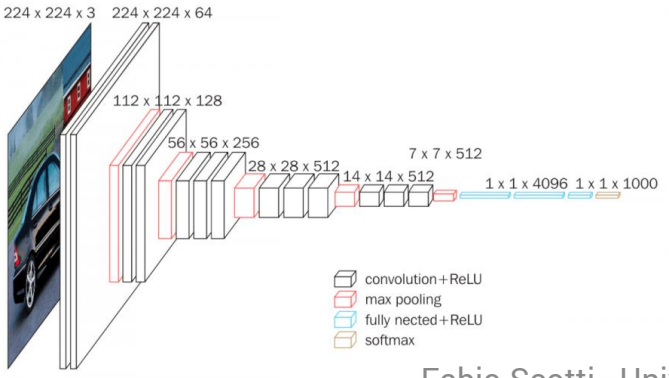
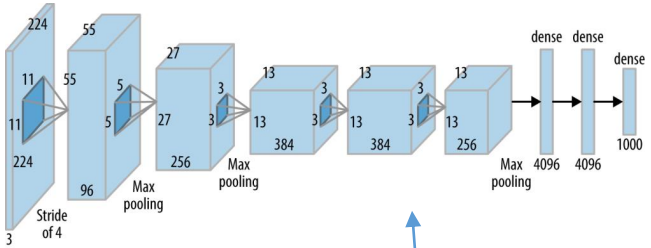


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.



“Weight Tying” : Sharing the weight matrix between input-to-embedding layer and output-to-softmax layer; That is, instead of using two weight matrices, we just use only one weight matrix. The intuition behind doing so is to combat the problem of overfitting.

Problems while training Deep Networks



- Early layers of MLP do not get trained well
  - Diffusion of Gradient – error attenuates as it propagates to earlier layers
  - Leads to very slow training
  - Exacerbated since top couple layers can usually learn any task "pretty well" and thus the error to earlier layers drops quickly as the top layers "mostly" solve the task
- Lower layers never get the opportunity to use their capacity to improve results, they just do a random feature map

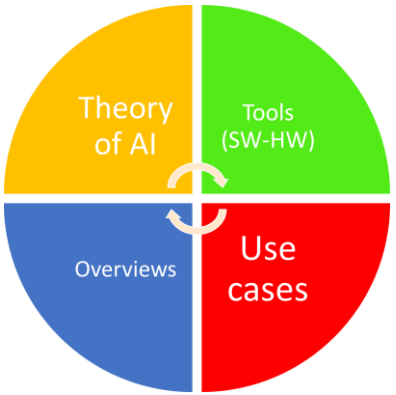
Problems in training deep networks (2)

- Need a way for early layers to do effective work
- Instability of gradient in deep networks: Vanishing or exploding gradient
  - Product of many terms, which unless "balanced" just right, is unstable
  - Either early or late layers stuck while "opposite" layers are learning
- Often not enough labeled data available while there may be lots of unlabeled data
  - Can we use unsupervised/semi-supervised approaches to take advantage of the unlabeled data
- Deep networks tend to have more sensitive training issues and local minima problems than shallow networks during supervised training!

THEORY

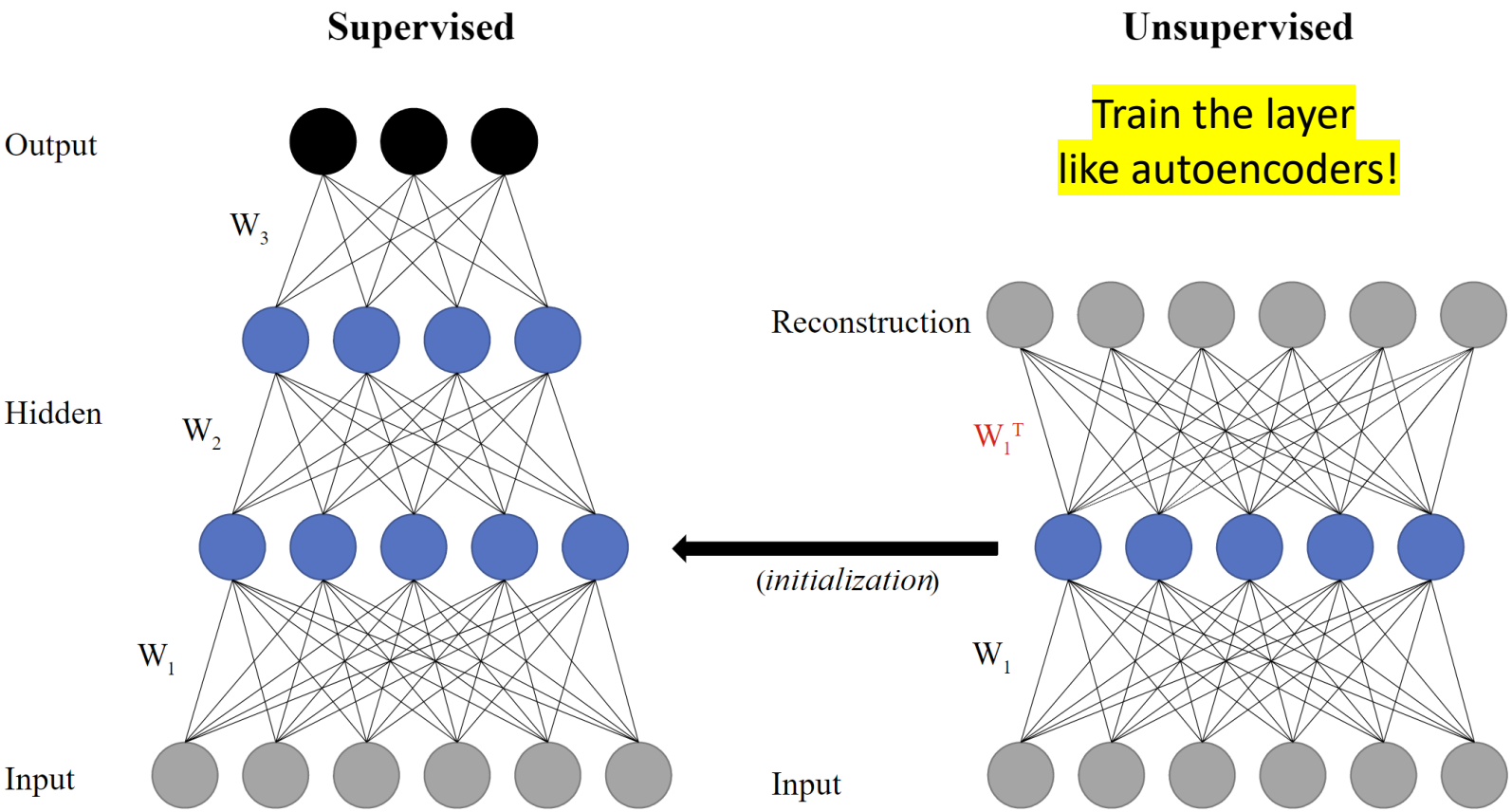
Greedy Layer-Wise Training

A very efficient solution to the problems in training of deep networks



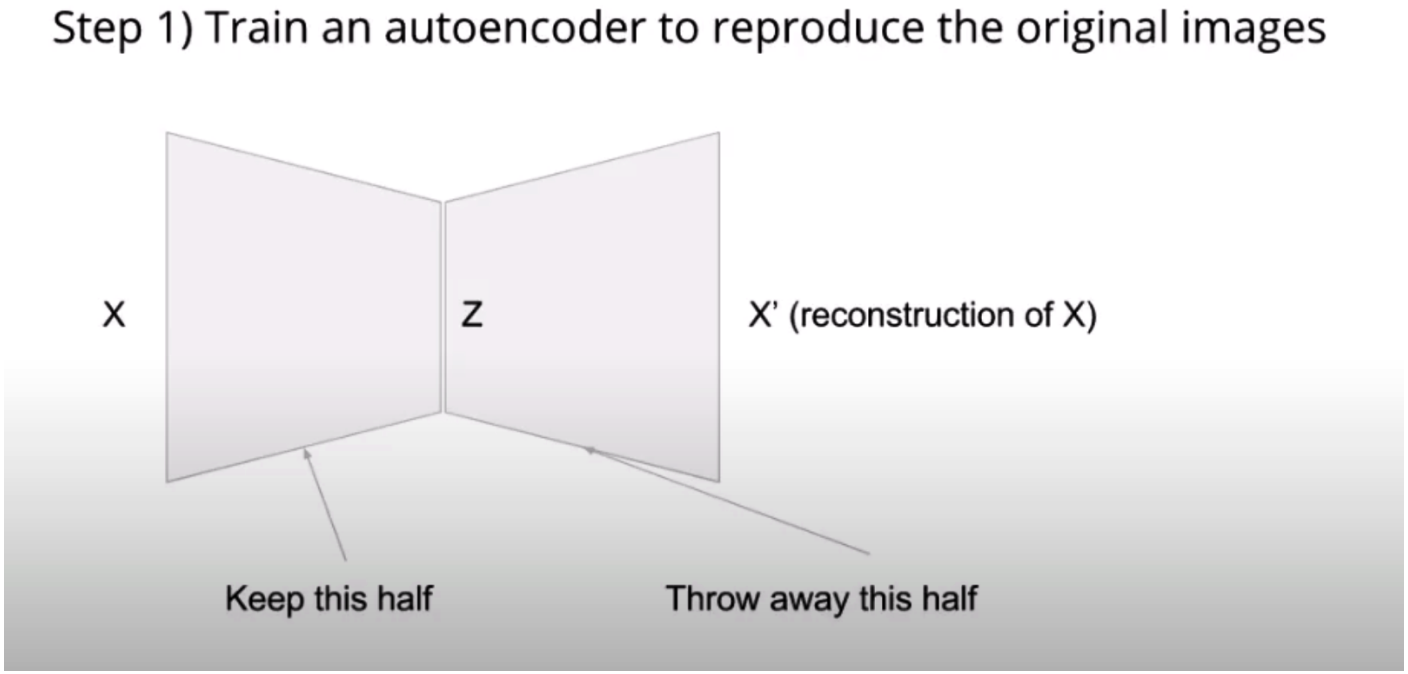
Solution:

Use the unsupervised learning as initialization
→ export the layer



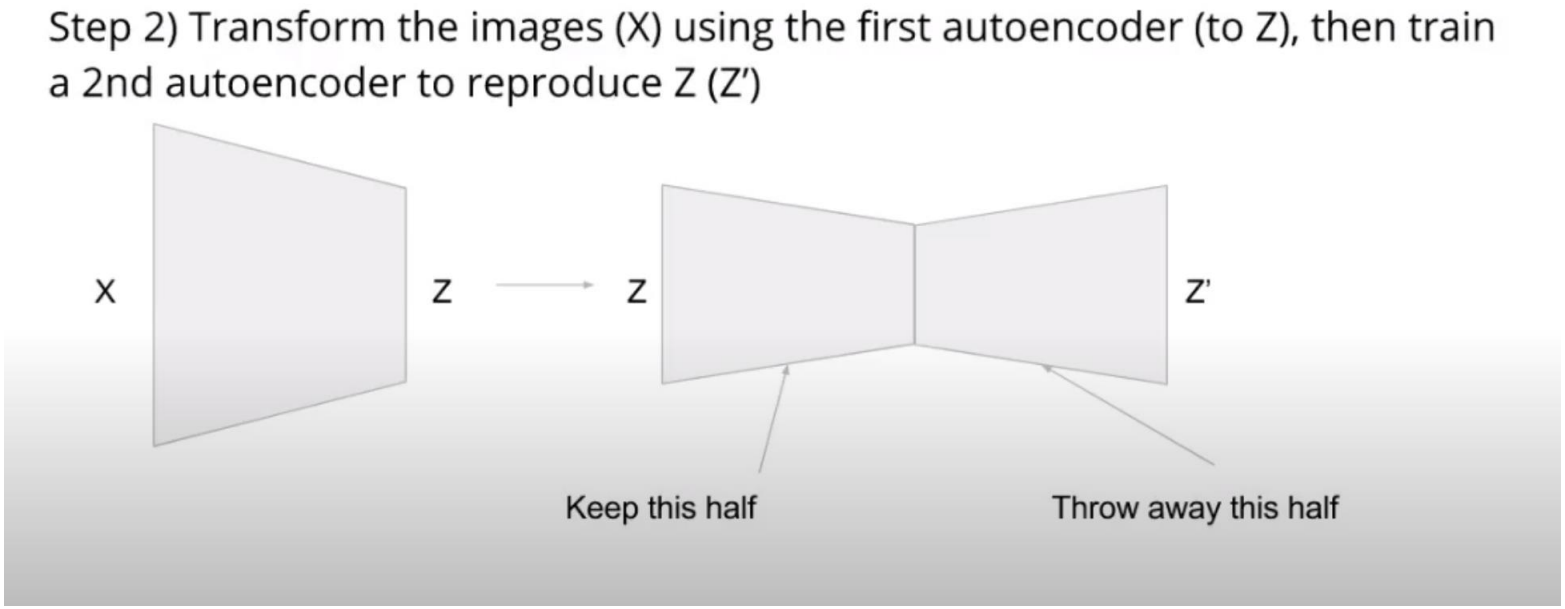
Solution: Greedy Layer-Wise Training

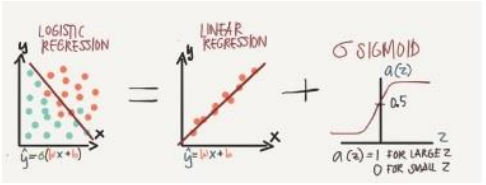
- 1) Train first layer using your data without the labels (unsupervised)



Solution: Greedy Layer-Wise Training

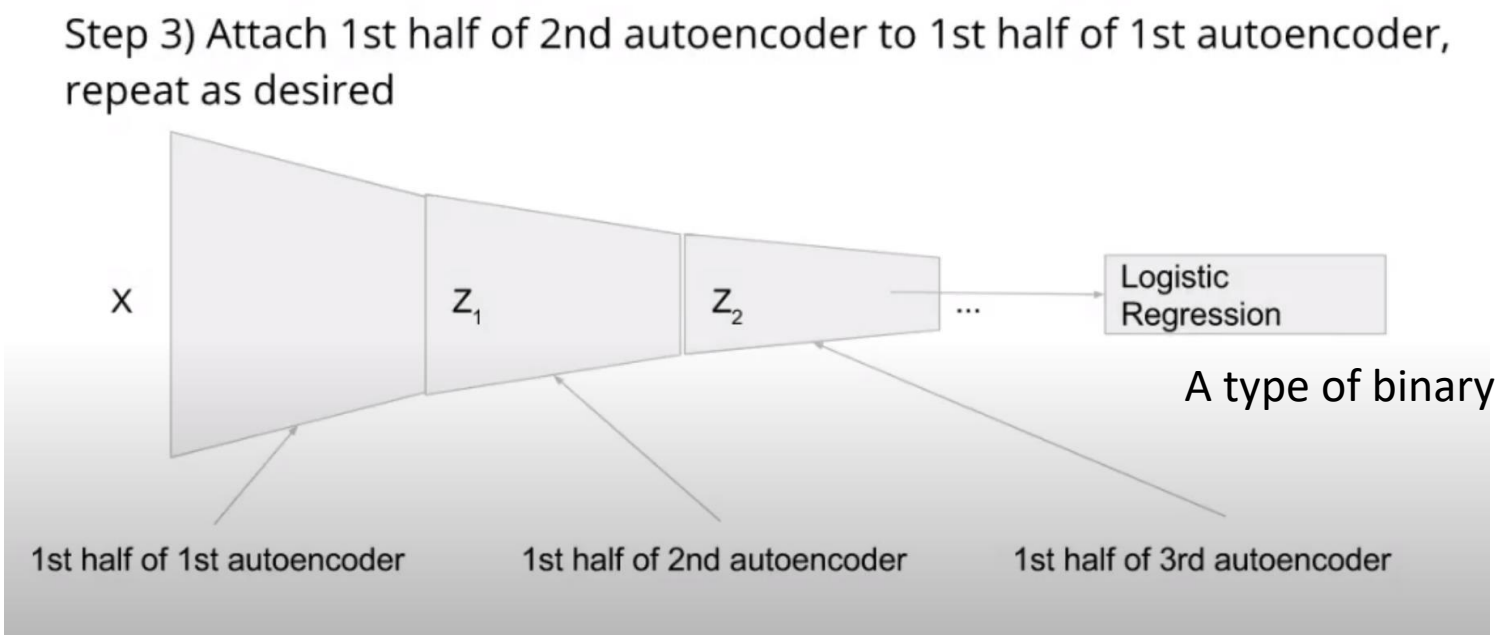
- 2) Then freeze the first layer parameters and start training the second layer using the output of the first layer as the unsupervised input to the second layer





Solution: Greedy Layer-Wise Training

- 3) Repeat this for as many layers as desired
- Use the outputs of the final layer as inputs to a supervised layer/model and train the last supervised layer(s) (leave early weights frozen)

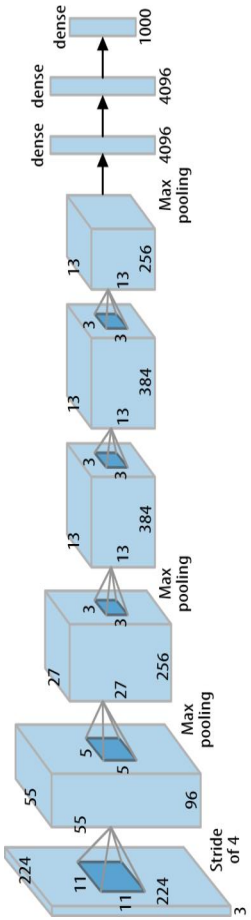


A type of binary classifier

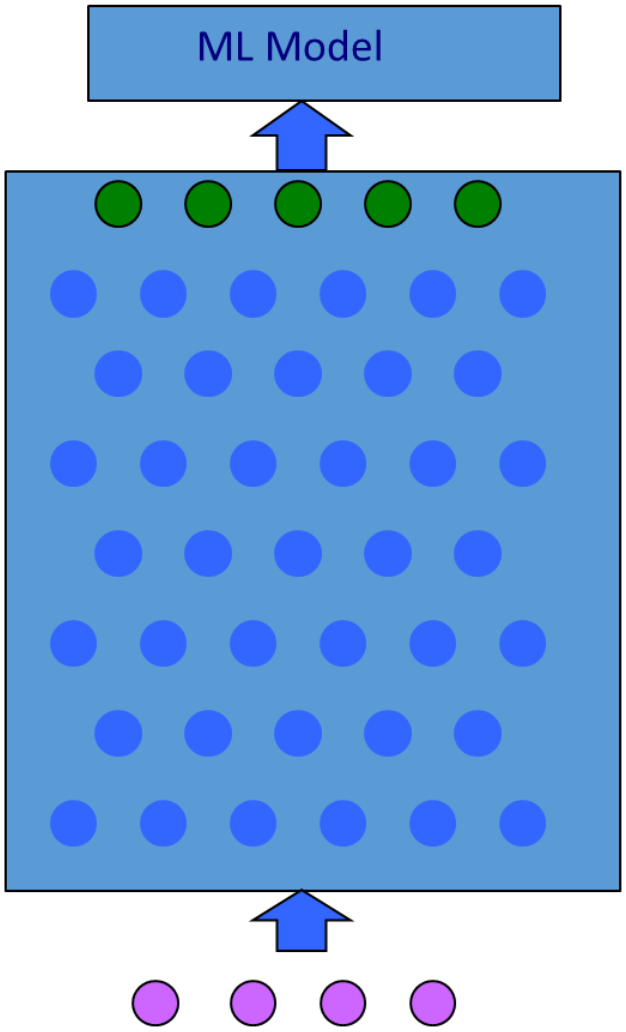
Greedy Layer-Wise Training:
resuming

1. **Train** first layer using your data without the labels (unsupervised)
2. Then **freeze** the first layer parameters and start training the second layer using the output of the first layer as the unsupervised input to the second layer
3. **Repeat** this for as many layers as desired
4. Use the outputs of the final layer as inputs to a supervised layer/model and **train the last supervised layer**(s) (leave early weights frozen)
5. Unfreeze all weights and **fine tune the full network** by training with a supervised approach, given the pre-training weight settings

Deep Net with Greedy Layer Wise Training



New Feature Space



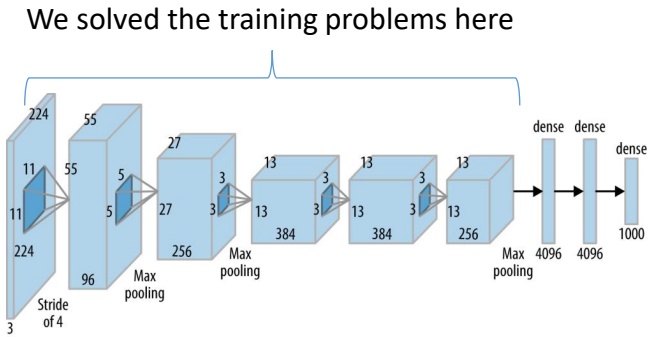
Supervised Learning

Unsupervised Learning

Original Inputs

Greedy Layer-Wise Training

- Greedy layer-wise training avoids many of the problems of trying to train a deep net in a supervised fashion
  - **Each layer gets full learning focus** in its turn since it is the only current "top" layer
  - Can take advantage of **unlabeled data**
  - When you finally **tune the entire network** with supervised training the network weights have already been adjusted so that you are in a good error basin and just need fine tuning. This helps with problems of
    - Ineffective early layer learning
    - Deep network local minima



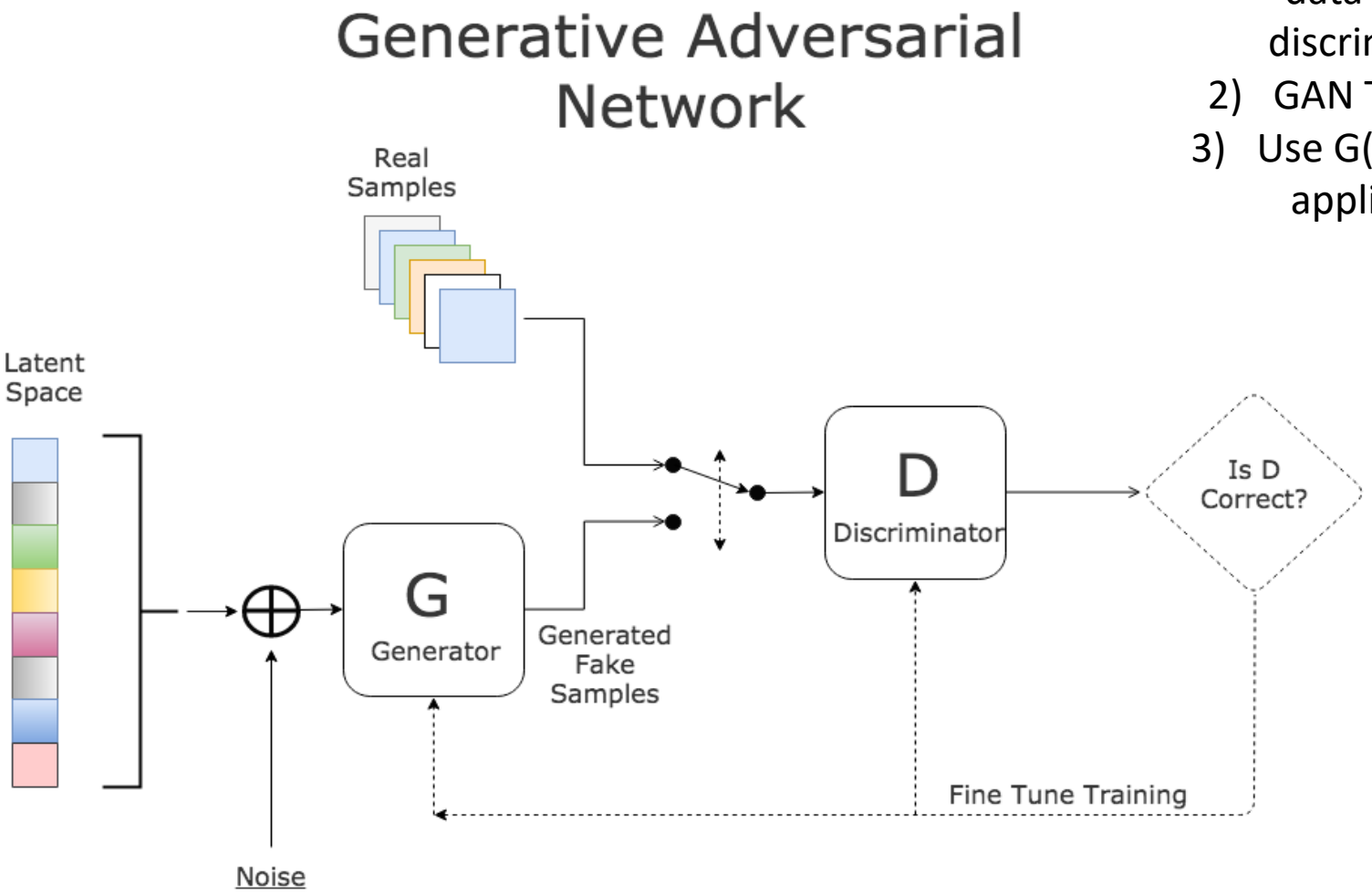
Generative Adversarial Networks (GAN)

Creating a pattern/image from a random number.

A different point of view to train Deep Neural Networks

GANs Net

1) A known dataset is used as initial training data for the discriminator.
2) GAN Training
3) Use G() in your application



Real or created by a GAN?

GAN

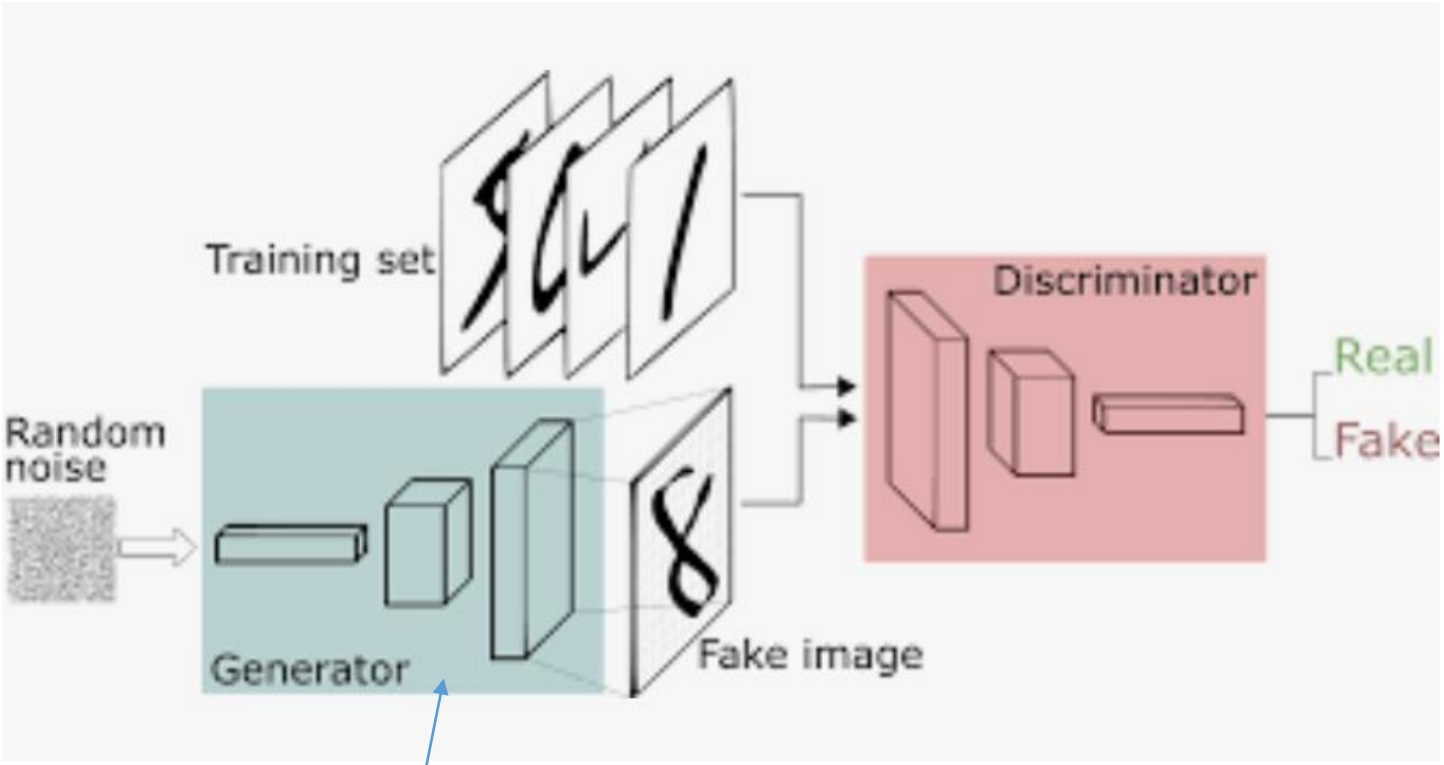
- Data augmentation
- Attacks
- Simulations of different conditions (style transfer)
- …

Age progression

Day/Night

Winter/
Summer

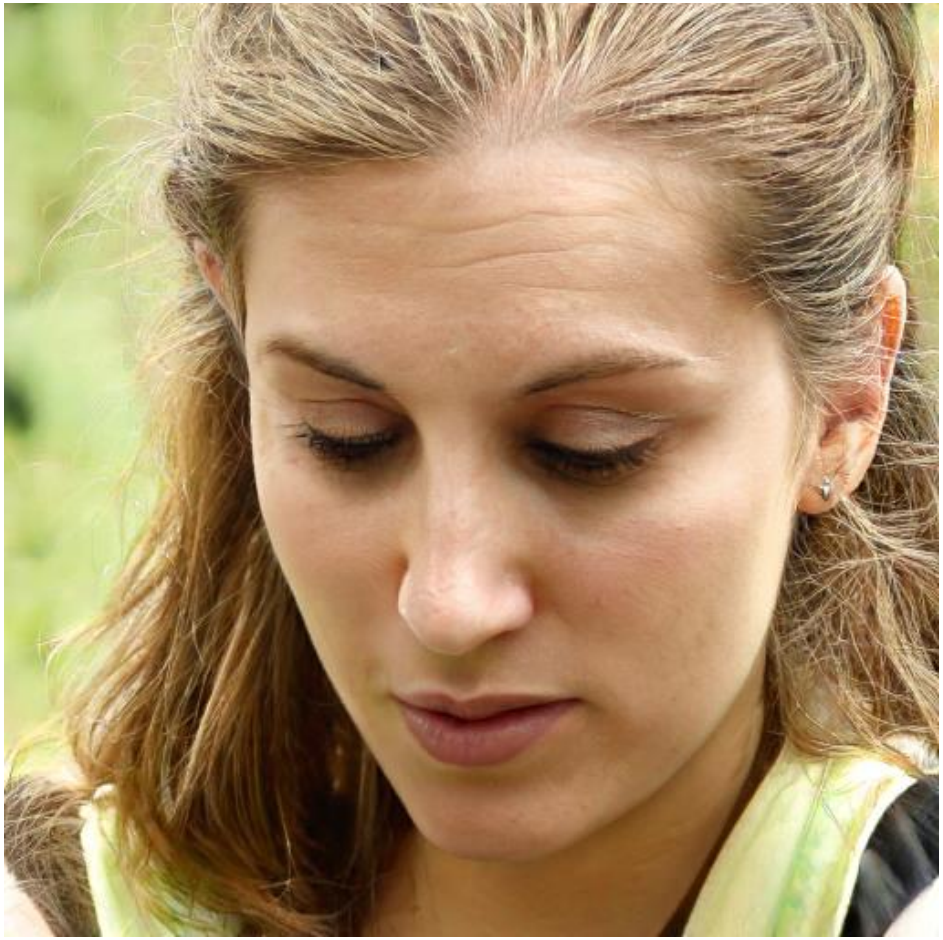
GANN:
Two networks competing

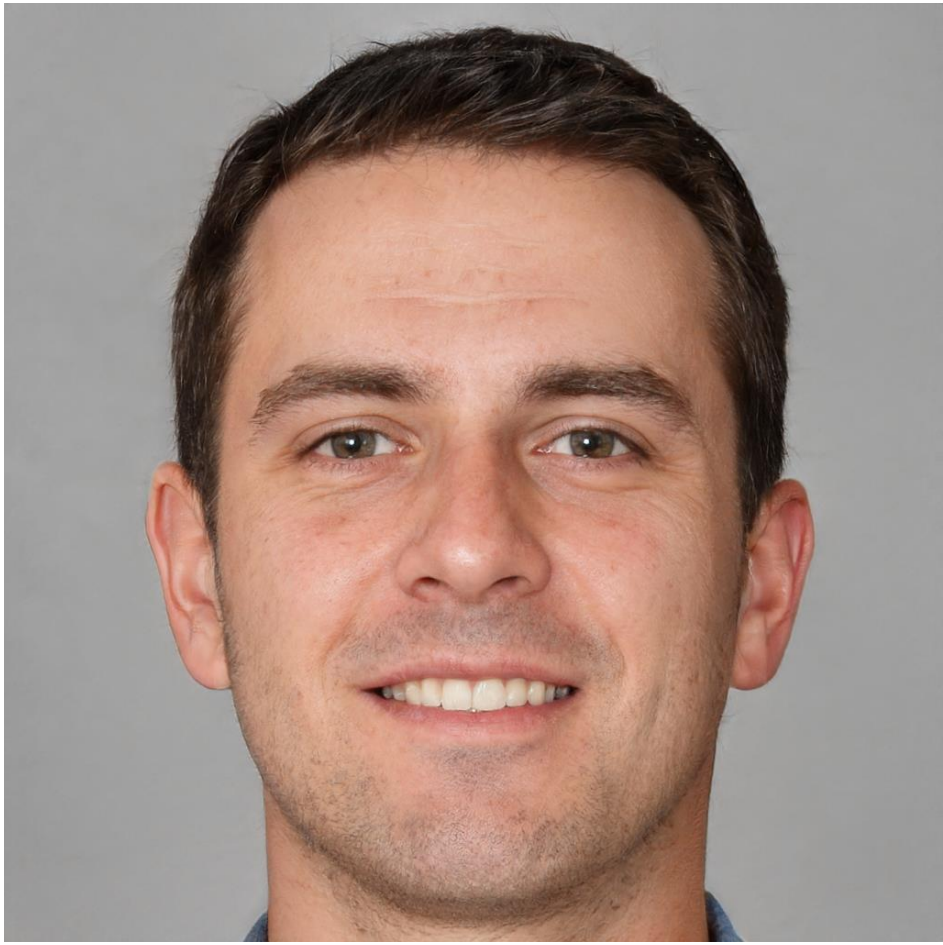


Note the «shape» of the Convolutional layers

Real or NN-generated?

Imagined by a GAN (generative adversarial network) StyleGAN2 (Dec 2019) - Karras et al. and Nvidia



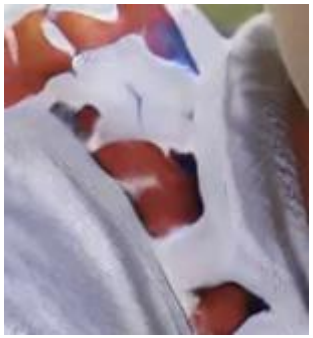


https://www.thispersondoesnotexist.com/

Example of fake faces

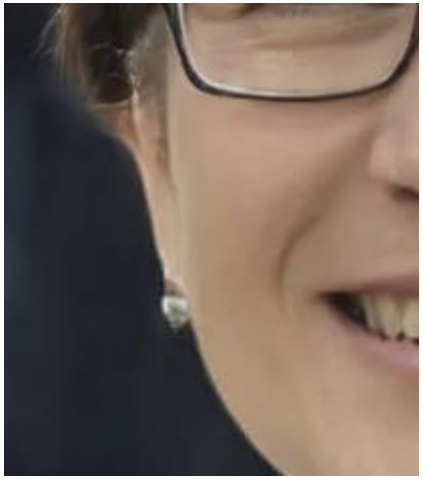
«Gestalt» is still missing…

The analysis of the context is revealing the GAN errors

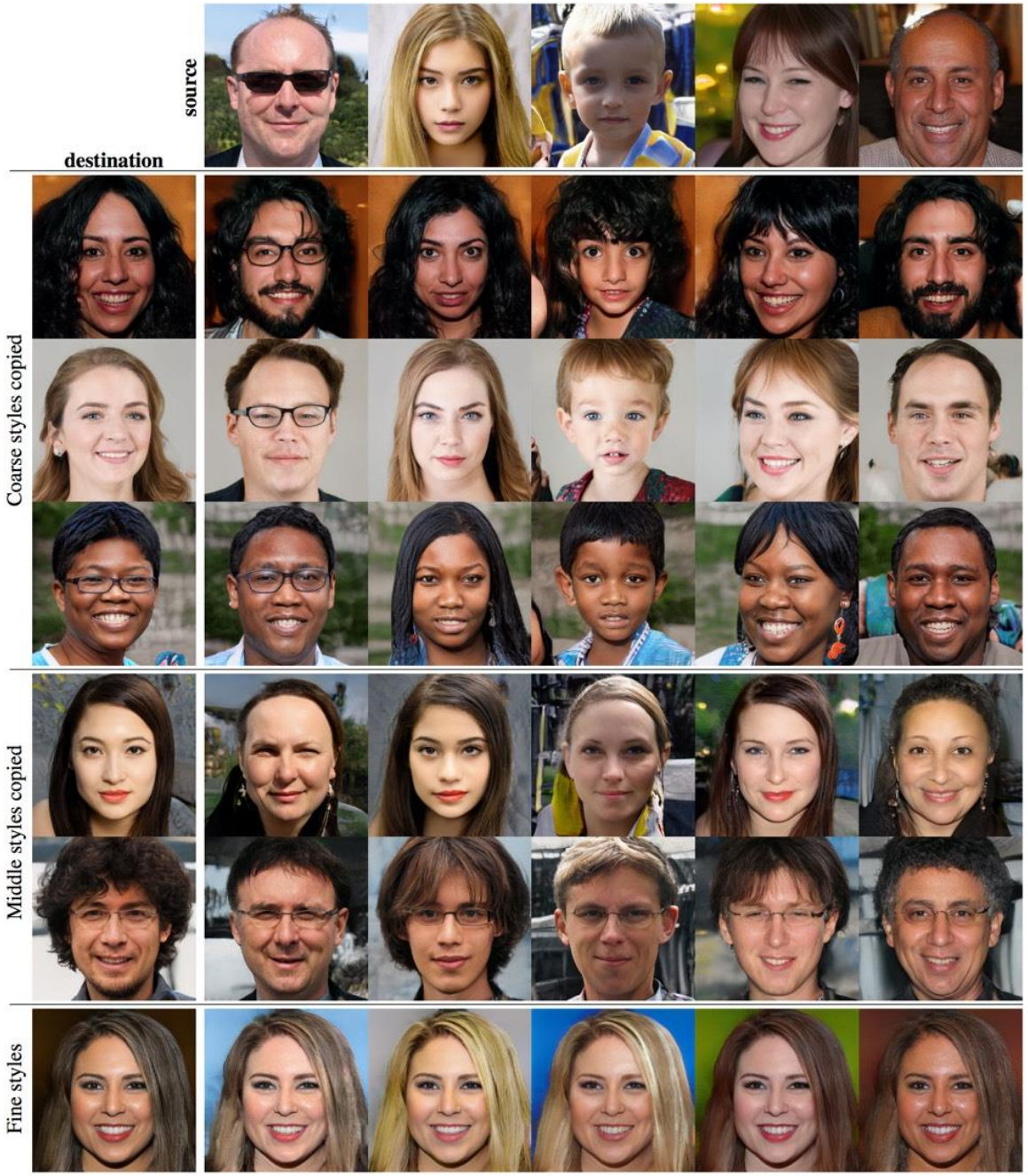


The fusion of the features of different levels is not (yet) perfectly realistic

Result using StyleGAN2 (Dec 2019) are improved…



Style transfer by CNN (GAN)



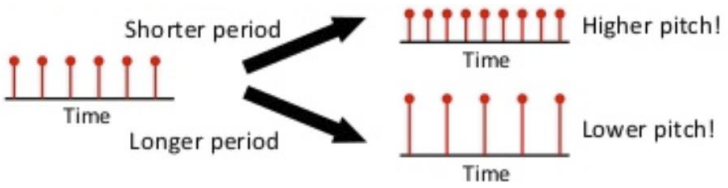
Take your time to appreciate the combinations…

In the following lessons we will study how to do it.

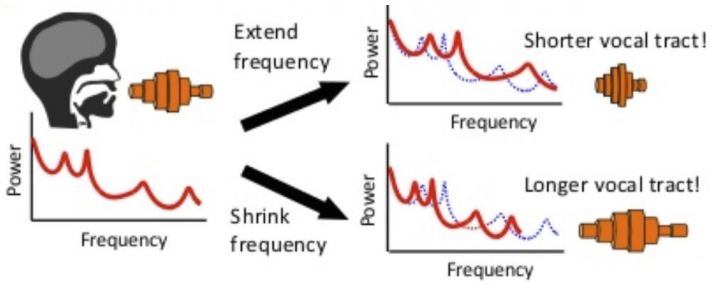
Voice conversion

- Voice transformation process aiming to convert one speaker's voice towards that of another.

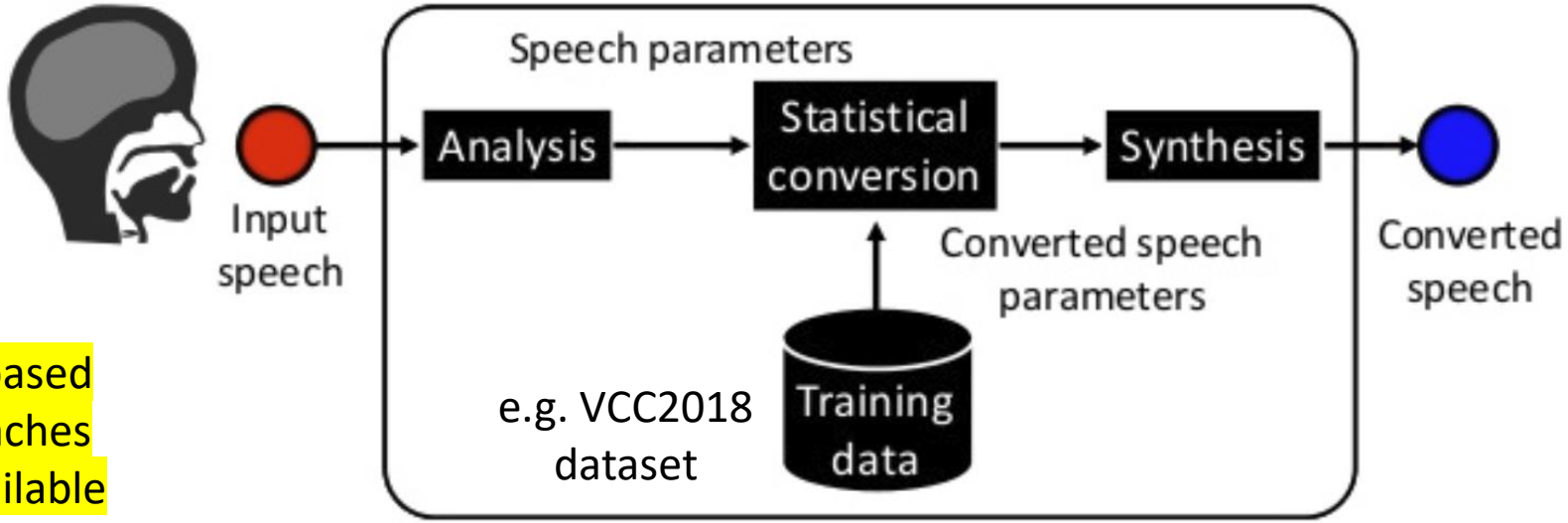
a) Excitation parameters (e.g., freq.)



b) Resonance parameters



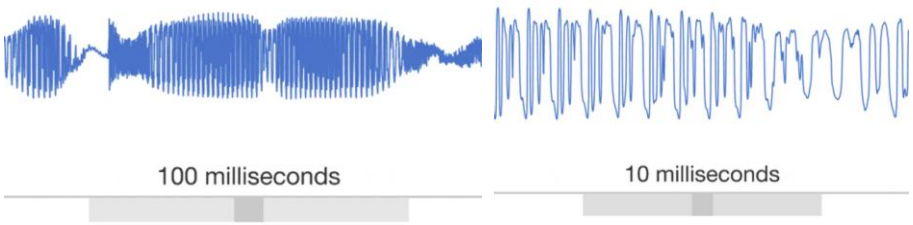
→Deep Autotuner



CNN-based approaches are available

Plots from Tokomi Toda, Advanced Voice Conversion, Nagoya Univesity, 2018

Speech Synthesis

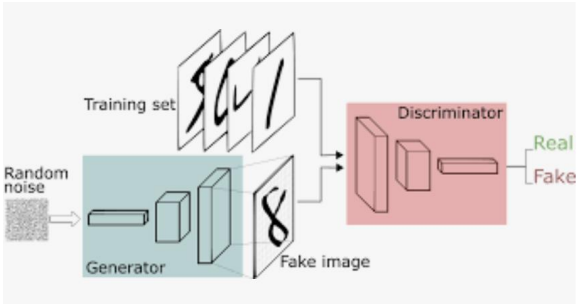


Example of WaveNet outputs

- Speech synthesis (commonly referred to as text-to-speech - TTS) generate intelligible, natural sounding artificial speech from any arbitrary text
- A general purpose method, but useful to attack the system in case of challenge-response methods
- Models used range from Hidden Markov Models to deeplearning systems
- Basic AVSs can be easily spoofed by speech synthesis inputs
- Recent public models
  - **WaveNet**, a deep generative model of raw audio waveforms
  - Tacotron: Towards End-to-End Speech Synthesis
- Countermeasures: at early stages..

Tacotron: Towards End-to-End Speech Synthesis, Interspeech 2017

GAN art



October 25th 2018

Christie's sold a portrait for $432,000 that had been generated by a GANN, based on open-source code written by Robbie Barrat of Stanford.

(congratulations!)

Main points

- Autoencoders
- Training and design of deep learning models
  - Fine Tuning
  - Transfer Knowledge
  - Data augmentation
- Automatic Feature Extraction
- Problems of the training of deep learning models.
- A solution: Greedy Layer-Wise Training
- Generative Adversarial Networks
  - Generation of realistic synthetic images