Section 1
From neural network to deep learning

Neurons

Limitation of traditional NN

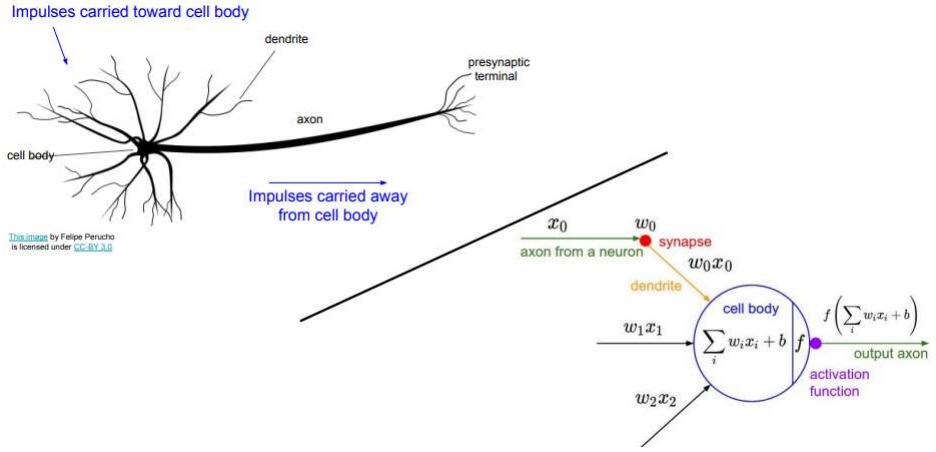
The success of convolution operation

Course resources

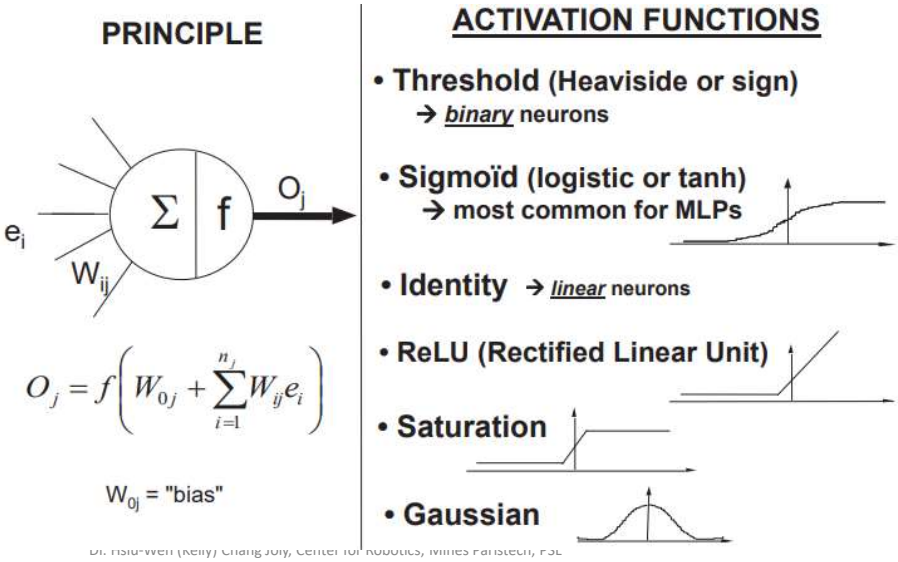
- Fabien Moutard:
https://github.com/fabienMoutarde/DLcourse
- Stanford class cs231n by Fei-Fei Li,A.Karpathy and J.Johnson:
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Review of pre-required concepts

- With the understanding of how our human brains works (although not completely), the most success "imitate" human brain modules are:

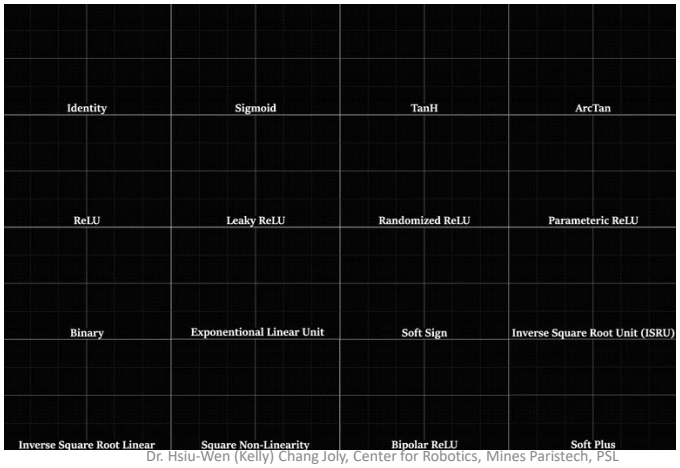


- However, the possible type of neurons (activation) might not be that simple as well as the way synaptic weights connect to each other



Modern approaches (1)

- In the past decades, more researches have found the new design of activation functions that improve specific tasks



Modern approaches (ReLU)

- Rectified Linear Unit (ReLU) is proposed by Hahnloser et al. in 2000

$$ReLU(\mathrm{x}) = \max(0, \mathrm{x})$$

$$ReLU'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

- Pros:
  - Less time and space complexity, because of sparsity, and compared to the sigmoid, it does not evolve the exponential operation, which are more costly.
  - Avoids the vanishing gradient problem.
- Cons:
  - Introduces the *dead relu* problem, where components of the network are most likely never updated to a new value. This can sometimes also be a pro.
  - ReLUs does not avoid the exploding gradient problem

Modern approaches (ELU)

- Exponential Linear Unit (ELU) is proposed by D.A. Clevert et al. 2016

$$ELU\ (x) = \begin{cases} x & \text{if} \quad x > 0 \\ \alpha(e^x - 1) & \text{if} \quad x \leq 0 \end{cases}$$

$$ELU'(x) = \begin{cases} 1 & \text{if} \quad x > 0 \\ \alpha e^x & \text{if} \quad x \leq 0 \end{cases}$$

α is around 0.1 to 0.3

- Pros
  - Avoids the *dead relu* problem.
  - Produces negative outputs, which helps the network nudge weights and biases in the right directions.
  - Produce activations instead of letting them be zero, when calculating the gradient.
- Cons
  - Introduces longer computation time, because of the exponential operation included
  - Does not avoid the exploding gradient problem
  - The neural network does not learn the alpha value

Modern approaches (Leaky ReLU)

- Leaky Rectified Linear Unit is proposed by AL Maas et al. 2013

$$LReLU\ (x) = \begin{cases} x & \text{if} \quad x > 0 \\ \alpha x & \text{if} \quad x \leq 0 \end{cases}$$

$$LReLU'(x) = \begin{cases} 1 & \text{if} \quad x > 0 \\ \alpha & \text{if} \quad x \leq 0 \end{cases}$$

α is around 0.1 to 0.3

- Pros
  - Like the ELU, we avoid the *dead relu* problem, since we allow a small gradient, when computing the derivative.
  - Faster to compute then ELU, because no exponential operation is included
- Cons
  - Does not avoid the exploding gradient problem
  - The neural network does not learn the alpha value
  - Becomes a linear function, when it is differentiated, whereas ELU is partly linear and nonlinear.

Modern approaches (GELU)

- Gaussian Error Linear Unit. An activation function used in the most recent Transformers – Google's BERT and OpenAI's GPT-2. It is proposed by Hendrycks, Dan; Gimpel, Kevin (2016)

$$GELU\ (x) = 0.5x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right)\right)$$

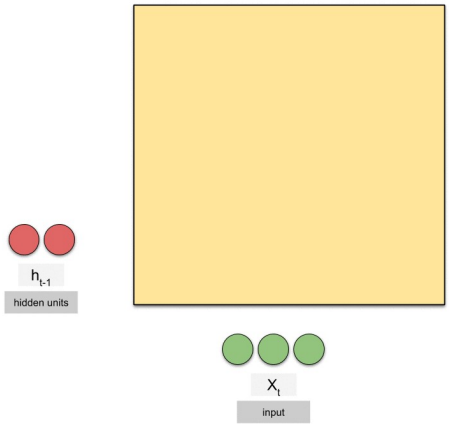
$$GELU'(x) = 0.5\tanh(0.0356774x^3 + 0.797885x) + (0.535161x^3 + 0.398942x)\,\mathrm{sech}^2(0.0356774x^3 + 0.797885x) + 0.5$$

- Pros
  - Seems to be state-of-the-art in NLP, specifically Transformer models – i.e. it performs best
  - Avoids vanishing gradients problem
- Cons
  - Fairly new in practical use, although introduced in 2016.

Good resource to understand how you choose/test activation functions: https://mlfromscratch.com/activation-functions-explained/#relu

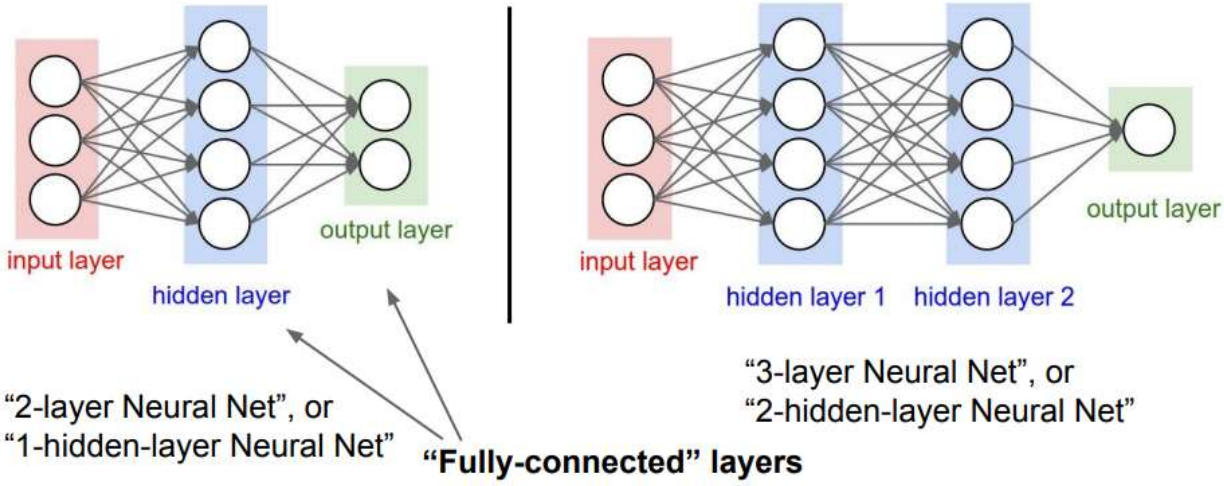
Network of formal neurons

- Feed-forward networks
  - No feedback connection
  - The output depends only on current input (No memory)
- Feedback or recurrent networks
  - Some internal feedback/backwards connection
  - All previous inputs can be the input (some memory system)
  - Tend to be used with temporal data or Natural Language Processing (NLP)



Vanilla RNN created by Raimi Karim

Networks architecture



*Also called Multi-layer perceptron (MLP)

Methodology for supervised training

Training set vs Test set

- Space of possible input values usually infinite, and training set is only a FINITE subset
- Zero error on all training examples ≠good results on whole space of possible inputs (generalization error ≠empirical error…)
- Need to collect enough and representative examples (very critical in deep neural network)
- Essential to keep aside a subset of examples that shall be used only as TEST SET for estimating final generalization (when training finished)
- Need also to use some "validation set" independent from training set, in order to tune all hyper-parameters (layer sizes, number of iterations, etc…)

Optimize hyper-parameters by Validation

- To avoid over-fitting and maximize generalization, absolutely essential to use some VALIDATION estimation, for optimizing training hyper-parameters (and stopping criterion):
  - either use a separate validation dataset (random split of data into Training-set + Validation-set)
  - or use CROSS-VALIDATION:
    - Repeat k times: train on (k-1)/k proportion of data + estimate error on remaining 1/k portion
    - Average the k error estimations

3-fold cross-validation:

- **Train on S1∪S2 then estimate errS3 error on S3**
- **Train on S1∪S3 then estimate errS2 error on S2**
- **Train on S2∪S3 then estimate errS1 error on S1**
- **Average validation error: (errS1+errS2+errS3)/3**

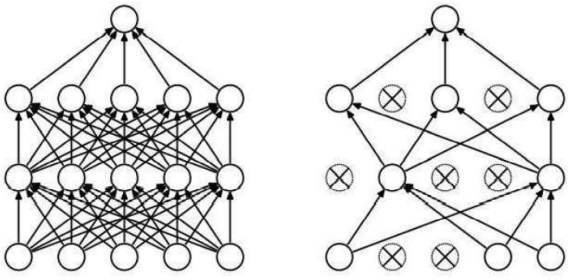
Some Neural Networks training "tricks"

- Importance of input normalization
  - zero mean, unit variance)
- Importance of weights initialization
  - random but SMALL and prop. to 1/sqrt(nbInputs)
- Decreasing (or adaptive) learning rate
- Importance of training set size
  - If a Neural Net has a LARGE number of free parameters

  ➔train it with a sufficiently large training-set!
- Avoid overfitting by Early Stopping of training iterations
- Avoid overfitting by use of L1 or L2 regularization

For ConvNet (or network with huge amount of training parameters), Dropout technique is used to avoid overfitting

Dropout



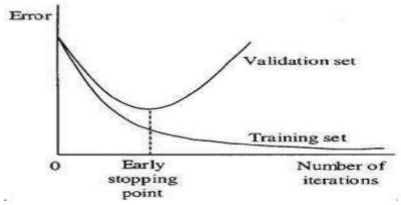
(a) Standard Neural Net (b) After applying dropout.

At each training stage, individual nodes can be temporarily "dropped out" of the net with probability p (usually ~0.5), or re-installed with last values of weights

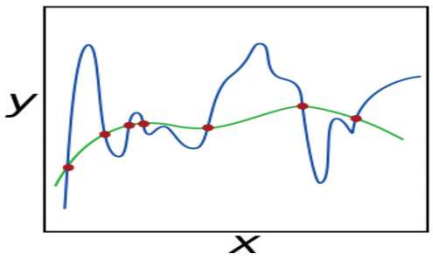
Early stopping

- For Neural Networks, a first method to avoid overfitting is to STOP LEARNING iterations as soon as the validation_error stops decreasing
- Generally, not a good idea to decide the number of iterations beforehand. Better to ALWAYS USE EARLY STOPPING

ALWAYS USE EARLY STOPPING



Regularization penalty



Trying to fit too many free parameters with not enough information can lead to overfitting

Regularization = penalizing too complex models
Often done by adding a special term to cost function

For neural network, the regularization term is just norm L2 or L1 of vector of all weights:

$$K = \sum_m (\text{loss}(Y_m, D_m)) + \beta \sum_{ij} |W_{ij}|^p \quad \text{with } p=2 \text{ (L2) or } p=1 \text{ (L1)}$$

→ name "Weight decay"

Breakthrough of Neural network

Deep-learning

Convolution neural network

So far…

- Aside from the problems of training, generalization, architecture, forward/backward connections, we can see the major obstacle of traditional NN is way it “SEE” the input data.
- In the old time, most of the practical tasks that implement NN are linear, smaller input dimension (such weather forecast, polynomial fitting, error estimation of navigation) and it involves expert experience in hidden layers design (what should be useful as feature?)
- It is roughly 53 years of slow development from the first workable neural network (1957, perceptron) to the modern architecture (2010)
- How can we encode our four dimensional world information in a much effective way to the computer ?

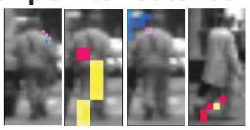
Importance of features in classical Machine learning

Traditional Machine Learning Flow

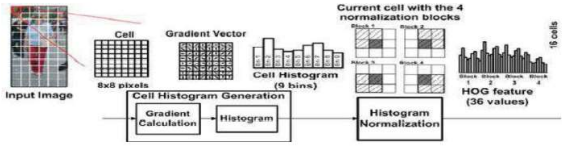
Examples of *hand-crafted* features

Haar features

Control-points features

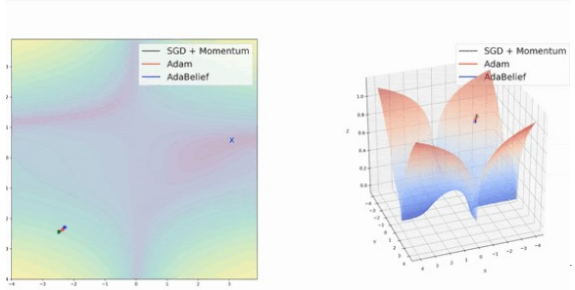


HoG (Histogram of Gradients)



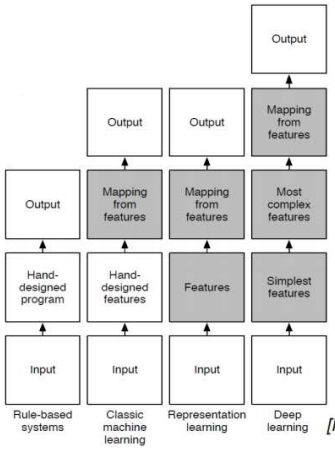
The turning point

- Convolution operation is proposed to solve the tasks especially in 2D or 3D images tasks➔ Section 2
- Recurrent network starts to have more mature architecture (LSTM, GRU) to “REMEMBER” and fuse information through time ➔ Section 4
- Reinforcement learning becomes more practical thanks to the computational power and the memory storage ➔ Section 5
- Aside from the type of neural network, deep learning succeeds to find several innovative ways (data augmentation, optimization algorithms) in training and it is on going research area



What is deep learning ?

Learning a *hierarchy* of increasingly abstract *representations*



[Figure from Goodfellow]

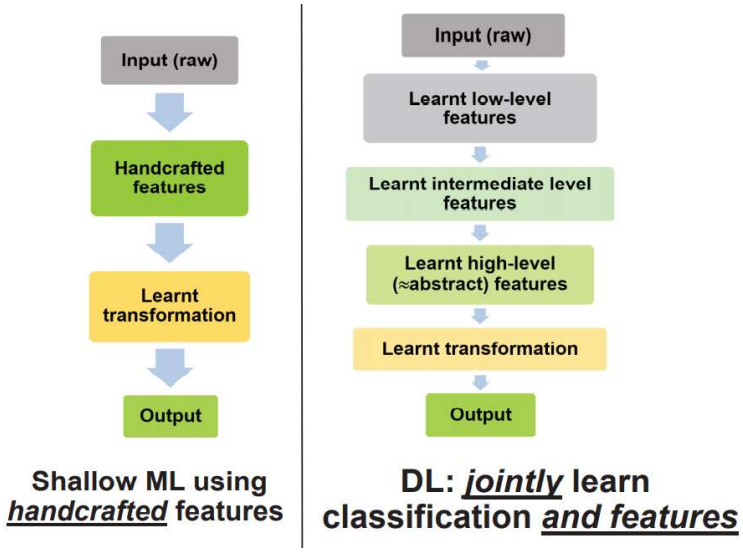
Increasing level of abstraction
Each stage ~ trainable feature transform

Image recognition
Pixel → edge → texton → motif → part → object

Speech
Sample → spectral band → … → phoneme → word

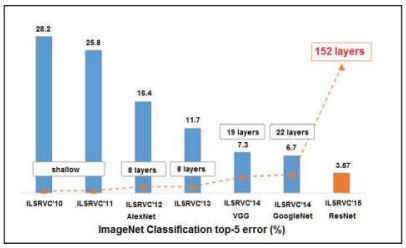
Text
Character → word → word group → clause → sentence → story

DL v.s. classical ML



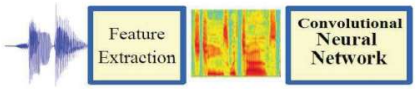
Deep learning recent breakthrough

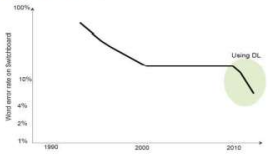
Very significant improvement over State-of-the-Art in Pattern Recognition / Image Semantic Analysis:



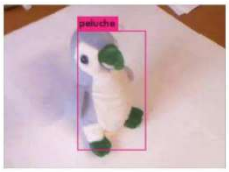
- **won many vision pattern recognition competitions (OCR, TSR, object categorization, facial expression,…)**
- **deployed in photo-tagging by Facebook, Google,Baidu,…**

Similar dramatic progress in Speech recognition + Natural Language Processing (NLP)





More applications



Object recognition

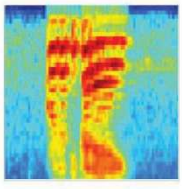
Scene analysis

Robotics

Все счастливые семьи похожи друг на друга, каждая несчастливая семья несчастлива по-своему.

Happy families are all alike. Every unhappy family is unhappy in its own way.

Language processing



Speech recognition

Medical diagnosis & Bio-informatics

Why features?

- Real data examples for a given task are usually not spread everywhere in input space, but rather clustered on a low-dimension « manifold »
- Example: Face images of 1000x1000 pixels

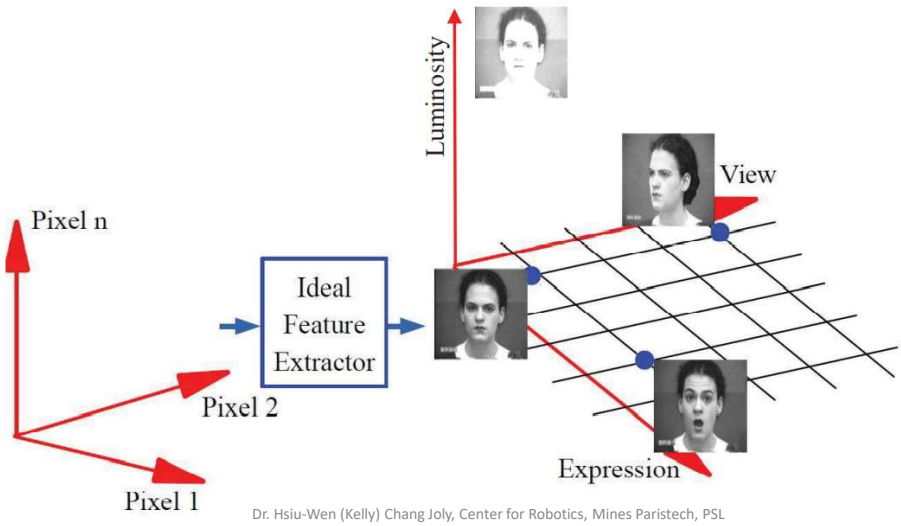
➔« raw » examples are vectors in R^1000000 !!

- BUT:
  - position = 3 cartesian coord
  - orientation 3 Euler angles
  - 50 muscles in face
  - Luminosity, color

  ➔Set of all images of ONE person has ≤ 69 dim

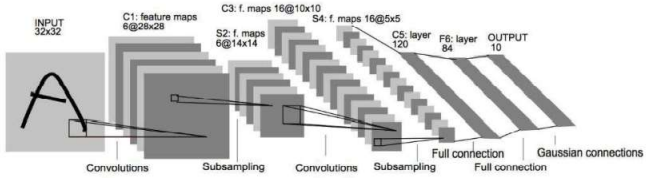
**This is why the powerful convolution operation is introduced!

Ideal features

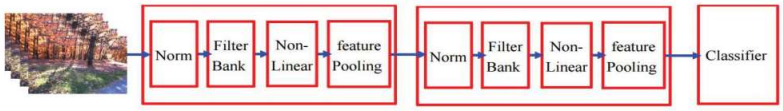


Convolution neural network (ConvNet)

- Proposed in 1998 by Yann LeCun (french prof.@ NYU, now also AI research director of Facebook). However, it took a long time to be popular



CNN called LeNet by Yann LeCun (1998)

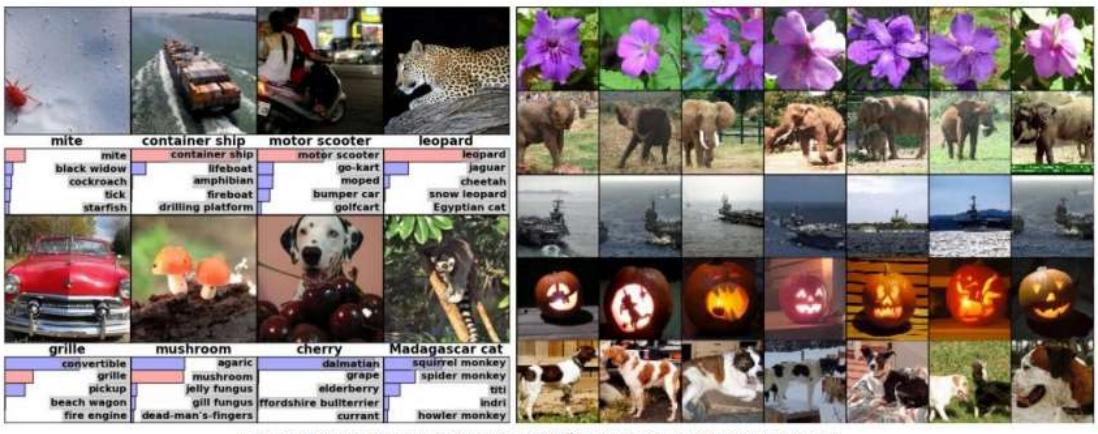


- **For inputs with correlated dims (2D *image*, 1D signal,…)**
- **Supervised learning**

ConvNet is everywhere

Classification

Retrieval



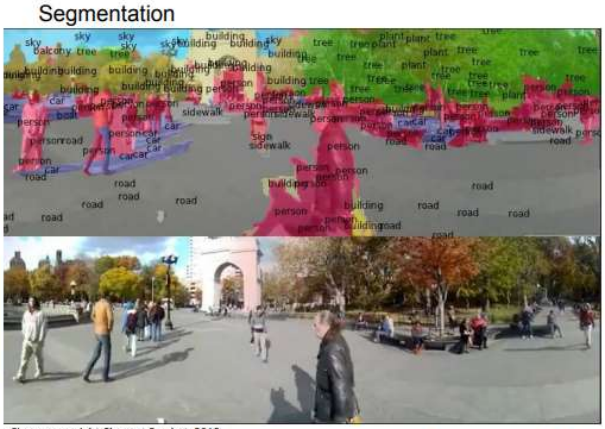
ConvNet is everywhere

Detection

Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

ConvNet is everywhere

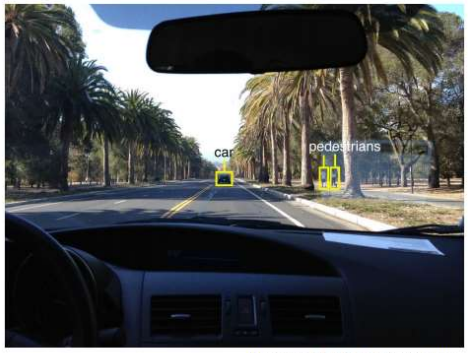
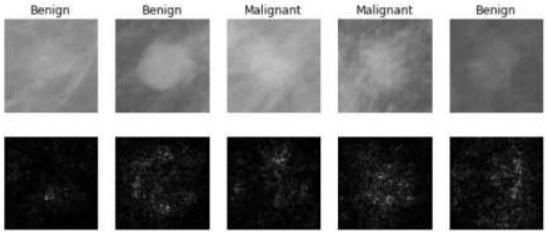


Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars



[Levy et al. 2016]

Figure copyright Levy et al. 2016. Reproduced with permission.

From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.

[Dieleman et al. 2014]

Dr. Hsiu-Wen (Kelly) Chang Joly, Center for Robotics, Mines Paristech, PSL

ConvNet is everywhere

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]

[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

ConvNet is everywhere

No errors | Minor errors | Somewhat related

A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

A woman is holding a cat in her hand

Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

A man riding a wave on top of a surfboard

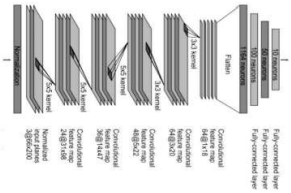
A cat sitting on a suitcase on the floor

A woman standing on a beach holding a surfboard

Captions generated by Justin Johnson using Neuraltalk2

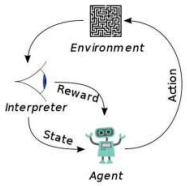
ConvNet is everywhere

ConvNet input:
Cylindrical projection of fisheye camera



ConvNet output:
steering angle

Imitation Learning from Human driving on real data



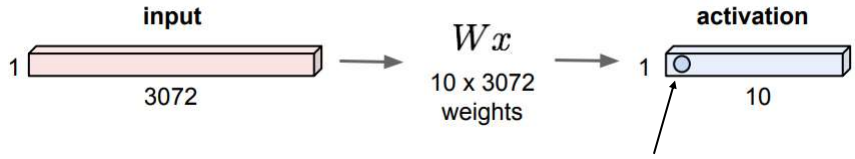
End-to-end driving via Deep ***Reinforcement*** Learning
[thèse CIFRE Valeo/MINES-ParisTech en cours]

Other image-based applications

- Image classification
- Visual object detection and categorization
- Semantic segmentation of images
- Tracking of human and cars

Fully connected layer

32x32x3 image -> stretch to 3072x1



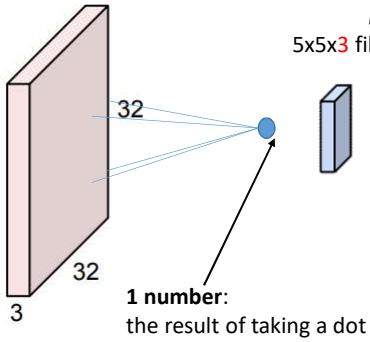
1 number:
the result of taking dot product between a row of W and the input (a 3072-dimensional dot product

Convolution layer

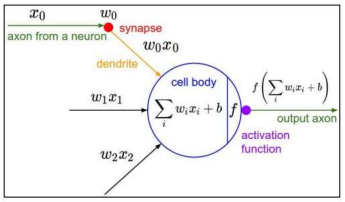
Filters always extend the full depth of the input volume

32x32x3 image (x)->
preserve spatial structure

5x5x3 filter (w)



Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"



It's just a neuron with local connectivity...

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias): $y = \phi(w^T x + b)$ where ϕ is the activation function

The output size of convolution

- There are three hyper parameters control the size of the output volume: the depth, stride and zero-padding
  - Depth(D): It corresponds to the number of filters would like to use, each learning to look for something different in the input.
  - Stride(S): the way we slide the filter. When the stride is 1 then we move the filter one pixel at a time. When the stride is N then the filter jump N pixels at a time. This will produce smaller output volumes spatially.
  - Padding(P): this number pad the input volume with zeros around the border. This is a nice feature that allow us to control the spatial size of the output volumes
- Formula for the calculating the output volume:

$$\frac{W - F + 2P}{S} + 1$$

Input volume: W, receptive field size:F

Summary of convolution layer

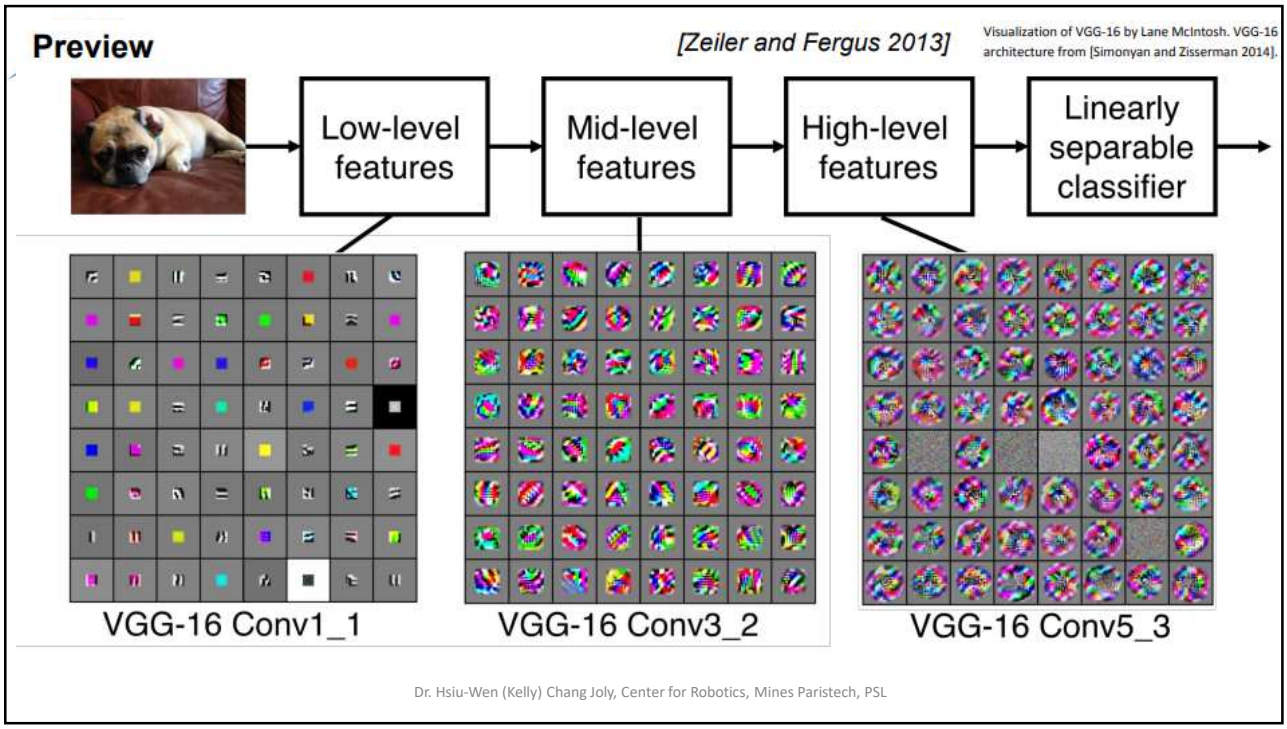
- Acceps a volume of $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - Their spatial extent F
  - The stride S
  - The amount of zero padding P
- Produces a volume of size $W_2 \times H_2 \times D_2$ where
  - $W_2 = \frac{W_1 - F + 2}{S} + 1$
  - $H_2 = \frac{H_1 - F + 2}{S} + 1$
  - $D_2 = K$

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

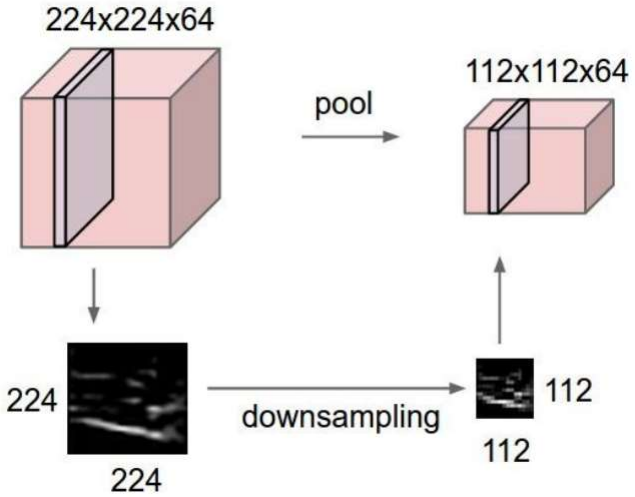
But wait! Why there is 1x1 convolution layers?

This is helpful and make sense because 1x1 convolution performs a K-dimensional dot product. It is used very often to reduce the depth



Pooling layer

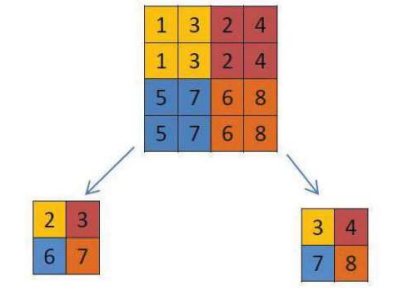
- makes the representations smaller and more manageable
- operates over each activation ma independently
- Noise reduction
- Small-translation invariance
- Small-scaling invariance



Pooling layers algorithm details

Parameters:

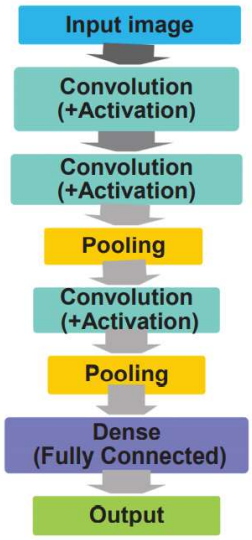
- **pooling size (often 2x2)**
- **pooling stride (usually = pooling_size)**
- **Pooling operation: *max*, average, Lp,…**



Example: 2x2 pooling, stride 2

Summary of ConvNet

Succession of Convolution (+ optional activation) layers and Pooling layers, which extract the hierarchy of features, followed by dense (fully connected) layer(s) for final classification



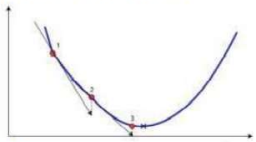
Overview of training

- One time step
  - activation functions, preprocessing, weight initialization, regularization, gradient checking
- Training dynamics
  - babysitting the learning process, parameter updates, hyperparameter optimization
- Evaluation
  - Model ensembles

ConvNet training

All successive layers of a convNet forms a Deep neural network (with weigh-sharing inside each conv. Layer, and specific pooling layers).

Training = optimizing values of weights&biases
Method used = gradient descent



➔ Stochastic Gradient Descent (SGD), using *back-propagation*:

- **Input 1 (or a few) random training sample(s)**
- **Propagate**
- **Calculate error (loss)**
- **Back-propagate through all layers from end to input, to compute gradient**
- **Update convolution filter weights**

They are the same like we face in machining learning problem

Datasets for image classification

Open source and public image dataset

Dataset	Total images	Categories	Images per category	Object per image	Size	Started year	Challenges competition
PASCAL VOC (Everinghame et al. 2010, 2015)	11540	20	303~4087	2.4	470x380	2005	VOC(2007~2012)
ImageNet (Deng et al. 2009)	14 millions+	21841		1.5	500x400	2009	ILSVRC(2010~2017)
MS COCO (Lin et al. 2014)	328000+	91		7.3	640x460	2014	MS COCO(2015~2018)
Open Images (Kuznetsova et al. 2018)	9 millions+	6000+		8.3	varied	2017	OICOD2018

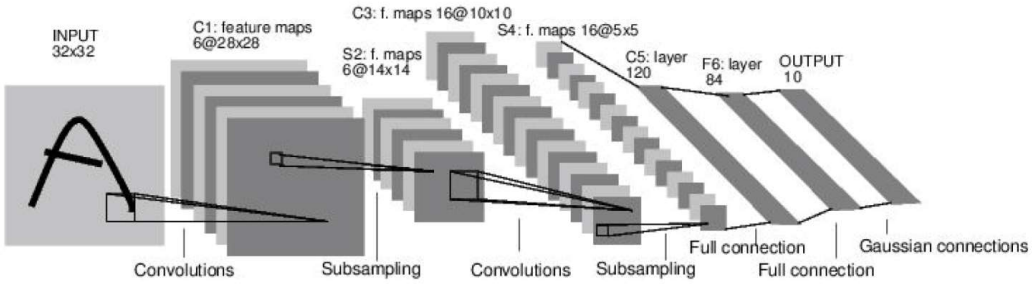
Revised from [Liu, L., Ouyang, W., Wang, X. *et al.* Deep Learning for Generic Object Detection: A Survey. *Int J Comput Vis* **128,** 261–318 (2020)]

Convolution Neural Network Architecture

Popular CNN architecture

- LeNet: 1st successful applications of ConvNets, by Yann LeCun in 1990's. Used to read zip codes, digits, etc.
- AlexNet: Beginning of ConvNet "buzz": largely outperformed competitors in ImageNet_ILSVRC2012 challenge. Architecture similar to LeNet (but deeper+larger, and some chained ConvLayers before Pooling). 60 M parameters !
- GoogLeNet: ILSVRC 2014 winner, developed by Google. Introduced an Inception Module, + AveragePooling instead of FullyConnected layer at output. Dramatic reduction of number of parameters (5M, compared to AlexNet with 60M).
- VGGNet: Runner-up in ILSVRC 2014. Very deep (16 CONV/FC layers) ➔140M parameters !!
- ResNet: ILSVRC 2015, "Residual Network" introducing "skip" connections. Currently ~ SoA in convNet. Very long training but fast execution.

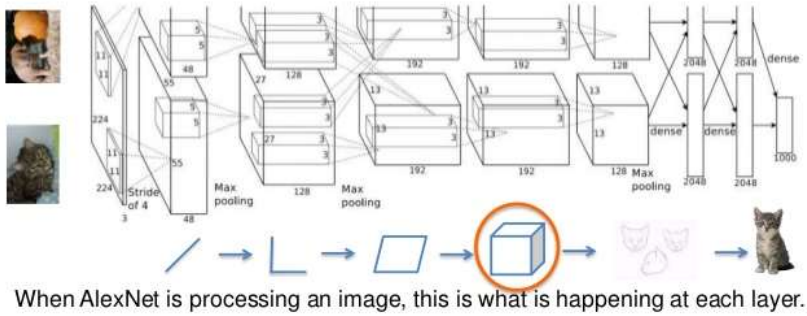
LeNet-5 architecture



Conv filters were 5x4, applied at stride 1 (C1)
Subsampling (Pooling) layers were 2x2 applied at stride 2
Finally there are 3 fully connected layers
[Conv-Pool-Conv-Pool-Conv-FC]

AlexNet architecture

- AlexNet (2012 winner) is still used today even though there are more accurate networks available, because of its relative simple structure and small depth. It is widely used in computer vision



When AlexNet is processing an image, this is what is happening at each layer.

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

ILSVRC winners

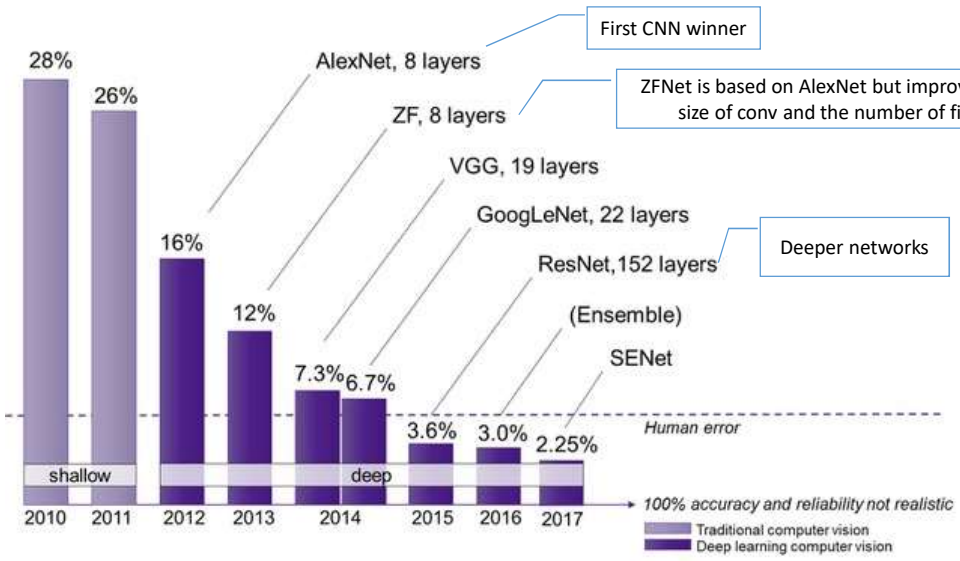
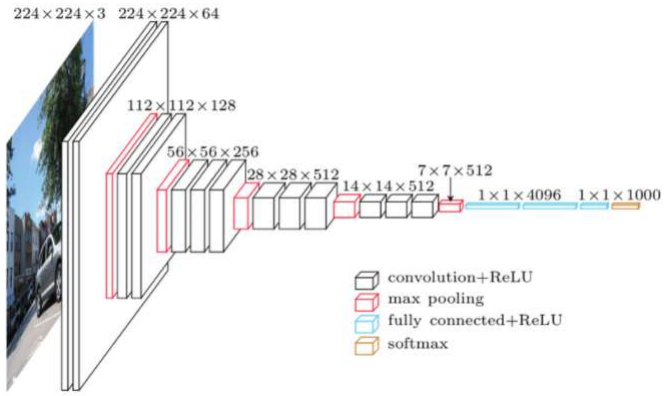


Figure copyright Gordon Cooper, 2019

VGGNet architecture



- VGG16 is a 16-layer neural network, not counting the max pooling layer and the softmax layer. Hence, it is known as VGG16.
- VGG19 consists of 19 layers.
- A pre-trained model is available in Keras for both Theano and TensorFlow backends.

VGGNet architecture

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

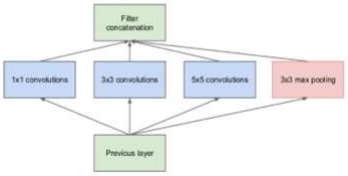
8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

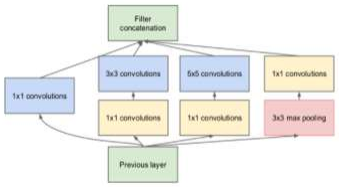
11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14

GoogLeNet architecture (2014)

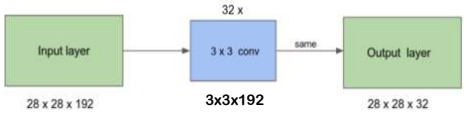
- In most of the architectures, the intuition is not clear why and when perform pooling and convolutional operation
  - AlexNet:1 conv+1 pooling
  - VGGNet: 3 convs+1 max pooling
- Inception model: a good local network topology (network within a network) and then stack these modules on top of each other
- It uses all the operations at the same time.It computes multiple kernels of different size over the same input map in parallel, concatenating their results into a single output.
- A pre-trained model is available in Keras for both Theano and TensorFlow backends.



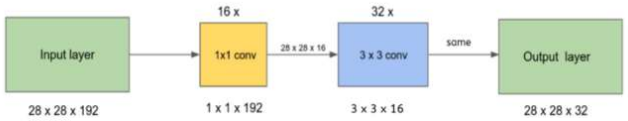
Q: What is the problem with this?



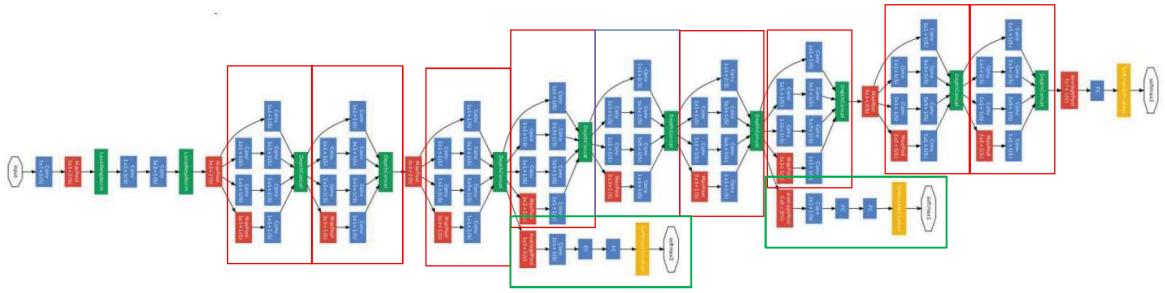
Inception module with dimension reduction



((28 x 28 x 3 x 3) x 192) x 32 ≃**43 M parameters**



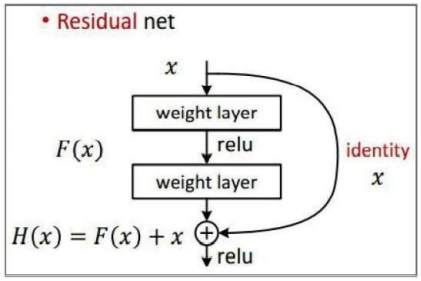
((28 x 28 x 1 x 1) x 192) x 16 + ((28 x 28 x 3 x 3) x 16) x 32 ≃**5.4 M parameters**



- Without expensive FC layers at the end. Instead, it use Global average
- 9 inception modules
- 2 auxiliaries softmax layer (AvgPool – 1x1 Conv – FC-FC-Softmax)
  - Their role is to push the network toward its goal and helps to ensure that the intermediate features are good enough for the network to learn
  - It turns out that softmax0 and sofmax1 gives regularization effect.
  - During training, their loss gets added to the total loss with a discount weight (the losses of the auxiliary classifiers were weighted by 0.3).
  - During inference, they are discarded
- 22 total layers with weights

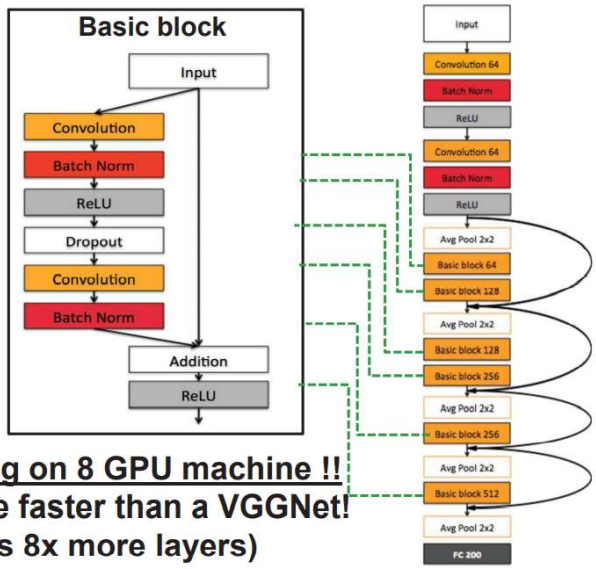
ResNet architecture (2015)

- After a certain depth, adding additional layers to feed-forward convNets results in a higher training error and higher validation error. When adding layers, performance increases only up to a certain depth, and then it rapidly decreases.
- In the **ResNet** (**Residual Network**) paper, the authors argued that this underfitting is unlikely due to the vanishing gradient problem, because this happens even when using the batch normalization technique.
- A new concept called **residual block** is proposed that the ResNet added connections that can skip layers
- Hypothesis: the problem is an optimization problem, deeper models are harder to optimize



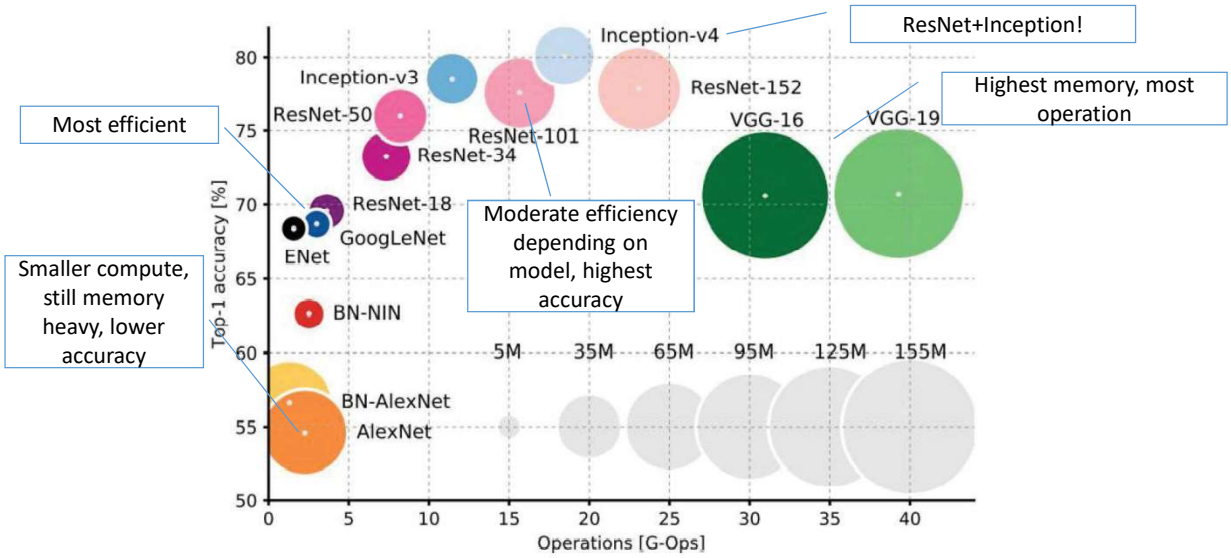
ResNet global architecture

- **152-layer module for ImageNet**
- **Swept all classification and detection competitions in ILSVRC15 and COCO15**



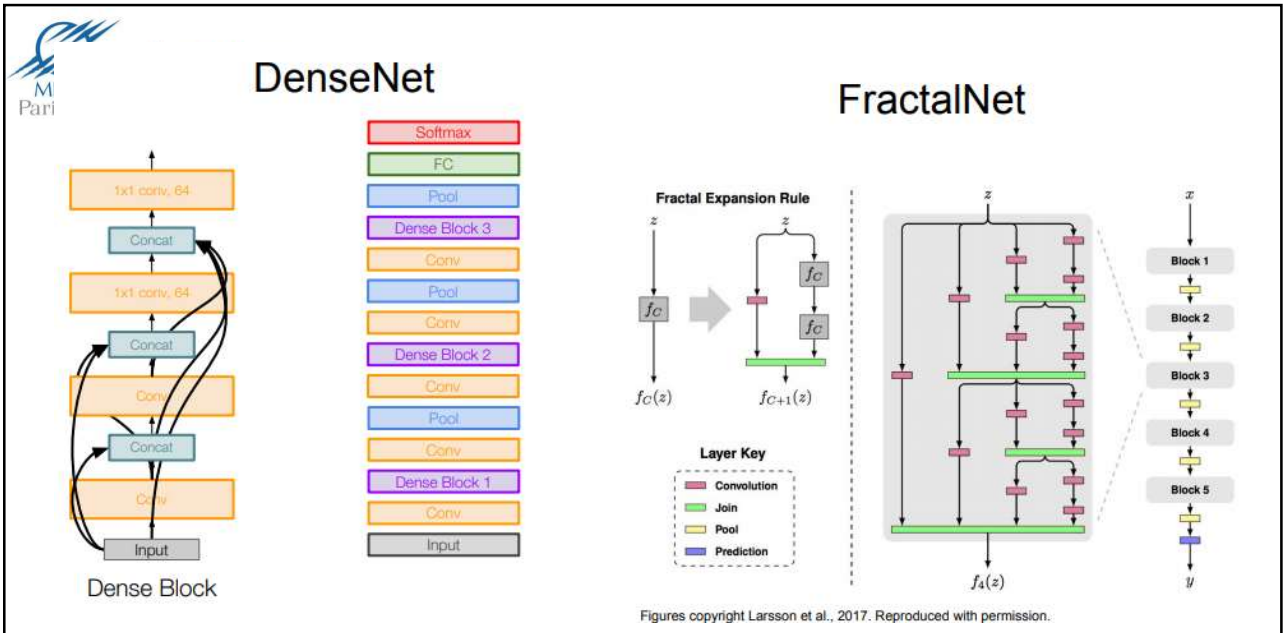
- **2-3 weeks of training on 8 GPU machine !!**
- **However, at runtime faster than a VGGNet! (even though it has 8x more layers)**

Performance comparison



Summary: CNN architecture

- VGG, GoogLeNet, ResNet all in wide use, available in model zoos
- ResNet current best default
- Trend towards extremely deep networks
- Significant research centers around design of layer / skip connections and improving gradient flow
- Even more recent trend towards examining necessity of depth vs. width and residual connections
- Other good architectures:
  - NiN [Lin et al. 2014]: (Network in Network. Insert MLP in between Conv. layer
  - Wide ResNet [azgoruyko et al. 2016]: User wider residual blocks (Fxk filters) is better
  - ResNeXT [Xie et al. 2016]: Wider residual blocks like inception module
  - Stochastic Depth [Huang et al. 2016]:randomly drop a subset of layers during training
  - DenseNet [Huang et al. 2017]: Dense blocks connected to every other layer in feedforward fashion
  - FractalNet [Larsson et al. 2017]: fusion of shallow and deep networks
  - SqueezeNet [Landola et al. 2017]: Fire modules consisting of squeeze and expand layers to increase the efficiency.



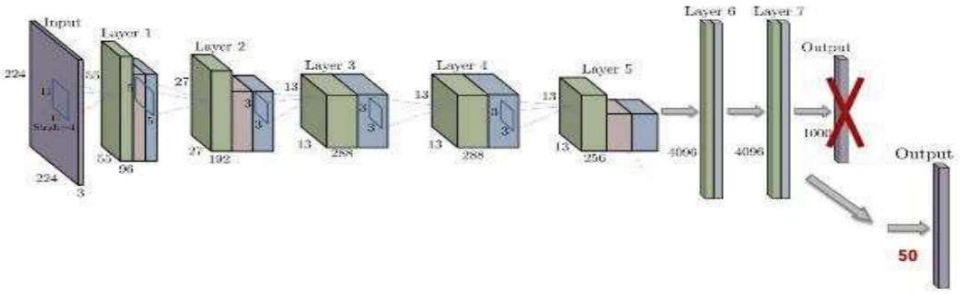
Figures copyright Larsson et al., 2017. Reproduced with permission.

Programming environments

- TensorFlow https://www.tensorflow.org
- KERAS https://keras.io
  - Python front-end APIs mapped either on Tensor-Flow or Theano back-end
- PyTorch https://pytorch.org/
- Caffe http://caffe.berkeleyvision.org/
  - C++ library, hooks from Python ➔ notebooks
- Theano http://www.deeplearning.net/software/theano/
- Lasagne http://lasagne.readthedocs.io
  - lightweight library to build+train neural nets in Theano

All of them handle transparent use of GPU, and most of them are used in Python code/notebook

Transfer learning



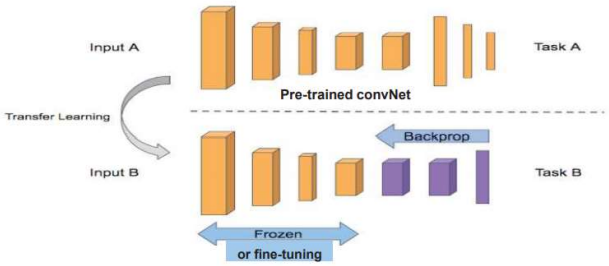
By removing *last layer(s)* (those for classification) of a convNet trained on ImageNet, one obtains a *transformation of any input image into a semi-abstract representation,* which can be used for learning SOMETHING ELSE (« transfer learning »):

- **either by just using learnt representation as features**
- **or by creating new convNet output and perform learning of new output layers + fine-tuning of re-used layers**

Transfer learning and fine-tuning

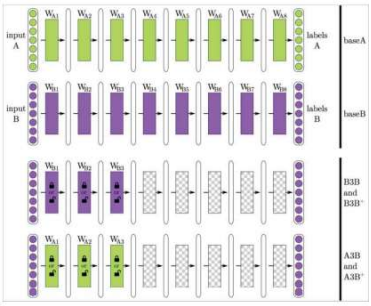
- **SoA convNets trained on ImageNet are image CLASSIFIERS for one object per image**
- **Many object categories can be irrelevant (e.g. boat in a office)**

➔ **For each application, models are usually obtained from state-of-the-art ConvNets pre-trained on ImageNet (winners of yearly challenge, eg: AlexNet, VGG, Inception, ResNet, etc…)**

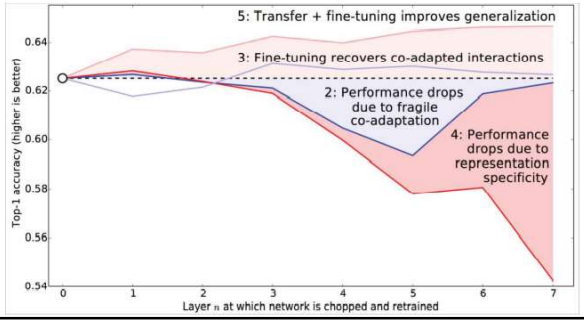


➔ **Adaptation is performed by *Transfer Learning*, ie modification+training of last layers and/or fine-tuning of pre-trained weights of lower layers**

Transfer-Learning even improves performances



[Yosinski, Clune, Bengio, Lipson, "How transferable are features in deep neural networks?", ICML'2014]



Some transfer-learning applications

- Learning on simulated synthetic images + fine-tuning on real-world images
- Recognition/classification for OTHER categories or classes
- Training an objects detector (or a semantic segmenter)
- Precise localization (position+bearing) = PoseNet
- Human posture estimation = openPose
- End-to-end driving (imitation Learning)
- 3D informations (depth map) from monovision!

Summary on ConvNets & Deep-Learning

- Proven advantage of learning features empirically from data
- Large ConvNets require huge amounts of labelled examples data for training
- Current research/progresses = finding efficient global architecture of ConvNets
- Enormous potential of TRANSFER-LEARNING on small datasets for restricted/specialized problems
- ConvNets also for multivariate time-series (1D temporal convolutions) and for 3D data (3D conv on voxels, etc…)
- ConvNets can potentially infer from image ANYTHING for which information is in the image (3D, movement, planning, …)