General rules of thumbs – Input Layer

- The size of the input layer should be divisible by 2 (hopefully a power of 2)
  - E.g., 256, 512

General rule of thumbs:
The conv layer

- Small filters with stride 1
- Usually zero-padding applied to keep the input size unchanged
- In general, for a certain F, if you choose

$$P = (F - 1)/2,$$

the input size is preserved (for S=1):

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

General rules of thumbs – Pooling Layer

- Max pooling with F=3, S=2 or F=2 and S=2 are common (pooling with bigger receptive field F can be destructive)

Taking care of downsampling

- At some point(s) in the network, we need to reduce the size
- If conv layers do not downsize, then only pooling layers take care of downsampling
- If conv layers also downsize, you need to be careful about strides etc. so that
  - (i) the dimension requirements of all layers are satisfied and
  - (ii) all layers tile up properly.
- S=1 seems to work well in practice
- However, for bigger input volumes, you may try bigger strides

How to initialize the weights?

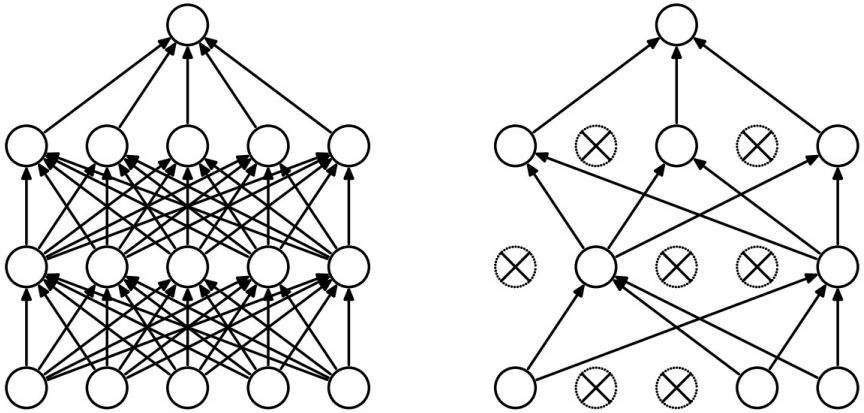
- Option 1: randomly
  - This has been shown to work nicely in the literature

- Option 2:
  - Train/obtain the "filters" elsewhere and use them as the weights
  - Unsupervised pre-training using image patches (windows)
  - Avoids full feedforward and backward pass, allows the search to start from a better position
  - You may even skip training the convolutional layers

Suggested Reading

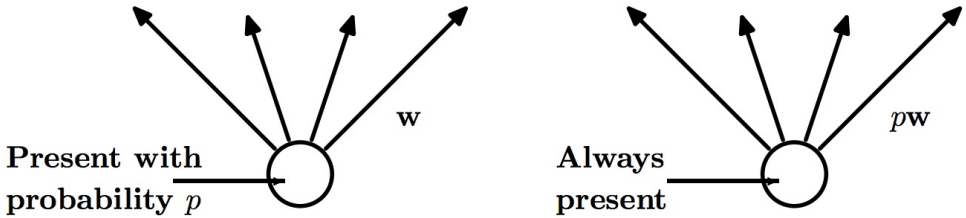
Dropout

A simple way to Prevent Neural Networks from Overfitting



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

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.



(a) At training time (b) At test time

Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Original Paper: http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Batch-Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Normalise the input and activation functions outputs to accelerate the training of deep networks. Higher learning rates can be used because batch normalization makes sure that there's no activation that's gone really high or really low.

It reduces overfitting because it has a slight regularization effects. Similar to dropout, it adds some noise to each hidden layer's activations. Therefore, if we use batch normalization, we will use less dropout, which is a good thing because we are not going to lose a lot of information. However, we should not depend only on batch normalization for regularization; we should better use it together with dropout.

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$
Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$

1: $N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma^2_{\mathcal{B}}]$$

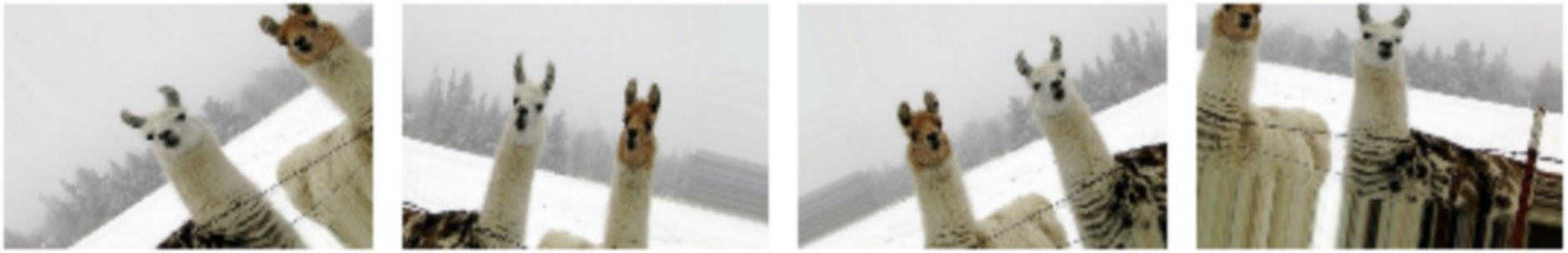
11: In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$
12: **end for**

Algorithm 2: Training a Batch-Normalized Network

Original Paper: http://proceedings.mlr.press/v37/ioffe15.pdf

Data Augmentation

- It is regularization scheme



- For a given dataset, you can generate an augmented dataset considering affine transformations to manipulate the training data.
- For each input image, you can can create a “duplicate” image that is shifted, zoomed in/out, rotated, flipped, distorted, or shaded with a hue. Both image and duplicate are fed into the neural net with the same label.
- Other augmentation schemes have been proposed in literature:
  - http://cs231n.stanford.edu/reports/2017/pdfs/300.pdf
  - https://arxiv.org/abs/1708.06020

Transfer Learning

- **The training of a deep learning model from scratch (with random initialization) is hard because it is relatively rare to have a dataset of sufficient size.** Usually, a network is **pre-trained on a large dataset** (e.g. ImageNet, EPIC-Kitchens), and **then used as an initialization or a fixed feature extractor for the task of interest.**

- Main Strategies:
  - **Fixed feature extractor**: take a model pretrained for a task (e.g., object detection), remove the last fully-connected layer and use model as a fixed feature extractor for the new dataset. Once you extract the features on the new dataset, train a classifier (e.g., Softmax classifier, or others) for the new dataset. Original network parameters used as feature extractor are fixed during the training of the classifier.

  - **Fine-tuning:** replace and retrain the classifier on top of the deep learning model on the new dataset, but to also fine-tune the weights of the pretrained network by continuing the backpropagation. It is possible to fine-tune all the layers of the deep model, or it's possible to 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 deep model contain more generic features (e.g. edge detectors or color blob detectors) that should be useful to many tasks, but later layers of the deep model becomes progressively more specific to the details of the classes contained in the original dataset.

  - **Pretrained models**: it is common to see researchers release their final deep model checkpoints for the benefit of others who can use the networks for fine-tuning. For example, Pytorch library has a models subpackage contains definitions for different model architectures https://pytorch.org/vision/stable/models.html

Finetuning

1. If the new dataset is small and similar to the original dataset used to train the CNN:
   - Finetuning the whole network may lead to overfitting
   - Just train the newly added layer
2. If the new dataset is big and similar to the original dataset:
   - The more, the merrier: go ahead and train the whole network
3. If the new dataset is small and different from the original dataset:
   - Not a good idea to train the whole network
   - However, add your new layer not to the top of the network, since those parts are very dataset (problem) specific
   - Add your layer to earlier parts of the network
4. If the new dataset is big and different from the original dataset:
   - We can "finetune" the whole network
   - This amounts to a new training problem by initializing the weights with those of another network

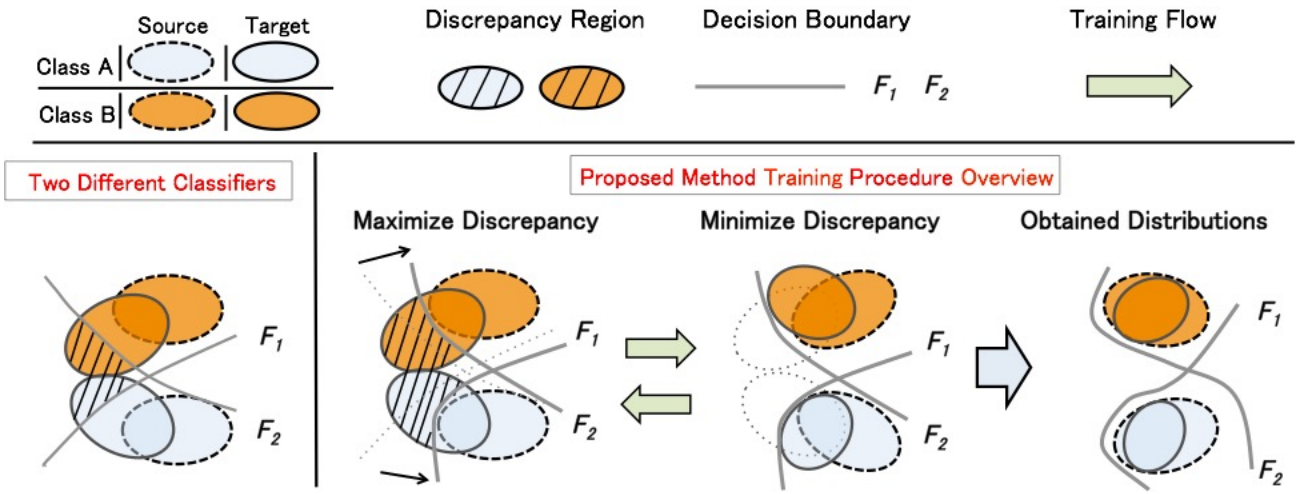
Domain Adaptation

Are a class of machine learning algorithms learning from a source data distribution a well performing model on a different (but related) target data distribution.

See a survey here:

https://arxiv.org/pdf/1702.05374.pdf

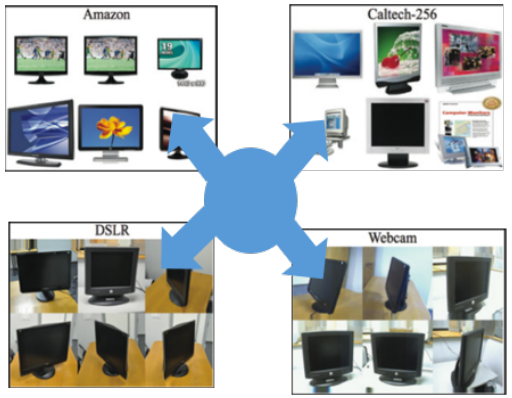
https://arxiv.org/pdf/1802.03601.pdf



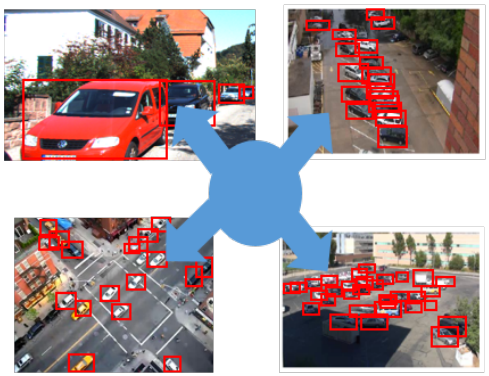
https://mil-tokyo.github.io/MCD_DA/

Domain Adaptation

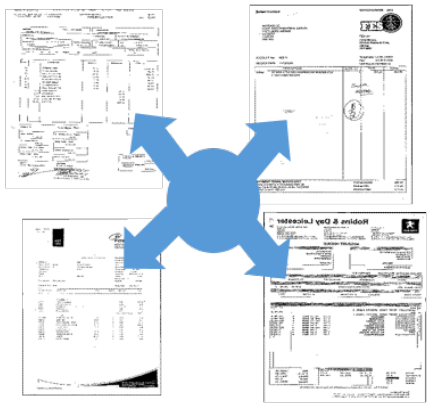
Object recognition



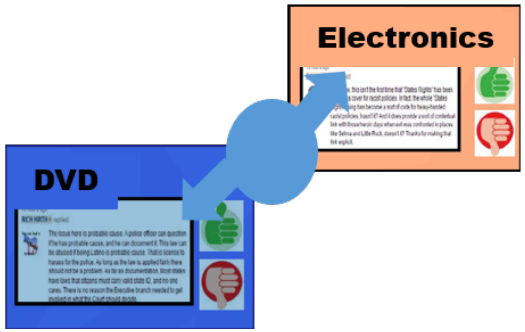
Object detection



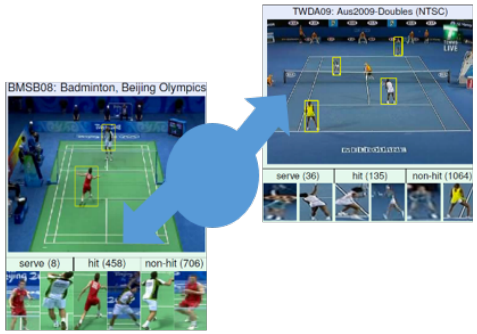
Document image categorization



Sentiment analyses



Action recognition



Speach recognition

ADDA Approach

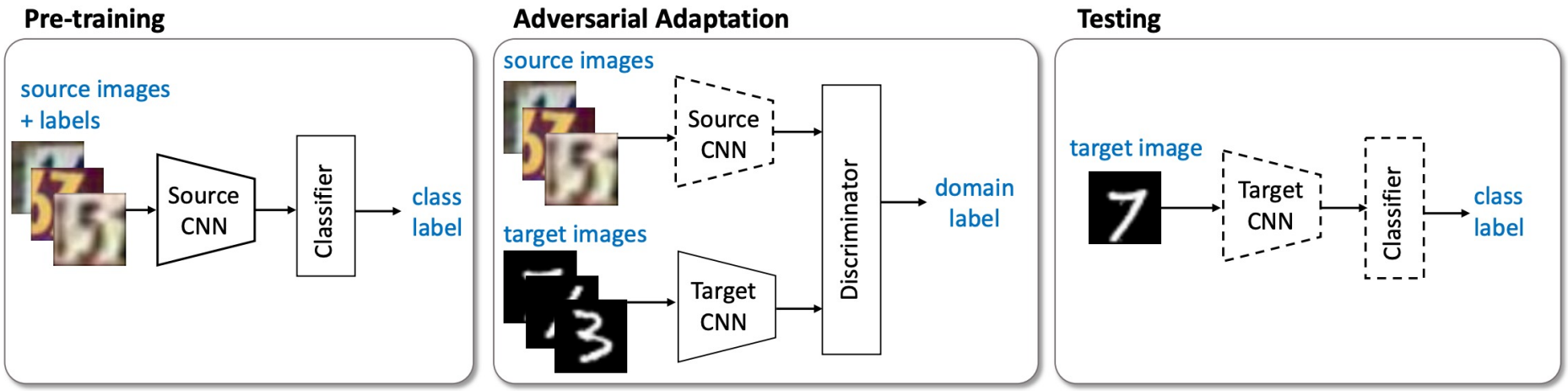


Figure 3: An overview of our proposed Adversarial Discriminative Domain Adaptation (ADDA) approach. We first pre-train a source encoder CNN using labeled source image examples. Next, we perform adversarial adaptation by learning a target encoder CNN such that a discriminator that sees encoded source and target examples cannot reliably predict their domain label. During testing, target images are mapped with the target encoder to the shared feature space and classified by the source classifier. Dashed lines indicate fixed network parameters.

https://arxiv.org/pdf/1702.05464.pdf

Image-to-Image Transformation

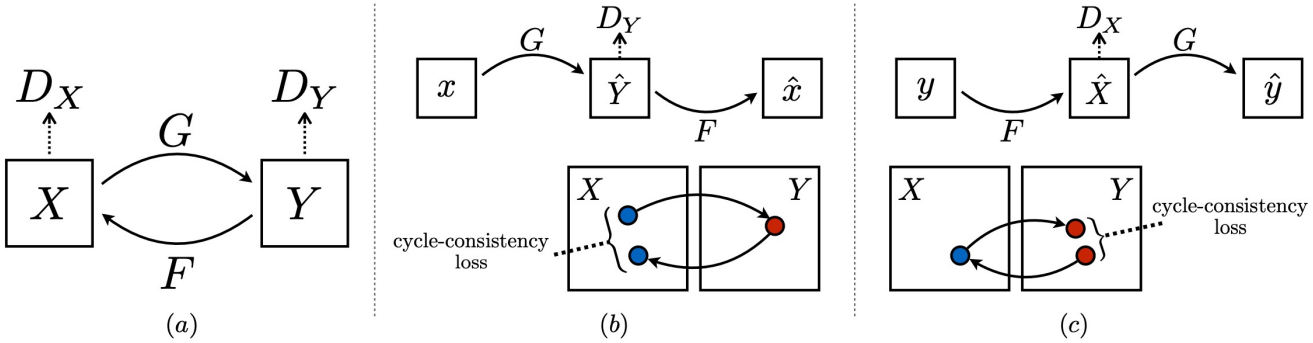


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

[CycleGan Page]

Image-to-Image Transformation

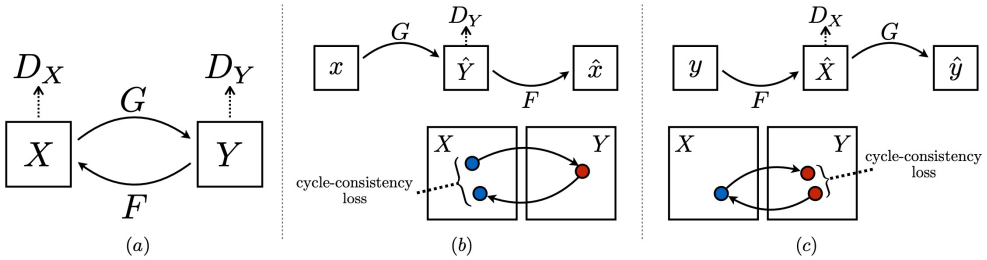


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F),$$

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))]$$

$$\min_G \max_{D_Y} \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y)$$

Ci sono 2 generatori (G e F) e 2 discriminatori (X e Y) che vengono addestrati qui.

- Il generatore `G` impara a trasformare l'immagine `X` nell'immagine `Y` ($G : X -> Y$)
- Il generatore `F` impara a trasformare l'immagine `Y` nell'immagine `X` ($F : Y -> X$)
- Il discriminatore `D_X` impara a distinguere tra l'immagine `X` e l'immagine generata `X` (`F(Y)`).
- Il discriminatore `D_Y` impara a distinguere tra l'immagine `Y` e l'immagine generata `Y` (`G(X)`).

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1]$$

- L'immagine X viene passata tramite il generatore G che restituisce l'immagine generata $hatY$.
- L'immagine generata $hatY$ viene passata tramite il generatore F che restituisce l'immagine ciclata $hatX$.
- L'errore assoluto medio viene calcolato tra X e $hatX$.

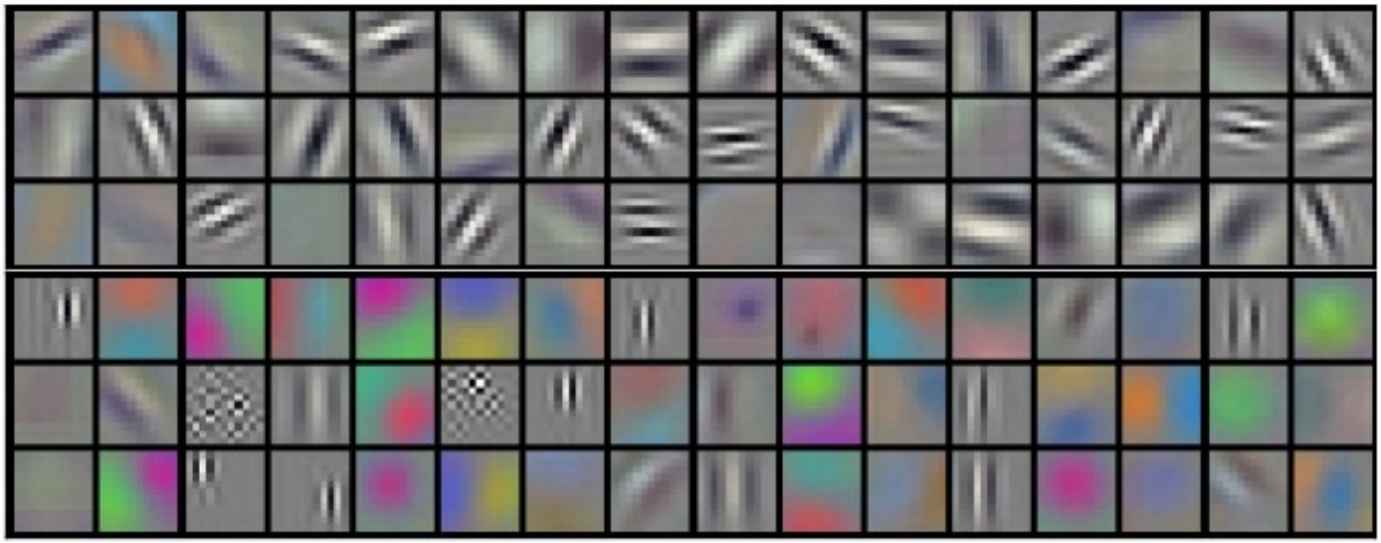
$$forward\ cycle\ consistency\ loss : X -> G(X) -> F(G(X)) \sim \hat{X}$$

$$backward\ cycle\ consistency\ loss : Y -> F(Y) -> G(F(Y)) \sim \hat{Y}$$

[CycleGan Paper]

What Does a CNN learn? First layer

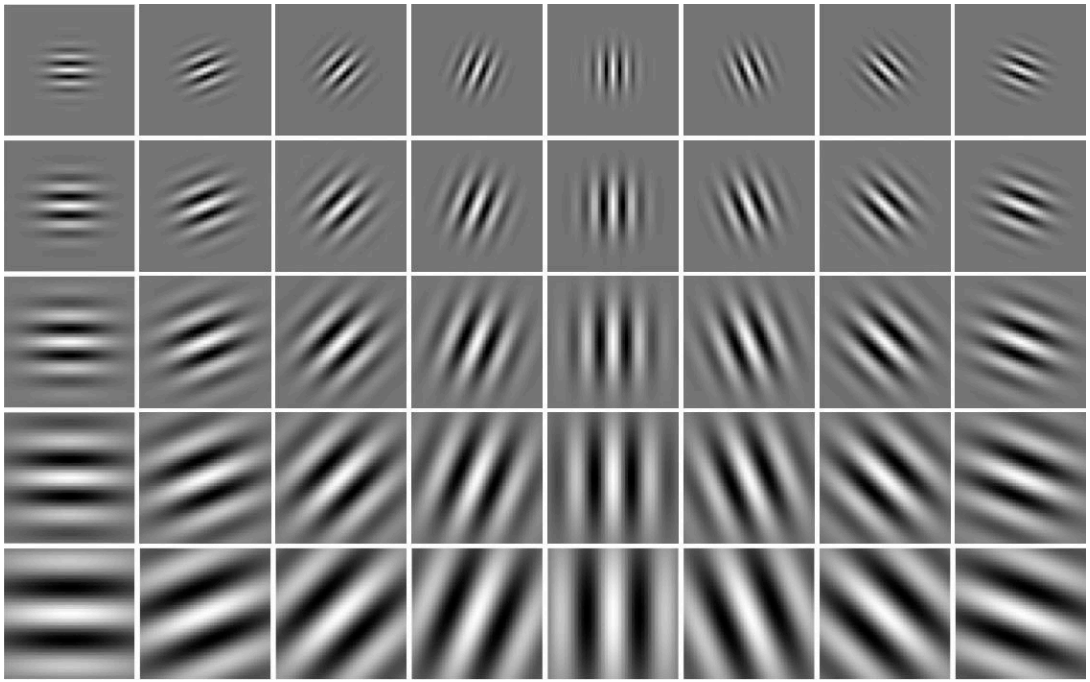
Krizhevsky et al. visualized the [11 x 11] kernels learned in the first layer:



The filters extract very low visual features like edges and color gradients.

They totally look like Gabor filters! ….but they have been learned automatically by the machine!

Drigression: Gabor Filters



Thus, image analysis using Gabor filters is thought to be similar to perception in human visual system.

Il filtro di Gabor è descritto dall'equazione

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda} + \psi\right)$$

dove:

$$x' = x\cos\theta + y\sin\theta$$

e

$$y' = -x\sin\theta + y\cos\theta$$

In questa equazione:

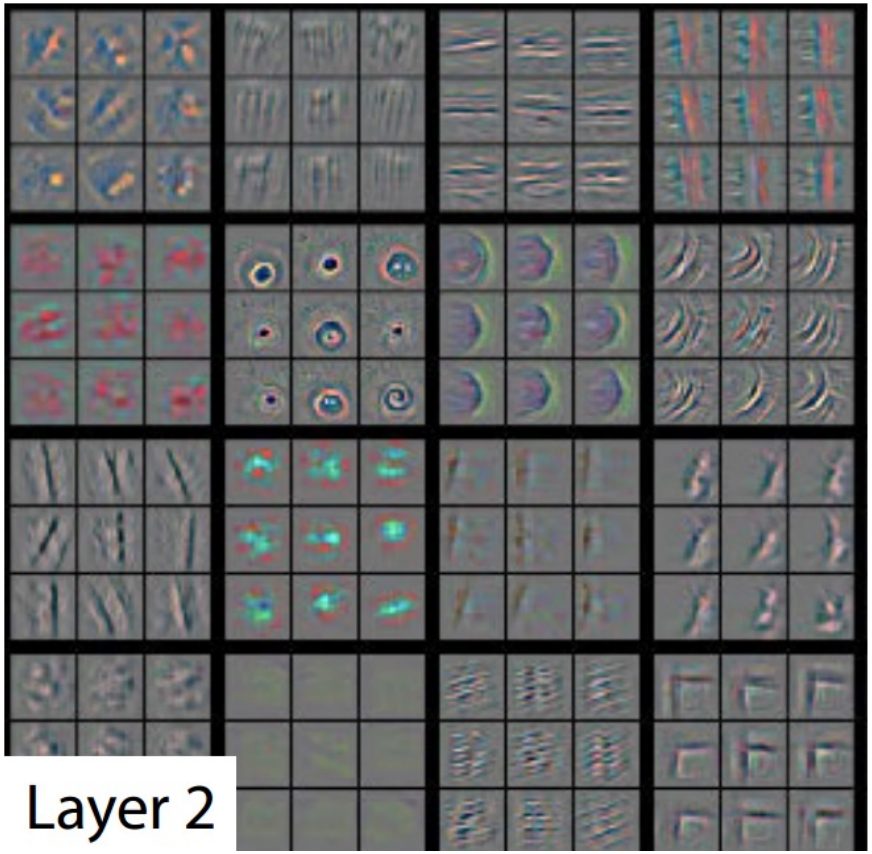
- λ rappresenta la lunghezza d'onda del fattore coseno
- θ rappresenta l'orientazione del filtro
- ψ è la fase del fattore coseno
- σ è il parametro che regola l'inviluppo
- γ specifica l'ellitticità del supporto della funzione di Gabor.

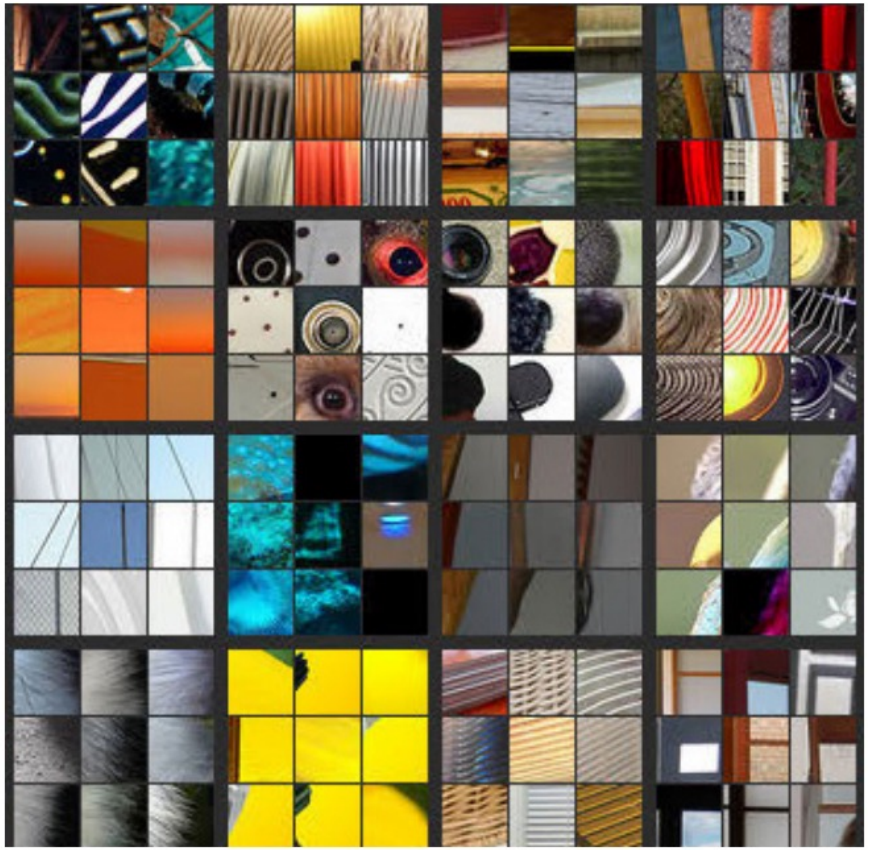
Gabor Filters are filters used for texture analysis. They are used to detect whether there are any specific frequency content in specific directions in localized regions of the image.

Simple cells in the visual cortex of mammalian brains can be modeled by Gabor functions.

https://en.wikipedia.org/wiki/Gabor_filter

What Does a CNN learn? Layer 2



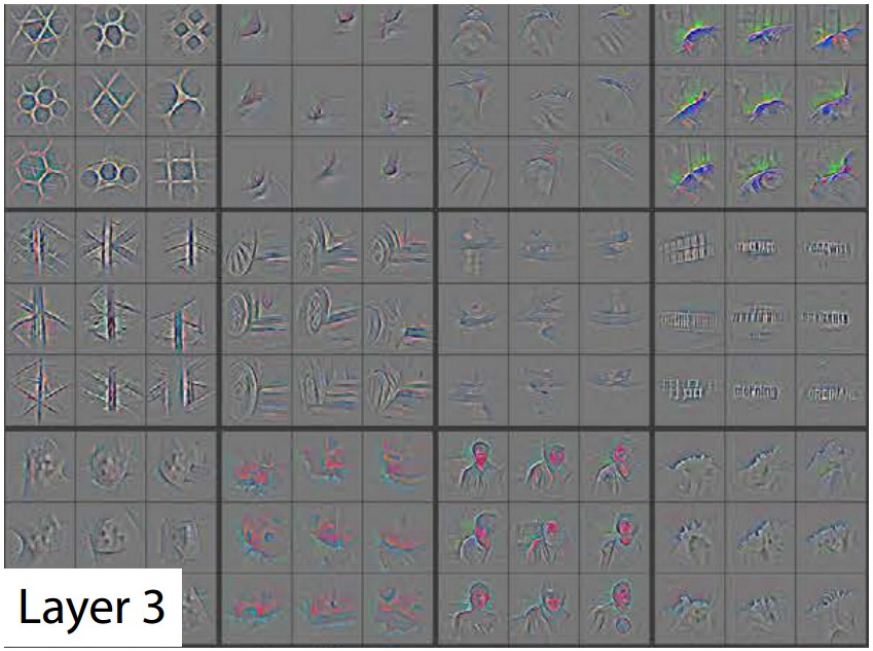


Patterns Causing High activations

Images Corresponding to Pattern

Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." *ECCV 2014*

What Does a CNN learn? Layer 3

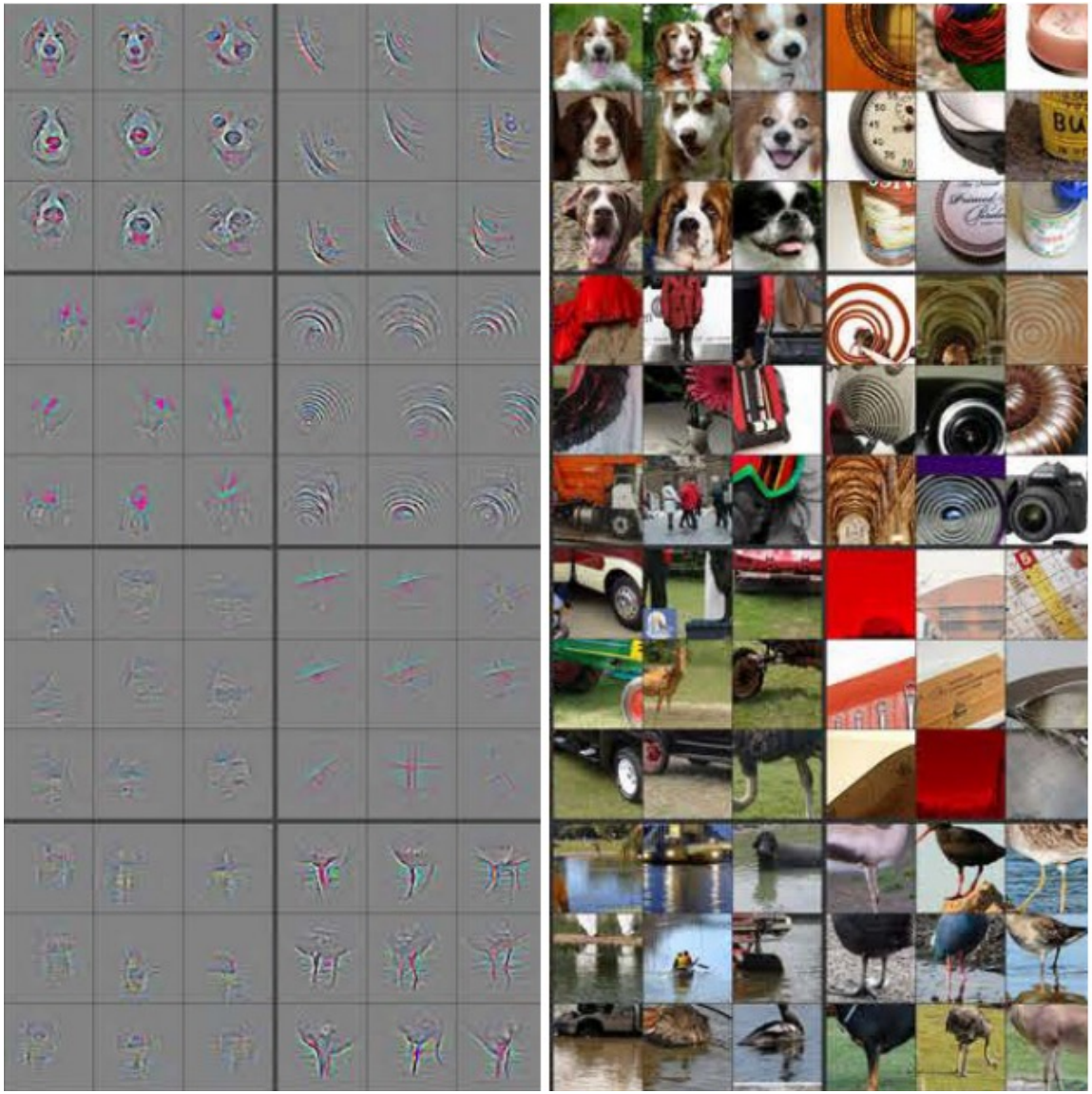


Patterns Causing High activations

Images Corresponding to Pattern

Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." *ECCV 2014*

What Does a CNN learn? Layer 4

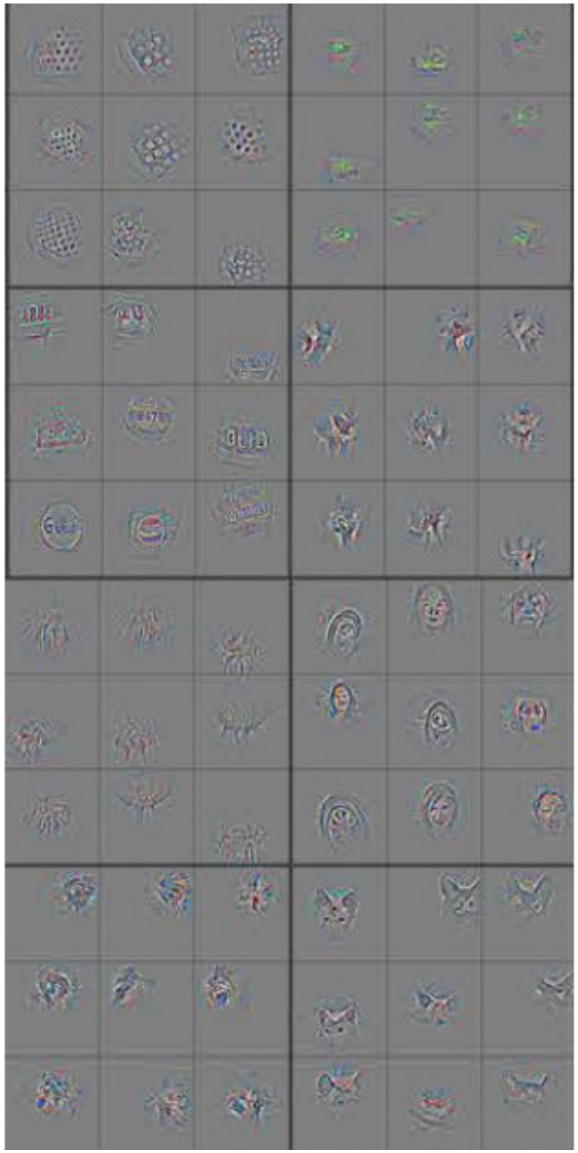


Patterns Causing High activations

Images Corresponding to Pattern

Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." *ECCV 2014*

What Does a CNN learn? Layer 5



Patterns Causing High activations

Images Corresponding to Pattern

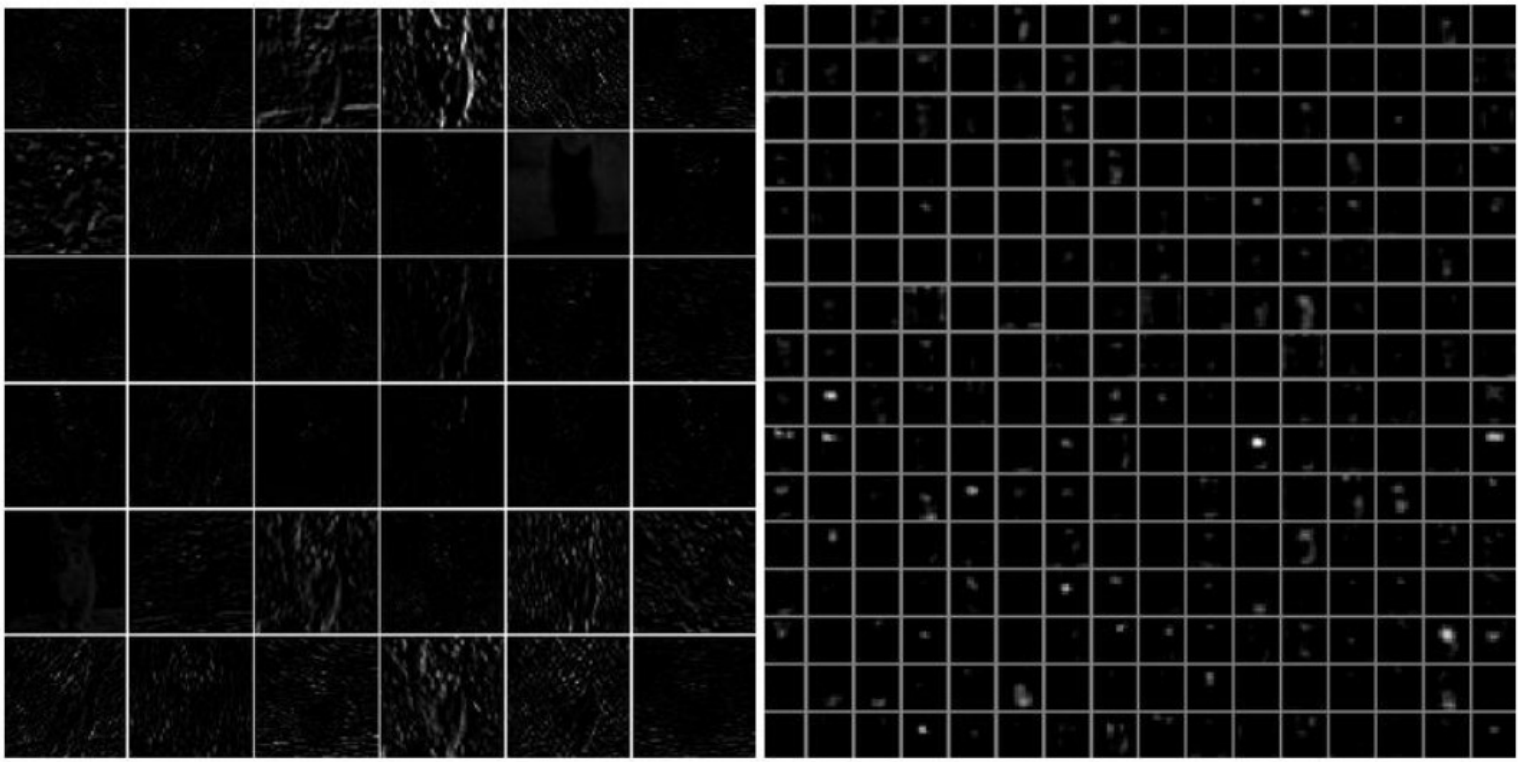
Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." *ECCV 2014*

Visualizing and Understanding CNNs

- Many different mechanisms
  - Visualize layer activations
  - Visualize the weights (i.e., filters)
  - Visualize examples that maximally activate a neuron
  - Visualize a 2D embedding of the inputs based on their CNN codes
  - Occlude parts of the image and see how the prediction is affected
  - Data gradients

Visualize activations during training

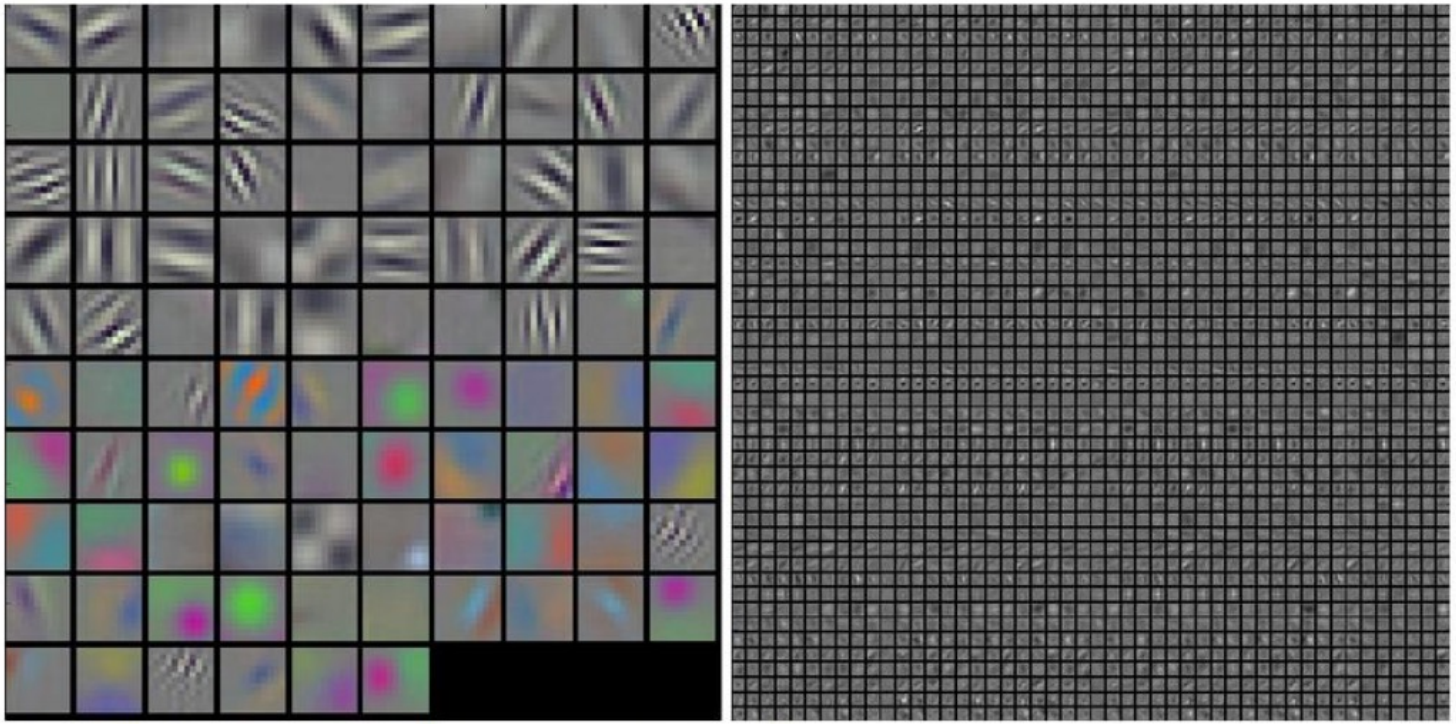
- Activations are dense at the beginning.
  - They should get sparser during training.
- If some activation maps are all zero for many inputs, dying neuron problem => high learning rate in the case of ReLUs.



Typical-looking activations on the first CONV layer (left), and the 5th CONV layer (right) of a trained AlexNet looking at a picture of a cat. Every box shows an activation map corresponding to some filter. Notice that the activations are sparse (most values are zero, in this visualization shown in black) and mostly local.

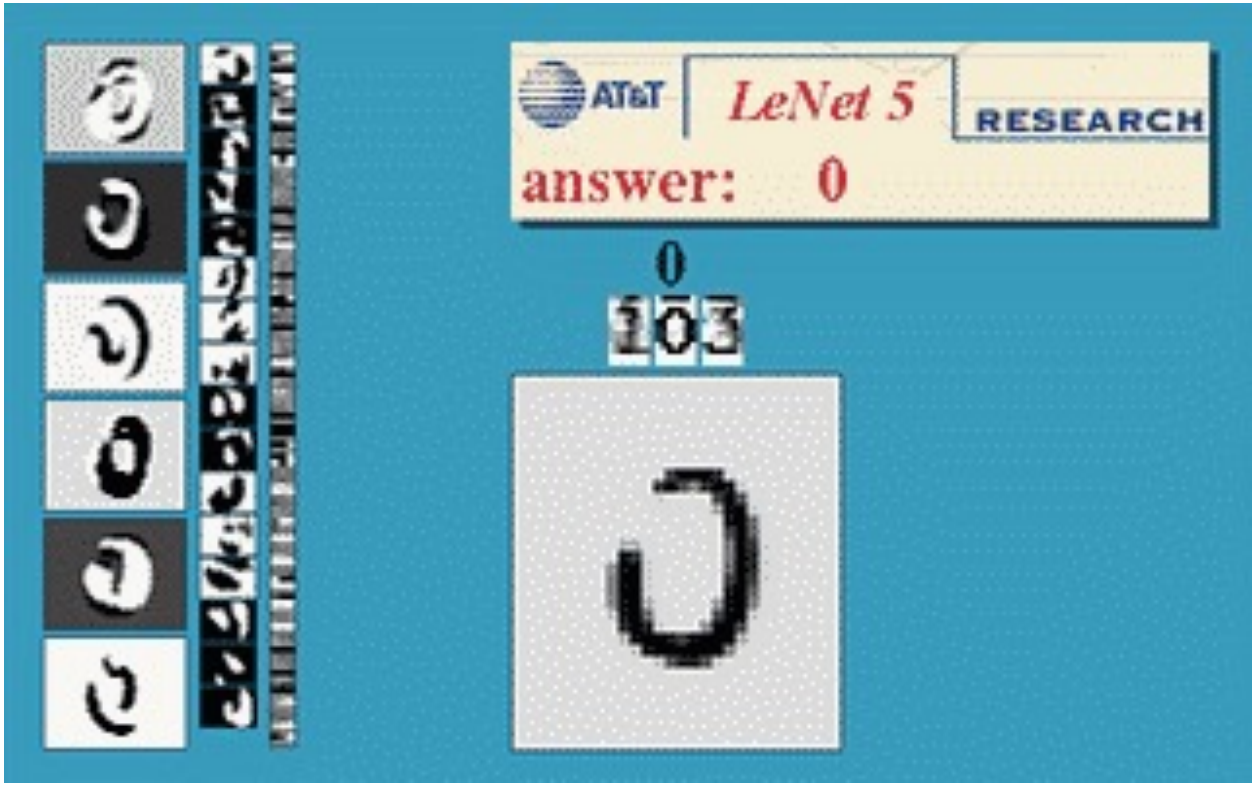
Visualize the weights

- We can directly look at the filters of all layers
- First layer is easier to interpret
- Filters shouldn't look noisy



Typical-looking filters on the first CONV layer (left), and the 2nd CONV layer (right) of a trained AlexNet. Notice that the first-layer weights are very nice and smooth, indicating nicely converged network. The color/grayscale features are clustered because the AlexNet contains two separate streams of processing, and an apparent consequence of this architecture is that one stream develops high-frequency grayscale features and the other low-frequency color features. The 2nd CONV layer weights are not as interpretable, but it is apparent that they are still smooth, well-formed, and absent of noisy patterns.

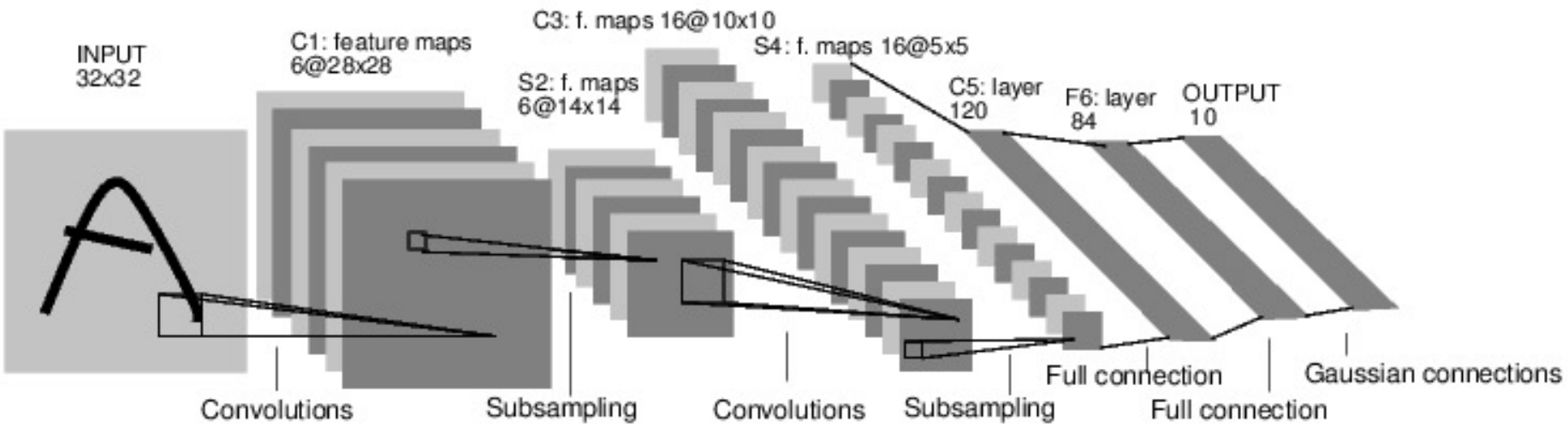
LeNet 5 Convolutional Neural Network Approach



http://yann.lecun.com/exdb/lenet/

LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

LeNet 5 Architecture



7 layers (5 excluding subsampling):

- C1, C3, C5 perform convolutions;
- S2, S4 perform subsampling (max pooling);
- F6 is a fully connected layer;
- Output layer gives a probability distribution over target classes.

How many parameters?

~60 000

vs

397 510

(MLP)

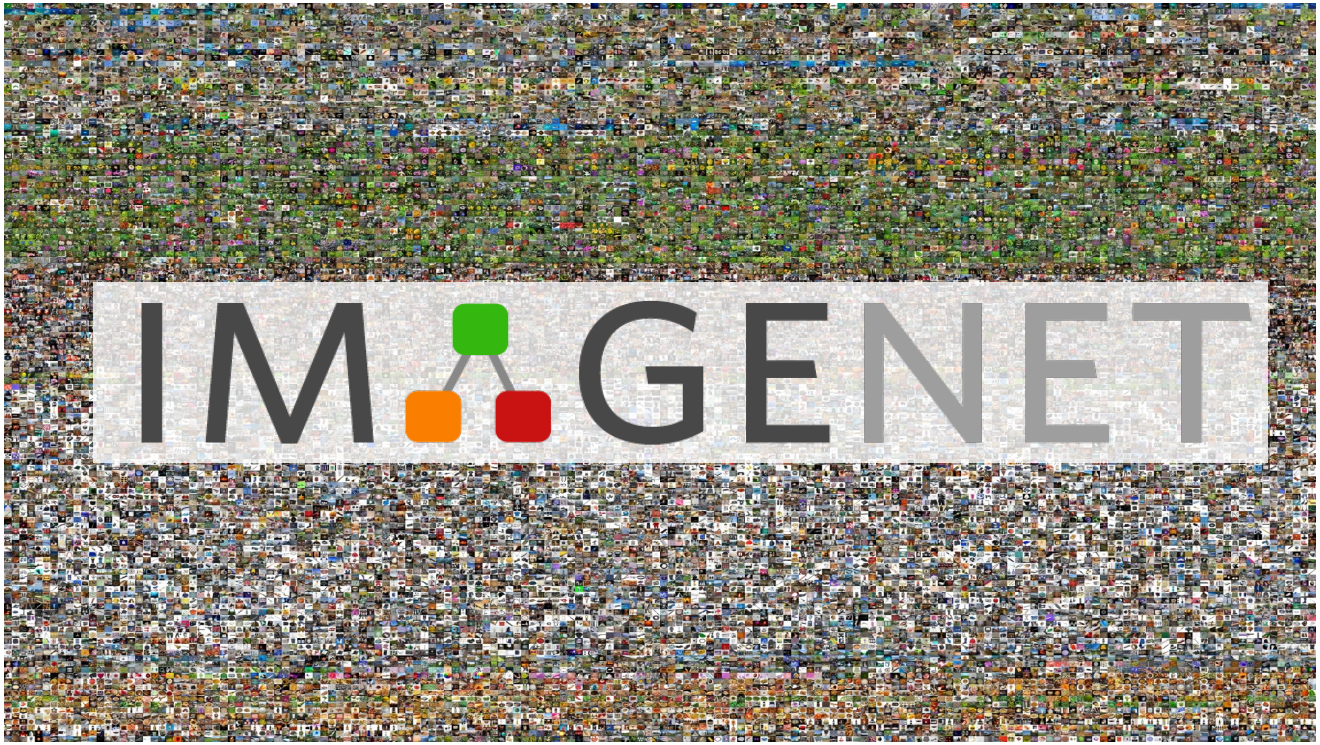
LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

ILSVRC Classification Challenge

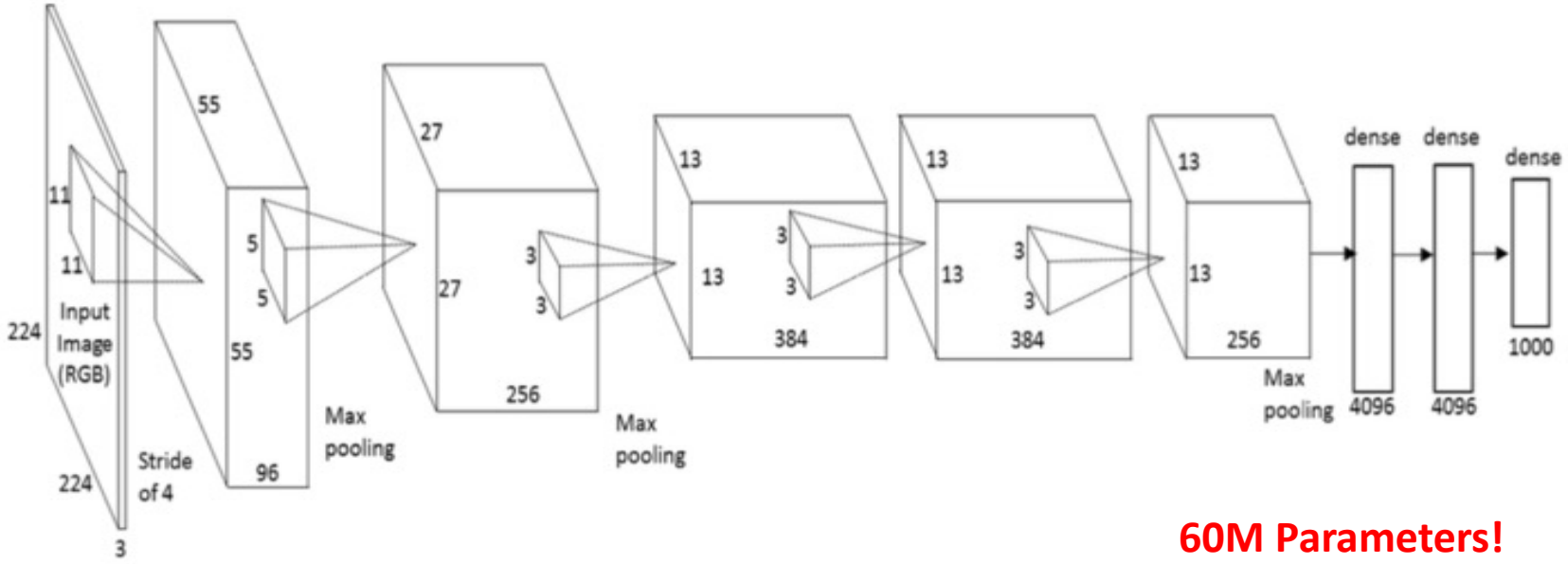
held every year since 2010

- 1000 object classes;
- 1,5M images.

Much more complicated than MNIST!



AlexNet Neural Network



60M Parameters!
(vs 60K)

Deeper than LeNet:

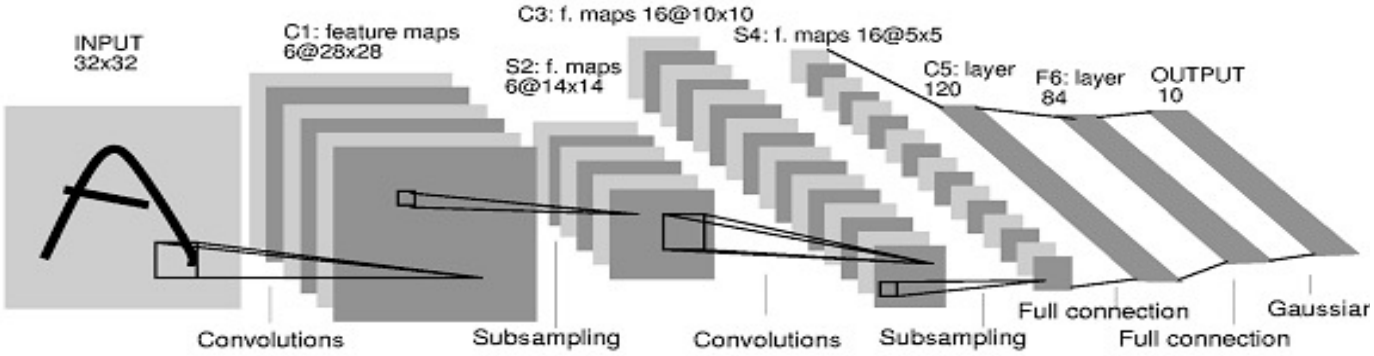
- 5 Conv layers (vs 3);
- Max pooling performed 3 times (vs 2);
- 2 fully connected layers (vs 1).

One Week Training
(on two GPUs)

Input size: 224 x 224 (vs 28 x 28). Output size: 1000 classes (vs 10).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

1998
LeCun et al.



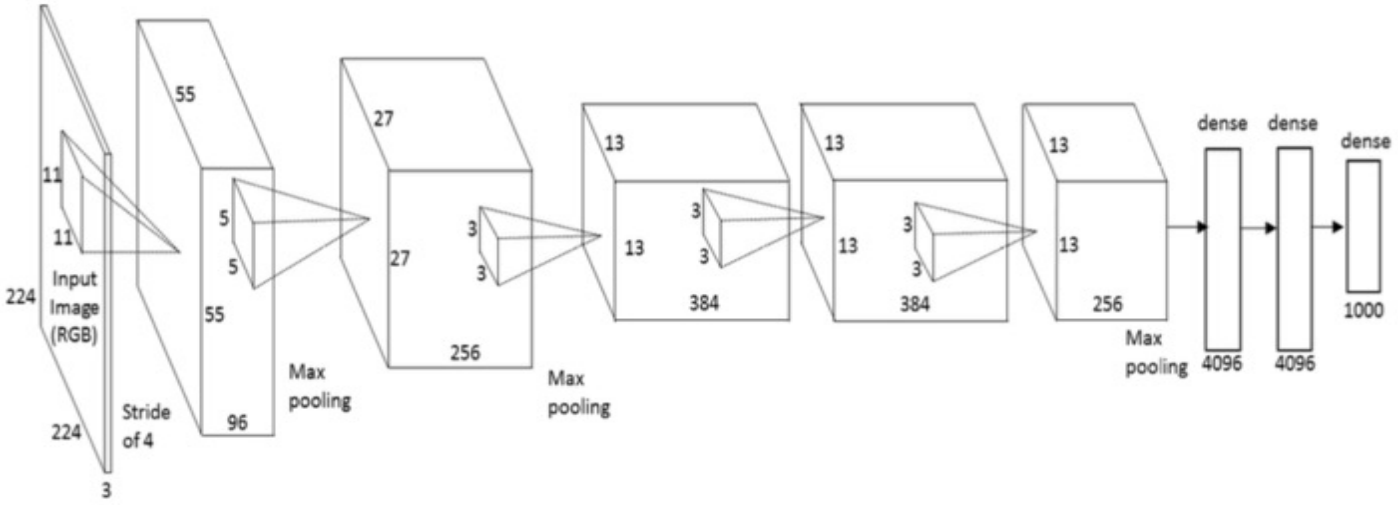
# of transistors

10^6

# of pixels used in training

10^7 NIST

2012
Krizhevsky et al.



# of transistors

10^9

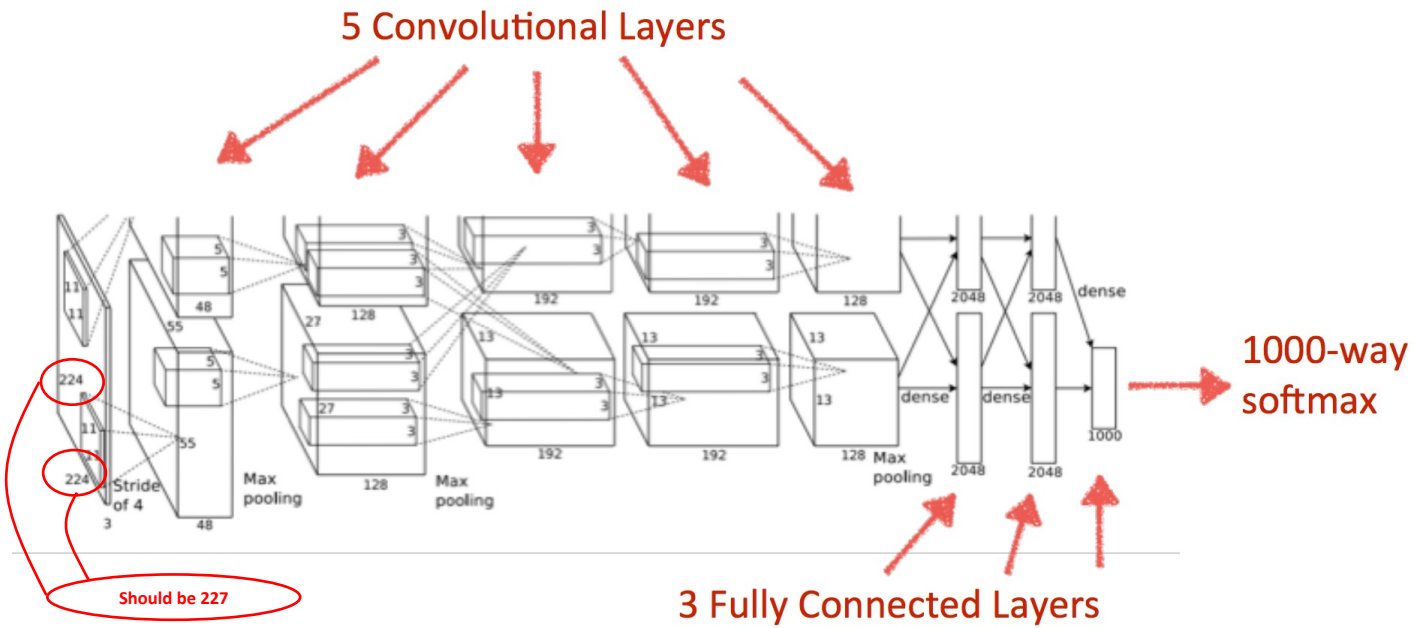
GPUs

# of pixels used in training

10^{14} IMAGENET

Slide credit: Fei Fei Li

AlexNet



Input: 227x227x3 images
First layer: 96 11x11 filters applied at stride 4, with padding 0 and no dilatation

Output volume size?
(N+2P-F)/S+1 = (227+0-11)/4+1 = 55 --> [55x55x96]

Number of parameters in this layer?
(11*11*3)*96 = 35K

Reminder:
Output size = (N+2P-F)/S + 1

Alexnet has 60 Million parameter compared to 60k parameter of LeNet-5
Training facilitated by GPUs, highly optimized convolution implementation and large datasets (ImageNet)

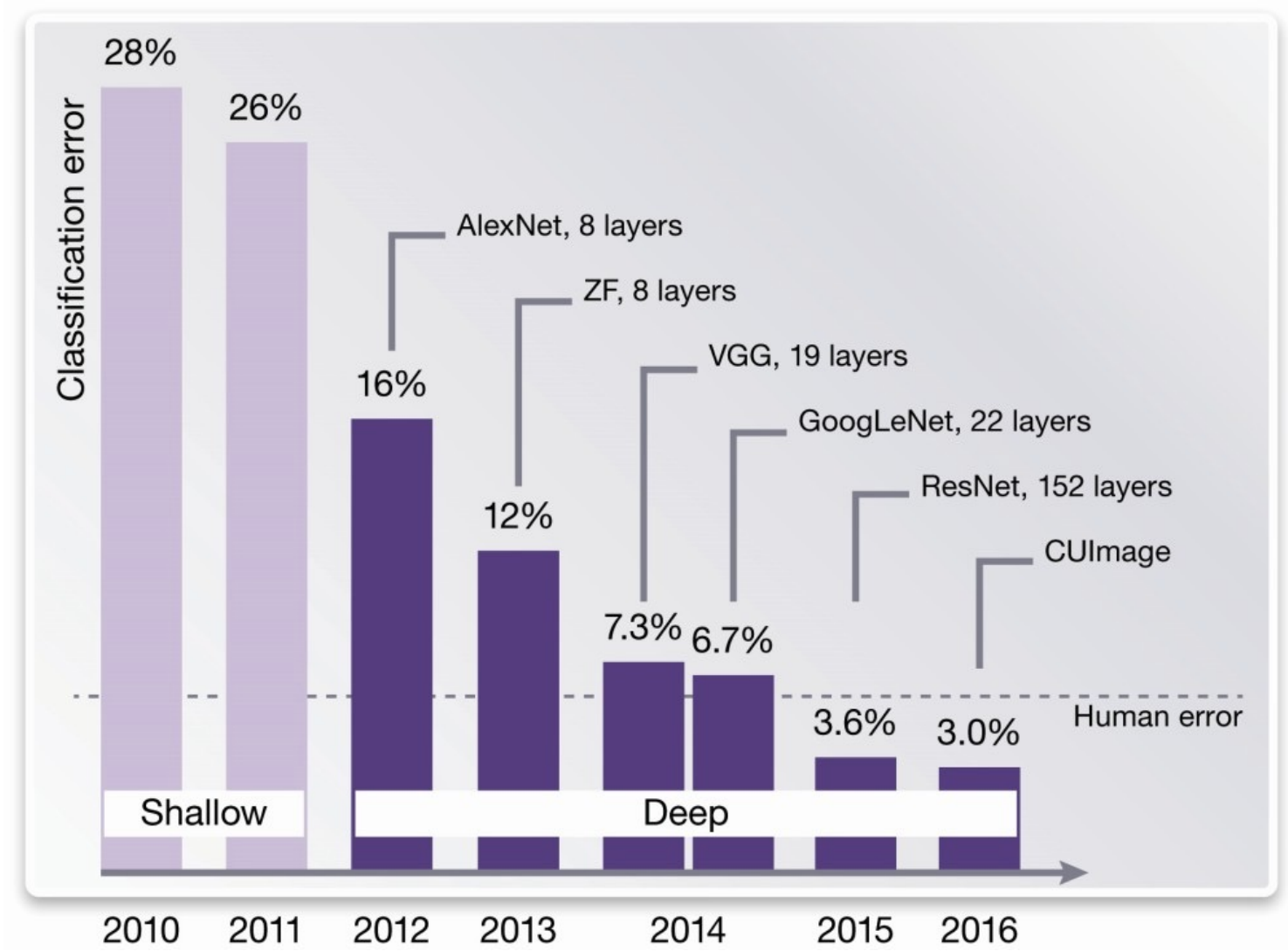
Architecture
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

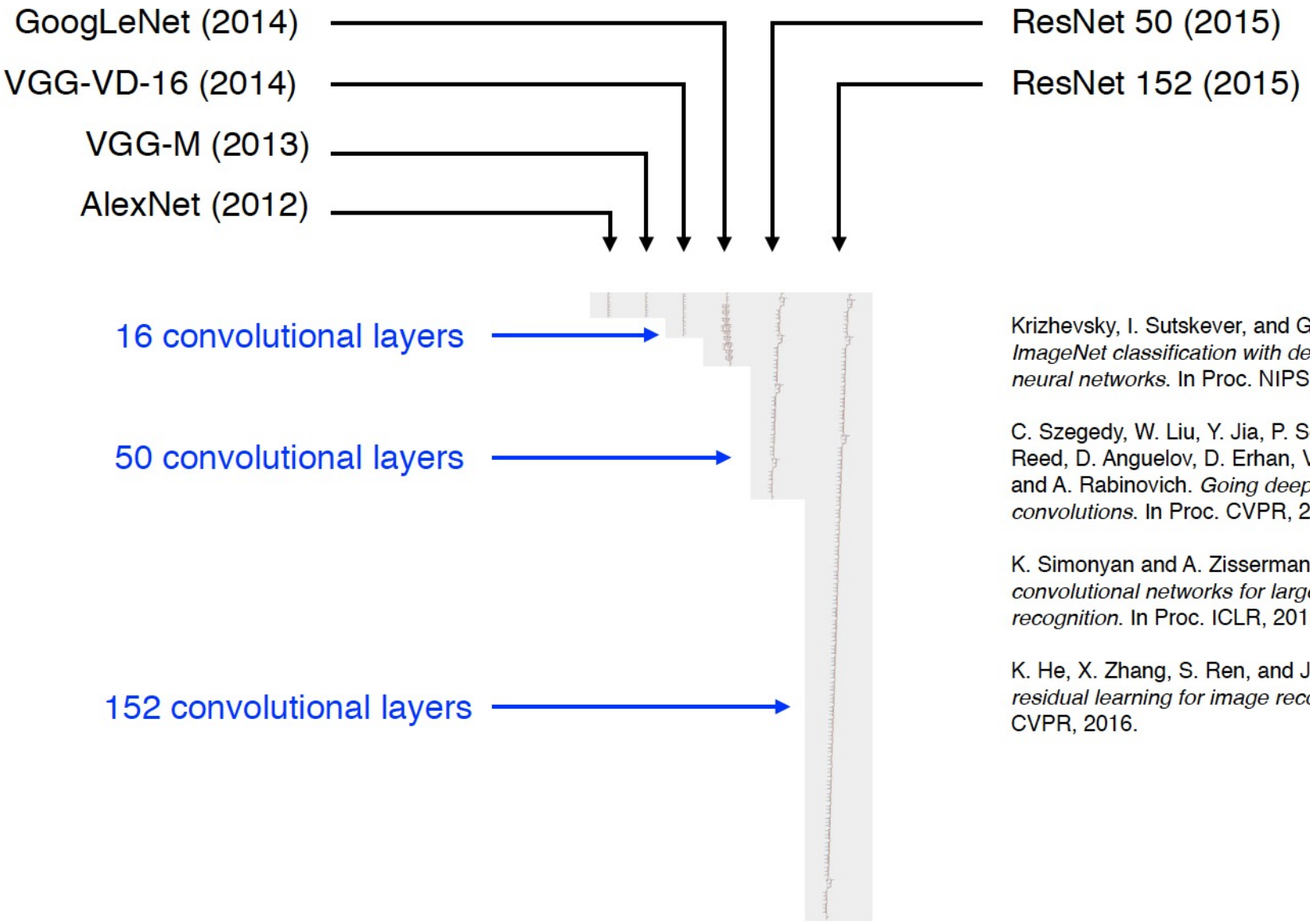
A. Krizhevsky, I. Sutskever, and G. E. Hinton. *Imagenet classification with deep convolutional neural networks*. In Proc. NIPS, 2012.

INNOVATIONS

- RELU : USE RELU INSTEAD OF TANH (ACCELERATE 6 TIMES)
- USE DROPOUT INSTEAD OF REGULARIZATION TO DEAL OVERFITTING
- OVERLAP POOLING TO REDUCE SIZE OF NETWORK
- 1.2M IMAGES 1000 CATEGORIES
- USED 2 GPUS
- LOCAL RESPONSE NORMALIZATION
- DATA AUGMENTATION
- SGD
- 8% reduced error top 1, 12% reduced error top 5

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners





Krizhevsky, I. Sutskever, and G. E. Hinton. *ImageNet classification with deep convolutional neural networks*. In Proc. NIPS, 2012.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. *Going deeper with convolutions*. In Proc. CVPR, 2015.

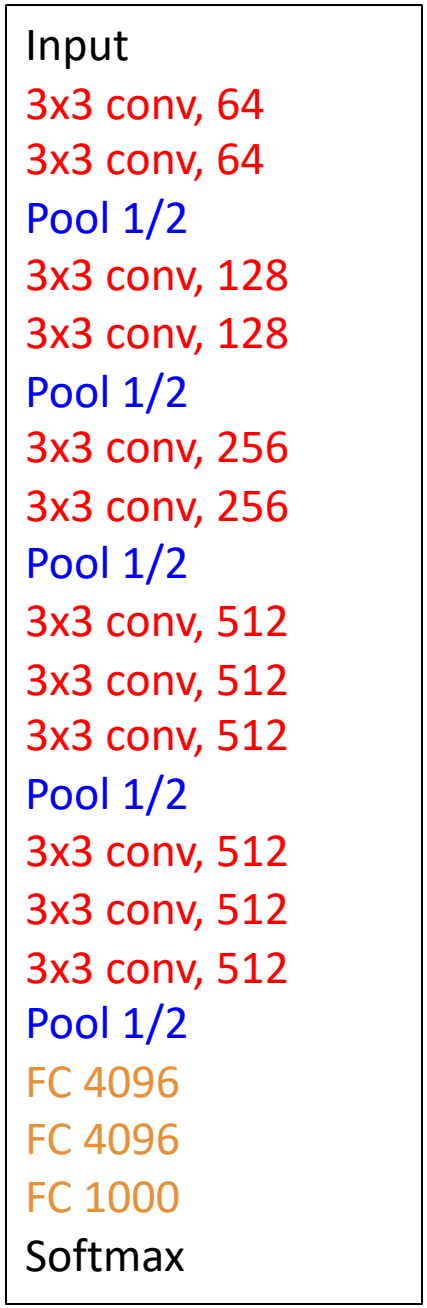
K. Simonyan and A. Zisserman. *Very deep convolutional networks for large-scale image recognition*. In Proc. ICLR, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. *Deep residual learning for image recognition*. In Proc. CVPR, 2016.

VGGNet

- *Very Deep Convolutional Networks For Large Scale Image Recognition - Karen Simonyan and Andrew Zisserman; 2015*
- The runner-up at the ILSVRC 2014 competition
- Significantly deeper than AlexNet
- 140 million parameters

[Simonyan and Zisserman, 2014]



VGGNet

- **Smaller filters**
  Only 3x3 CONV filters, stride 1, pad 1
  and 2x2 MAX POOL , stride 2

- **Deeper network**
  AlexNet: 8 layers
  VGGNet: 16 - 19 layers

- **VGG16:**

TOTAL memory: 24M * 4 bytes ~= 96MB

TOTAL params: 138M parameters

[Simonyan and Zisserman, 2014]

VGGNet

- **Why use smaller filters? (3x3 conv)**

Stack of three 3x3 conv (stride 1) layers has the same effective receptive field as one 7x7 conv layer.

- **What is the effective receptive field of three 3x3 conv (stride 1) layers?**

7x7

But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer

[Simonyan and Zisserman, 2014]

VGGNet

Details/Retrospectives :

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as AlexNet
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks
- Trained on 4 Nvidia Titan Black GPUs for **two to three weeks**.

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[Simonyan and Zisserman, 2014]

VGGNet

VGG Net reinforced the notion that **convolutional neural networks have to have a deep network of layers in order for this hierarchical representation of visual data to work**.

Keep it deep.

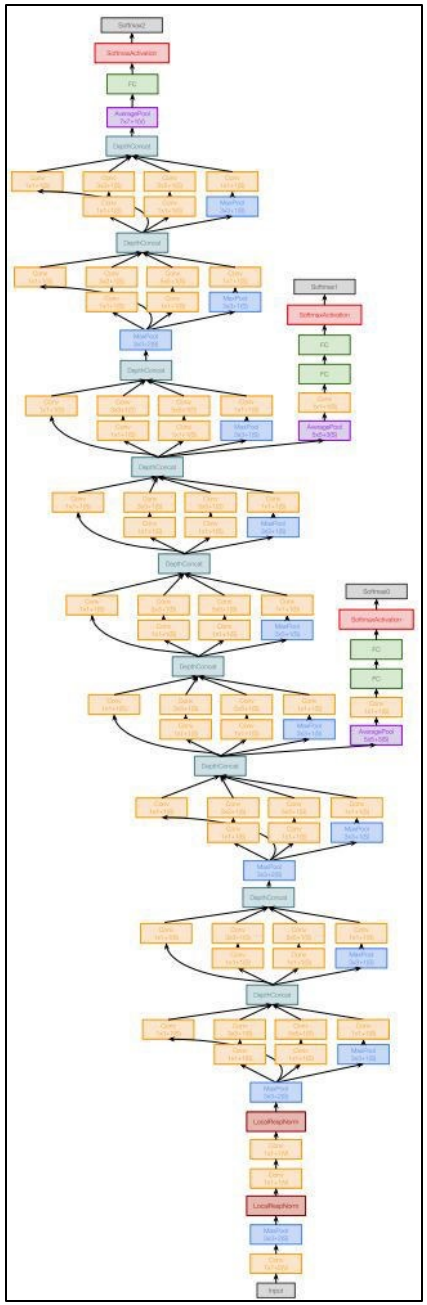
Keep it simple.

[Simonyan and Zisserman, 2014]

GoogleNet

- *Going Deeper with Convolutions - Christian Szegedy et al.; 2015*
- ILSVRC 2014 competition winner
- Also significantly deeper than AlexNet
- x12 less parameters than AlexNet
- Focused on computational efficiency

[Szegedy et al., 2014]



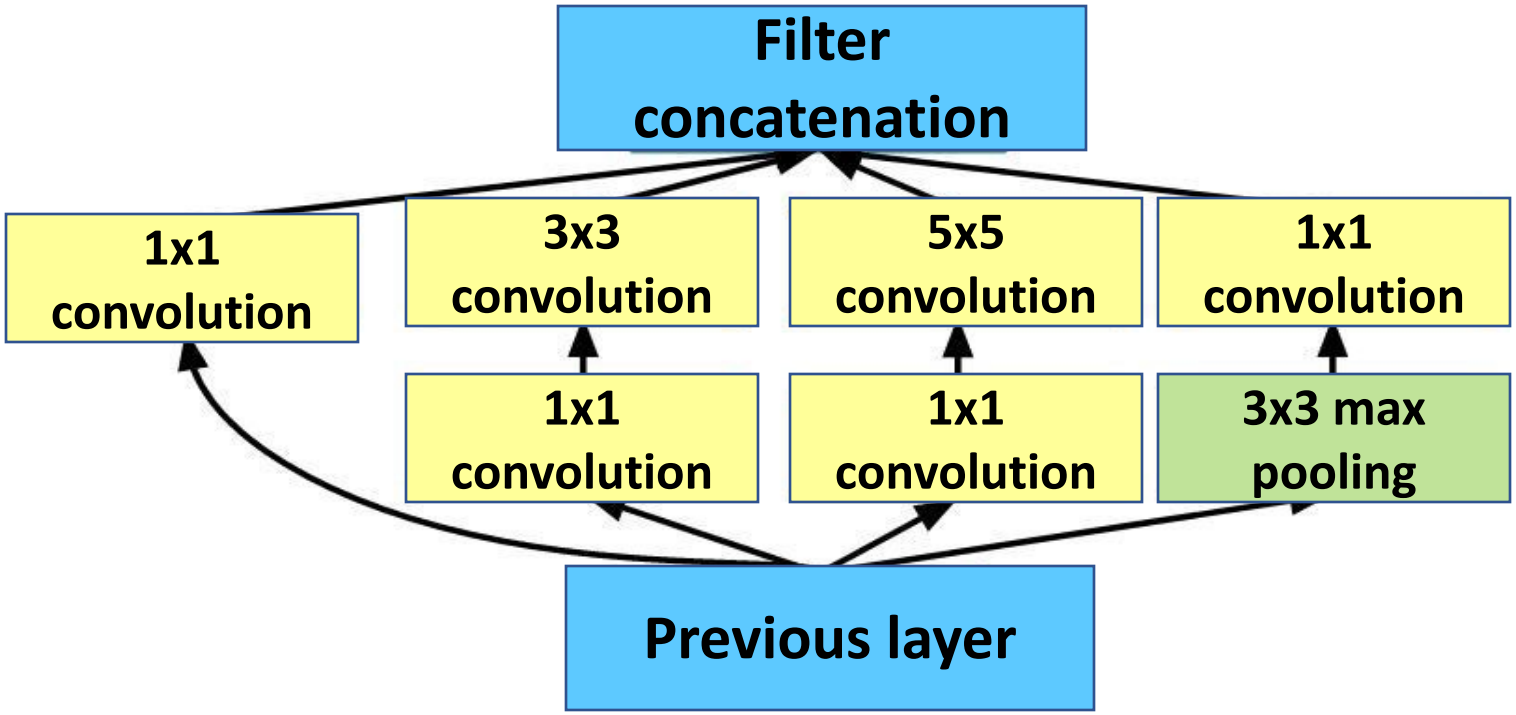
GoogleNet

- 22 layers
- Efficient **“Inception” module** - strayed from the general approach of simply stacking conv and pooling layers on top of each other in a sequential structure
- No FC layers
- Only 5 million parameters!
- ILSVRC’14 classification winner (6.7% top 5 error)

[Szegedy et al., 2014]

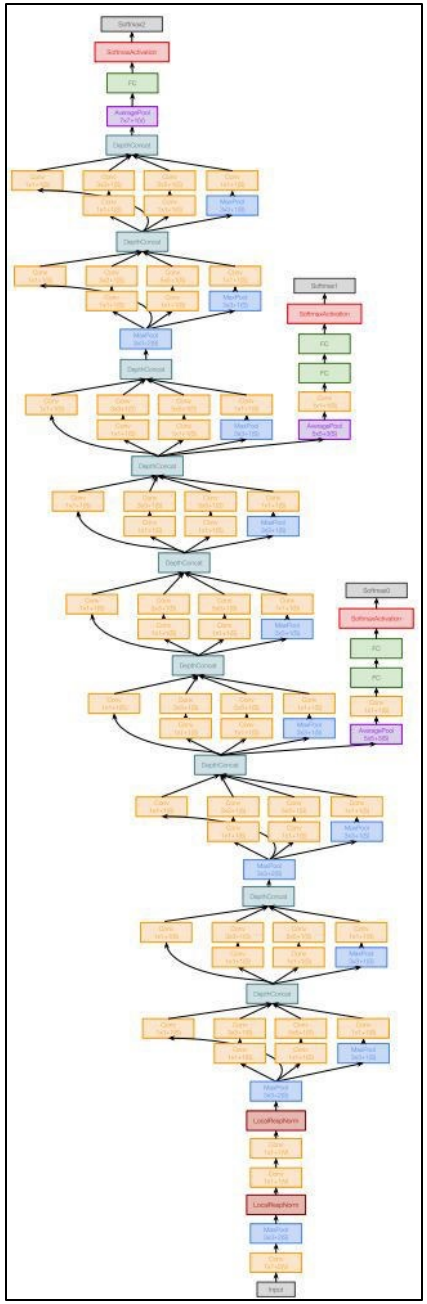
GoogleNet

"**Inception module":** design a good local network topology (network within a network) and then stack these modules on top of each other



Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[Szegedy et al., 2014]

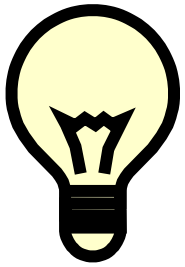


GoogleNet

Details/Retrospectives :

- Deeper networks, with computational efficiency
- 22 layers
- Efficient “Inception” module
- No FC layers
- 12x less params than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)

[Szegedy et al., 2014]



GoogleNet

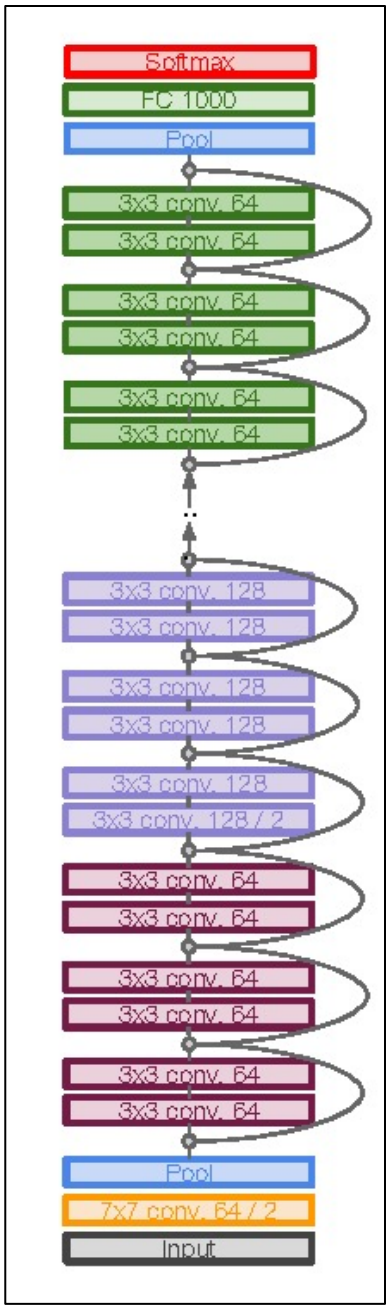
Introduced the idea that CNN layers **didn't always have to be stacked up sequentially**. Coming up with the Inception module, the authors showed that a creative structuring of layers can lead to improved performance and **computationally efficiency**.

[Szegedy et al., 2014]

ResNet

- *Deep Residual Learning for Image Recognition - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; 2015*
- **Extremely deep network – 152 layers**
- **Deeper neural networks are more difficult to train.**
- **Deep networks suffer from vanishing and exploding gradients.**
- **Present a residual learning framework to ease the training of networks that are substantially deeper than those used previously.**

[He et al., 2015]



ResNet

- ILSVRC'15 classification winner (3.57% top 5 error, humans generally hover around a 5-10% error rate)
  Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

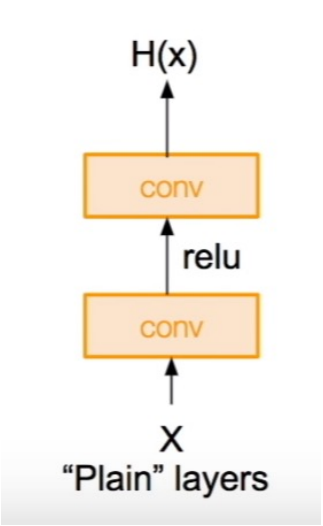
[He et al., 2015]

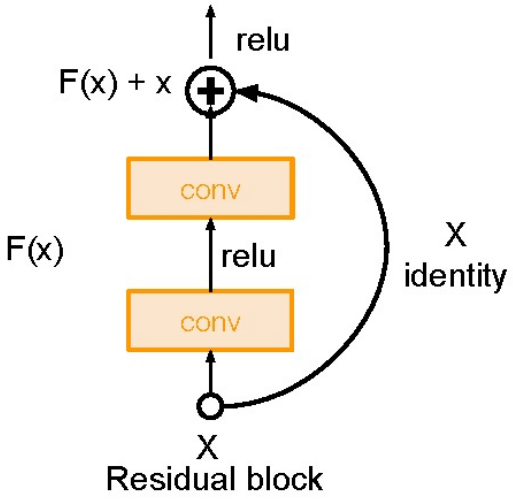
ResNet

Residual Block

Input x goes through conv-relu-conv series and gives us F(x). That result is then added to the original input x. Let's call that H(x) = F(x) + x.

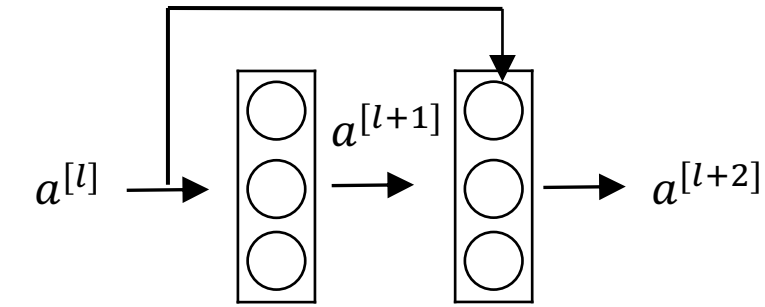
In traditional CNNs, H(x) would just be equal to F(x). So, instead of just computing that transformation (straight from x to F(x)), we're computing the term that we have to *add*, F(x), to the input, x.





[He et al., 2015]

ResNet



Short cut/ skip connection

$a^{[l]}$ → **Linear** → **ReLU** → **Linear** → **ReLU** → $a^{[l+2]}$

$a^{[l+1]}$

$$\mathbf{z}^{[l+1]} = \boldsymbol{\theta}^{[l+1]}\, \mathbf{a}^{[l]} + \mathbf{b}^{[l+1]}$$

$$\mathbf{a}^{[l+1]} = \mathbf{f}(\mathbf{z}^{[l+1]})$$

$$\mathbf{z}^{[l+2]} = \boldsymbol{\theta}^{[l+2]}\mathbf{a}^{[l+1]} + \mathbf{b}^{[l+2]}$$

$$\mathbf{a}^{[l+2]} = \mathbf{f}(\mathbf{z}^{[l+2]})$$

$$\mathbf{a}^{[l+2]} = \mathbf{f}\left(\mathbf{z}^{[l+2]} + \mathbf{a}^{[l]}\right) = \mathbf{f}\left(\boldsymbol{\theta}^{[l+2]}\mathbf{a}^{[l+1]} + \mathbf{b}^{[l+2]} + \mathbf{a}^{[l]}\right)$$

[He et al., 2015]

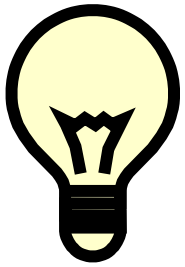
ResNet

Experimental Results:

- Able to train very deep networks without degrading
- Deeper networks now achieve lower training errors as expected

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[He et al., 2015]



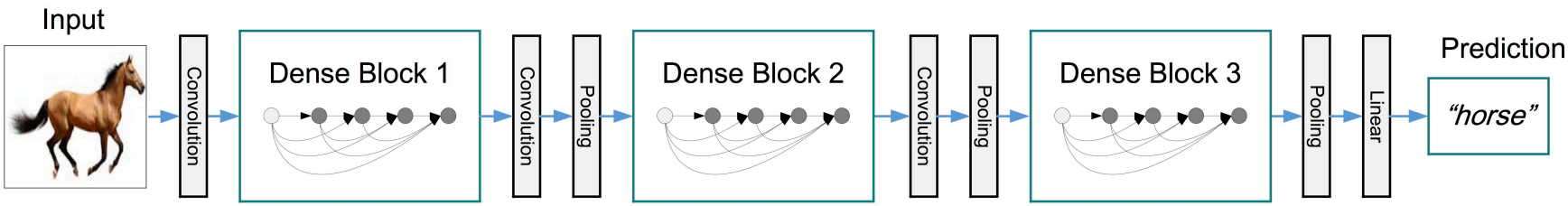
ResNet

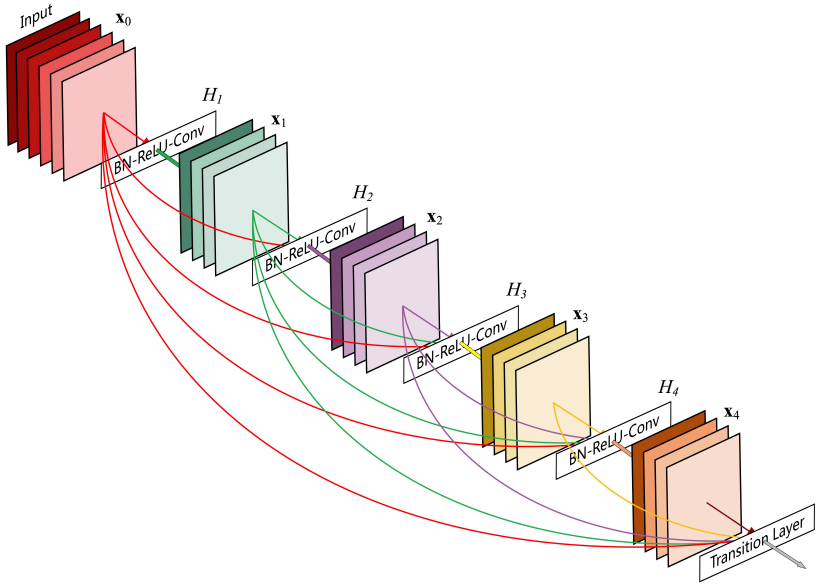
One of the best CNN architecture that we currently have and is a great innovation for the idea of residual learning.

Even better than human performance in some task!

[He et al., 2015]

DENSENET





Paper:
http://openaccess.thecvf.com/content_cvpr_2017/papers/Huang_Densely_Connected_Convolutional_CVPR_2017_paper.pdf
Poster:
https://liuzhuang13.github.io/posters/DenseNet.pdf
Slides:
https://pdfs.semanticscholar.org/c3d9/26a85d85a83126f405ad40ff453611148c15.pdf
Code:
https://github.com/liuzhuang13/DenseNet

Pre-Trained models

- Try to use the original implementation!
- Every peper usually have a model online.

- E.g., Mask R-CNN, YOLO, ecc..