Introduction to Deep Learning

Elad Hoffer

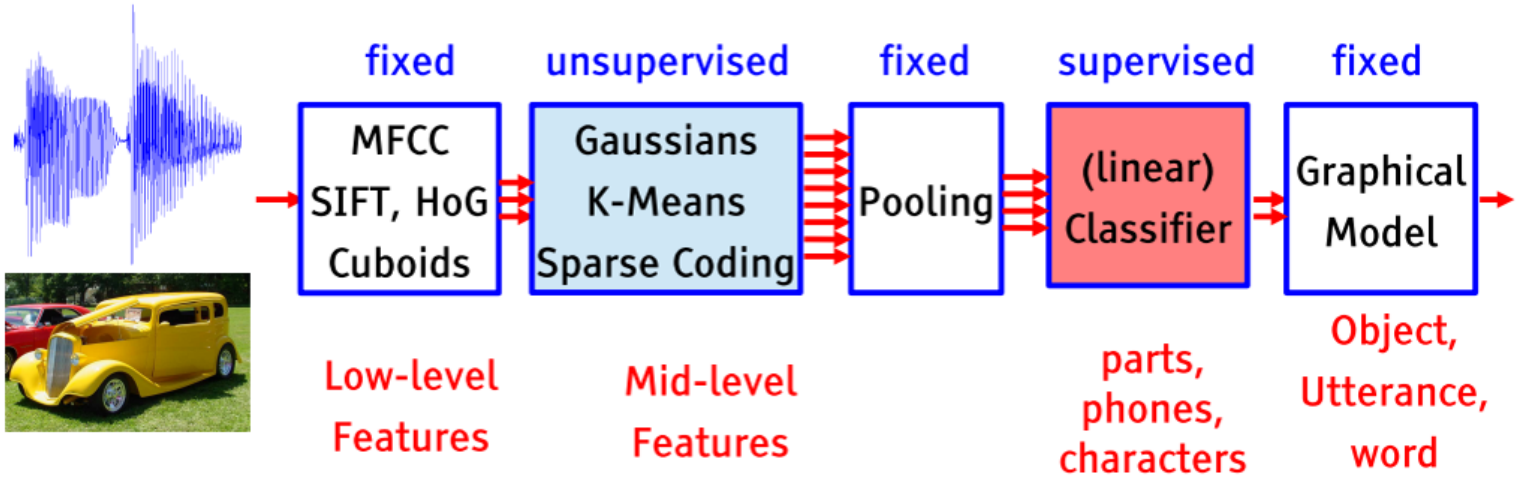
Technion Israel Institute Of Technology

Background

Traditional machine learning

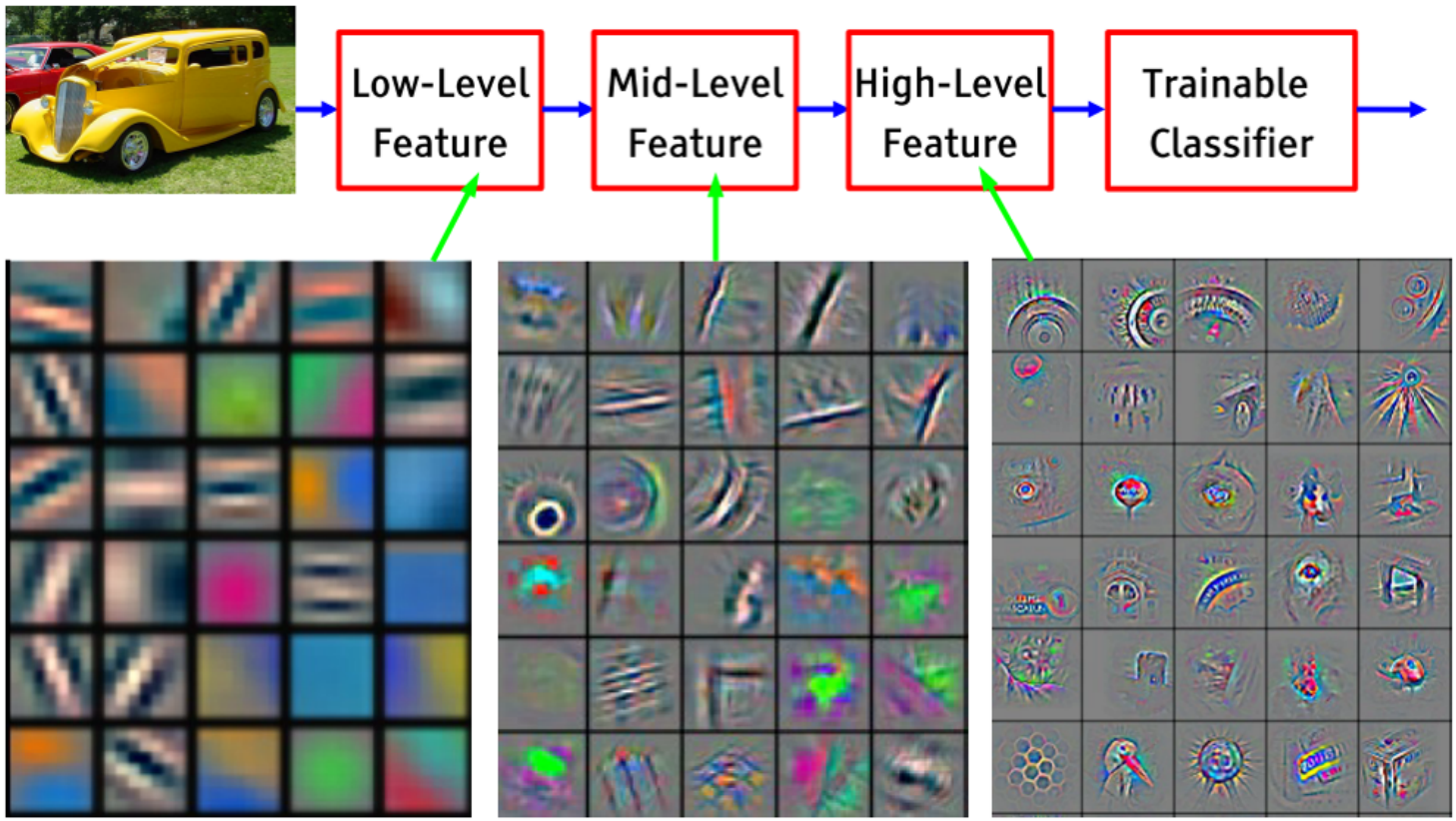
Traditional machine learning is about:

- Feature engineering
- Creating models and representations
- Optimizing and predicting



Feature learning

Can we create a model to learn and extract hierarchical feature representations from data?



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

What is Deep Learning?

Deep learning (deep structured learning or hierarchical learning) is a set of algorithms in machine learning that attempt to model high-level abstractions in data by using model architectures composed of multiple non-linear transformations.

Deep learning models appear as deep networks, deep Boltzman-machines, deep belief networks, deep auto-encoders and more.

But usually a deep learning model is simply a: Neural networks with MORE THAN 1 HIDDEN LAYERS

Short history of neural networks

Neural networks were used as machine learning models as early as the 1940s (McCulloch & Pitts).
They had a re-surge in the 1980s, but were largely abandoned in the 1990s.

Why?

- Researchers were unable to train large multi-layered networks.
- Training time was too long.
- Needed lots of data.

Lately, the availability of bigger, faster and better suited computational devices (GPUs) together with lots of usable data caused a NN renaissance (again).

Why you should know about deep learning?

You are already using it:

- Image classification (Google, Facebook, Microsoft...)
- Face recognition
- Voice recognition
- Machine translation

With a lot of new advancements ahead

- Reinforcement learning (Atari games, GO winner)
- Content generation: images, audio, art

Basics of neural networks

Basic models of neural networks

We will start by examining the most basic building block of a neural network - a *fully-connected* layer. This layer computes and affine function of its input: For $x \in \mathbb{R}^n$, weights $W \in \mathbb{R}^{m \times n}$ and bias vector $b \in \mathbb{R}^m$

$$f(x) = \varphi(Wx + b)$$

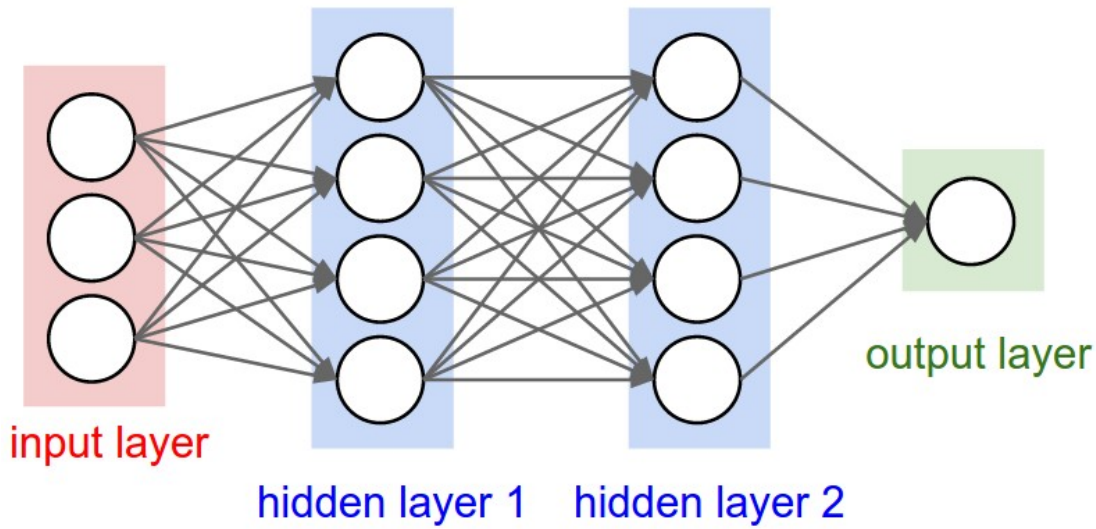
Where φ non-linear *activation function*.

Using the step-function as activation, this is the well-known *Perceptron* (Rosenblatt '57)

neural networks

We can now stack multiple layers to form a simple network.

- Networks are fed with inputs and trained to output the corresponding targets.
- Hidden layers (or hidden activations/representations), are said to *represent* the data, by applying non-linear transformations.



Why are non-linearities important?

Common non-linearities used:

- Sigmodial function - $Sigmoid(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic tangent - $TanH(x) = \frac{1-e^{-x}}{1+e^{x}}$
- Rectified linear unit - $ReLU(x) = \max(x, 0)$

We use non-linearities to prevent our network from "collapsing" to a linear model.

VISUALIZING SIMPLE NN

Objective and Loss functions

We will focus on problem of the form: given training examples $\{(x^{(1)}, t^{(1)}), \ldots, (x^{(N)}, t^{(N)})\}$ such that $x^{(i)}$ are the inputs and $t^{(i)}$ are the targets, we wish to minimize an objective function between the targets, and the network output $y = F(x)$.

In order to optimize our model according to the desired objective, we will need to choose an *error function*.

For example, we can perform regression using a network with a *Mean-square-error (MSE)* - computing for the network outputs $y^{(i)}$ vs targets $t^{(i)}$

$$E_{MSE}(y, t) = \frac{1}{N} \sum_{i=1}^{N} \|y^{(i)} - t^{(i)}\|^2$$

Optimizing neural networks

Stochastic-Gradient-Descent

Training a neural network with a large number of parameters requires a simple optimization technique.

- Usually, a a variant of *Stochastic gradient descent (SGD)* is used, using noisy subset estimation of the gradient.
- It requires $O(n)$ number of computations and memory use, where n is the number of parameters
- The most simple update-rule is:

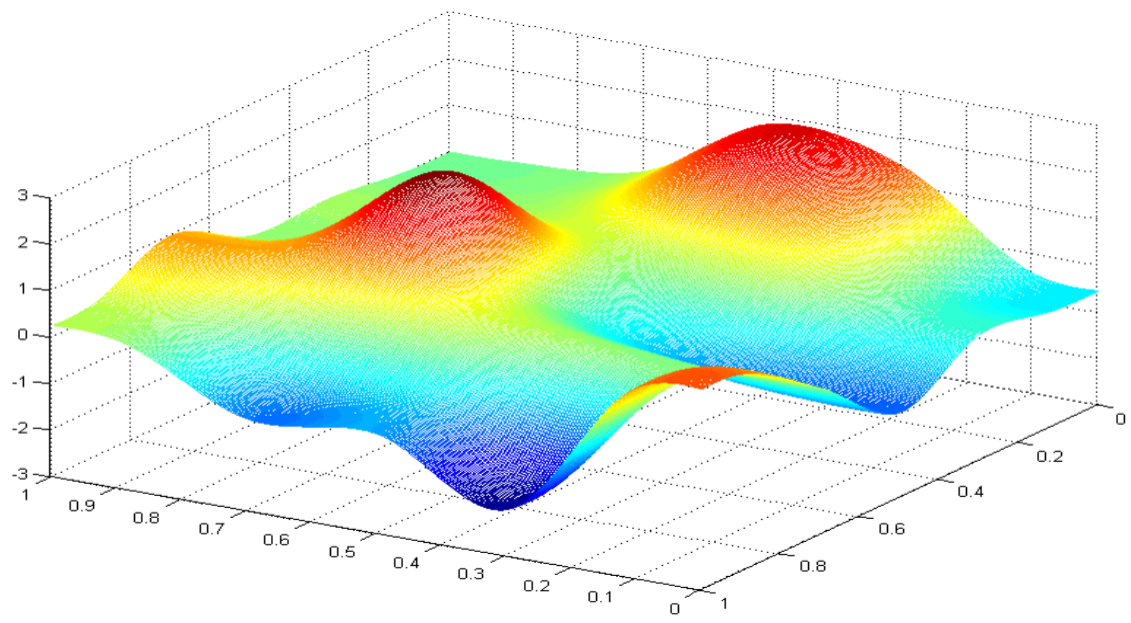
SGD update rule

$$w_{t+1} = w_t - \varepsilon \cdot \frac{\partial E}{\partial w_t}$$

where w is the optimized parameter (weight), E is the error (loss function estimation) and ε is the *Learning-Rate*

Efficient optimization of NNs

Optimizing neural networks can be very challenging, as we're dealing with a highly non-convex problem, residing in a high dimensional space.
Problems can be observed even in a very simple non-convex error landscape:



Efficient optimization of NNs

Some obvious difficulties:

- Strong dependency on initial weights
- Possibly poor error estimation leading to noisy gradients
- Exploring a (very) high-dimensional space using only local information

But before that, we need to compute the gradient of each parameter with regard to our optimized loss:

$$\frac{\partial E}{\partial W}?$$

Calculating the gradient

We will start by differentiating a *fully-connected* layer without the non-linearity. For $x \in \mathbb{R}^n$, weights $W \in \mathbb{R}^{m \times n}$ and bias vector $b \in \mathbb{R}^m$

$$f(x) = Wx + b$$

Knowing the error gradient with regard to the output $\frac{\partial E}{\partial f}$, we can compute the gradient with regard to input

$$\frac{\partial E}{\partial x} = \frac{\partial f}{\partial x} \cdot \frac{\partial E}{\partial f} = W^T \frac{\partial E}{\partial f}$$

and with regard to parameters (weight+bias)

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial f} x^T \quad , \quad \frac{\partial E}{\partial b} = \frac{\partial E}{\partial f}$$

Backpropagation

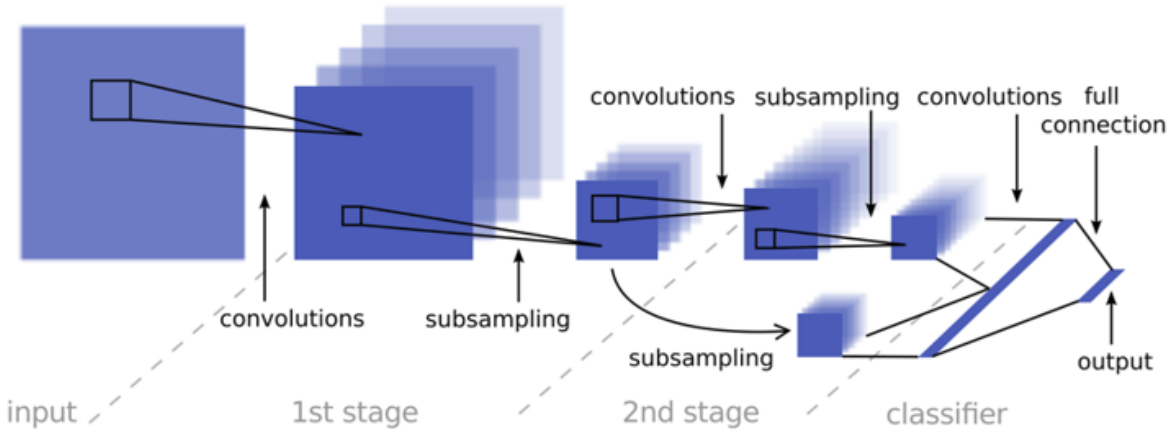
This can be expanded to arbitrary graph (DAG) using the chain rule for derivatives - a.k.a *Backpropagation*

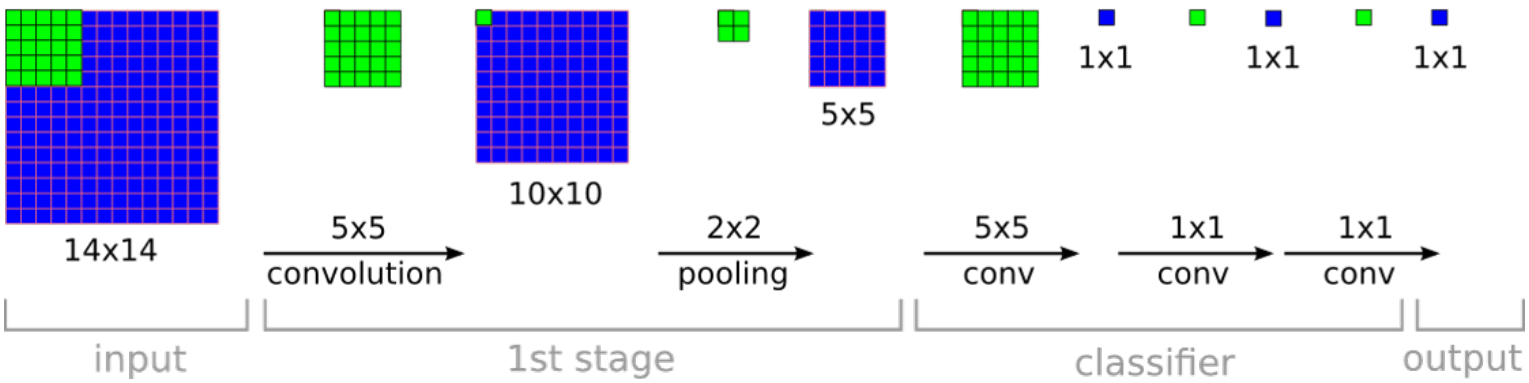
- Starting at the output and the gradient of our error function, we can backpropagate our gradients towards previous layers
- Each step is local - requiring the (saved) output of previous layer, and the gradient w.r.t to the next layer
- For each layer we calculate the gradient w.r.t to input, and gradient w.r.t weights

Convolutional networks

ConvNets

A convolutional network (ConvNet/CNN), is composed of multiple layers of convolutions, pooling and non-linearities.





Background

- Convolutional networks are rooted in ideas from "NeoCognitron" (Fukoshima 80') and inspired from Hubel's and Wiesel's work on visual primary cortex (59')
- Their first modern form appeared in Lecun's 98' work to recognize handwritten digits (LeNet).
- They reached their current status as visual recognition de-facto standard, after beating traditional computer-vision techniques in 2012 ImageNet competition by huge margin (AlexNet).

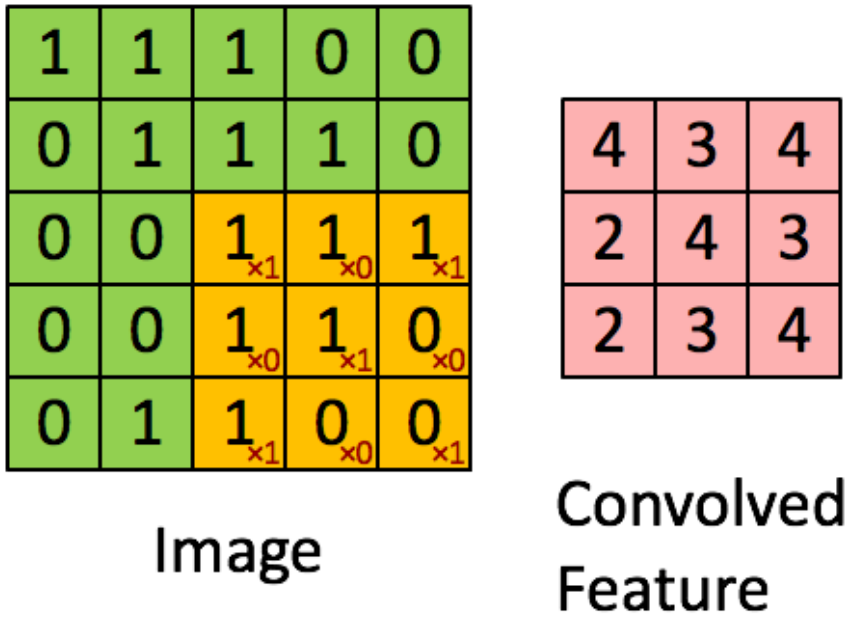
Convolution layer

Spatial Convolution

- Natural images have the property that many patches share statistical properties, and can often be described with a fairly small set of the same features
- This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and we can use the same features at all locations.
- This is a widely used property in computer-vision and image-processing related tasks.

Spatial Convolution

For example, having learned features over small (say 5×5) patches sampled from the image, we can then convolve them with the larger image, thus obtaining a different feature activation value at each location in the image.



Convention and terminology

- $2d$ grids that represent image space are called *feature maps*. Each value in a feature map represent information about a specific location in the input image.
- The parameters learned for each convolution layer are also ordered as a $2d$ map - usually called “kernels” or “filters”
- Convolutions can have *strides* - moving the filters with a fixed step and *padded* - adding zero values to avoid spatial size decrease
- Bias values are added per-map (number of bias elements is the number of output feature maps)

Forward function

It is most natural to think of this as correlation between two cubes:

- Our input is $X \in \mathbb{R}^{N \times W \times H}$ where N is the number of feature maps, each of spatial size $W \times H$.
- We wish to correlate them with $W \in \mathbb{R}^{M \times N \times K \times K}$ - M different filters of size $N \times K \times K$
- Output is $Y \in \mathbb{R}^{M \times (W-K+1) \times (H-K+1)}$ (no stride, no padding)

$$y_{m,i,j} = \sum_{n=0}^{N-1} \sum_{k=0}^{K-1} \sum_{l=0}^{K-1} w_{m,k,l} \cdot x_{n,(i+k),(j+l)} + b_m$$

It can also be expressed as sum of M 2d-convolutions

$$Y_m = \sum_{n=0}^{N-1} W_m * X_n + b_m$$

Properties

Convolutional layers can be seen as a specific configuration of a fully-connected layer. The assumptions we made about images are embedded as stationary entities allow us to use:

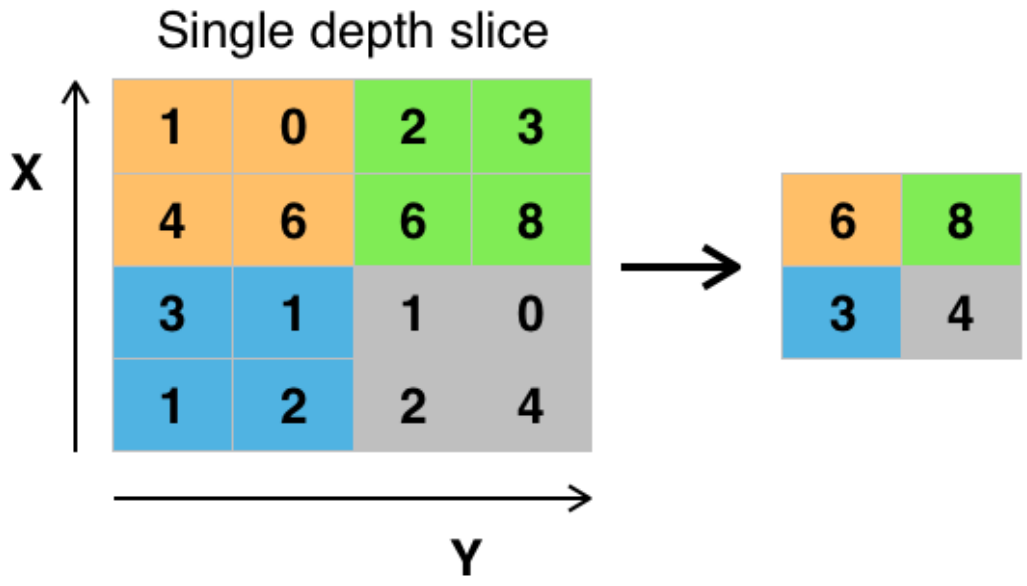
- Local connectivity - units in subsequent layer are connected only to a small subset in previous layer.
- Shared weights - weights used to extract information are shared across all spatial locations (this is not always the case - e.g faces).

Both these changes allow a dramatic reduction in the number of trainable parameters.

Pooling layers

Pooling

To describe a large image, one natural approach is to aggregate statistics of features at various locations. For example, one could compute the mean (or max) value of a particular feature over a region of the image.



Pooling

Pooling operations serve two main purposes:

- Dimensionality reduction - pooling is usually done with stride bigger than 1, effectively reducing the spatial size of the maps and allowing easier processing.
- Local invariance to translation/transformation - by pooling a spatial area to single point, we allow small invariance to location and deformations. This also reduces the ability to overfit on specific images (generalizes from patches).

Pooling functions

Most effective pooling methods are

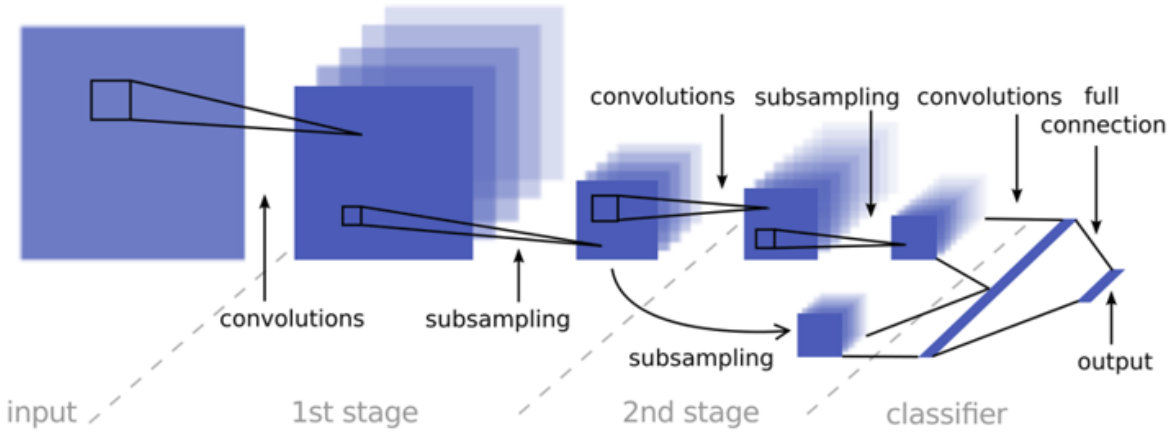
- Max Pooling - taking the maximum element from the pooled area.
- Average Pooling - taking the average of the pooled area.

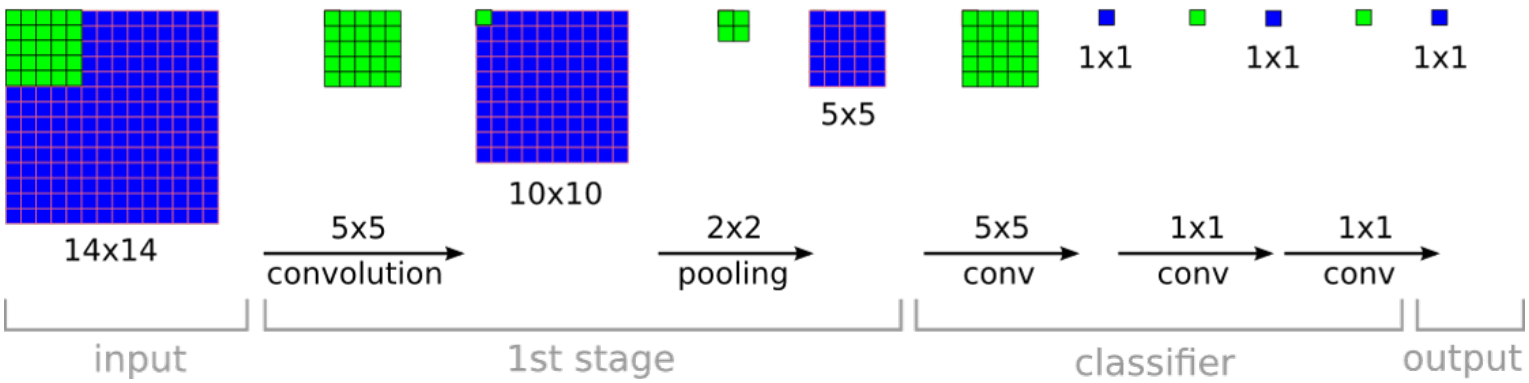
Less popular, but useful in some cases:

- Stochastic pooling - taking a random value.
- Fractional pooling - using a random pool size (2-3).
- L_p Pooling - taking the L_p norm of the pooled area.

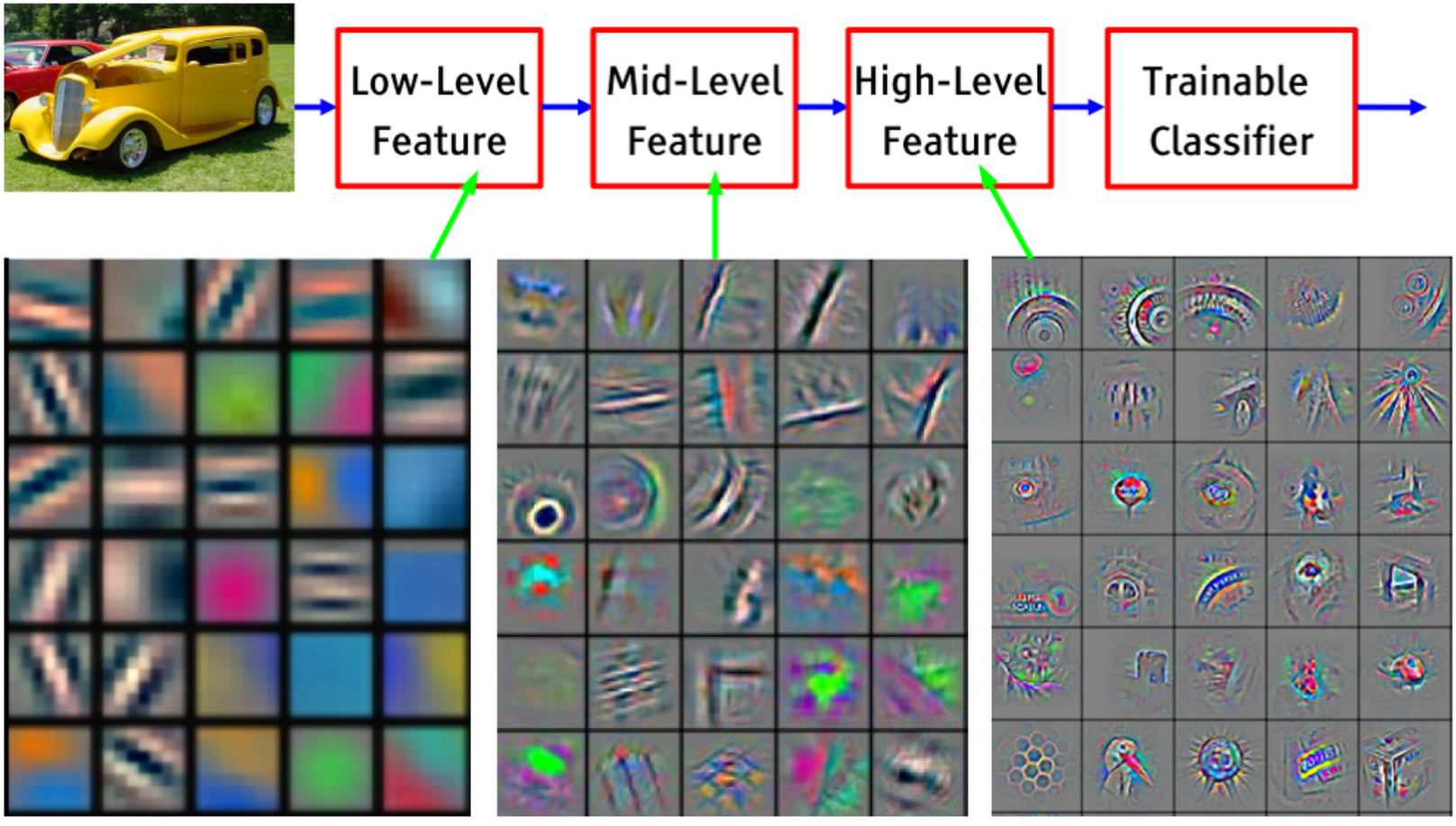
ConvNets

A convolutional network (ConvNet/CNN), is composed of multiple layers of convolutions, pooling and non-linearities.





What do convolutional networks learn?



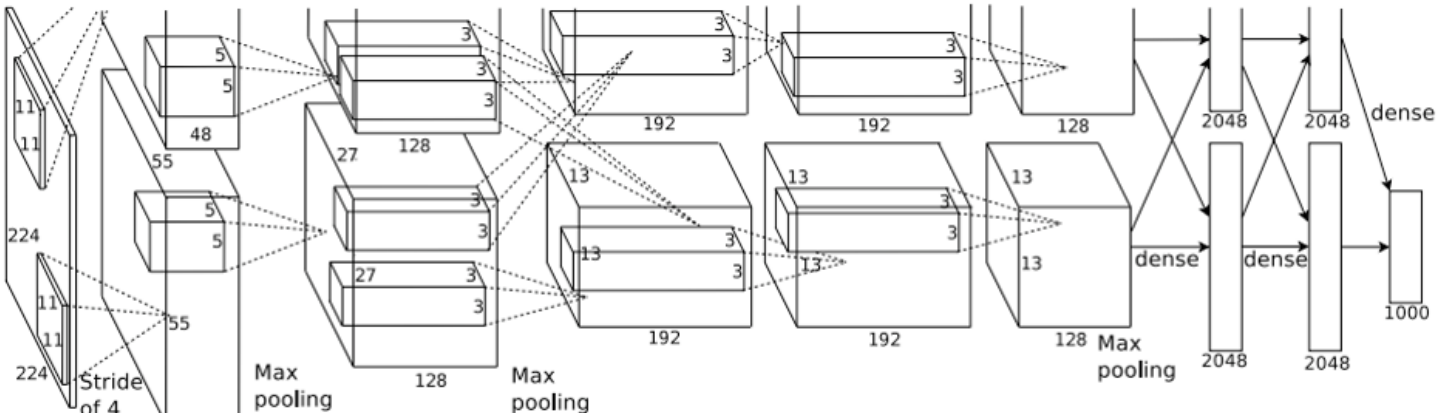
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

CNN ARCHITECTURES

AlexNet

The first work that popularized Convolutional Networks in Computer Vision was the "AlexNet", developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton.

- AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% vs 26% error).
- The Network was deeper, bigger, than previous networks - $60M$ parameters, 8 trainable layers.

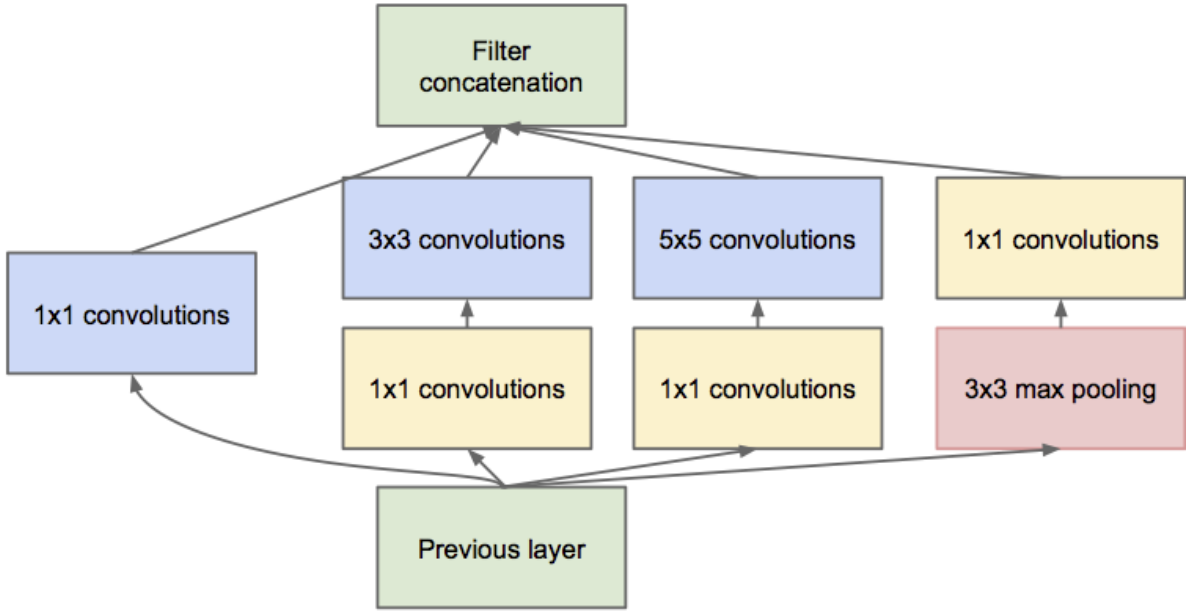


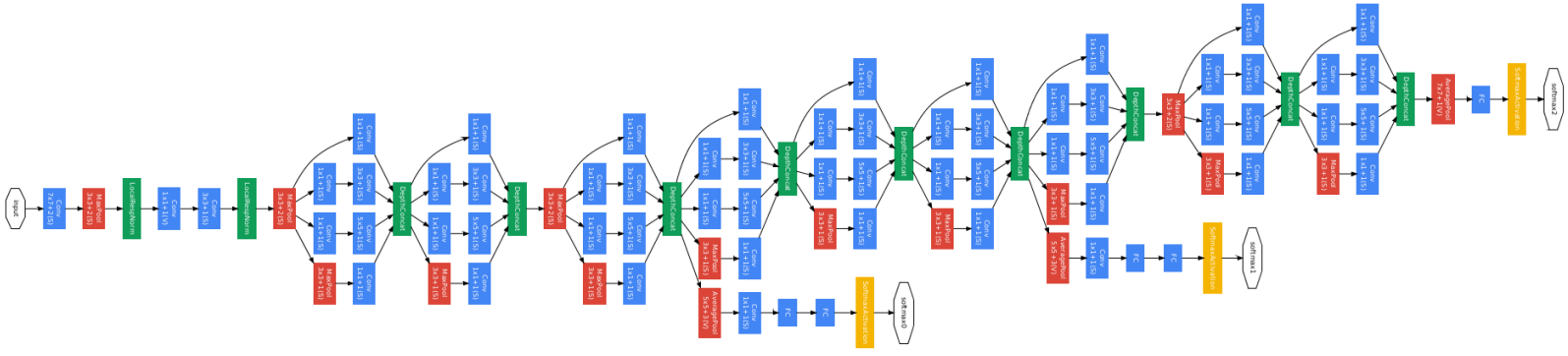
GoogLeNet

The ILSVRC 2014 winner was a Convolutional Network from Szegedy et al. from Google called *GoogLeNet*. It featured some new ideas

- Inception Module - concatenation of several convolution sizes
- 1×1 convolutions + Average Pooling instead of Fully Connected layers (both introduced by Lin in "Network-in-network" paper from 2013).
- This dramatically reduced the number of parameters in the network (5M, compared to AlexNet with 60M).
- Google later improved this network by introducing batch-normalization + different inception configuration (Inception v2-v4)

GoogLeNet


Filter concatenation
3x3 convolutions
5x5 convolutions
1x1 convolutions
1x1 convolutions
1x1 convolutions
1x1 convolutions
3x3 max pooling
Previous layer



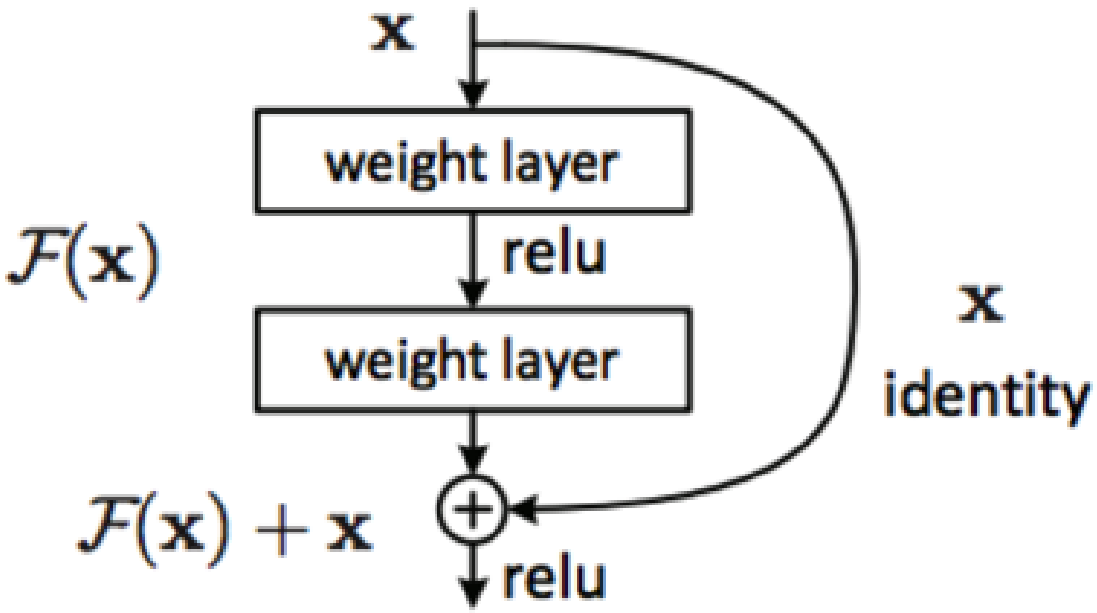
VGG Network

The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman that became known as the VGGNet. Its main contribution was in showing that the depth of the network is a critical component for good performance.

- Their best network contains 16 trainable layers
- Features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end.
- Expensive to evaluate and uses a lot more memory and parameters (140M).

Residual networks

- Residual Network developed by Kaiming He et al. was the winner of ILSVRC 2015. It features a very deep architecture (152 layers) with special skip connections and heavy use of batch normalization. The architecture is also missing fully connected layers at the end of the network.



Trends in designing CNNs

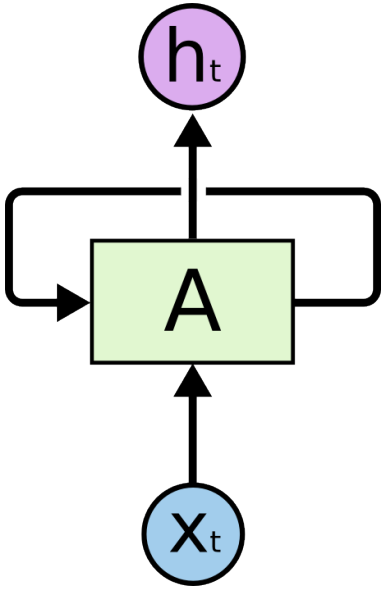
Some noticeable trends in recent CNN architectures:

- Smaller convolution kernels - 3×3 is very dominant.
- Deeper - using smaller kernels is rectified by stacking more layers - effectively increasing the receptive field of the network.
- Less pooling - going deeper means we do not want to reduce spatial size too quickly. Modern network have small number (if any) of pooling layers.

Recurrent neural networks

Recurrent neural networks

Recurrent neural networks (RNNs) are networks where connections between units form a directed cycle.



$$h_t = \phi\left(W_i x_t + W_r h_{t-1} + b\right)$$

$x_t \in \mathbb{R}^N$ – input at time t

$h_t \in \mathbb{R}^M$ – *state* at time t

$W_i \in \mathbb{R}^{M \times N}, W_r \in \mathbb{R}^{M \times M}, b \in \mathbb{R}^M$

ϕ is usually a bounded non-linearity $(Tanh, Sigmoid)$

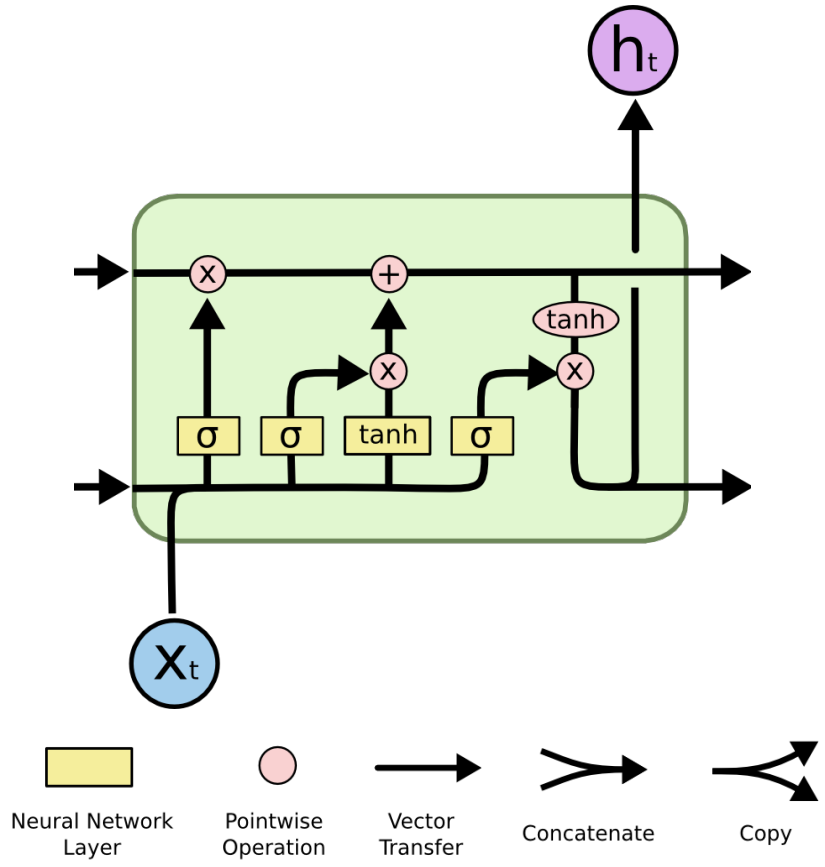
Recurrent neural networks

Recurrent neural networks properties:

- RNN hidden states work as memory on previous inputs and states
- Temporal dependency and internal memory allow processing of sequences of inputs
- Able to address wide range of time-dependencies
- Able to learn and infer sequences of varying length

Long-short-term-memory

Most succesful recurrent model used today:
Long Short-Term Memory (LSTM) architecture (Hochreiter and Schmidhuber, '97).



Vision tasks & usages

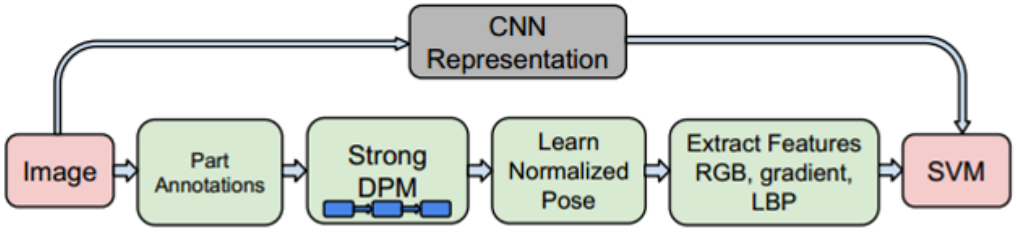
Transfer learning using deep networks

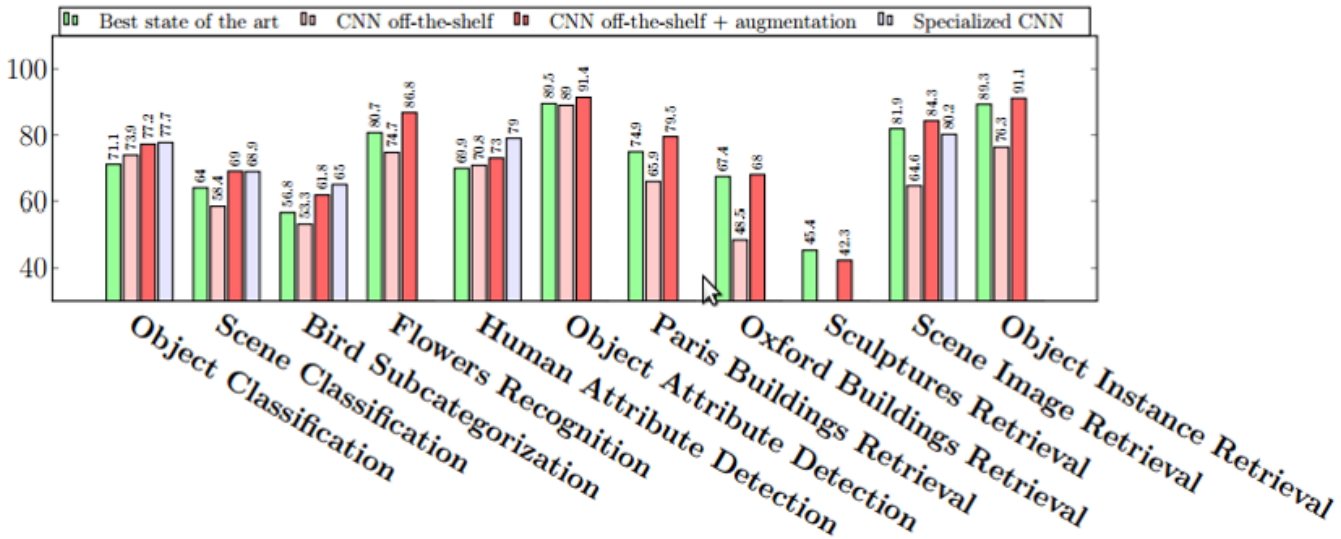
We've seen how, given enough data, we can train a convolutional network to learn complicated visual tasks.

- In practice, very few people train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size.
- Instead, it is common to use a network that was pretrained on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories),

Transfer learning

This turned out to work pretty well in practice - :





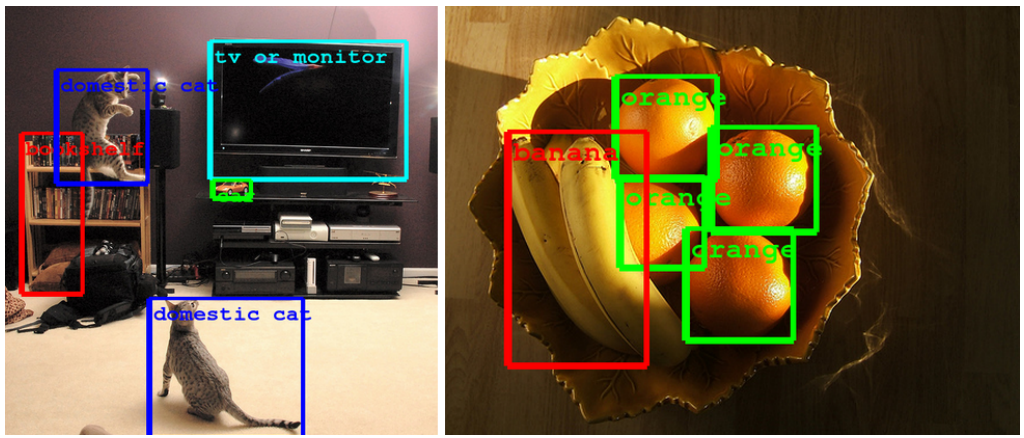
Transfer learning considerations using CNNs

When transfering our model to the new task, our two main approaches for using the trained CNN are

- *Generic feature extractor* - Take the pretrained network, remove the last fully-connected layer, then treat the rest of the ConvNet as a fixed feature extractor for the new dataset. You can then train a linear classifier (e.g. Linear SVM or Softmax classifier) on those features for the new dataset.
- *Fine-tuning* - Replace the classifier with one for the new task, then fine-tune the weights of the pretrained network by continuing the backpropagation. It is possible to fine-tune all the layers of the ConvNet, or it's possible to keep some of the layers fixed.

Localization + Detection

As we've seen, we can adapt a trained network for new sizes ands scales. A natural additional tasks in computer-vision is to try to *localize* and *detect* objects in an image.



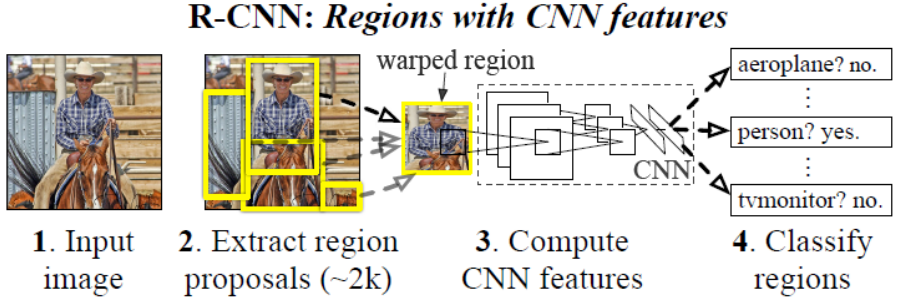
We need both to classify our object and to localize it

- For detection - we also need to handle multiple objects and reject a lot of false-positives (not-an-object)

Localization + Detection with CNNs

Some methods for detection using CNNs when we posses ground-truth bounding boxes

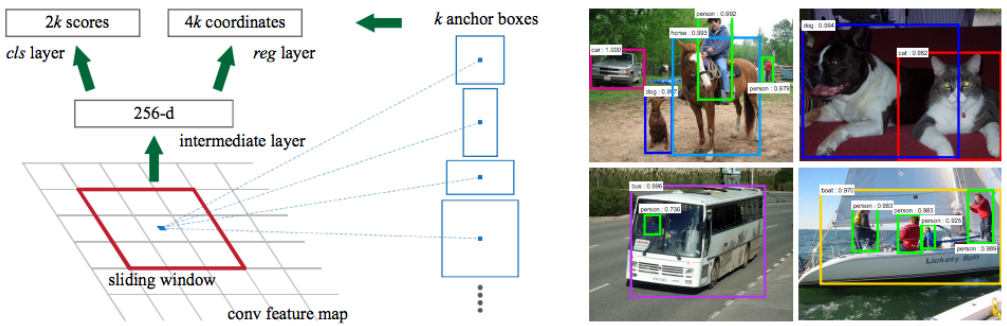
- Most naive approach is to add a regression objective - estimating the bounding boxes of each object (OverFeat - Sermanent 13').
- A more popular and effective method is to use *region-proposals* (often using "selective-search"), suggested in RCNN (Girshick 14')



Localization + Detection with CNNs

The RCNN approach, although accurate, had a bottleneck - generating and evaluating on ~ 2000 proposals. To remedy this, future works suggested

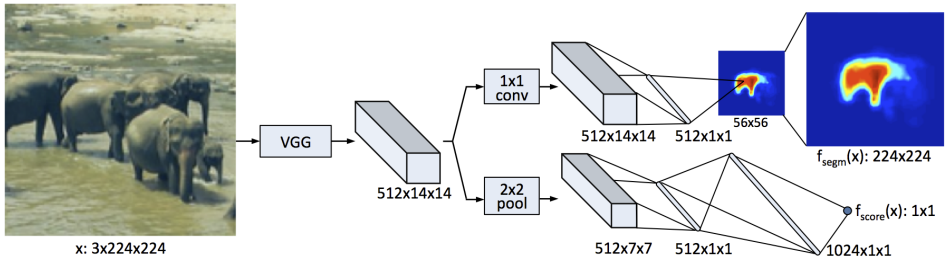
- Evaluating most of the network on entire image, and pooling the proposals from feature space instead of image space (SPP, Fast-RCNN)
- Moved the region-proposal to be an inherent part of the network (Faster-RCNN)



Segmentation using CNNs

Another important computer-vision task is to segment whole images. This can be viewed as a classification at the pixel level.

- We might wish to incorporate both global, abstract features, as well as localized low-level features
- This means using multiple layers for final segmentation (Long 14')
- As with detection, trend is to build whole end-to-end models by creating object-candidates (Pinheiro 15')

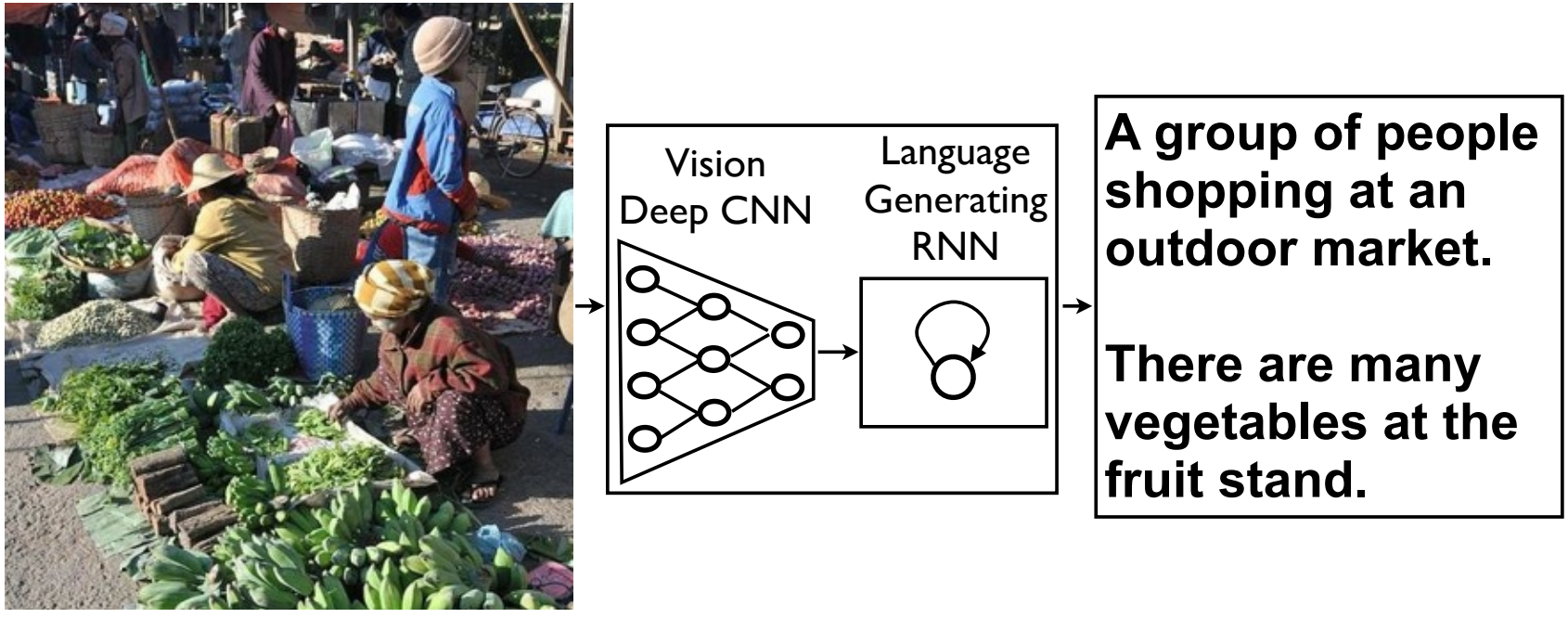


Segmentation using CNNs

Image segmentation (*DeepMask* Pinheiro 15′)

Caption generation

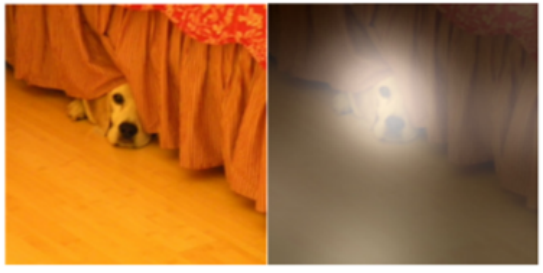
Another usage is transferring visual knowledge into language domain - such as generating captions or question answering. This is done by augmenting a convolutional network with a recurrent one.



Caption generation

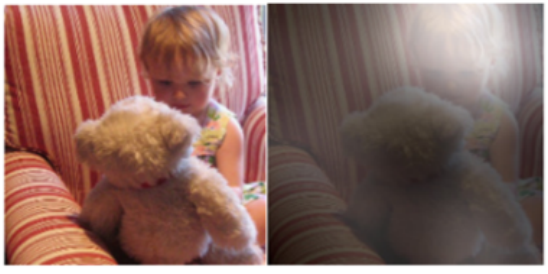
This can be coupled with attention mechanisms - learn a state-dependent weighting over the input space in order to maximize an objective.

A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

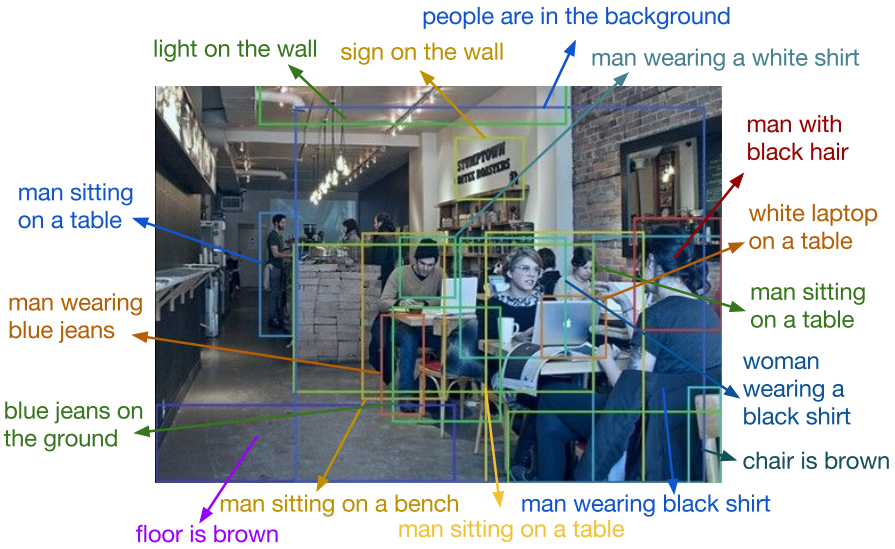


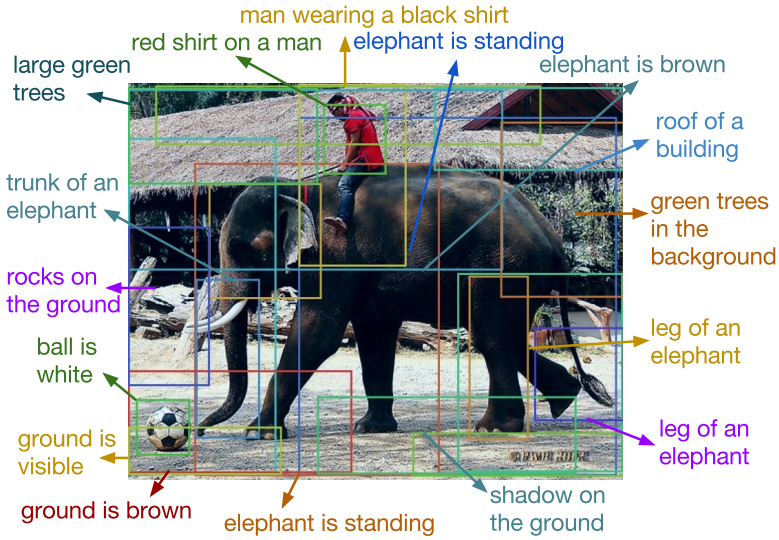
A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Caption generation





Style transfer

We can quantify an image "style" representation with the correlations between the different feature maps. These feature correlations are given by the Gram matrix $G^l \in \mathbb{R}^{N_l \times N_l}$, where for a given layer l:

$$S_{ij}^l = \sum_k F_{ik}^l F_{jk}^l.$$

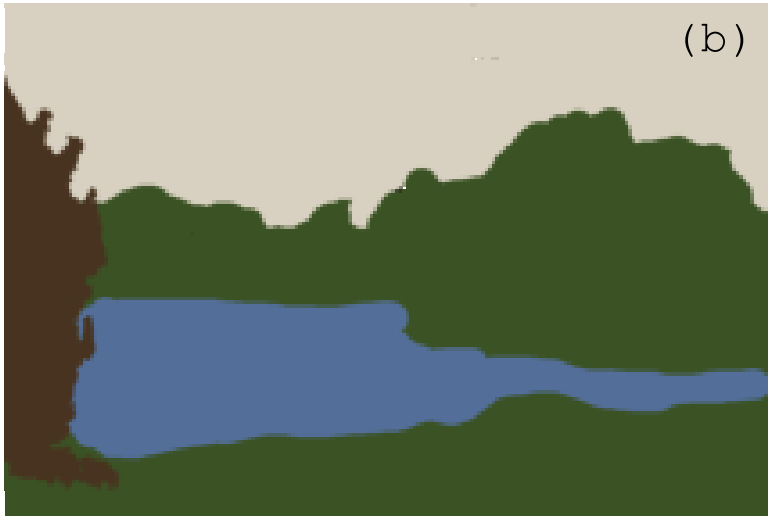
Thus we can change the target image x until it generates the same response in a certain layer of the CNN as the original image y.

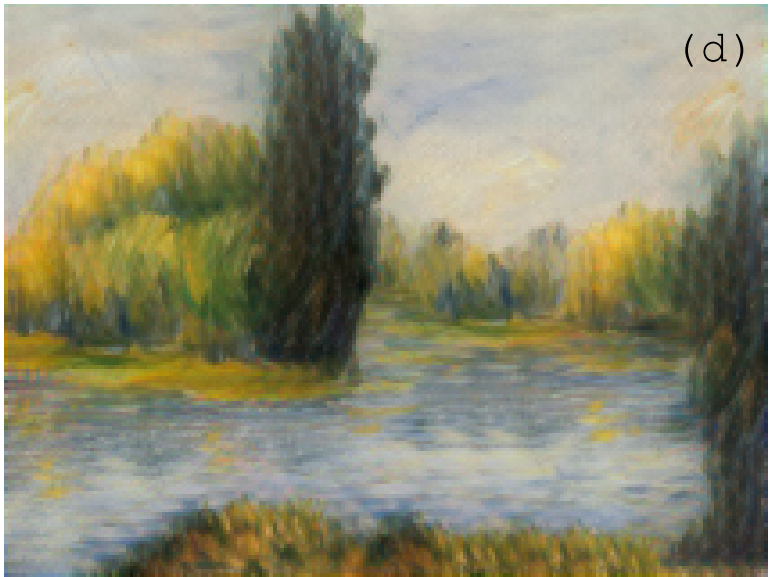
$$\mathscr{L}_{style} = \sum_{l=0}^{L} \frac{1}{N_l^2 M_l^2} \sum_{i,j} \left(S(x)_{ij}^l - S(y)_{ij}^l \right)^2$$

"A Neural Algorithm of Artistic Style" (Gatys et al)

Style transfer

Style transfer





Generative Models

Generative adversarial networks

One new and and exciting framework for unsupervised, generative models is the *Adversrial network* (Goodfellow 14'). It consists of two networks trained simultaneously:

- A *generative* model G that tries to capture the data distribution and generate new samples.
- A *discriminative* model D that estimates the probability that a sample came from the training data rather than G.

The training procedure for G is to maximize the probability of D making a mistake - it is "cheating" by using the gradients with respect to D.

Generative adversarial networks

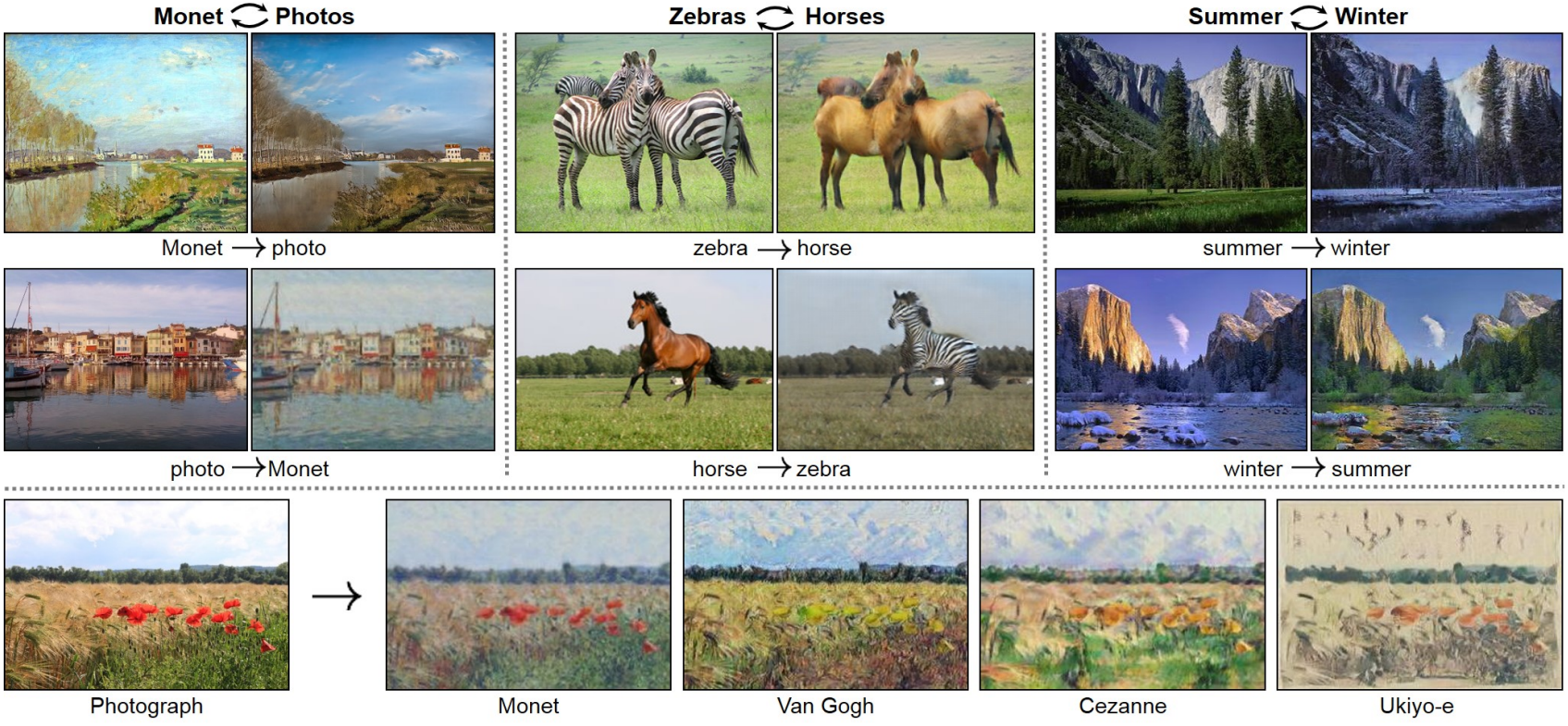
"Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks" (Alec Radford, Luke Metz, Soumith Chintala 15')

Generative adversarial networks

BE-GAN (Berthelot et al. 17')

Many GAN papers over the past year - with impressive improvement over time.

Cycle-GAN (Zhu et al. 17')



Deep dreaming

Another interesting usage is to try and maximize the L_2 norm of intermediate layer activations, by optimizing an input image. This can help visualize the role each layer plays in the model.