Convolutional Neural Networks

E. Scornet

Fall 2020

Neural Network reborn

Renewed interest in 2006: ["A fast learning algorithm for deep belief nets", Hinton et al. 2006]

Propose a way to train deep neural nets:

- Train the first layer.
- Add a layer on top of it and train only this layer.
- Repeat the process until the network is deep enough.
- Use this network as a warm start to train the whole network.

Technical reasons for this new growing interest:

- Larger datasets
- More powerful computers
- Software infrastructure
- Small number of algorithmic changes
  1. MSE replaced by cross-entropy
  2. ReLU (Fukushima, 1975, 1980)

Using classical networks for images?

No, for two reasons:

- Do not take into account the spatial organization of pixels (if the pixels are permuted, the output of the network would be the same, whereas the image would change drastically)
- Non robust to image shifting

Idea:

- Apply local transformation to a set of nearby pixels (spatial nature of image is used)
- Repeat this transformation over the whole image (resulting in a shift-invariant output)

Not a new idea: trace back to perceptron and studies about the visual cortex of a cat. The cat is able to

- detect oriented edges, end-points, corners (low-level features)
- combine them to detect more complex geometrical forms (high-level features)

Outline

1. Foundations of CNN
   - Convolution layer
   - Pooling layer
   - Data preprocessing
2. Famous CNN
   - LeNet (1998)
   - AlexNet (2012)
   - ZFNet (2013)
   - VGGNet (2014)
   - GoogLeNet (2014)
   - ResNet (2016)
   - DenseNet (2017)
   - Many other CNN
3. Applications
   - Image classification
   - Pose, action detection
   - Object detection
   - Scene labeling - Semantic segmentation
   - Object tracking - videos
   - Text detection and recognition

Outline

1. Foundations of CNN
   - Convolution layer
   - Pooling layer
   - Data preprocessing
2. Famous CNN
   - LeNet (1998)
   - AlexNet (2012)
   - ZFNet (2013)
   - VGGNet (2014)
   - GoogLeNet (2014)
   - ResNet (2016)
   - DenseNet (2017)
   - Many other CNN
3. Applications
   - Image classification
   - Pose, action detection
   - Object detection
   - Scene labeling - Semantic segmentation
   - Object tracking - videos
   - Text detection and recognition

Convolutional neural networks (CNNs)

- Neural networks that use **convolution instead of matrix product** in one of the layers
- A CNN layer typically includes 3 operations: **convolution**, **activation** and **pooling**
- Using the more general idea of **parameters sharing**, instead of **full connection** (convolution instead of matrix product)

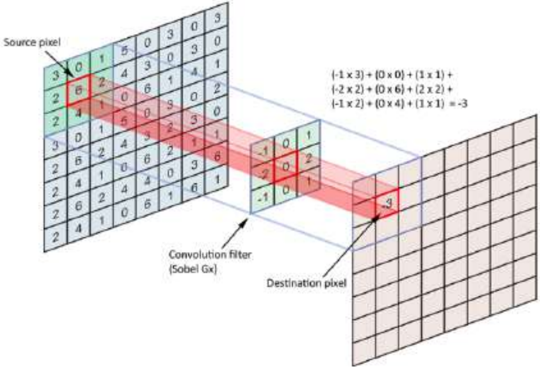
Convolution operator in neural networks is as follows

$$O(i,j) = (I \star K)(i,j) = \sum_k \sum_l I(i+k, j+l) K(k,l)$$

- I is the input and K is called the kernels
- The kernel K will be **learned** (replaces the weights $\mathbf{W}$ in a fully connected layer)

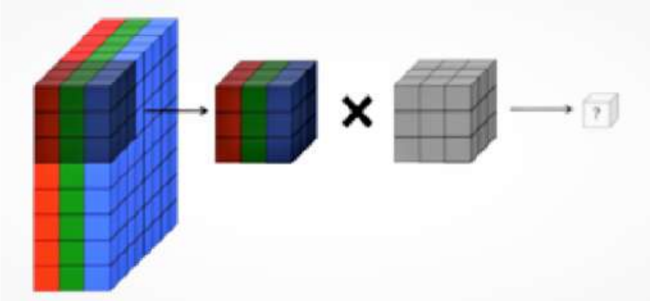
Convolution - Grey scale

- Size of the input image is $8 \times 8 \times 1$ (height, width, depth)
- Size of the kernel is $3 \times 3 \times 1$



Convolution - RGB

- Size of the input image is $8 \times 8 \times 3$ (height, width, depth)
- Size of the kernel is $3 \times 3 \times 3$



Warning: every filter is small spatially (along width and height), but extends through the full depth of the input volume.

Parameters of convolutional layer 1/4

Four hyperparameters control the size of the output volume: the kernel size, the depth, the stride and the zero-padding.

- The size of the kernel (typically 3×3, 5×5).

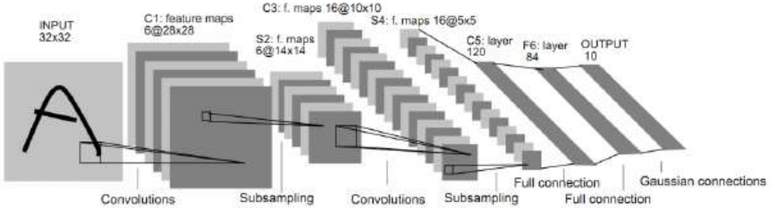


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Parameters of convolutional layer 2/4

Four hyperparameters control the size of the output volume: the kernel size, the depth, the stride and the zero-padding.

- The size of the kernel,
- The depth of the output volume, i.e., the number of filters/activation maps/feature maps.

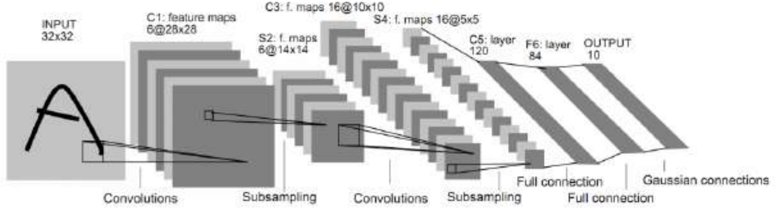


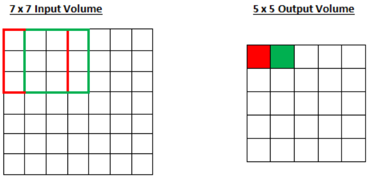
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Parameters of convolutional layer 3/4

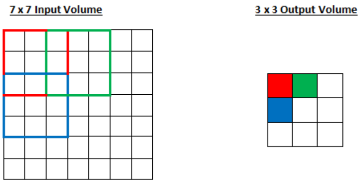
Four hyperparameters control the size of the output volume: the kernel size, the depth, the stride and the zero-padding.

- The size of the kernel,
- The depth of the output volume,
- The stride, i.e., of how many pixels do we move the filter horizontally and vertically. Usually, stride is equal to one (rarely to two, and even more rarely larger).

Stride $= 1$



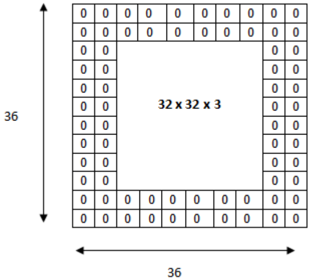
Stride $= 2$



Parameters of convolutional layer 4/4

Four hyperparameters control the size of the output volume: the kernel size, the depth, the stride and the zero-padding.

- The size of the kernel,
- The depth of the output volume,
- The stride,
- The size of the zero-padding, i.e. the number of zeros we add to the borders of the image. This can be used to obtain a constant image size between the input and the output.



Zero-padding of 2

How to choose zero-padding?

Let

- I the height/width of the input
- O the height/width of the output
- P the size of the zero-padding
- K the height/width of the filter
- S the stride

What is the relation between these quantities? How do we choose the zero-padding to obtain an output of the same size as the input?

How to choose zero-padding?

Let

- I the height/width of the input
- O the height/width of the output
- P the size of the zero-padding
- K the height/width of the filter
- S the stride

What is the relation between these quantities? How do we choose the zero-padding to obtain an output of the same size as the input?

$$O = \left\lfloor \frac{2P + I - K}{S} \right\rfloor + 1$$

Why convolution?

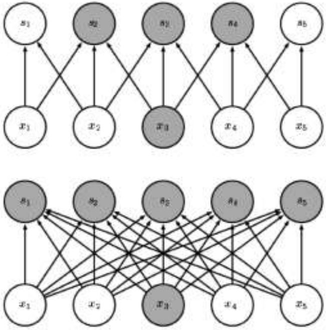
- Same transformation applied to all parts of the image (takes into account the spatial dependence between pixels and object-shift invariance)

- Input image contains millions of pixel values, but we want to detect small meaningful features such as edges with kernels that use only few hundred of pixels

- When using a matrix product, all input and output units are connected, whereas convolution connects only output neurons with several pixels of the input image.

  Convolution involves weight sharing (a form of regularization) and requires less parameters which improves memory, is more statistically efficient and computationally faster.

Sparse connections



[from *Deep Learning*, Goodfellow, Bengio and Courville]

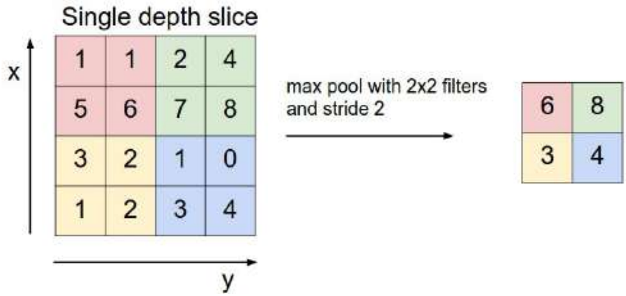
- Top: in a convolution with a kernel of width 3, only three outputs are affected by the input x. We say that the **connectivity is sparse**
- Bottom: when using matrix multiplication, all outputs are connected to an input. We say that **connectivity is dense**

Outline

1. Foundations of CNN
   - Convolution layer
   - Pooling layer
   - Data preprocessing
2. Famous CNN
   - LeNet (1998)
   - AlexNet (2012)
   - ZFNet (2013)
   - VGGNet (2014)
   - GoogLeNet (2014)
   - ResNet (2016)
   - DenseNet (2017)
   - Many other CNN
3. Applications
   - Image classification
   - Pose, action detection
   - Object detection
   - Scene labeling - Semantic segmentation
   - Object tracking - videos
   - Text detection and recognition

Pooling

The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the max function.



Parameters:

- Stride S
- Spatial extend F

Usually, $S = F = 2$ and more rarely $F = 3, S = 2$ (overlapping pooling).

Pooling

- Pooling layers compute each pixel of the output as a summary statistic of neighboring input pixels at the corresponding location.
- The most widely used is the max aggregation, called max-pooling
- Pooling helps the representation to become approximately invariant to small translations of the input
- If a small translation is applied, output of the layer is almost unchanged
- Very useful if we care more about the presence of some feature than its position in the image: for face detection (presence of eyes is more important than where they are)
- Pooling also allows to handle inputs with different sizes: pictures can have different sizes, but the output classification layer must be of fixed size

Convolutional Neural Network

Typically, one or more convolutional layers are followed by one pooling layer and so on. At the end of the network, several fully connected layers are typically used to compute probabilities.

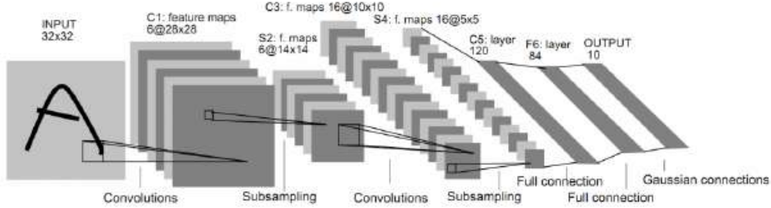


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Outline

1. Foundations of CNN
   - Convolution layer
   - Pooling layer
   - Data preprocessing
2. Famous CNN
   - LeNet (1998)
   - AlexNet (2012)
   - ZFNet (2013)
   - VGGNet (2014)
   - GoogLeNet (2014)
   - ResNet (2016)
   - DenseNet (2017)
   - Many other CNN
3. Applications
   - Image classification
   - Pose, action detection
   - Object detection
   - Scene labeling - Semantic segmentation
   - Object tracking - videos
   - Text detection and recognition

Data processing

Normalizing data

For each channel R, G, B, compute the pixels mean over all images in the whole data set. Substract this value to each channel of each image. → you do not lose relative information between images.

Data augmentation

1. Sampling ["Imagenet large scale visual recognition challenge", Russakovsky et al. 2015]
2. Translation/shifting ["Deep convolutional neural networks and data augmentation for environmental sound classification", Salamon and Bello 2017]
3. Horizontal reflection/mirroring ["Mirror, mirror on the wall, tell me, is the error small?", H. Yang and Patras 2015]
4. Rotating ["Holistically-nested edge detection", Xie and Tu 2015]
5. Various photometric transformations ["Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture", Eigen and Fergus 2015]

Prediction

At test time, patches are extracted from the new images together with some of its reflection/translation/... A prediction is made for each of these artificial images and they are aggregated to make the final prediction.

Adding noise - Data augmentation and regularization

- Add noise to input

  ["Training with noise is equivalent to Tikhonov regularization", Bishop 1995]

  ["Adding noise to the input of a model trained with a regularized objective", Rifai et al. 2011]

  ["Explaining and harnessing adversarial examples", Goodfellow et al. 2014]

- Add noise to weights

  ["An analysis of noise in recurrent neural networks: convergence and generalization", Jim et al. 1996]

  ["Practical variational inference for neural networks", Graves 2011]

- Add noise to output

  ["Randomizing outputs to increase prediction accuracy", Breiman 2000]

- Select the best data transformations (computationally expensive, many re-training steps).

  ["Transformation pursuit for image classification", Paulin et al. 2014]

Outline

1 Foundations of CNN
- Convolution layer
- Pooling layer
- Data preprocessing

2 Famous CNN
- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2016)
- DenseNet (2017)
- Many other CNN

3 Applications
- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

Outline

1 Foundations of CNN
- Convolution layer
- Pooling layer
- Data preprocessing

2 Famous CNN
- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2016)
- DenseNet (2017)
- Many other CNN

3 Applications
- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

LeNet

["Generalization and network design strategies", LeCun et al. 1989]

["Gradient-based learning applied to document recognition", LeCun et al. 1998]

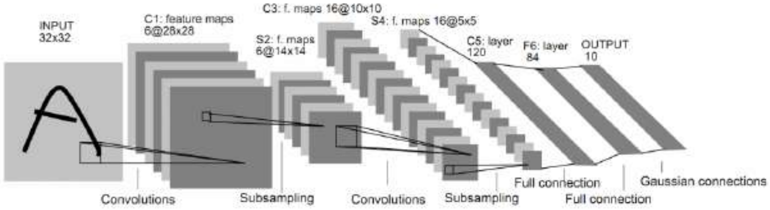


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

First layer: convolutional layer C1

- Kernel size $= 5 \times 5 +$ a bias
- Stride $= 1$ (overlapping contiguous receptive fields)
- Zero-padding $= 0$
- Output: 6 different feature maps, each one resulting from the convolution with a kernel 5×5 to which the activation function σ is applied.

Second layer: subsampling/pooling layer S2

- Type of pooling: averaging.
- Kernel size $= 2 \times 2$
- Stride $= 2$ (non-overlapping receptive fields)
- Zero-padding $= 0$
- Output: one feature map per input feature map resulting from the operation $\sigma((2 \times 2 \text{ averaging})w + b)$.

Third-layer: convolutional layer C3

- Warning: this layer operates on several feature maps whereas layer C1 operates on the input image (depth $= 1$).
- Here each feature map is connected to some specific input feature maps in order to
  - Reduce the number of connections
  - Break the symmetry between the different layers of the network.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

What about the remaining layers

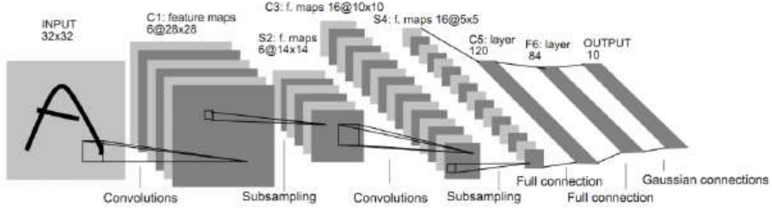


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

What about the remaining layers

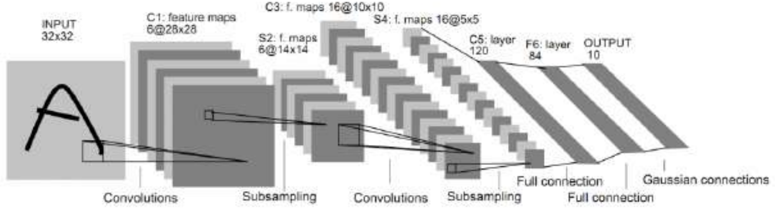


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

- S4: Pooling layer as before
- C5: Convolutional layer connected to all previous feature maps.
- F6: fully-connected layer with 84 units
- Output: a specific layer

Bi-pyramidal structure: the number of feature maps increases while the spatial resolution decreases.

Output layer

Radial Basis function units
The jth neuron of the output layer computes

$$\|z - w_j\|_2^2 = \sum_{i=1}^{84} (x_i - w_{j,i})^2,$$

where z is the vector of size 84 produced by layer F6 and $w_j = (w_{j,1}, \ldots, w_{j,84})$ is the weight vector of the jth neuron.

Gaussian connections
Assuming that the vector in layer F6 are Gaussian, neuron j outputs the negative log likelihood of a Gaussian distribution with mean w_j and covariance matrix I.

In other words, each neuron outputs the square euclidean distance between its parameter vector and the input.

Question.
How to choose $w_j \in \{-1, 1\}^{84}$?

Output layer and activation function

To choose $w_0 \in \{-1, 1\}^{84}$, use a stylized version of the image of 0 of size $7 \times 12 = 84$. The pixel of this image are the parameters w_j of the output neuron $j = 0$.

Why do not use a one-hot encodage?

LeCun et al. 1998 states that it does not work with more than few dozens of classes since it requires output units to be off most of the time which is difficult to achieve with sigmoid functions.

Activation function

$$\sigma(x) = A \tanh(\alpha x),$$

where $A = 1.7159$, $\alpha = 2/3$.

$\rightarrow$ Prevent saturation since neurons outputs belong to $\{-1, 1\}$

- $\sigma(1) = 1$
- $\sigma(-1) = -1$.

Criterion to optimize

Let $[f_\theta(x)]_j = \|z - w_j\|_2^2$ be the output of the jth neuron of the output layer, where z is the vector produced by layer F6.
Then the error for one observation (x, y) is defined as

$$E(\theta) = \sum_{j=0}^{9} [f_\theta(x)]_j \mathbb{1}_{y=j} + \log\left(e^{-C} + \sum_{j=0}^{9} e^{-[f_\theta(x)]_j} \right),$$

where $C > 0$ is a constant.

The second term acts as a regularization since it forces the parameters of the neurons $j \neq y$ to be far from the input vector of layer F6.

Criterion to optimize

Let $[f_\theta(x)]_j = \|z - w_j\|_2^2$ be the output of the jth neuron of the output layer, where z is the vector produced by layer F6.
Then the error for one observation (x, y) is defined as

$$E(\theta) = \sum_{j=0}^{9} [f_\theta(x)]_j \mathbb{1}_{y=j} + \log\left(e^{-C} + \sum_{j=0}^{9} e^{-[f_\theta(x)]_j} \right),$$

where $C > 0$ is a constant.

The second term acts as a regularization since it forces the parameters of the neurons $j \neq y$ to be far from the input vector of layer F6.

This is equivalent to

$$E(\theta) = -\log\left(\frac{e^{-[f_\theta(x)]_y}}{e^{-C} + \sum_{k=0}^{9} e^{-[f_\theta(x)]_k}} \right),$$

which is very close to the negative log likelihood of a softmax output layer.

Optimization procedure

Related to stochastic gradient descent:

$$\theta_j^{(k+1)} = \theta_j^{(k)} - \frac{\eta}{\mu + h_{jj}} \frac{\partial E_i}{\partial \theta_j},$$

where E_i is the loss of a single observation, η is the initial learning rate, μ a hand-picked constant and h_{jj} is the jth diagonal element of the Hessian matrix associated to E_i.

The expression of h_{jj} is quite complicated since θ_j appears in different connections:

$$h_{jj} = \sum_{(i,m)\in V_j} \sum_{(k,l)\in V_j} \frac{\partial^2 E_i}{\partial u_{im} \partial u_{kl}},$$

where u_{im} is the connection between units i and m, and V_j is the set of pairs (i, m) such that the connection between i and m involves the weight θ_j.

An approximation of each diagonal terms h_{jj} is performed at the beginning of each epoch, using the first 500 observations (whole data set being composed of 60000 observations).

Parameters

Weight initialization: uniform distribution $U([-2.4/F_i, 2.4/F_i])$, where F_i is the number of inputs (fan-in) of the unit which the connection belongs to.

$\rightarrow$ Keep the weighted sum in the same range for each unit.

Gradient descent

$$\theta_j^{(k+1)} = \theta_j^{(k)} - \frac{\eta}{\mu + h_{jj}} \frac{\partial E_i}{\partial \theta_j},$$

with $\mu = 0.02$.

Optimization lasts 20 epochs:

- $\eta = 0.0005$ for the first two epochs,
- $\eta = 0.0002$ for the next three epochs,
- $\eta = 0.0001$ for the next three epochs,
- $\eta = 0.00005$ for the next four epochs,
- $\eta = 0.00001$ for the remaining epochs,

Results

The 82 patterns misclassified by LeNet5. Below each image is displayed the correct answer (left) and the prediction (right). These errors are mostly caused by genuinely ambiguous patterns, or by digits written in a style that are under represented in the training set.

Outline

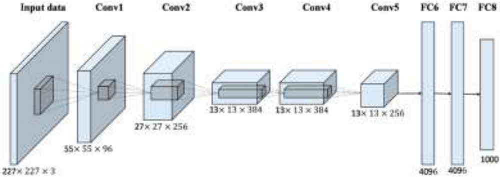
1. Foundations of CNN
   - Convolution layer
   - Pooling layer
   - Data preprocessing
2. Famous CNN
   - LeNet (1998)
   - AlexNet (2012)
   - ZFNet (2013)
   - VGGNet (2014)
   - GoogLeNet (2014)
   - ResNet (2016)
   - DenseNet (2017)
   - Many other CNN
3. Applications
   - Image classification
   - Pose, action detection
   - Object detection
   - Scene labeling - Semantic segmentation
   - Object tracking - videos
   - Text detection and recognition

AlexNet

["Imagenet classification with deep convolutional neural networks", Krizhevsky et al. 2012]

Ingredients:

- Activation function (ReLU)
- Local Response Normalization (LRN)
- Overlapping pooling
- Dropout
- Data augmentation



ReLU activation function

According to Krizhevsky et al. 2012, Convolutional neural networks with ReLU activation functions can be trained several times faster than the same networks using tanh function.

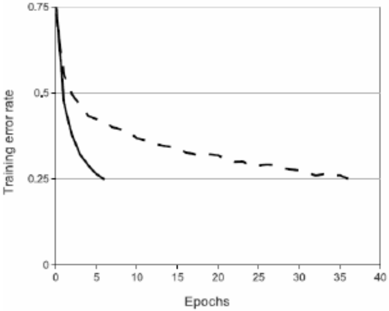


Figure: A four-layer convolutional neural network with ReLU (solid line) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh (dashed line). The learning rates for each network were chosen independently to make training as fast as possible.

Local Response Normalization/ Brightness normalization

Let $a^i_{x,y}$ the activity of a neuron resulting of kernel i applied to the position (x, y) followed by a ReLU function and $b^i_{x,y}$ the corresponding renormalized activity which is given by

$$b^i_{x,y} = a^i_{x,y} \left(C + \alpha \sum_{j=\max(0,i-q/2)}^{\min(Q-1,i+q/2)} (a^j_{x,y})^2 \right)^{-\beta},$$

where the sum is taken over q adjacent feature maps at the same spatial position, and Q is the total number of feature maps in this layer.

Constants (determined with validation set): $C = 2, q = 5, \alpha = 10^{-4}, \beta = 0.75$.

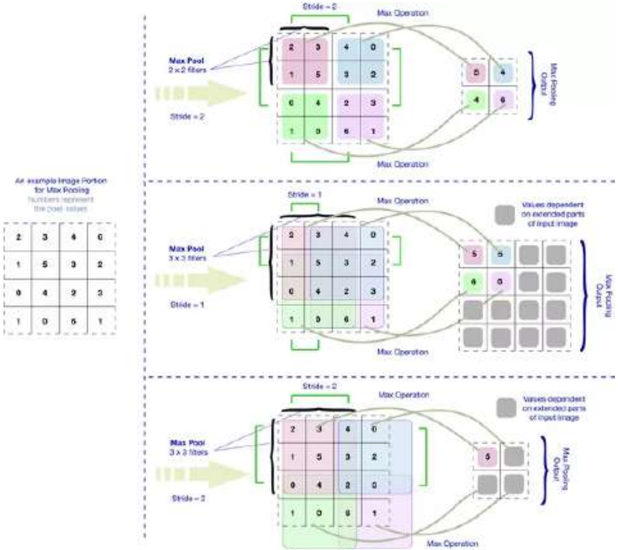
Note that the ordering of feature maps is arbitrary and determined before training. This renormalization creates a competition between the different feature maps.

["What is the best multi-stage architecture for object recognition?", Jarrett et al. 2009]

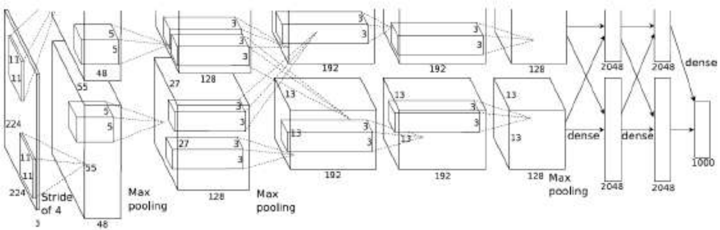
They propose a similar normalization procedure where the mean activity is substracted (local contrast normalization).

Overlapping pooling

Usually receptive fields in pooling layers do not overlap. Here, they use a grid of size 3×3 with a stride $s = 2$. Resulting network is slightly less prone to overfitting.



Overall architecture



Key-point: architecture is split across two GPU, which, most of the time, do not communicate with each other.

- Connectivity of each convolutional layer
- ReLu are applied right after all convolutional layers and fully connected layers
- Local Response Normalization is applied after ReLU in the first and second convolutional layer
- Max-pooling is applied after the first, second and fifth convolutional layers.

Optimization

Initialization:

- Weights: $\mathcal{N}(0, 0.0001)$
- Biases of second, fourth and fifth convolutional layers and biases of fully connected layers set to 1 (seems to accelerate the early stages of learning, prevent dying ReLU phenomenon). Other biases are set to 0.

Stochastic gradient descent with momentum

$$v^{(k+1)} = 0.9v^{(k)} - 0.0005\eta\theta^{(k)} - \frac{\eta}{B}\sum_{i\in\mathcal{B}} \nabla \ell_i(\theta^{(k)})$$

$$\theta^{(k+1)} = \theta^{(k)} + v^{(k+1)},$$

with batch size $|\mathcal{B}| = B = 128$.

The second term in the first equation corresponds to the L_2 regularization of the losswith a constant $\lambda = 0.0005$ (weight decay of 0.0005).

Learning rate is the same for all layers with the following heuristic:

- Initialization: $\eta = 0.01$
- Divide η by 10 when the validation error stop improving (done three times here).
- 90 epochs on 1.2 million images: 6 days.

Numerical results

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs[7]	–	–	26.2%
1CNN	40.7%	18.2%	–
5CNNs	38.1%	16.4%	16.4%
1CNN*	39.0%	16.6%	–
7CNNs*	36.7%	15.4%	15.3%

- First line is the second runner-up.
- Second and third lines are results output by the averaging over 1 or 5 CNN described before.
- Last two lines correspond to networks with an extra convolutional layer after the last pooling layer which has been trained on Image Net Fall 2011 then "fine-tuned" on the ImageNet 2012 data base.

AlexNet has a very similar architecture to LeNet, but is deeper, bigger, and features Convolutional Layers stacked on top of each other: previously, pooling layers followed immediately each convolutional layer.

Results

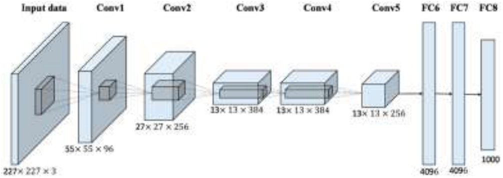
Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

Outline

1. Foundations of CNN
   - Convolution layer
   - Pooling layer
   - Data preprocessing
2. Famous CNN
   - LeNet (1998)
   - AlexNet (2012)
   - ZFNet (2013)
   - VGGNet (2014)
   - GoogLeNet (2014)
   - ResNet (2016)
   - DenseNet (2017)
   - Many other CNN
3. Applications
   - Image classification
   - Pose, action detection
   - Object detection
   - Scene labeling - Semantic segmentation
   - Object tracking - videos
   - Text detection and recognition

ZFNet: Improve upon AlexNet

["Visualizing and understanding convolutional networks", Zeiler and Fergus 2014]



Aim at finding out what the different feature maps are searching for in order to obtain a better tuning of network architecture.

In ZFNet, feature maps are not divided across two different GPU. Thus connections between layers are less sparse than for AlexNet.

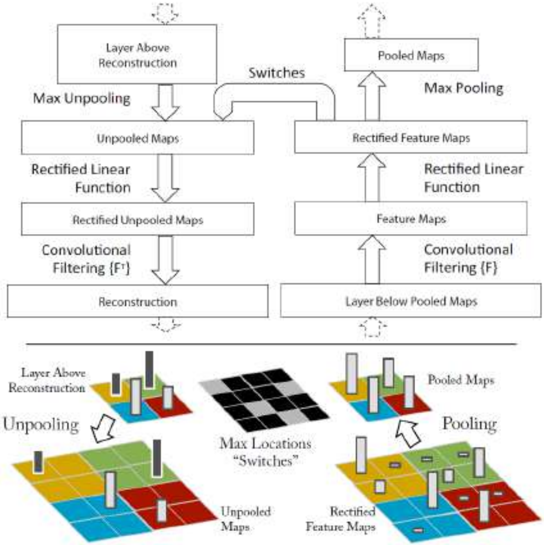
Deconvnet

Find the pixels that maximize the activation of a given feature map.

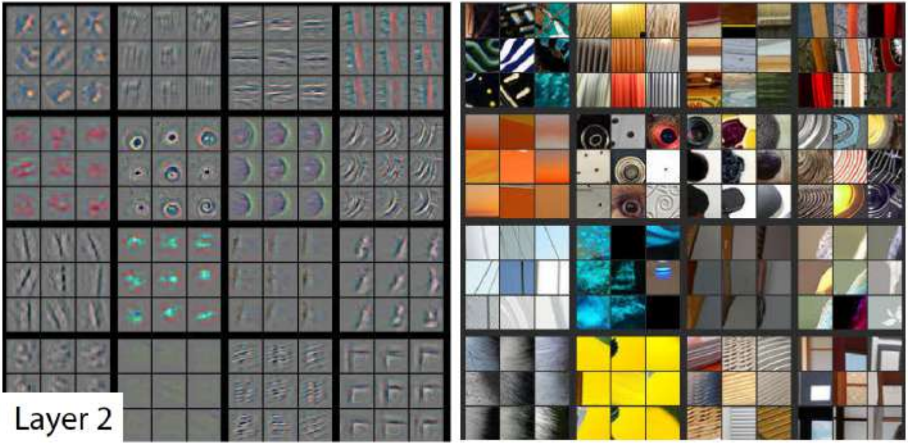
How? Invert the network.

Precisely:

- Choose a layer
- Choose a feature map
- Run the network on a validation set
- Choose the image maximizing the activation of this feature map
- "Backpropagate" this activation to obtain a stylized image in the pixel space

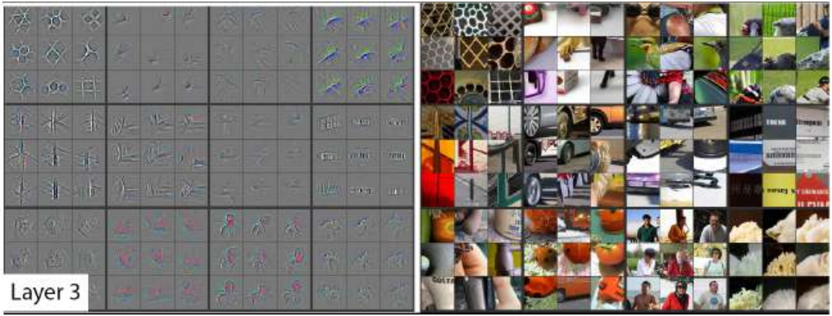


Results

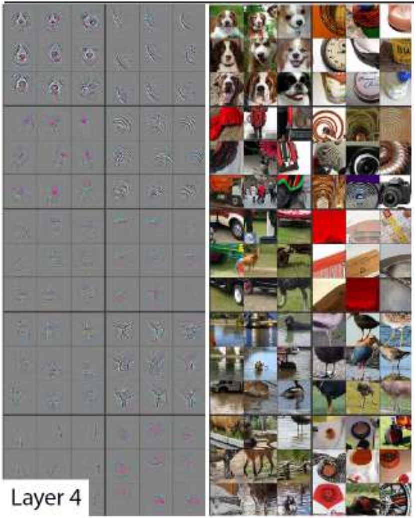


Top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using the previous deconvolutional network approach.

Results



Results

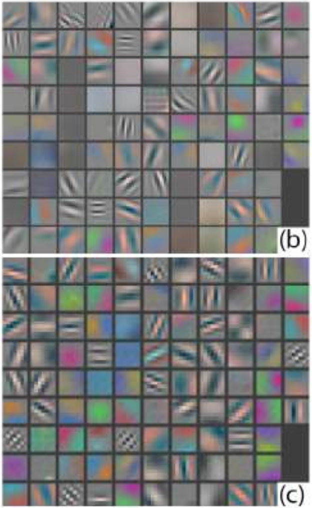


Results

Remarks

- strong grouping within each feature map,
- greater invariance at higher layers
- exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1).

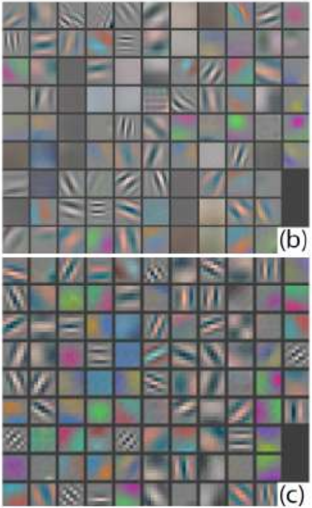
Visualization of previous modifications



(b): 1st layer features from Krizhevsky et al. 2012.

(c): 1st layer features of ZFNet.

Visualization of previous modifications



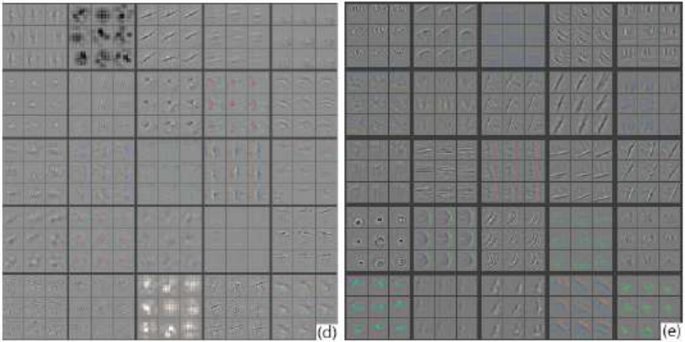
(b): 1st layer features from Krizhevsky et al. 2012.

(c): 1st layer features of ZFNet.

Differences: smaller stride (2 vs 4) and filter size (7x7 vs 11x11)

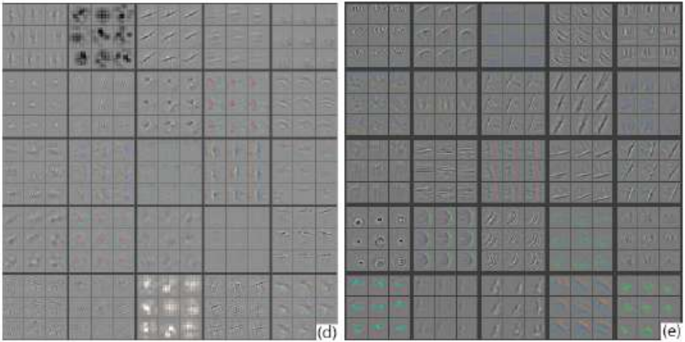
Results in more distinctive features and fewer dead features.

Visualization of previous modifications



(d): Visualizations of 2nd layer features from Krizhevsky et al. 2012; (e): Visualizations of the 2nd layer features of ZFNet.

Visualization of previous modifications



(d): Visualizations of 2nd layer features from Krizhevsky et al. 2012; (e): Visualizations of the 2nd layer features of ZFNet.

Feature maps in (e) are cleaner, with no aliasing artefacts that are visible in (d).

Conclusion regarding AlexNet

- First layer filters are a mix of high and low frequency information, with little coverage of middle frequencies.

  $\rightarrow$ Reduced the first layer filter size from 11×11 to 7×7.

- Aliasing artifacts are present in second layer because of the large stride of 4 used in the first convolutional layer.

  $\rightarrow$ change the stride from 4 to 2.

With these modifications:

- Winner of the ILSVRC 2013
- Improvement on AlexNet by
  - expanding the size of the middle convolutional layers
  - making the stride and filter size on the first layer smaller.

ZF Net final structure

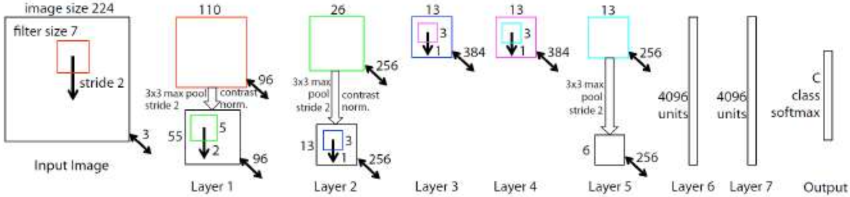
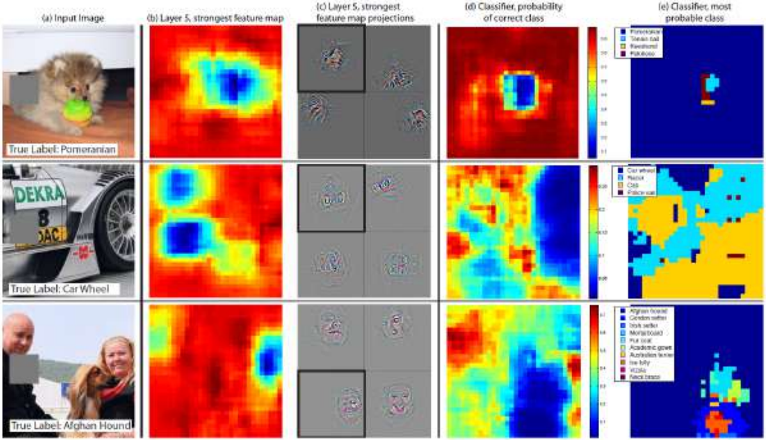


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form ($6 \cdot 6 \cdot 256 = 9216$ dimensions). The final layer is a C-way softmax function, C being the number of classes. All filters and feature maps are square in shape.

Results in classification

Error %	Val Top-1	Val Top-5	Test Top-5
(Gunji et al., 2012)	–	–	26.2
(Krizhevsky et al., 2012), 1 convnet	40.7	18.2	––
(Krizhevsky et al., 2012), 5 convnets	38.1	16.4	16.4
(Krizhevsky et al., 2012)*, 1 convnets	39.0	16.6	––
(Krizhevsky et al., 2012)*, 7 convnets	36.7	15.4	15.3
Our replication of (Krizhevsky et al., 2012), 1 convnet	40.5	18.1	––
1 convnet as per Fig. 3	38.4	16.5	––
5 convnets as per Fig. 3 - (a)	36.7	15.3	15.3
1 convnet as per Fig. 3 but with layers 3, 4, 5: 512,1024,512 maps - (b)	37.5	16.0	16.1
6 convnets, (a) & (b) combined	**36.0**	**14.7**	**14.8**

Occlusion



Three test examples where we systematically cover up different portions of the scene with a gray square (1st column) and see how the top (layer 5) feature maps ((b) & (c)) and classifier output ((d) & (e)) changes.

(b): for each position of the gray scale, we record the total activation in one layer 5 feature map (the one with the strongest response in the unoccluded image).

(c): a visualization of this feature map projected down into the input image (black square), along with visualizations of this map from other images. The first row example shows the strongest feature to be the dog's face. When this is covered-up the activity in the feature map decreases (blue area in (b)).

(d): a map of correct class probability, as a function of the position of the gray square. E.g. when the dog's face is obscured, the probability for pomeranian drops significantly.

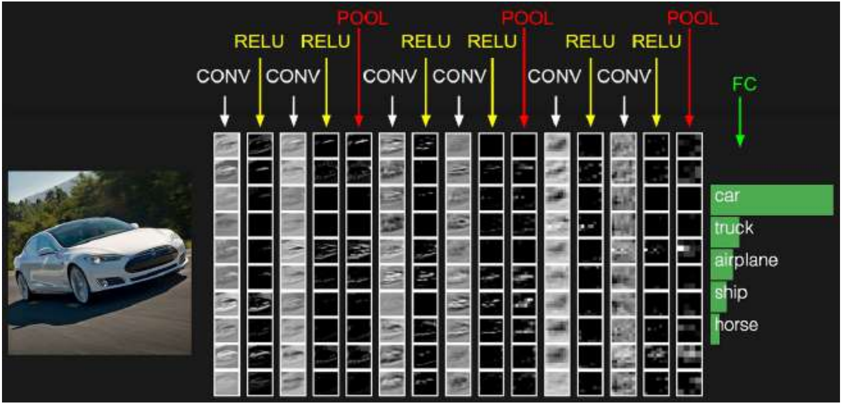
(e): the most probable label as a function of occluder position.

Outline

1. Foundations of CNN
   - Convolution layer
   - Pooling layer
   - Data preprocessing
2. Famous CNN
   - LeNet (1998)
   - AlexNet (2012)
   - ZFNet (2013)
   - VGGNet (2014)
   - GoogLeNet (2014)
   - ResNet (2016)
   - DenseNet (2017)
   - Many other CNN
3. Applications
   - Image classification
   - Pose, action detection
   - Object detection
   - Scene labeling - Semantic segmentation
   - Object tracking - videos
   - Text detection and recognition

Tiny VGGnet

["Very deep convolutional networks for large-scale image recognition", Simonyan and Zisserman 2014b]



Network features

Convolutional layers:

- Small receptive field: 3×3 (smallest ones capable of capturing the notion of top/down, left/right!)
- Stride of 1
- Spatial resolution is preserved after convolution

Max-pooling layers:

- 2×2 kernel
- Stride of 2

All hidden layers use ReLU activation functions.

Local Response Normalization layers do not improve performance.

Insightful remark...

If you stack 3 convolutional layers with receptive fields 3×3, you obtain a convolutional layer with receptive fields 7×7. What is the interest?

1. Stack of 3 convolutional layers of size 3×3: complexity of $3(3^2 C^2) = 27C^2$, where C is the number of channels.
2. One standard convolutional layer of size 7×7: complexity of $49C^2$.

In the first case, we cannot obtain every possible layer: the resulting object is a decomposition of three consecutive convolutional layers. There are less possibilities hence less parameters.

VGGNet

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

Parameters

Initialization:

- Network A: $\mathcal{N}(0, 0.01)$ for weights and 0 for biases.
- For other networks: first four conv layers and last three fully connected layers were initialized using network A and the remaining layers were initialized randomly.

Stochastic gradient descent with momentum

$$v^{(k+1)} = 0.9 v^{(k)} - 0.0005 \eta \theta^{(k)} - \eta \frac{1}{B} \sum_{i \in \mathcal{B}} \nabla L_i(\theta^{(k)})$$

$$\theta^{(k+1)} = \theta^{(k)} + v^{(k+1)},$$

with batch size $B = 128$.

Learning rate is the same for all layers with the following heuristic:

- Initialization: $\eta = 0.01$
- Divide η by 10 when the validation error stop improving (done three times here).
- 74 epochs.

- L_2 penalty with constant 5.10^{-4}
- Dropout regularization for the first two fully connected layers (probability $p = 0.5$)

Results

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	**23.7**	**6.8**	**6.8**
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-	7.9	
GoogLeNet (Szegedy et al., 2014) (7 nets)	-	**6.7**	
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

A downside of the VGGNet is that it is more expensive to evaluate and uses a lot more memory and parameters (140M).
Most of these parameters are in the first fully connected layer, and it was since found that these FC layers can be removed with no performance downgrade, significantly reducing the number of necessary parameters.

Results

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	23.7	6.8	6.8
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	–	7.9	
GoogLeNet (Szegedy et al., 2014)(7 nets)	–	6.7	
MSRA (He et al, 2014)(11 nets)	–	–	8.1
MSRA (He et al., 2014)(1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	–	–	11.7
Clarifai (Russakovsky et al., 2014)(1 net)	–	–	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013)(1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al, 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al, 2014) (1 net)	35.7	14.2	–
Krizhevsky et al. (Krizhevsky et al., 2012)(5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	–

A downside of the VGGNet is that it is more expensive to evaluate and uses a lot more memory and parameters (140M).
Most of these parameters are in the first fully connected layer, and it was since found that these FC layers can be removed with no performance downgrade, significantly reducing the number of necessary parameters.

Outline

1. Foundations of CNN
   - Convolution layer
   - Pooling layer
   - Data preprocessing
2. Famous CNN
   - LeNet (1998)
   - AlexNet (2012)
   - ZFNet (2013)
   - VGGNet (2014)
   - GoogLeNet (2014)
   - ResNet (2016)
   - DenseNet (2017)
   - Many other CNN
3. Applications
   - Image classification
   - Pose, action detection
   - Object detection
   - Scene labeling - Semantic segmentation
   - Object tracking - videos
   - Text detection and recognition

GoogLeNet

["Going deeper with convolutions", Szegedy, W. Liu, et al. 2015]

Aim.

Increasing the depth and width of state-of-the-art convolutional neural networks while keeping the number of parameters small:

- Can approximate more complex functions
- while being robust to overfitting and computationally appealing.

How.

Specifically, use of 1×1 convolution layers to reduce the number of parameters + apply filters of different sizes 3×3, 5×5 or 3×3 max pooling (on each feature maps).

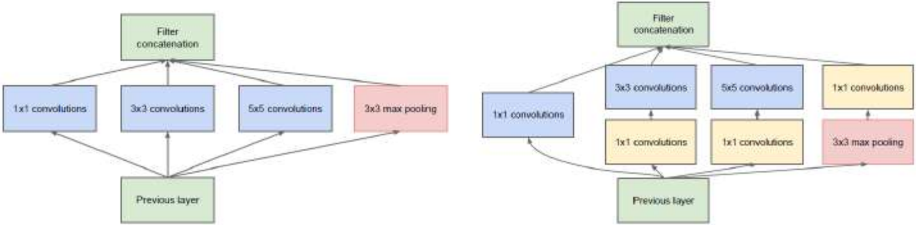
Details.

- All convolution layers use ReLU activation functions.
- Same spatial resolution for each feature map.

GoogLeNet - Inception module

Same spatial resolution for each feature map.

Use of 1×1 convolution layers to reduce the number of parameters then apply filters of different sizes 3×3, 5×5 or 3×3 max pooling (on each feature maps).



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2: Inception module

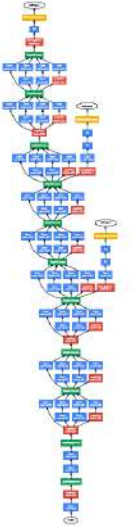
GoogLeNet - Inception module

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

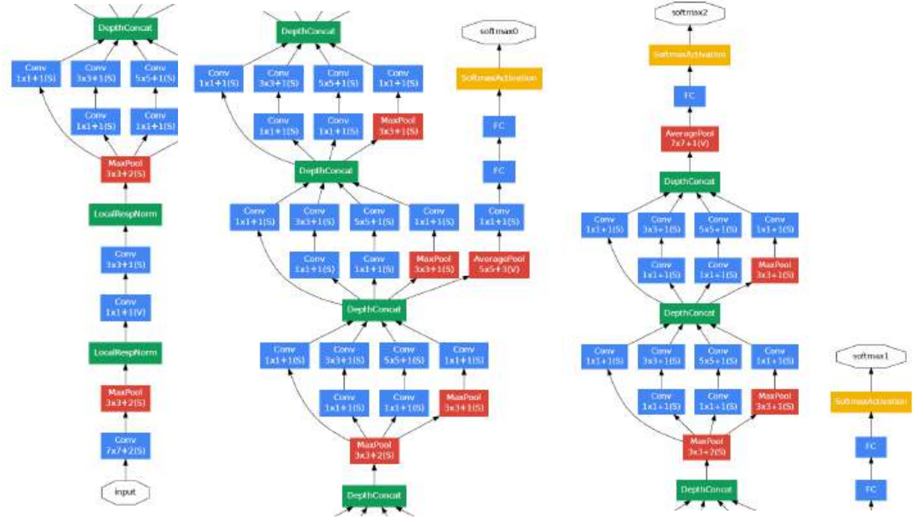
Table 1: GoogLeNet incarnation of the Inception architecture

"3x3 reduce" and "5x5 reduce" stands for the number of 1x1 filters in the reduction layer used before the 3x3 and 5x5 convolutions. One can see the number of 1x1 filters in the projection layer after the built-in max-pooling in the pool proj column.

Structure of GoogLeNet



Structure of GoogLeNet



Deep network - A concern

In order to backpropagate gradient, the authors add some auxiliary classifiers connected to intermediate layers.

During training the loss of auxiliary classifiers is weighted by 0.3 and added to the total loss of the network. Auxiliary networks are removed at inference time.

Auxiliary network put after $(4a)$ and $(4d)$:

- Average pooling layer 5×5, stride of 3
- A 1×1 convolution with 128 filters, with ReLU.
- A fully connected layer with 1024 neurons and ReLU
- A dropout layer with a dropout ratio of 70%.
- A linear layer with softmax loss, predicting the same 1000 classes as the main classifier.

Parameters

Initialization:

- Weights are drawn from $\mathcal{N}(0,1)$ and biases are set to 0.

["Deep learning via Hessian-free optimization.", Martens 2010]

Stochastic gradient descent with momentum

$$v^{(k+1)} = \mu v^{(k)} - \eta \frac{1}{B} \sum_{i \in \mathcal{B}} \nabla \ell_i(\theta^{(k)} + \mu v^{(k)})$$

$$\theta^{(k+1)} = \theta^{(k)} + v^{(k+1)},$$

with batch size $B = 200$, where

$$\mu^{(k)} = \min(1 - 2^{-1-\log_2(\lfloor k/250 \rfloor + 1)}, \mu_{max}),$$

where $\mu_{max} \in \{0, 0.9, 0.99, 0.995, 0.999\}$.

Learning rate is the same for all layers with the following heuristic:

- Initialization: $\eta = 0.01$
- Multiply η by 0.96 every 8 epochs.
- Training lasts 125 epochs.

Results

- Polyak averaging is used to create the final model at inference time.
- 7 different versions of GoogleNet were trained and aggregated to make predictions.

Team	Year	Place	Error (top-5)	Uses external data
SuperVision	2012	1st	16.4%	no
SuperVision	2012	1st	15.3%	Imagenet 22k
Clarifai	2013	1st	11.7%	no
Clarifai	2013	1st	11.2%	Imagenet 22k
MSRA	2014	3rd	7.35%	no
VGG	2014	2nd	7.32%	no
GoogLeNet	2014	1st	6.67%	no

Table 2: Classification performance

Number of models	Number of Crops	Cost	Top-5 error	compared to base
1	1	1	10.07%	base
1	10	10	9.15%	-0.92%
1	144	144	7.89%	-2.18%
7	1	7	8.09%	-1.98%
7	10	70	7.62%	-2.45%
7	144	1008	6.67%	-3.45%

Main contribution: development of an Inception Module that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).

Outline

1 Foundations of CNN
- Convolution layer
- Pooling layer
- Data preprocessing

2 Famous CNN
- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2016)
- DenseNet (2017)
- Many other CNN

3 Applications
- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

ResNet (2016)

["Deep residual learning for image recognition", He et al. 2016]

Statement: Optimization can be hard for some deep networks.

Solution: Ease optimization by adding simple paths in the network

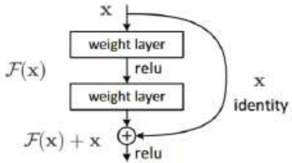


Figure 2. Residual learning: a building block.

$\rightarrow$ No extra parameters, no additional computational complexity

Literature on shortcut connections

Early practice for training multi-layer perceptrons was to add a linear layer between the inputs and the outputs

[*Pattern recognition and neural networks*, Ripley 2007]

Few intermediate classifiers can also be added in intermediary levels in order to ease the optimization:

- [“Going deeper with convolutions”, Szegedy, W. Liu, et al. 2015]
- [“Deeply-supervised nets”, Lee et al. 2015]

Highway networks have shortcut connections with gating functions. Here, gates are data dependent and have parameters.

- [“Highway networks”, Rupesh Kumar Srivastava et al. 2015]
- [“Training very deep networks”, Rupesh K Srivastava et al. 2015]

General Idea

Inspired from VGG nets:

- For the same output feature map size, the layers have the same numbers of filters
- If the feature map size is halved, then the number of filters is doubled to preserve the time complexity per layer

$$\mathbf{y} = f(\mathbf{x}, \mathbf{W}_i) + \mathbf{x},$$

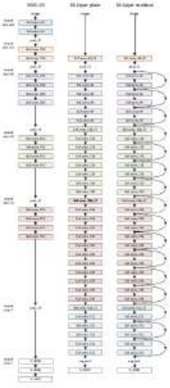
where $\mathbf{x}$ and $\mathbf{y}$ are respectively the input and the output of a (stack of) layer(s), $\mathbf{W}_i$ are the weights of this/these layer(s) and $f(\mathbf{x}, \mathbf{W}_i)$ the output of this/these layer(s).

If dimensions do not match between $\mathbf{x}$ and $\mathbf{y}$, there are two solutions:

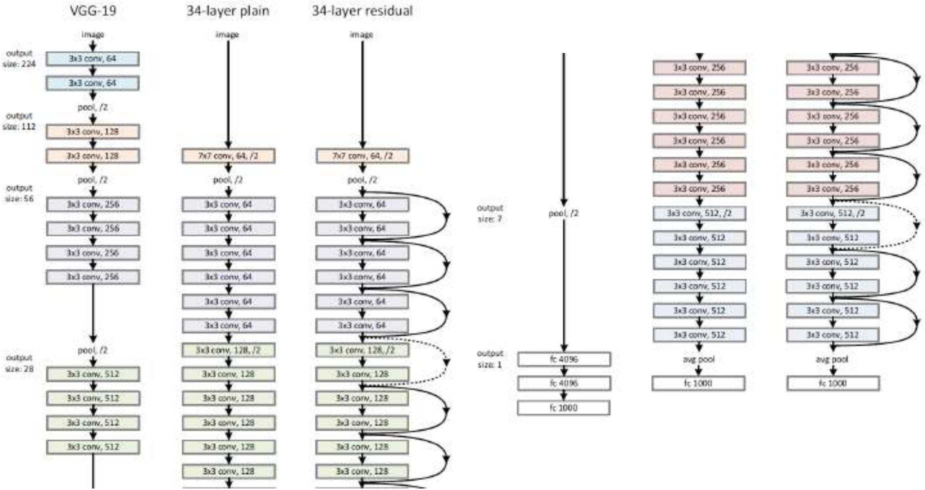
- identity mapping is coupled with extra zero entries padded for increasing dimensions
- Projection shortcut is used to match dimensions via 1×1 convolution filters
$$\mathbf{y} = f(\mathbf{x}, \mathbf{W}_i) + W_s\mathbf{x},$$
where W_s is a projection.

Besides, "when the shortcuts go across feature maps of two different sizes, they are performed with a stride of 2".

Structure of ResNet



Structure of ResNet



Parameters

Initialization, as in He et al. 2015: weights are drawn from $\mathcal{N}(0, 2/n_L)$ (n_L is the number of neurons in the previous layer); biases are set to 0.

Stochastic gradient descent with momentum

$$v^{(k+1)} = 0.9v^{(k)} - 0.0001\eta\theta^{(k)} - \eta\frac{1}{B}\sum_{i\in\mathcal{B}} \nabla L_i(\theta^{(k)})$$

$$\theta^{(k+1)} = \theta^{(k)} + v^{(k+1)},$$

with batch size $B = 256$.

Learning rate is the same for all layers with the following heuristic:

- Initialization: $\eta = 0.1$
- Divide η by 10 when the validation error stop improving (done three times here).
- 120 epochs.

Miscellaneous:

- Batch normalization after each convolution and before activation
- No dropout

Results

method	top-5 err. (test)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

- Winner of ILSVRC 2015
- Special skip connections and heavy use of batch normalization
- No fully connected layers at the end of the network.

Outline

1. Foundations of CNN
   - Convolution layer
   - Pooling layer
   - Data preprocessing
2. Famous CNN
   - LeNet (1998)
   - AlexNet (2012)
   - ZFNet (2013)
   - VGGNet (2014)
   - GoogLeNet (2014)
   - ResNet (2016)
   - DenseNet (2017)
   - Many other CNN
3. Applications
   - Image classification
   - Pose, action detection
   - Object detection
   - Scene labeling - Semantic segmentation
   - Object tracking - videos
   - Text detection and recognition

DenseNet

["Densely Connected Convolutional Networks.", G. Huang et al. 2017]

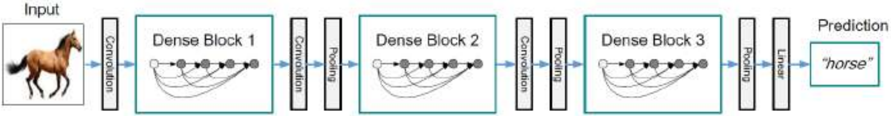


Figure: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling

DenseNet

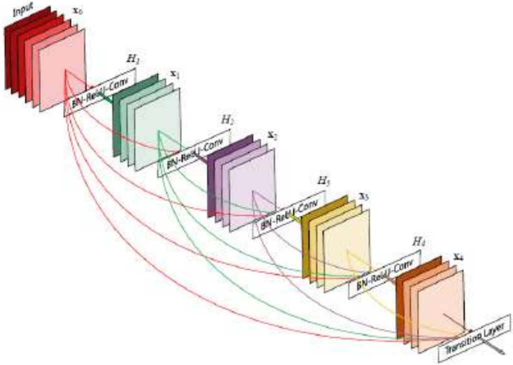


Figure: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Ingredients

Let $\mathbf{x}_\ell$ be the input of the ℓth layer. Usually,

$$\mathbf{x}_\ell = f_\ell(\mathbf{x}_{\ell-1}).$$

Dense Block. Inside a dense block,

$$\mathbf{x}_\ell = f_\ell(\mathbf{x}_0, \ldots, \mathbf{x}_{\ell-1}).$$

The functions f_ℓ are composed of three consecutive operations:

1. First, a batch normalization
2. Then, activation function ReLU
3. Finally, 3×3 convolutional layer (feature map sizes are kept fixed)

Between dense blocks.

1. Batch normalization
2. 1×1 convolution
3. 2×2 average pooling

Ingredients

Growth rate k
If each function f_ℓ produces k feature maps, the inputs of the ℓth layer has $k_0 + k(\ell - 1)$ channels. Narrow layers (typically $k = 12$) give good results.
$\rightarrow$ Indeed, each layer has access to each previous layer and thus to the "collective knowledge" of the network.

Bottleneck layer - DenseNet-B
A way to improve computational efficiency is to introduce 1×1 convolutional layers: inside dense block, for each layer

$$\text{BN - ReLU - Conv } (1 \times 1) \text{ - BN - ReLU - Conv } (3 \times 3)$$

Conv 1×1 are set to produce $4k$ feature maps.

Compression layer - DenseNet-C
Throw away a fraction $\theta \in [0, 1]$ (typically $\theta = 0.5$) of feature maps at transition layers.

Architecture

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Parameters

Initialization, as in He et al. 2015: weights are drawn from $\mathcal{N}(0, 2/n_L)$ (n_L is the number of neurons in the previous layer); biases are set to 0.

Stochastic gradient descent with momentum

$$v^{(k+1)} = 0.9v^{(k)} - 0.0001\eta\theta^{(k)} - \eta\frac{1}{B}\sum_{i\in\mathcal{B}}\nabla L_i(\theta^{(k)})$$

$$\theta^{(k+1)} = \theta^{(k)} + v^{(k+1)},$$

with batch size $B = 256$.

Learning rate is the same for all layers with the following heuristic:

- Initialization: $\eta = 0.1$
- Divide η by 10 at epoch 30 and 60.
- 90 epochs.

Miscellaneous:

- Batch normalization after each convolution and before activation
- No dropout

DenseNet Results

Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [32]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [34]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**	-

Table 2: Error rates (%) on CIFAR and SVHN datasets. k denotes network's growth rate. Results that surpass all competing methods are **bold** and the overall best results are **blue**. "+" indicates standard data augmentation (translation and/or mirroring). $*$ indicates results run by ourselves. All the results of DenseNets without data augmentation (C10, C100, SVHN) are obtained using Dropout. DenseNets achieve lower error rates while using fewer parameters than ResNet. Without data augmentation, DenseNet performs better by a large margin.

Outline

1 Foundations of CNN
- Convolution layer
- Pooling layer
- Data preprocessing

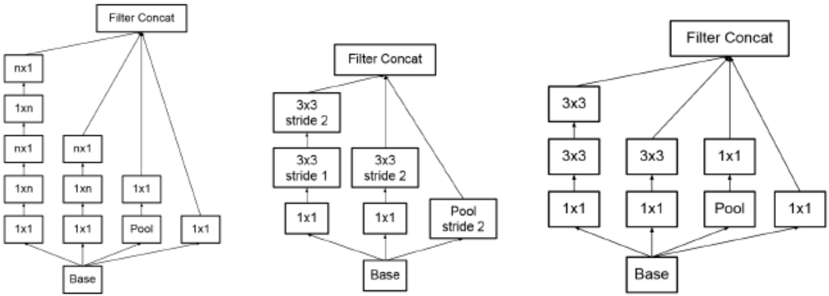
2 Famous CNN
- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2016)
- DenseNet (2017)
- Many other CNN

3 Applications
- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

Inception V2-V3

Based on GoogLeNet Inception module

["Rethinking the inception architecture for computer vision", Szegedy, Vanhoucke, et al. 2016]



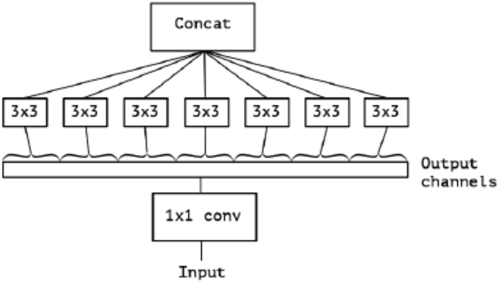
New ideas:

- Using asymmetric convolutions $1 \times n$ and $n \times 1$ (for $n = 3, 5, 7$) can be useful in the middle layers of the networks for feature maps of size $m \times m$ (for $12 \leq m \leq 20$).
- Label smoothing using a uniform distribution over labels

Xception

["Xception: Deep learning with depthwise separable convolutions", Chollet 2017]

Stands for "Extreme Inception" and builds upon Inception module in GoogLeNet.

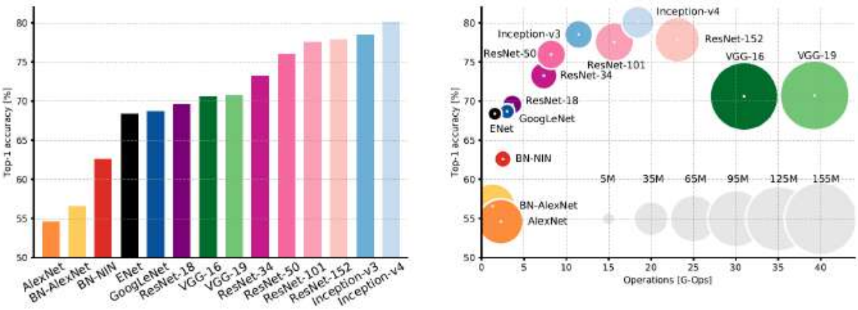


The main ideas:

- Perform 1×1 convolutions
- Apply 3×3 (or other filter size) convolutions to each previous feature map (the one created by 1×1 convolutions) separately.

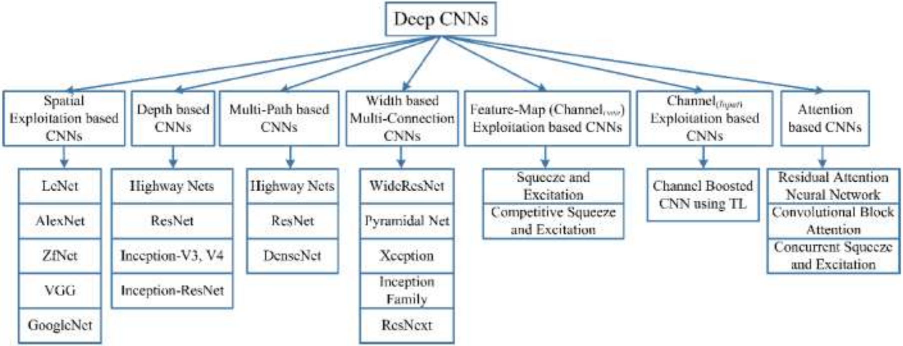
$\rightarrow$ Decoupled the depth (1×1 convolutions) and the spatial transformations (convolutions on each feature map separately).

Comparison of several CNN



["An analysis of deep neural network models for practical applications", Canziani et al. 2016]

CNN Taxonomy



See this very detailed review paper ["A survey of the recent architectures of deep convolutional neural networks", Khan et al. 2020]

Outline

1. Foundations of CNN
   - Convolution layer
   - Pooling layer
   - Data preprocessing
2. Famous CNN
   - LeNet (1998)
   - AlexNet (2012)
   - ZFNet (2013)
   - VGGNet (2014)
   - GoogLeNet (2014)
   - ResNet (2016)
   - DenseNet (2017)
   - Many other CNN
3. Applications
   - Image classification
   - Pose, action detection
   - Object detection
   - Scene labeling - Semantic segmentation
   - Object tracking - videos
   - Text detection and recognition

Applications

This section is based on ["Recent advances in convolutional neural networks", Gu et al. 2015].

More applications domain and more references are presented in this paper.

Outline

1 Foundations of CNN
- Convolution layer
- Pooling layer
- Data preprocessing

2 Famous CNN
- LeNet (1998)
- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2016)
- DenseNet (2017)
- Many other CNN

3 Applications
- Image classification
- Pose, action detection
- Object detection
- Scene labeling - Semantic segmentation
- Object tracking - videos
- Text detection and recognition

Image classification - Hierarchy of classifiers

["Error-driven incremental learning in deep convolutional neural network for large-scale image classification", Xiao et al. 2014]

$\rightarrow$ They propose a training method that grows a network not only incrementally but also hierarchically. In their method, classes are grouped according to similarities and are self-organized into different levels.

["HD-CNN: hierarchical deep convolutional neural networks for large scale visual recognition", Yan et al. 2015]

$\rightarrow$ They introduce a hierarchical deep CNNs (HD-CNNs) by embedding deep CNNs into a category hierarchy. They decompose the classification task into two steps. The coarse category CNN classifier is first used to separate easy classes from each other, and then those more challenging classes are routed downstream to fine category classifiers for further prediction. This architecture follows the coarse-to-fine classification paradigm and can achieve lower error at the cost of an affordable increase of complexity.

Image classification - CNN Tree

Z. Wang et al. 2018 build a tree of CNN to learn fine-grained features for subcategory recognition.

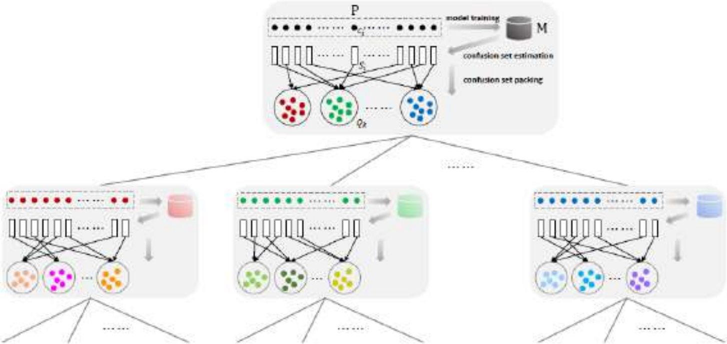


Image classification - CNN Tree

Figure: Confusion set outputs by AlexNet softmax prediction on validation set of ILSVRC 2015.

Image classification - CNN Tree

		AlexNet					
	#Basic Models	1	2	3	4	5	6
Top-1 errors	T_0	43.09%	41.28%	40.41%	40.21%	39.82%	39.63%
	T_1	40.68%(-2.41%)	38.95%(-2.33%)	38.07%(-2.34%)	37.80%(-2.41%)	37.49%(-2.33%)	37.39%(-2.24%)
	T_2	40.40%(-2.69%)	38.60%(-2.68%)	37.84%(-2.57%)	37.62%(-2.59%)	37.33%(-2.49%)	37.19%(-2.44%)
Top-5 errors	T_0	20.04%	18.53%	18.03%	17.72%	17.52%	17.43%
	T_1	18.58%(-1.46%)	17.52%(-1.01%)	16.93%(-1.10%)	16.59%(-1.13%)	16.39%(-1.13%)	16.24%(-1.19%)
	T_2	18.55%(-1.49%)	17.39%(-1.14%)	16.81%(-1.22%)	16.53%(-1.19%)	16.36%(-1.16%)	16.23%(-1.20%)

		GoogleNet					
	#Basic Models	1	2	3	4	5	6
Top-1 errors	T0	32.75%	30.96%	30.27%	29.89%	29.72%	29.56%
	T1	28.37%(-4.38%)	26.51%(-4.45%)	25.99%(-4.28%)	25.57%(-4.32%)	25.4%(-4.32%)	25.15%(-4.41%)
Top-5 erros	T0	12.00%	10.89	10.53%	10.32%	10.17%	10.08%
	T1	10.09%(-1.91%)	8.98%(-1.91%)	8.68%(-1.85%)	8.33%(-1.99%)	8.23%(-1.94%)	8.12%(-1.96%)

Image classification - CNN Tree

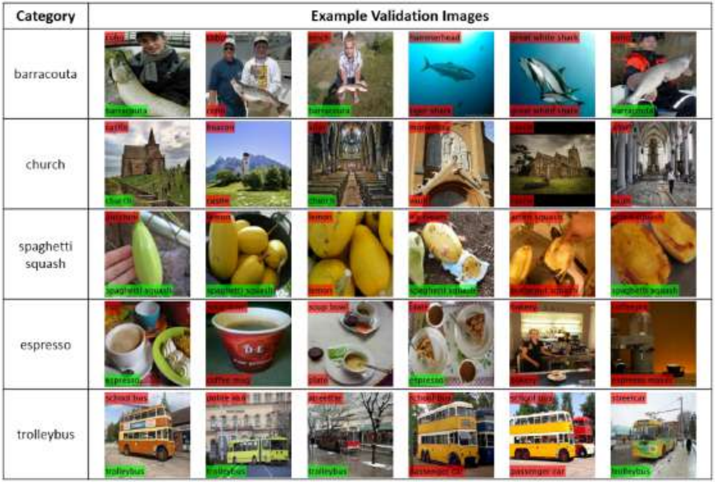


Figure: Top label is given by basic AlexNet CNN while bottom one is given by CNNTree (green color corresponds to a correct prediction)

Outline

1. Foundations of CNN
   - Convolution layer
   - Pooling layer
   - Data preprocessing
2. Famous CNN
   - LeNet (1998)
   - AlexNet (2012)
   - ZFNet (2013)
   - VGGNet (2014)
   - GoogLeNet (2014)
   - ResNet (2016)
   - DenseNet (2017)
   - Many other CNN
3. Applications
   - Image classification
   - Pose, action detection
   - Object detection
   - Scene labeling - Semantic segmentation
   - Object tracking - videos
   - Text detection and recognition

Pose estimation - Deeppose

["Deeppose: Human pose estimation via deep neural networks", Toshev and Szegedy 2014]

DeepPose is the first application of CNNs to human pose estimation problem. It captures the full context of each body joint by taking the whole image as the input.

Previous works:

- Limited expressiveness – the use of local detectors, which reason in many cases about a single part
- Modeling only a small subset of all interactions between body parts.

Pose estimation - Deeppose

Structure:

- Normalizing images
- Regression problem, i.e., prediction of k joints
$$\text{Image} \mapsto \mathbf{y} \in \mathbb{R}^{2k}.$$
- Use a cascade of 7 layers, each one taking a zoom of the previous image as input (refinement of the prediction at each stage).

Pose estimation - Deeppose

Figure 6. Predicted poses in red and ground truth poses in green for the first three stages of a cascade for three examples.

Pose estimation - Deeppose

Action recognition - images

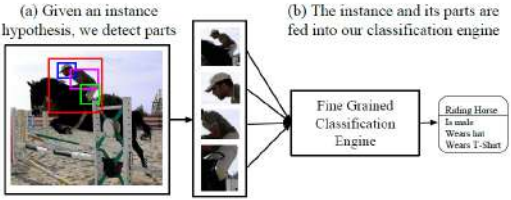
Action recognition aims at classifying human activities based on their visual appearance and motion dynamics.

In Simonyan and Zisserman 2014b (VGG),they use the outputs of the penultimate layer of a pre-trained CNN to represent full images of actions as well as the human bounding boxes inside them, and achieve a high level of performance in action classification.

Gkioxari et al. 2015 add a part detection to this framework. Their part detector is a CNN based extension to the original Poselet Pishchulin et al. 2013 method.

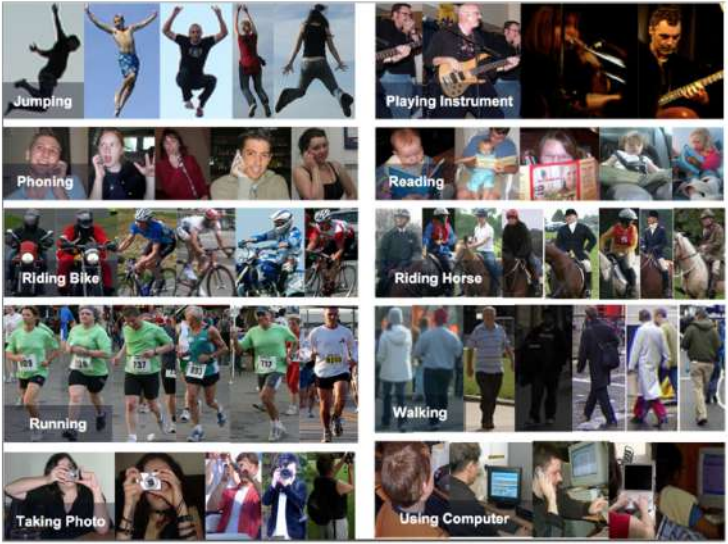
Action recognition

["Actions and attributes from wholes and parts", Gkioxari et al. 2015]



Given an R-CNN person detection (red box), they detect parts using a novel, deep version of poselets. The detected whole-person and part bounding boxes are input into a fine-grained classification engine to produce predictions for actions and attributes.

Action recognition

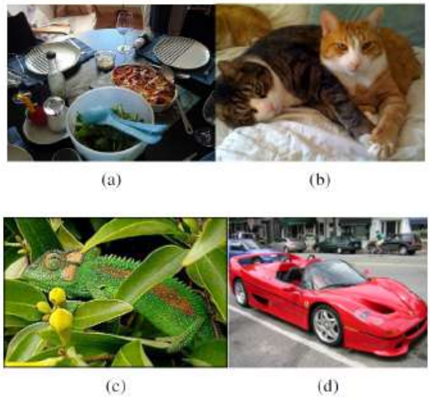


Outline

1. Foundations of CNN
   - Convolution layer
   - Pooling layer
   - Data preprocessing
2. Famous CNN
   - LeNet (1998)
   - AlexNet (2012)
   - ZFNet (2013)
   - VGGNet (2014)
   - GoogLeNet (2014)
   - ResNet (2016)
   - DenseNet (2017)
   - Many other CNN
3. Applications
   - Image classification
   - Pose, action detection
   - Object detection
   - Scene labeling - Semantic segmentation
   - Object tracking - videos
   - Text detection and recognition

Object detection - Exhaustive search vs segmentation

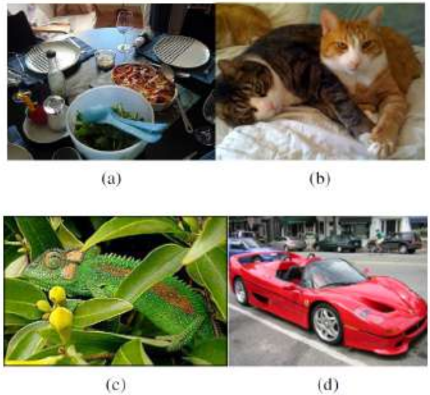
Segmentation: aims for a unique partitioning of the image through a generic algorithm, where there is one part for all object silhouettes in the image.



High variety of reasons that an image region forms an object:

Object detection - Exhaustive search vs segmentation

Segmentation: aims for a unique partitioning of the image through a generic algorithm, where there is one part for all object silhouettes in the image.



High variety of reasons that an image region forms an object:

(b) the cats can be distinguished by colour, not texture.

(c) the chameleon can be distinguished from the surrounding leaves by texture, not colour.

(d) the wheels can be part of the car because they are enclosed, not because they are similar in texture or colour.

(a) many different scales needed

→ Necessity to use a variety of diverse strategies.

Object detection - Exhaustive search vs segmentation

Alternative approach: do localisation through the identification of an object.

Exhaustive search: With an appearance model learned from examples, an exhaustive search is performed where every location within the image is examined as to not miss any potential object location.

Searching every possible location is computationally infeasible.

$\rightarrow$ restrictions need to be imposed: the classifier is simplified and the appearance model needs to be fast.

Selective search: data-driven selective search using bottom up grouping.

Object detection - Exhaustive search vs segmentation

Bottom-up grouping generates hierarchical nested partitioning of the input image.

["Mean shift: A robust approach toward feature space analysis"; "Efficient graph-based image segmentation", Comaniciu and Meer 2002; Felzenszwalb and Huttenlocher 2004]

(a) (b)

Object detection - Exhaustive search vs segmentation

Generic algorithm:

- They first use Felzenszwalb and Huttenlocher 2004 to create initial regions. This method is the fastest, publicly available algorithm that yields high quality starting locations.
- Then they use a greedy algorithm to iteratively group regions together
  - First the similarities between all neighbouring regions are calculated.
  - The two most similar regions are grouped together, and new similarities are calculated between the resulting region and its neighbours.
  - The process of grouping the most similar regions is repeated until the whole image becomes a single region.

Variety of partitionings:

- Different variant of input images
- Similarities based on color, texture, size, shared pixels

colour spaces	RGB	I	Lab	rgI	HSV	rgb	C	H
Light Intensity	-	-	+/-	2/3	2/3	+	+	+
Shadows/shading	-	-	+/-	2/3	2/3	+	+	+
Highlights	-	-	-	-	1/3	-	+/-	+

Object detection - naive approach

Generally, the difficulties mainly lie in how to accurately and efficiently localize objects in images or video frames.

In some early works by Vaillant et al. 1994; Nowlan and Platt 1995; Girshick, Iandola, et al. 2015, they use the sliding window based approaches to densely evaluate the CNN classifier on windows sampled at each location and scale. Since there are usually hundreds of thousands of candidate windows in a image, these methods suffer from highly computational cost, which makes them unsuitable to be applied on the large-scale dataset

More references on object proposal based methods:

["Human detection from images and videos: A survey", Nguyen et al. 2016]

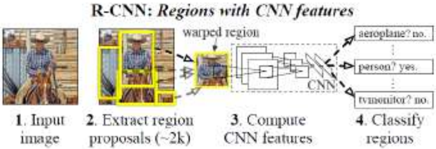
["Category-independent object proposals with diverse ranking", Endres and Hoiem 2014]

["Textproposals: a text-specific selective search algorithm for word spotting in the wild", Gómez and Karatzas 2017]

Object detection - R-CNN - Regions with CNN features

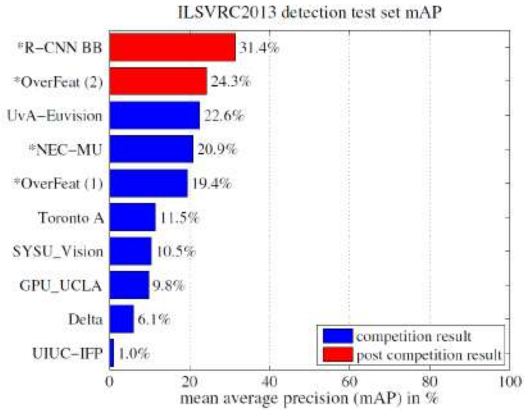
One of the most famous object proposal based CNN detector is Region-based CNN (R-CNN) by Girshick, Donahue, et al. 2014, aiming at

- localizing objects with a deep network
- training a high-capacity model with only a small quantity of annotated detection data



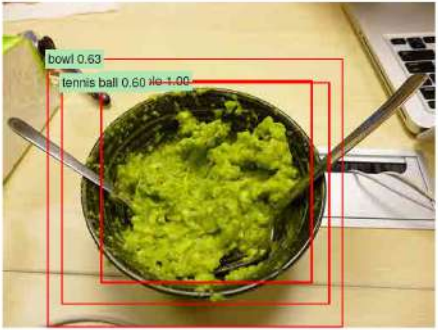
1. Generating category-independent region proposals via selective search.
2. Training large CNN that extracts a fixed-length feature vector from each region (Supervised pre-training on the large auxiliary dataset ILSVRC, followed by domainspecific fine-tuning on the small dataset PASCAL).
3. Learning a set of class- specific linear SVMs.

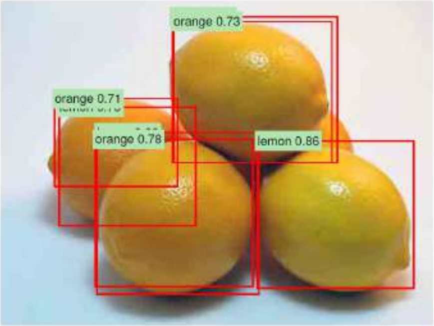
Object detection - R-CNN - Regions with CNN features



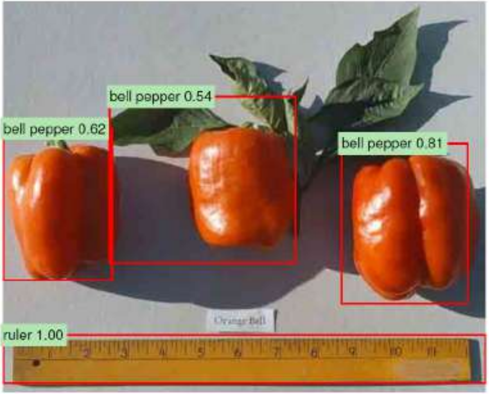
However, computational cost is high since the time-consuming CNN feature extractor will be performed for each region separately.

Object detection - R-CNN - Regions with CNN features





Object detection - R-CNN - Regions with CNN features



Object detection - improving R-CNN

["Spatial pyramid pooling in deep convolutional networks for visual recognition", He et al. 2014]

Spatial Pyramid Pooling network (SPP net) is a pyramid-based version of R-CNN, which introduces an SPP layer to relax the constraint that input images must have a fixed size. Unlike R-CNN, SPP net extracts the feature maps from the entire image only once, and then applies spatial pyramid pooling on each candidate window to get a fixed-length representation.

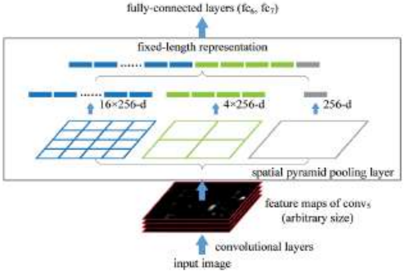


Figure 3: A network structure with a **spatial pyramid pooling layer**. Here 256 is the filter number of the conv_5 layer, and conv_5 is the last convolutional layer.

Object detection - improving R-CNN

Drawback: multi-stage pipeline $\Rightarrow$ CNN feature extractor and SVM classifier are impossible to train jointly.

["Faster r-cnn: Towards real-time object detection with region proposal networks", Ren et al. 2015]

Fast RCNN improves SPP net by using an end-to-end training method. All network layers can be updated during fine-tuning, which simplifies the learning process and improves detection accuracy.

["Attentionnet: Aggregating weak directions for accurate object detection", Yoo et al. 2015]

They treat the object detection problem as an iterative classification problem. It predicts an accurate object boundary box by aggregating quantized weak directions from their detection network.

Object detection - YOLO, SDD

More recently, YOLO Redmon et al. 2016 and SSD W. Liu et al. 2016 allow single pipeline detection that directly predicts class labels.

YOLO (You Only Look Once) treats object detection as a regression problem to spatially separated bounding boxes and associated class probabilities.

SDD (Single Shot Detector) discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. With this multiple scales setting and their matching strategy, SSD is significantly more accurate than YOLO.

With the benefits from super-resolution, Lu et al. 2016 propose a top-down search strategy to divide a window into sub-windows recursively, in which an additional network is trained to account for such division decisions.

YOLO

["You only look once: Unified, real-time object detection", Redmon et al. 2016]

The whole detection pipeline is a single network which predicts bounding boxes and class probabilities from the full image in one evaluation, and can be optimized end-to-end directly on detection performance.

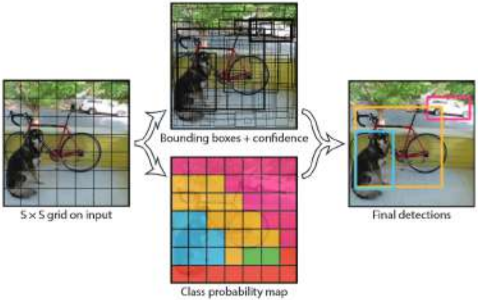


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

Drawback

Fails to detect small numerous objects.

YOLO

YOLO still lags behind state-of-the-art detection systems in accuracy. While it can quickly identify objects in images it struggles to precisely localize some objects, especially small ones.

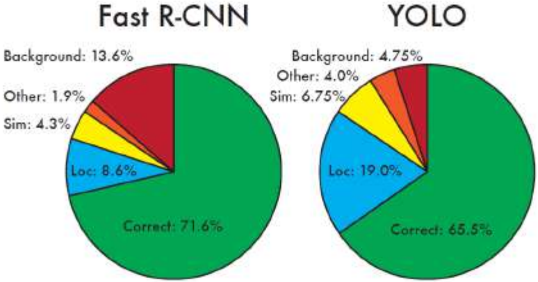
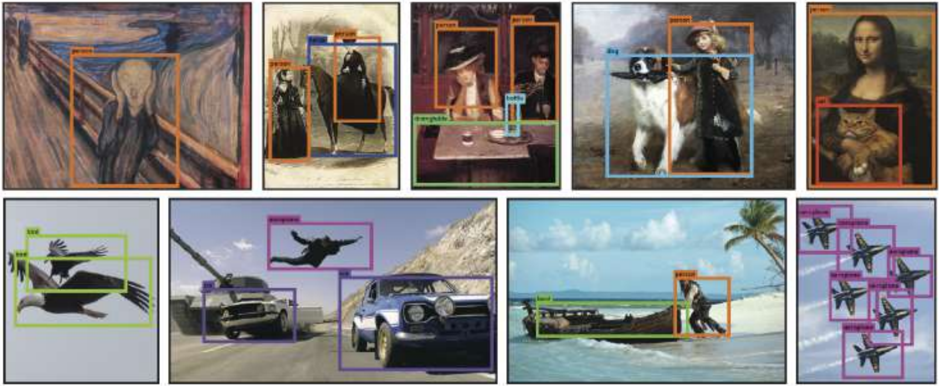
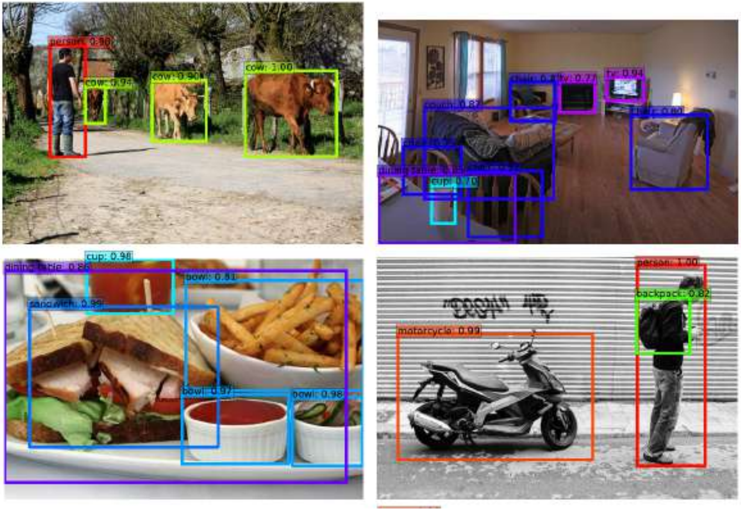


Figure 4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

YOLO



SSD



person: 0.98
cow: 0.94
cow: 0.90
cow: 1.00
tv: 0.77
tv: 0.94
couch: 0.87
cup: 0.70
cup: 0.98
dining table: 0.86
bowl: 0.81
sandwich: 0.99
bowl: 0.97
bowl: 0.98
person: 1.00
backpack: 0.82
motorcycle: 0.99

Image classification - Going further

Lin et al. 2015 incorporate part localization, alignment, and classification into one recognition system which is called Deep LAC.

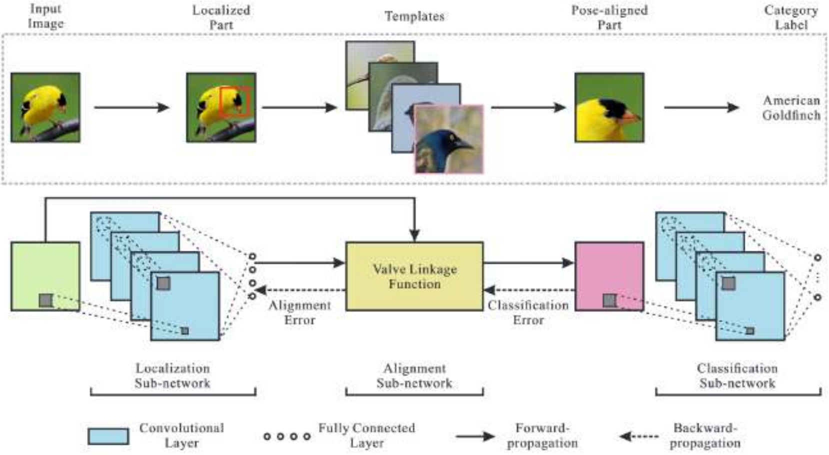


Image classification - Going further

Annotations are not easy to collect and these systems have difficulty in scaling up and to handle many types of fine-grained classes.

["Fine-grained recognition without part annotations", Krause et al. 2015] combine co-segmentation and alignment in a discriminative mixture to generate parts for facilitating fine-grained classification.

["Weakly supervised fine-grained categorization with part-based image representation", Zhang et al. 2016] use the unsupervised selective search to generate object proposals, and then select the useful parts from the multi-scale generated part proposals.

Object detection and classification: see also ["Deep neural networks for object detection", Szegedy, Toshev, et al. 2013]

Outline

1. Foundations of CNN
    - Convolution layer
    - Pooling layer
    - Data preprocessing
2. Famous CNN
    - LeNet (1998)
    - AlexNet (2012)
    - ZFNet (2013)
    - VGGNet (2014)
    - GoogLeNet (2014)
    - ResNet (2016)
    - DenseNet (2017)
    - Many other CNN
3. Applications
    - Image classification
    - Pose, action detection
    - Object detection
    - Scene labeling - Semantic segmentation
    - Object tracking - videos
    - Text detection and recognition

Scene labeling

Scene labeling aims to relate one semantic class (road, water, sea...) to each pixel of the input image

→ ["Recurrent convolutional neural networks for scene labeling", Pinheiro and Collobert 2014]
To enable the CNNs to have a large field of view over pixels, they develop the recurrent CNNs. More specifically, the identical CNNs are applied recurrently to the output maps of CNNs in the previous iterations. By doing this, they can achieve slightly better labeling results while significantly reduces the inference times.

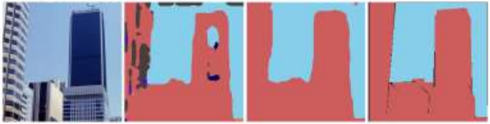
→ ["Dag-recurrent neural networks for scene labeling", Shuai et al. 2016]
They use the recurrent neural networks to model the contextual dependencies among image features from CNNs, and dramatically boost the labeling performance.

Object semantic segmentation

["Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs", L.-C. Chen et al. 2018]
They apply pre-trained deep CNNs to emit the labels of pixels. Considering that the imperfectness of boundary alignment, they further use fully connected Conditional Random Field (CRF) to boost the labeling performance.

Scene labeling - DAG-RNN

Input Image CNN DAG-RNN Ground Truth



Input Image CNN DAG-RNN Ground Truth

Outline

1. Foundations of CNN
   - Convolution layer
   - Pooling layer
   - Data preprocessing
2. Famous CNN
   - LeNet (1998)
   - AlexNet (2012)
   - ZFNet (2013)
   - VGGNet (2014)
   - GoogLeNet (2014)
   - ResNet (2016)
   - DenseNet (2017)
   - Many other CNN
3. Applications
   - Image classification
   - Pose, action detection
   - Object detection
   - Scene labeling - Semantic segmentation
   - Object tracking - videos
   - Text detection and recognition

Object tracking

The success in object tracking relies heavily on how robust the representation of target appearance is against several challenges such as view point changes, illumination changes, and occlusions

["Deeptrack: Learning discriminative feature representations online for robust visual tracking", Li et al. 2016]
They propose a target-specific CNN for object tracking, where the CNN is trained incrementally during tracking with new examples obtained online. They employ a candidate pool of multiple CNNs as a data-driven model of different instances of the target object.

["Cnntracker: Online discriminative object tracking via deep convolutional neural network", Y. Chen et al. 2016]
A CNN object tracking method is proposed to address limitations of handcrafted features and shallow classifier structures in object tracking problem.

["Online tracking by learning discriminative saliency map with convolutional neural network", Hong et al. 2015]
They propose a visual tracking algorithm based on a pre-trained CNN. They put an additional layer of an online SVM to learn a target appearance discriminatively against background.

`https://pjreddie.com/darknet/yolo/`

Pose/Action recognition - videos

Applying CNNs on videos is challenging because traditional CNNs are designed to represent two dimensional pure spatial signals but in videos a new temporal axis is added which is essentially different from the spatial variations in images

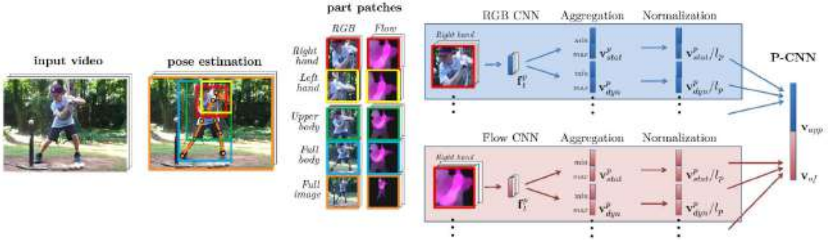
[“3D convolutional neural networks for human action recognition”, Ji et al. 2013]
They consider the temporal axis in a similar manner as other spatial axes and introduce a network of 3D convolutional layers to be applied on video inputs.

[“Two-stream convolutional networks for action recognition in videos”, Simonyan and Zisserman 2014a]
Separating the representation to spatial and temporal variations and train individual CNNs for each of them. First stream of this framework is a traditional CNN applied on all the frames and the second receives the dense optical flow of the input videos and trains another CNN which is identical to the spatial stream in size and structure. The output of the two streams are combined in a class score fusion step.

[“P-cnn: Pose-based cnn features for action recognition”, Chéron et al. 2015]
They use the two stream CNN on the localized parts of the human body and show the aggregation of part-based local CNN descriptors can effectively improve the performance of action recognition.

Pose estimation - P-CNN

["P-cnn: Pose-based cnn features for action recognition", Chéron et al. 2015]



["End-to-end learning of deformable mixture of parts and deep convolutional neural networks for human pose estimation", W. Yang et al. 2016]
https://www.youtube.com/watch?v=MKVvQK8FawE

["Segnet: A deep convolutional encoder-decoder architecture for image segmentation", Badrinarayanan et al. 2015]
https://www.youtube.com/watch?v=CxanE_W46ts

["Realtime multi-person 2d pose estimation using part affinity fields", Cao et al. 2016]
https://www.youtube.com/watch?v=pW6nZXeWlGM

Outline

1. Foundations of CNN
   - Convolution layer
   - Pooling layer
   - Data preprocessing
2. Famous CNN
   - LeNet (1998)
   - AlexNet (2012)
   - ZFNet (2013)
   - VGGNet (2014)
   - GoogLeNet (2014)
   - ResNet (2016)
   - DenseNet (2017)
   - Many other CNN
3. Applications
   - Image classification
   - Pose, action detection
   - Object detection
   - Scene labeling - Semantic segmentation
   - Object tracking - videos
   - Text detection and recognition

Text detection and recognition

Optical Character Recognition (OCR) can be categorized into three types:

1. text detection and localization without recognition,
2. text recognition on cropped text images,
3. end-to-end text spotting that integrates both text detection and recognition.

Several proposed methods:

- CNN model originally trained for character classification to perform text detection
  ["End-to-end text recognition with convolutional neural networks", T. Wang et al. 2012]
- CNN model allowing feature sharing across four different subtask: text detection, character case-sensitive and insensitive classification, and bigram classification.
  ["Deep features for text spotting", Jaderberg, Vedaldi, et al. 2014]
- Elementary subtasks as text bounding box filtering, text bounding box regression, and text recognition are each tackled by a separate CNN model.
  ["Reading text in the wild with convolutional neural networks", Jaderberg, Simonyan, et al. 2016]

References

Chris M Bishop. "Training with noise is equivalent to Tikhonov regularization". In: *Neural computation* 7.1 (1995), pp. 108–116.

Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation". In: *arXiv preprint arXiv:1511.00561* (2015).

Leo Breiman. "Randomizing outputs to increase prediction accuracy". In: *Machine Learning* 40.3 (2000), pp. 229–242.

Zhe Cao et al. "Realtime multi-person 2d pose estimation using part affinity fields". In: *arXiv preprint arXiv:1611.08050* (2016).

Yan Chen et al. "Cnntracker: Online discriminative object tracking via deep convolutional neural network". In: *Applied Soft Computing* 38 (2016), pp. 1088–1098.

Liang-Chieh Chen et al. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2018), pp. 834–848.

References

François Chollet. “Xception: Deep learning with depthwise separable convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.

Guilhem Chéron, Ivan Laptev, and Cordelia Schmid. “P-cnn: Pose-based cnn features for action recognition”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 3218–3226.

Dorin Comaniciu and Peter Meer. “Mean shift: A robust approach toward feature space analysis”. In: *IEEE Transactions on pattern analysis and machine intelligence* 24.5 (2002), pp. 603–619.

Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. “An analysis of deep neural network models for practical applications”. In: *arXiv preprint arXiv:1605.07678* (2016).

David Eigen and Rob Fergus. “Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2650–2658.

References

Ian Endres and Derek Hoiem. “Category-independent object proposals with diverse ranking”. In: *IEEE transactions on pattern analysis and machine intelligence* 36.2 (2014), pp. 222–234.

Pedro F Felzenszwalb and Daniel P Huttenlocher. “Efficient graph-based image segmentation”. In: *International journal of computer vision* 59.2 (2004), pp. 167–181.

Georgia Gkioxari, Ross Girshick, and Jitendra Malik. “Actions and attributes from wholes and parts”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2470–2478.

Ross Girshick, Jeff Donahue, et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

Ross Girshick, Forrest Iandola, et al. “Deformable part models are convolutional neural networks”. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2015, pp. 437–446.

References

Lluıs Gómez and Dimosthenis Karatzas. "Textproposals: a text-specific selective search algorithm for word spotting in the wild". In: *Pattern Recognition* 70 (2017), pp. 60–74.

Alex Graves. "Practical variational inference for neural networks". In: *Advances in Neural Information Processing Systems*. 2011, pp. 2348–2356.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples". In: *arXiv preprint arXiv:1412.6572* (2014).

Jiuxiang Gu et al. "Recent advances in convolutional neural networks". In: *arXiv preprint arXiv:1512.07108* (2015).

Kaiming He et al. "Spatial pyramid pooling in deep convolutional networks for visual recognition". In: *European conference on computer vision*. Springer. 2014, pp. 346–361.

Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

References

Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

Seunghoon Hong et al. "Online tracking by learning discriminative saliency map with convolutional neural network". In: *International Conference on Machine Learning*. 2015, pp. 597–606.

Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554.

Gao Huang et al. "Densely Connected Convolutional Networks." In: *CVPR*. Vol. 1. 2. 2017, p. 3.

Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun, et al. "What is the best multi-stage architecture for object recognition?" In: *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE. 2009, pp. 2146–2153.

Max Jaderberg, Karen Simonyan, et al. "Reading text in the wild with convolutional neural networks". In: *International Journal of Computer Vision* 116.1 (2016), pp. 1–20.

References

Kam-Chuen Jim, C Lee Giles, and Bill G Horne. "An analysis of noise in recurrent neural networks: convergence and generalization". In: *IEEE Transactions on neural networks* 7.6 (1996), pp. 1424–1438.

Shuiwang Ji et al. "3D convolutional neural networks for human action recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 35.1 (2013), pp. 221–231.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. "Deep features for text spotting". In: *European conference on computer vision*. Springer. 2014, pp. 512–528.

Asifullah Khan et al. "A survey of the recent architectures of deep convolutional neural networks". In: *Artificial Intelligence Review* 53.8 (2020), pp. 5455–5516.

Jonathan Krause et al. "Fine-grained recognition without part annotations". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 5546–5555.

References

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

Yann LeCun et al. "Generalization and network design strategies". In: *Connectionism in perspective* (1989), pp. 143–155.

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

Chen-Yu Lee et al. "Deeply-supervised nets". In: *Artificial Intelligence and Statistics*. 2015, pp. 562–570.

Di Lin et al. "Deep lac: Deep localization, alignment and classification for fine-grained recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1666–1674.

Wei Liu et al. "Ssd: Single shot multibox detector". In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

References

Yongxi Lu, Tara Javidi, and Svetlana Lazebnik. "Adaptive object detection using adjacency and zoom prediction". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2351–2359.

Hanxi Li, Yi Li, and Fatih Porikli. "Deeptrack: Learning discriminative feature representations online for robust visual tracking". In: *IEEE Transactions on Image Processing* 25.4 (2016), pp. 1834–1848.

James Martens. "Deep learning via Hessian-free optimization." In: *ICML*. Vol. 27. 2010, pp. 735–742.

Duc Thanh Nguyen, Wanqing Li, and Philip O Ogunbona. "Human detection from images and videos: A survey". In: *Pattern Recognition* 51 (2016), pp. 148–175.

Steven J Nowlan and John C Platt. "A convolutional neural network hand tracker". In: *Advances in neural information processing systems* (1995), pp. 901–908.

References

Mattis Paulin et al. "Transformation pursuit for image classification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 3646–3653.

Pedro HO Pinheiro and Ronan Collobert. "Recurrent convolutional neural networks for scene labeling". In: *31st International Conference on Machine Learning (ICML)*. EPFL-CONF-199822. 2014.

Leonid Pishchulin et al. "Poselet conditioned pictorial structures". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 588–595.

Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

Shaoqing Ren et al. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems*. 2015, pp. 91–99.

Salah Rifai et al. "Adding noise to the input of a model trained with a regularized objective". In: *arXiv preprint arXiv:1104.3250* (2011).

References

Brian D Ripley. *Pattern recognition and neural networks*. Cambridge university press, 2007.

Olga Russakovsky et al. "Imagenet large scale visual recognition challenge". In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

Justin Salamon and Juan Pablo Bello. "Deep convolutional neural networks and data augmentation for environmental sound classification". In: *IEEE Signal Processing Letters* 24.3 (2017), pp. 279–283.

Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. "Training very deep networks". In: *Advances in neural information processing systems*. 2015, pp. 2377–2385.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. "Highway networks". In: *arXiv preprint arXiv:1505.00387* (2015).

Bing Shuai et al. "Dag-recurrent neural networks for scene labeling". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3620–3629.

References

Christian Szegedy, Alexander Toshev, and Dumitru Erhan. "Deep neural networks for object detection". In: *Advances in neural information processing systems*. 2013, pp. 2553–2561.

Karen Simonyan and Andrew Zisserman. "Two-stream convolutional networks for action recognition in videos". In: *Advances in neural information processing systems*. 2014, pp. 568–576.

Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

Christian Szegedy, Wei Liu, et al. "Going deeper with convolutions". In: Cvpr. 2015.

Christian Szegedy, Vincent Vanhoucke, et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

References

Alexander Toshev and Christian Szegedy. "Deeppose: Human pose estimation via deep neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1653–1660.

Régis Vaillant, Christophe Monrocq, and Yann Le Cun. "Original approach for the localisation of objects in images". In: *IEE Proceedings-Vision, Image and Signal Processing* 141.4 (1994), pp. 245–250.

Tao Wang et al. "End-to-end text recognition with convolutional neural networks". In: *Pattern Recognition (ICPR), 2012 21st International Conference on*. IEEE. 2012, pp. 3304–3308.

Zhenhua Wang, Xingxing Wang, and Gang Wang. "Learning fine-grained features via a CNN tree for large-scale classification". In: *Neurocomputing* 275 (2018), pp. 1231–1240.

Tianjun Xiao et al. "Error-driven incremental learning in deep convolutional neural network for large-scale image classification". In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM. 2014, pp. 177–186.

References

Saining Xie and Zhuowen Tu. "Holistically-nested edge detection". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1395–1403.

Zhicheng Yan et al. "HD-CNN: hierarchical deep convolutional neural networks for large scale visual recognition". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2740–2748.

Wei Yang et al. "End-to-end learning of deformable mixture of parts and deep convolutional neural networks for human pose estimation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 3073–3082.

Donggeun Yoo et al. "Attentionnet: Aggregating weak directions for accurate object detection". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2659–2667.

Heng Yang and Ioannis Patras. "Mirror, mirror on the wall, tell me, is the error small?" In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 4685–4693.

References

Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

Yu Zhang et al. "Weakly supervised fine-grained categorization with part-based image representation". In: *IEEE Transactions on Image Processing* 25.4 (2016), pp. 1713–1725.