Deep Learning

Chapter 6: Modern Convolutional Neural Networks

Mina Rezaei

Department of Statistics – LMU Munich

Winter Semester 2020

LECTURE OUTLINE

From LeNet to AlexNet

Networks Using Blocks (VGG)

Network in Network (NiN)

Networks with Parallel Concatenations (GoogLeNet)

Residual Networks (ResNet)

Densely Connected Networks (DenseNet)

From LeNet to AlexNet

LENET ARCHITECTURE

- Pioneering work on CNNs by Yann Lecun in 1998.
- Applied on the MNIST dataset for automated handwritten digit recognition.
- Consists of convolutional, "subsampling" and dense layers.
- Complexity and depth of the net was mainly restricted by limited computational power back in the days.

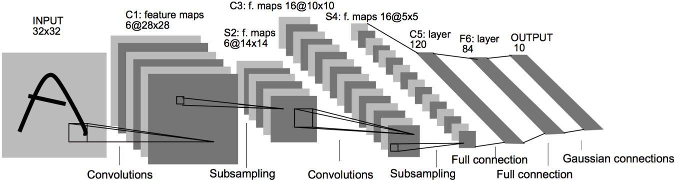


Figure: LeNet architecture: two conv layers with subsampling, followed by dense layers and a ’Gaussian connections’ layer.

LENET ARCHITECTURE

- A neuron in a subsampling layer looks at a 2×2 region of a feature map, sums the four values, multiplies it by a trainable coefficient, adds a trainable bias and then applies a sigmoid activation.
- A stride of 2 ensures that the size of the feature map reduces by about a half.
- The 'Gaussian connections' layer has a neuron for each possible class.
- The output of each neuron in this layer is the (squared) Euclidean distance between the activations from the previous layer and the weights of the neuron.

ALEXNET

- AlexNet, which employed an 8-layer CNN, won the ImageNet Large Scale Visual Recognition (LSVR) Challenge 2012 by a phenomenally large margin.
- The network trained in parallel on two small GPUs, using two streams of convolutions which are partly interconnected.
- The architectures of AlexNet and LeNet are very similar, but there are also significant differences:
  - First, AlexNet is deeper than the comparatively small LeNet5. AlexNet consists of eight layers: five convolutional layers, two fully-connected hidden layers, and one fully-connected output layer.
  - Second, AlexNet used the ReLU instead of the sigmoid as its activation function.

ALEXNET

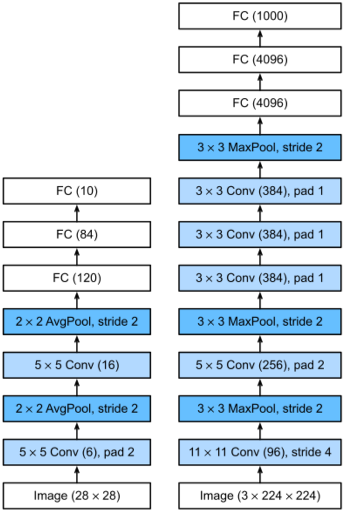


Figure: From LeNet (left) to AlexNet (right).

ALEXNET

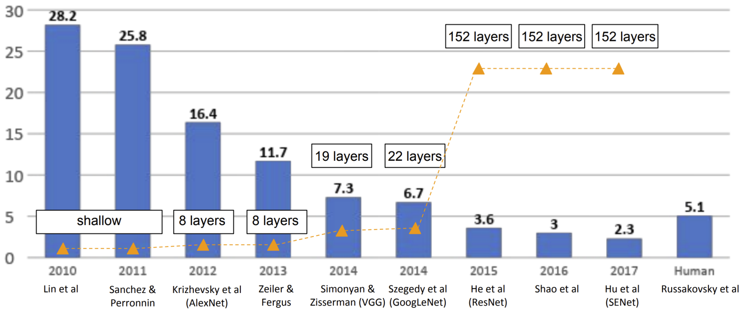
Full (simplified) AlexNet architecture:
[224x224x3]] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

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

credit : Stanford Cs231n

Figure: Implementation detail by AlexNet.

IMAGENET LARGE SCALE VISUAL RECOGNITION CHALLENGE (ILSVRC) WINNERS



credit : Stanford Cs231n

Figure: ILSVRC Compitions winners.

ZFNET

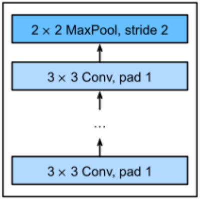
It has similar architecture of AlexNet but:

- CONV1: change from (11 $\times$ 11 stride 4) to (7 x 7 stride 2)
- CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512
- ImageNet top 5 error: 16.4% to 11.7%

Networks Using Blocks (VGG)

VGG BLOCKS

- The block composed of convolutions with 3×3 kernels with padding of 1 (keeping height and width) and 2×2 max pooling with stride of 2 (having the resolution after each block).
- The use of blocks leads to very compact representations of the network definition.
- It allows for efficient design of complex networks.



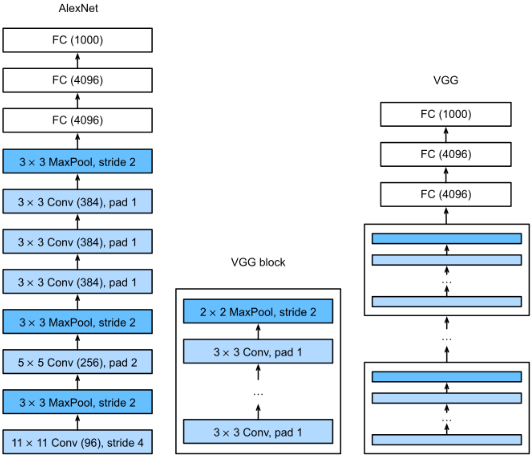
credit : D2DL

Figure: VGG block.

VGG NETWORK

- Architecture introduced by Simonyan and Zisserman, 2014 as "Very Deep Convolutional Network".
- A deeper variant of the AlexNet.
- Basic idea is to have small filters and Deeper networks
- Mainly uses many cnn layers with a small kernel size 3×3.
- Stack of three 3×3 cnn (stride 1) layers has same effective receptive field as one 7×7 conv layer. But It's deeper, more non-linearities, and fewer parameters
- Performed very well in the ImageNet Challenge in 2014.
- Exists in a small version (VGG16) with a total of 16 layers (13 cnn layers and 3 fc layers) and 5 VGG blocks while a larger version (VGG19) with 19 layers (16 cnn layers and 3 fc layers) and 6 VGG blocks.

VGG NETWORK



credit : D2DL

Figure: From AlexNet to VGG that is designed from building blocks.

VGG NETWORK

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0

CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728

CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864

POOL2: [112x112x64] memory: 112*112*64=800K params: 0

CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728

CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456

POOL2: [56x56x128] memory: 56*56*128=400K params: 0

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824

POOL2: [28x28x256] memory: 28*28*256=200K params: 0

CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648

CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296

CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296

POOL2: [14x14x512] memory: 14*14*512=100K params: 0

CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

POOL2: [7x7x512] memory: 7*7*512=25K params: 0

FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448

FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216

FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

VGG16

TOTAL memory: 24M * 4 bytes ~= 96MB / image (for a forward pass)

TOTAL params: 138M parameters

credit : Stanford Cs231n

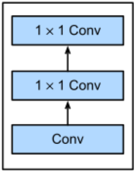
Figure: Computational complexity by VGG network.

Network in Network (NiN)

NIN BLOCKS

- The idea behind NiN is to apply a fully-connected layer at each pixel location (for each height and width). If we tie the weights across each spatial location, we could think of this as a 1×1 convolutional layer.
- The NiN block consists of one convolutional layer followed by two 1×1 convolutional layers that act as per-pixel fully-connected layers with ReLU activations.
- The convolution window shape of the first layer is typically set by the user. The subsequent window shapes are fixed to 1×1.

NIN BLOCKS



credit : D2DL

Figure: NiN block.

GLOBAL AVERAGE POOLING

- Problem setting: tackle overfitting in the final fully connected layer.
  - Classic pooling removes spatial information and is mainly used for dimension and parameter reduction.
  - The elements of the final feature maps are connected to the output layer via a dense layer. This could require a huge number of weights increasing the danger of overfitting.
  - Example: 256 feature maps of dim 100x100 connected to 10 output neurons lead to 256×10^5 weights for the final dense layer.

GLOBAL AVERAGE POOLING

- Solution:
  - Average each final feature map to the element of one global average pooling (GAP) vector.
  - Example: 256 feature maps are now reduced to GAP-vector of length 256 yielding a final dense layer with 2560 weights.

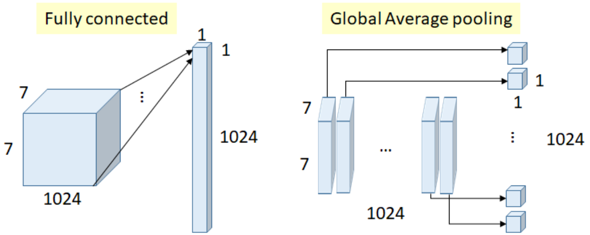


Figure: An Example of Fully Connected Layer VS Global Average Pooling Layer

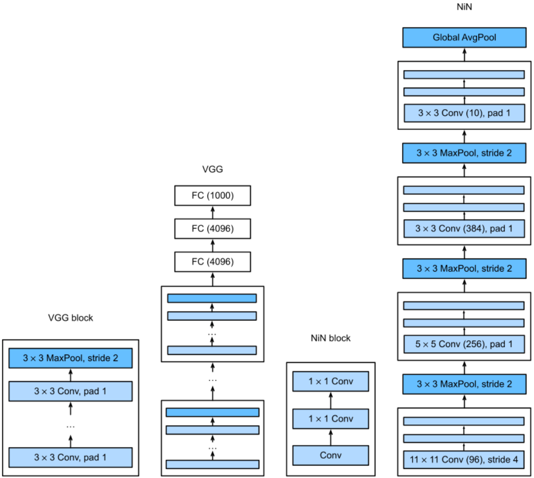
GLOBAL AVERAGE POOLING

- GAP preserves whole information from the single feature maps whilst decreasing the dimension.
- Mitigates the possibly **destructive** effect of pooling.
- Each element of the GAP output represents the activation of a certain feature on the input data.
- Acts as an additional regularizer on the final fully connected layer.

NETWORK IN NETWORK (NIN)

- NiN uses blocks consisting of a convolutional layer and multiple 1×1 convolutional layers. This can be used within the convolutional stack to allow for more per-pixel nonlinearity.
- NiN removes the fully-connected layers and replaces them with global average pooling (i.e., summing over all locations) after reducing the number of channels to the desired number of outputs (e.g., 10 for Fashion-MNIST).
- Removing the fully-connected layers reduces overfitting. NiN has dramatically fewer parameters.
- The NiN design influenced many subsequent CNN designs.

NETWORK IN NETWORK (NIN)



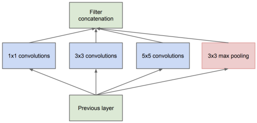
credit : D2DL

Figure: Comparing architectures of VGG and NiN, and their blocks.

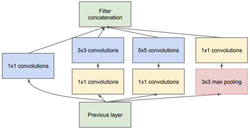
Networks with Parallel Concatenations (GoogLeNet)

INCEPTION BLOCK

- Problem setting: how do we choose the kernel size in each layer?
- This is often an arbitrary decision.
- Solution: offer the model kernels of different sizes in each layer through which it can propagate information and let it decide, which one to use to which extent.
- Side-effect: massive parameter reduction allowing for deeper architectures.



(a) Inception module, naïve version



(b) Inception module with dimension reductions

Figure: Inception block.

INCEPTION BLOCK

- The Inception block is equivalent to a subnetwork with four paths.
- It extracts information in parallel through convolutional layers of different window shapes and max-pooling layers.
- 1×1 convolutions reduce channel dimensionality on a per-pixel level. Max-pooling is used as it is ought to increase the robustness of the feature maps. The kernels are padded accordingly to yield feature maps of equal dimensions..

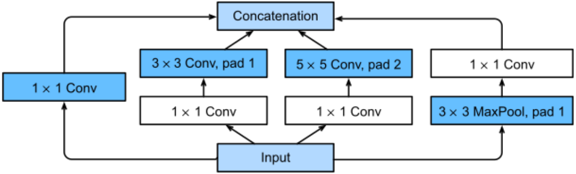
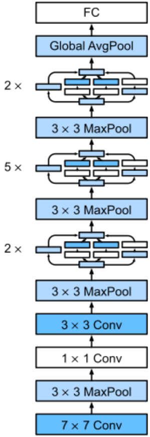


Figure: Inception block with dimention reduction.

GOOGLENET ARCHITECTURE

- GoogLeNet connects multiple well-designed Inception blocks with other layers in series.
- The ratio of the number of channels assigned in the Inception block is obtained through a large number of experiments on the ImageNet dataset.
- GoogLeNet, as well as its succeeding versions, was one of the most efficient models on ImageNet, providing similar test accuracy with lower computational complexity.

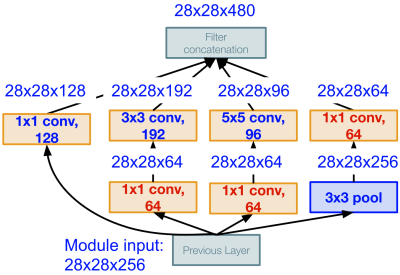
GOOGLENET ARCHITECTURE



credit : D2DL

Figure: The GoogLeNet architecture.

GOOGLENET ARCHITECTURE



Inception module with dimension reduction

Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

Compared to 854M ops for naive version
Bottleneck can also reduce depth after pooling layer

credit : Stanford cs231n

Figure: Computational complexity by Inceptation module

Residual Networks (ResNet)

RESIDUAL BLOCK (SKIP CONNECTIONS)

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

credit : Stanford cs231n

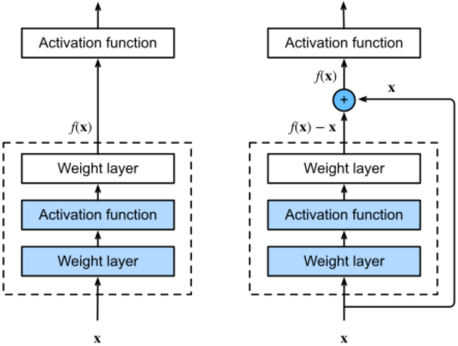
56-layer model performs worse on both training and test error! The deeper model performs worse, but it’s not caused by overfitting!

- Fact: Deep models have more representation power (more parameters) than shallower models.
- Hypothesis: the problem is an optimization problem, **deeper models are harder to optimize**

RESIDUAL BLOCK (SKIP CONNECTIONS)

- Problem setting: theoretically, we could build infinitely deep architectures as the net should learn to pick the beneficial layers and skip those that do not improve the performance automatically.
- But: this skipping would imply learning an identity mapping $\mathbf{x} = \mathcal{F}(\mathbf{x})$. It is very hard for a neural net to learn such a 1:1 mapping through the many non-linear activations in the architecture.
- Solution: offer the model explicitly the opportunity to skip certain layers if they are not useful.
- Introduced in [He et. al , 2015] and motivated by the observation that stacking evermore layers increases the test- as well as the train-error ($\neq$ overfitting).

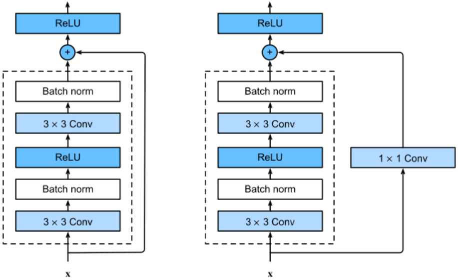
RESIDUAL BLOCK (SKIP CONNECTIONS)



credit : D2DL

Figure: A regular block (left) and a residual block (right).

RESIDUAL BLOCK (SKIP CONNECTIONS)



credit : D2DL

Figure: ResNet block with and without 1×1 convolution.The information flows through two layers and the identity function. Both streams of information are then element-wise summed and jointly activated.

RESIDUAL BLOCK (SKIP CONNECTIONS)

- Let $\mathcal{H}(\mathbf{x})$ be the optimal underlying mapping that should be learned by (parts of) the net.
- $\mathbf{x}$ is the input in layer l (can be raw data input or the output of a previous layer).
- $\mathcal{H}(\mathbf{x})$ is the output from layer l.
- Instead of fitting $\mathcal{H}(\mathbf{x})$, the net is ought to learn the residual mapping $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$ whilst $\mathbf{x}$ is added via the identity mapping.
- Thus, $\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$, as formulated on the previous slide.
- The model should only learn the **residual mapping** $\mathcal{F}(\mathbf{x})$
- Thus, the procedure is also referred to as **Residual Learning**.

RESIDUAL BLOCK (SKIP CONNECTIONS)

- The element-wise addition of the learned residuals $\mathcal{F}(\mathbf{x})$ and the identity-mapped data $\mathbf{x}$ requires both to have the same dimensions.
- To allow for downsampling within $\mathcal{F}(\mathbf{x})$ (via pooling or valid-padded convolutions), the authors introduce a linear projection layer W_s .
- W_s ensures that $\mathbf{x}$ is brought to the same dimensionality as $\mathcal{F}(\mathbf{x})$ such that:

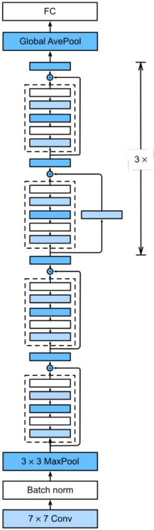
$$y = \mathcal{F}(\mathbf{x}) + W_s\mathbf{x},$$

- y is the output of the skip module and W_s represents the weight matrix of the linear projection (# rows of W_s = dimensionality of $\mathcal{F}(\mathbf{x})$).
- This idea applies to fully connected layers as well as to convolutional layers.

RESNET ARCHITECTURE

- The residual mapping can learn the identity function more easily, such as pushing parameters in the weight layer to zero.
- We can train an effective deep neural network by having residual blocks.
- Inputs can forward propagate faster through the residual connections across layers.
- ResNet had a major influence on the design of subsequent deep neural networks, both for convolutional and sequential nature.

RESNET ARCHITECTURE



credit : D2DL

Figure: The ResNet-18 architecture.

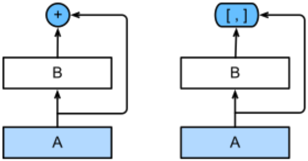
Densely Connected Networks (DenseNet)

FROM RESNET TO DENSENET

- ResNet significantly changed the view of how to parametrize the functions in deep networks.
- DenseNet (dense convolutional network) is to some extent the logical extension of this [Huang et al., 2017].
- Dense blocks where each layer is connected to every other layer in feedforward fashion.
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse.
- To understand how to arrive at it, let us take a small detour to mathematics:
    - Recall the Taylor expansion for functions. For the point $x = 0$ it can be written as:
    $f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \dots.$

FROM RESNET TO DENSENET

- The key point is that it decomposes a function into increasingly higher order terms. In a similar vein, ResNet decomposes functions into : $f(\mathbf{x}) = \mathbf{x} + g(\mathbf{x})$.
- That is, ResNet decomposes f into a simple linear term and a more complex nonlinear one. What if we want to capture (not necessarily add) information beyond two terms? One solution was DenseNet [Huang et al., 2017].



credit : D2DL

Figure: DensNet Block.

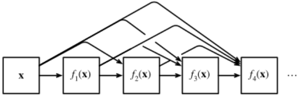
FROM RESNET TO DENSENET

As shown in previous Figure, the key difference between ResNet and DenseNet is that in the latter case outputs are concatenated (denoted by $[,]$) rather than added. As a result, we perform a mapping from x to its values after applying an increasingly complex sequence of functions:

$$\mathbf{x} \to [\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})]), f_3([\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})])]), \ldots] .$$

In the end, all these functions are combined in MLP to reduce the number of features again. In terms of implementation this is quite simple: rather than adding terms, we concatenate them.
The name DenseNet arises from the fact that the dependency graph between variables becomes quite dense. The last layer of such a chain is densely connected to all previous layers.

FROM RESNET TO DENSENET



credit : D2DL

Figure: The DensNet architecture.

REFERENCES

B. Zhou, Khosla, A., Labedriza, A., Oliva, A. and A. Torralba (2016)
Deconvolution and Checkerboard Artifacts
http://cnnlocalization.csail.mit.edu/Zhou_Learning_Deep_Features_CVPR_2016_paper.pdf

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich (2014)
Going deeper with convolutions
https://arxiv.org/abs/1409.4842

Kaiming He, Zhang, Xiangyu, Ren, Shaoqing, and Jian Sun (2015)
Deep Residual Learning for Image Recognition
https://arxiv.org/abs/1512.03385

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva and Antonio Torralba (2016)
Learning Deep Features for Discriminative Localization
http://cnnlocalization.csail.mit.edu/Zhou_Learning_Deep_Features_CVPR_2016_paper.pdf