ImageNet: A BIG Dataset with 15 Millions Labeled Images

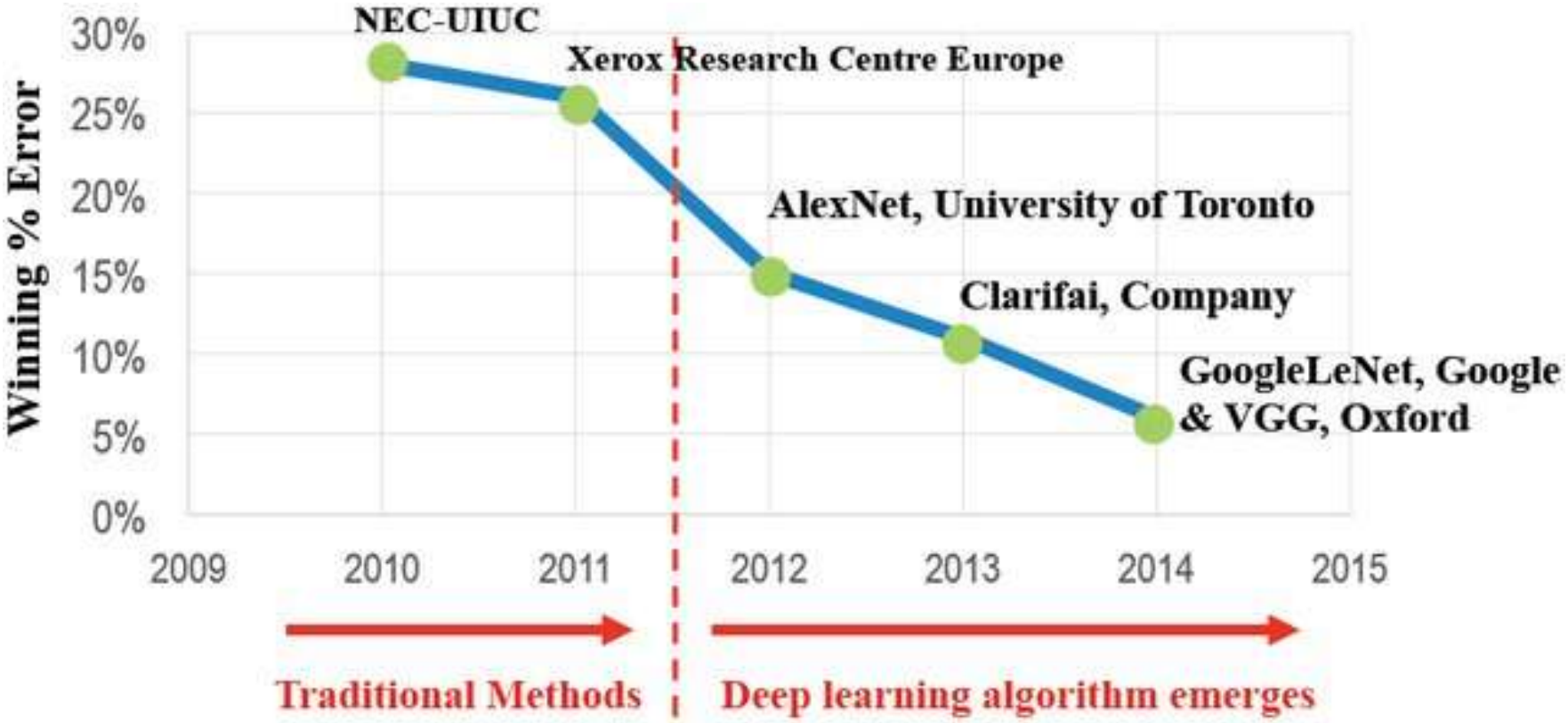
ImageNet and ILSVRC

- **ImageNet** is a dataset of over 15 million labeled images in 22000 categories.
  - Images were collected from the web
  - Images have variable resolutions.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and **Li Fei-Fei**. "Imagenet: A large-scale hierarchical image database." In IEEE CVPR **2009**.

- **ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) – 2010 to 2017**
  - it was an annual competition which has very big impact.
  - It uses a subset of ImageNet around 1000 images in **1000 categories**.
    - Around 1.2M training,
    - 50K validation and 100K test.
  - Tasks:
    - Image Classification,
    - Single-object localization,
    - Object detection
  - Usually, **Top-1** error and **Top-5** errors are reported.

ILSVRC Results (2010 to 2014)



Feature Engineering

2011: SVM was used as the classifier. => A shallow model.

AlexNet (2012) – Winner of ILSVRC 2012 – Breakthrough of CNNs

❖ LeNet achieved good results; But:

- On small datasets (e.g., MNIST)
- On raw pixels values as input. (w/o feature extraction)
- CV practitioners would never feed raw pixels into models
- Later other algorithms such as SVM achieved similar or even better results.

❖ Original AlexNet architecture

- Realized the bottleneck of CNNs are convolutions and matrix multiplication.
  - Used **Grouped convolution** for model parallelization across 2 GPUs

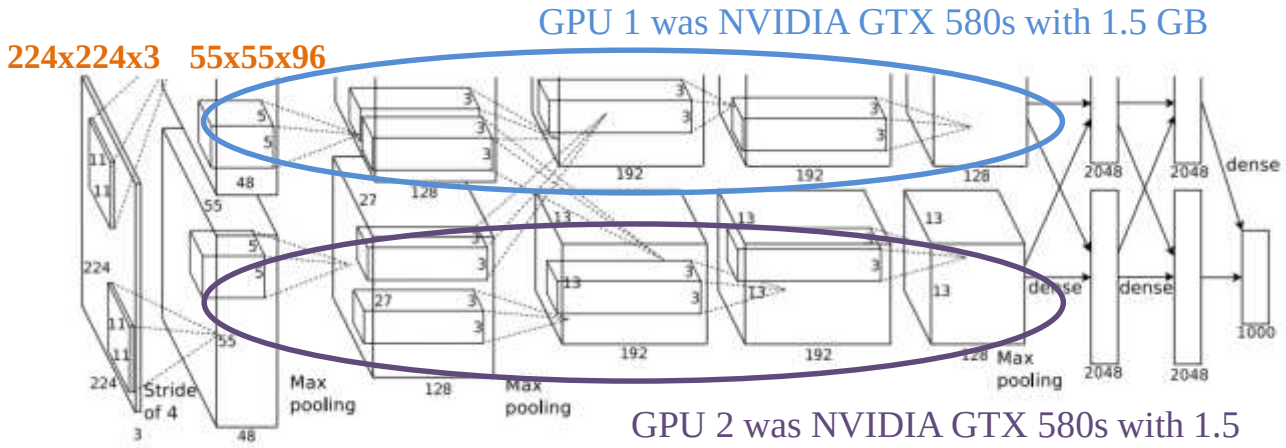


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

55*55*96 = 290,400 neurons in the 1st layer.

[Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (**2012**): 1097-1105.]

AlexNet Features

- Left: LeNet-5; Right: [**Simplified**] AlexNet
- AlexNet is a DEEPER and WIDER than LeNet-5.
- AlexNet depth = 8
  - ~ 650K neurons; 60M parameters.
- Overlapping pooling strategy
  - Reduced Top-1 error by 0.4% and Top-5 by 0.3%.
  - Claimed it prevent overfitting!
- Categorical cross-entropy loss was used.

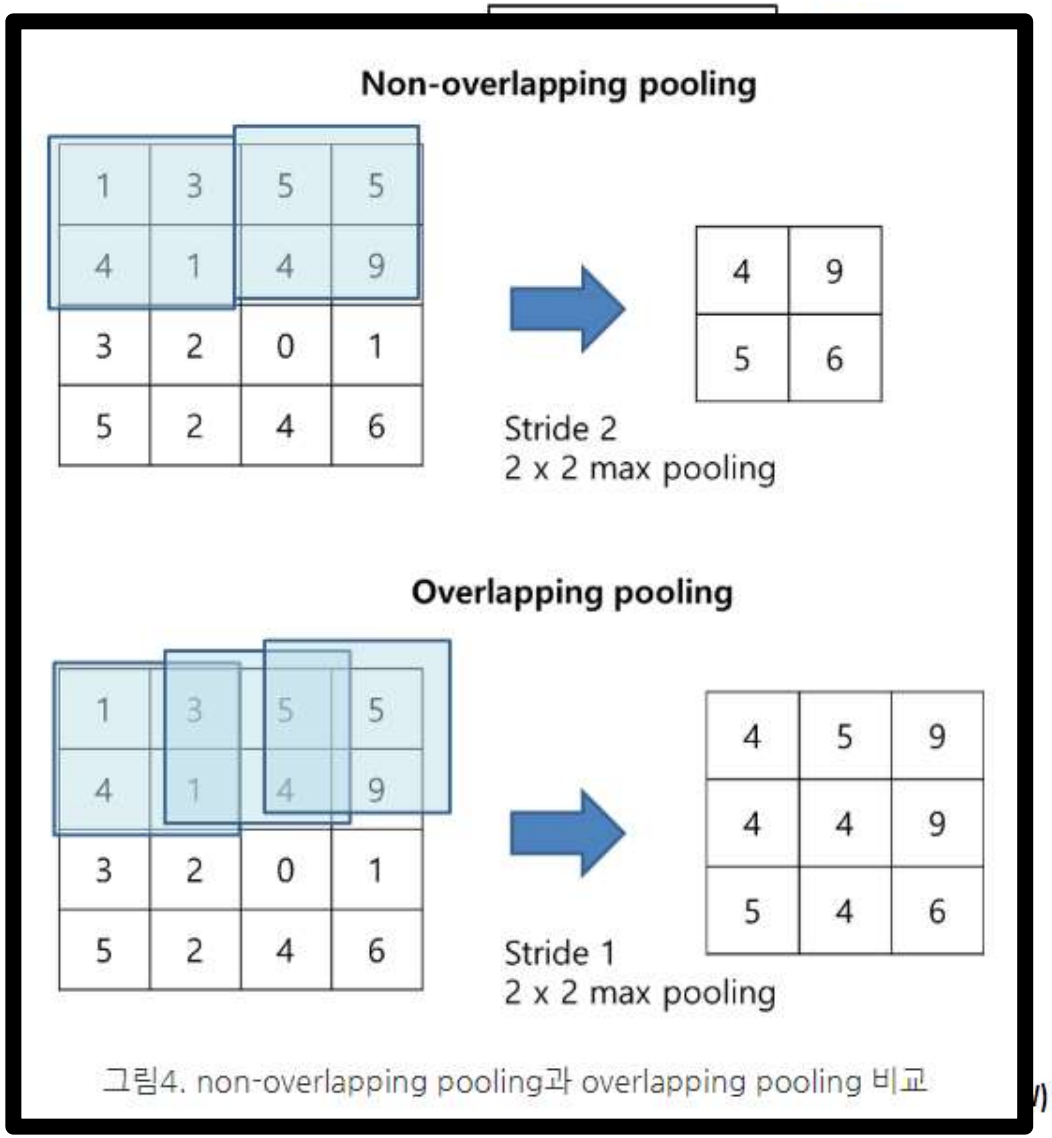
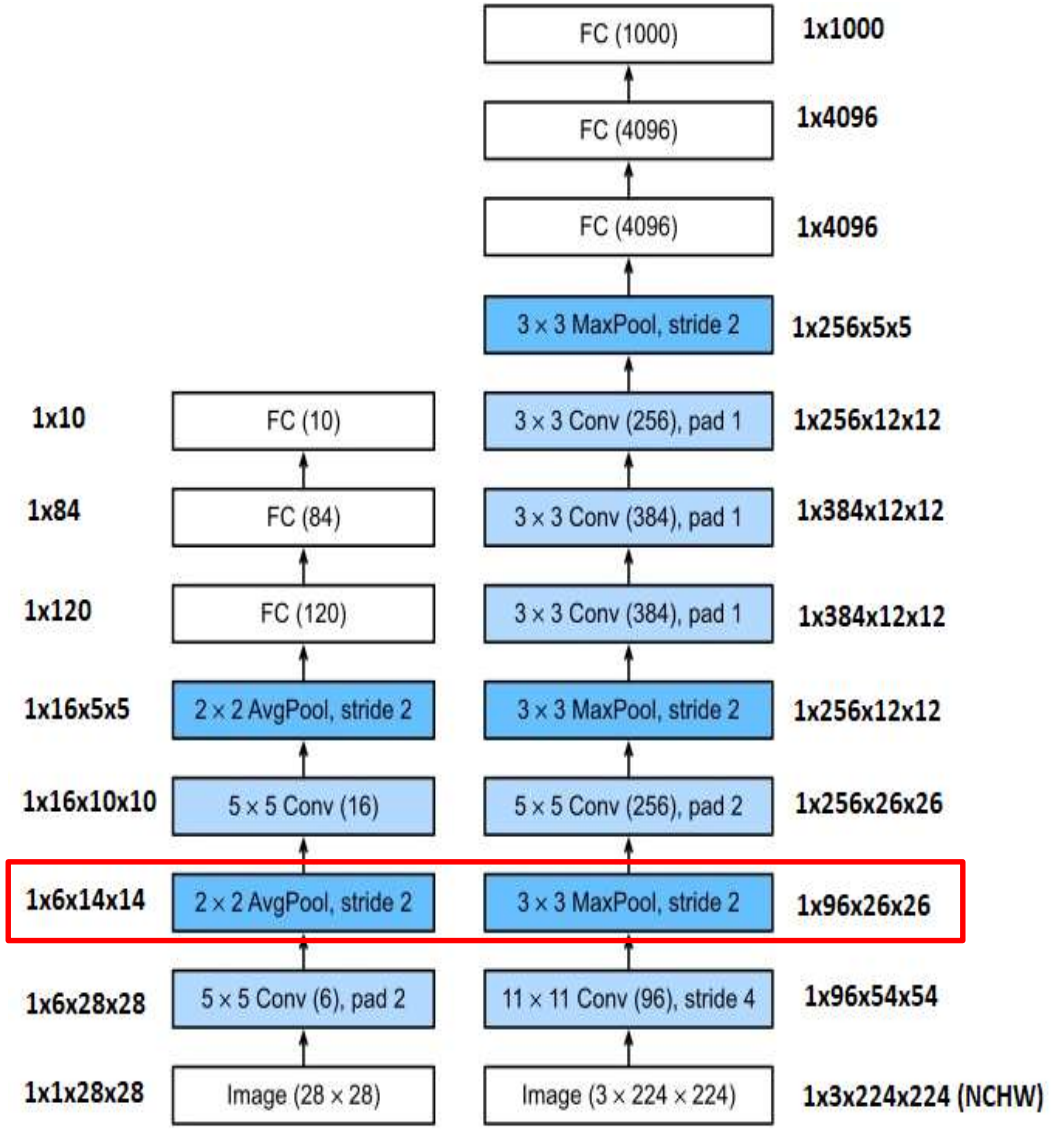


그림4. non-overlapping pooling과 overlapping pooling 비교

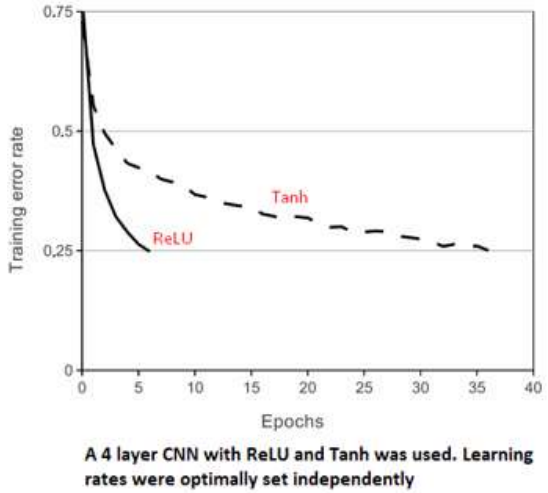
AlexNet Features

- Left: LeNet-5; Right: [**Simplified**] AlexNet
- AlexNet is a DEEPER and WIDER than LeNet-5.
- AlexNet depth = 8
  - ~ 650K neurons; 60M parameters.
- Overlapping pooling strategy
  - Reduced Top-1 error by 0.4% and Top-5 by 0.3%.
  - Claimed it prevent overfitting!
- Categorical cross-entropy loss was used.
- Weight initialization: N(0, 0.01)
- Used Local Response Normalization (LRN layer): => Not used anymore [VGG paper].



AlexNet Features

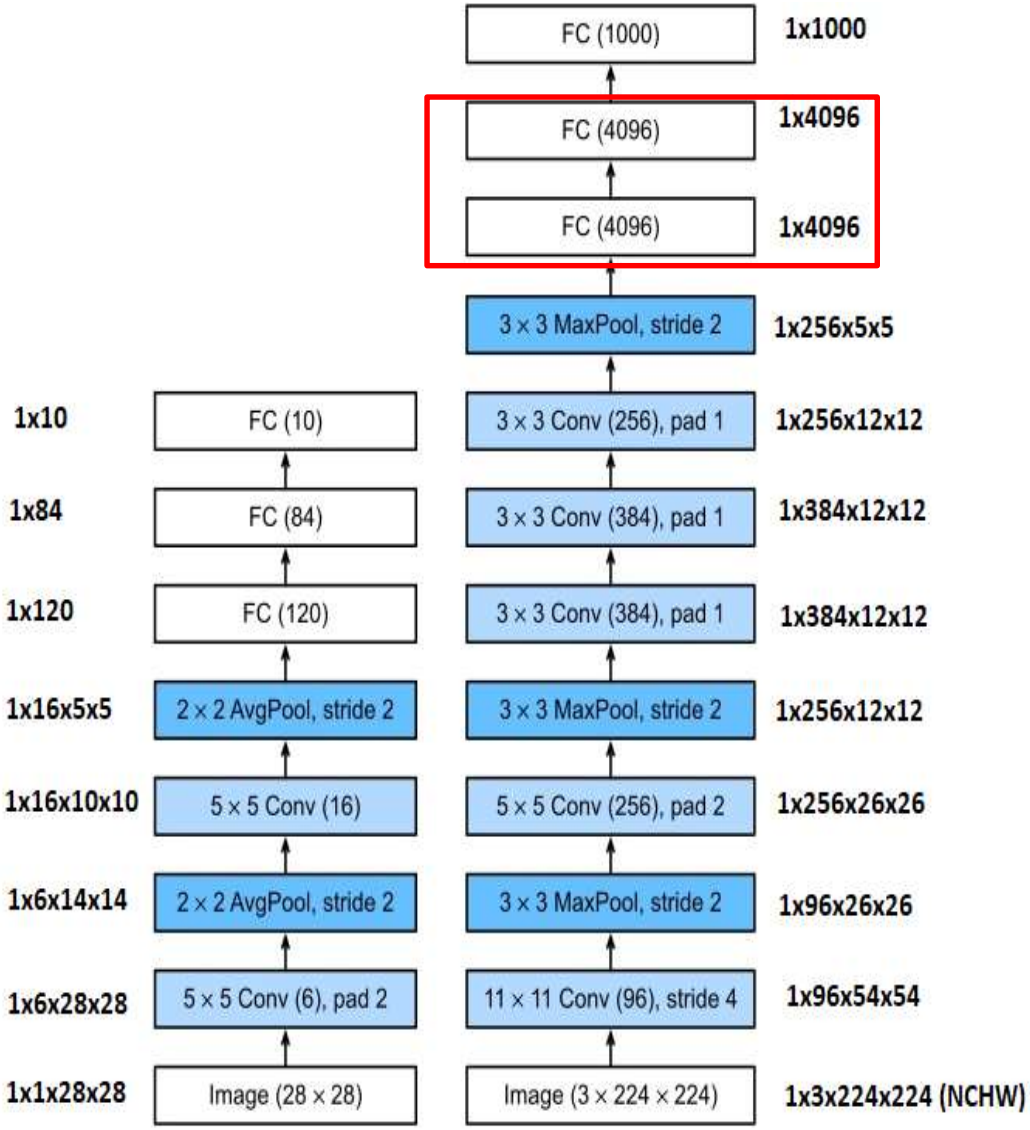
- ReLU instead of Sigmoid or Tanh.



A 4 layer CNN with ReLU and Tanh was used. Learning rates were optimally set independently

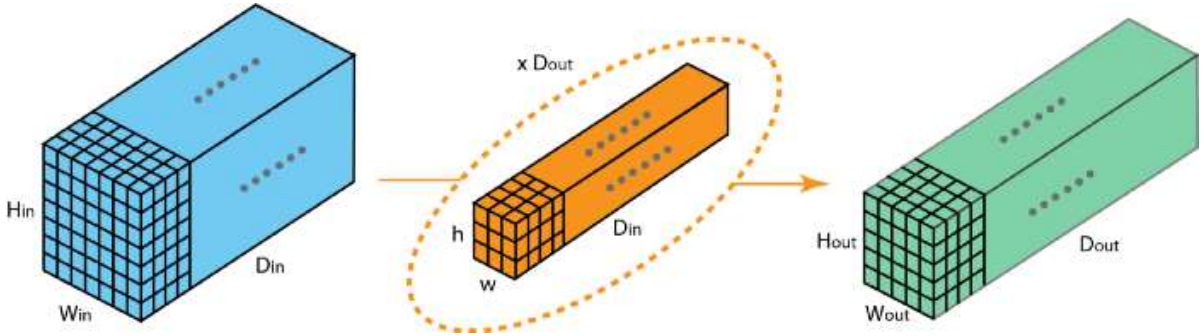
- Avoided overfitting using:
  - Dropout w/ 0.5 in the first two FC layers.
  - Data augmentation (random transformation and random intensity variation)

- Learning: mini-batch SGD w/ momentum 0.9 + L2 weight decay (5×10^{-5})

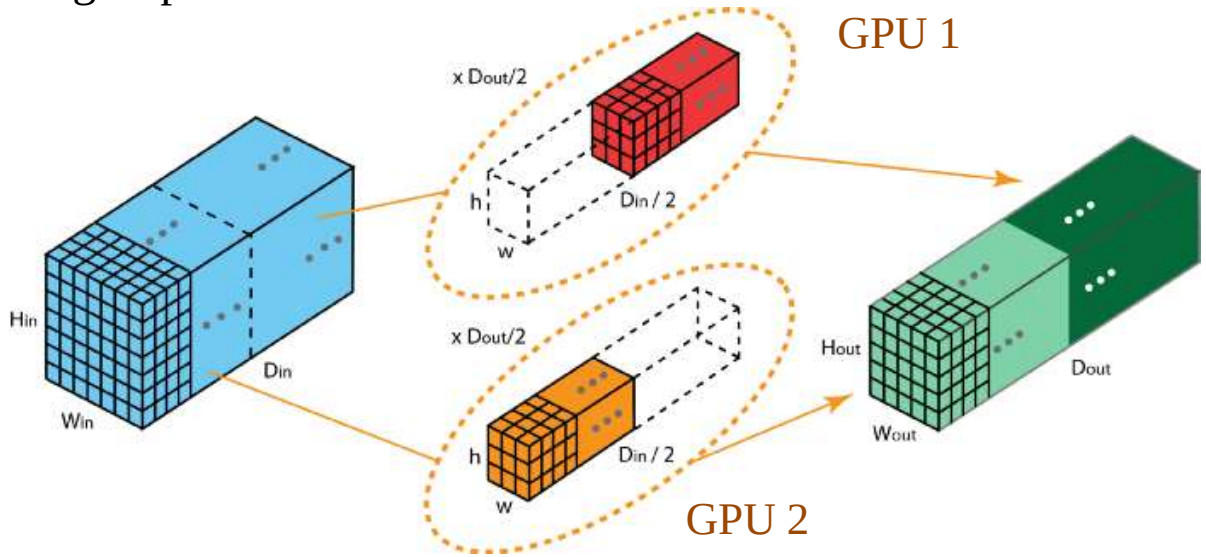


Grouped Convolution

- Recall standard 2D convolution



- Grouped Convolution w/ 2 filter groups



Some AlexNet Results at a Glance

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.

- Frequency-selective
- Orientation-selective
- Color blobs.
- Redundant filters

Similar to *Gabor* filters.

Note:
Simple Cells in Primary Visual Cortex (V1) function as Gabor filters.

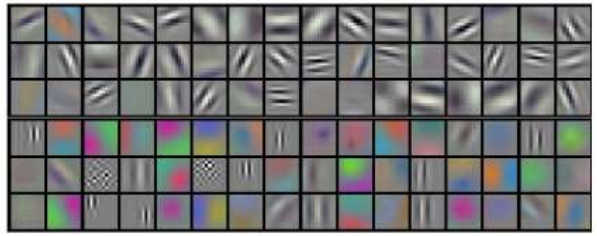


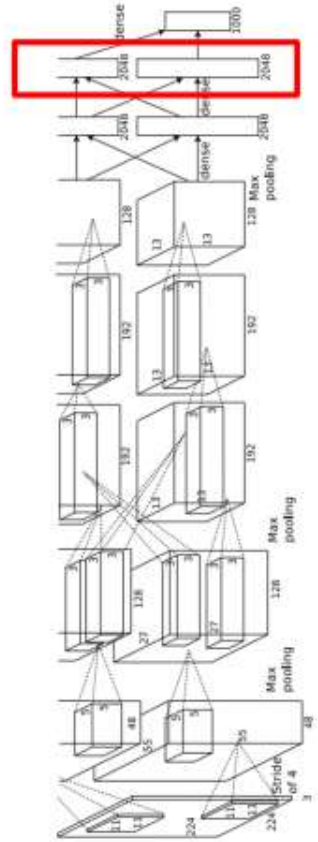
Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

Top predictions (=with the highest probabilities) are expected to be relevant.

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).

What does the last layer learn?

FC7 layer

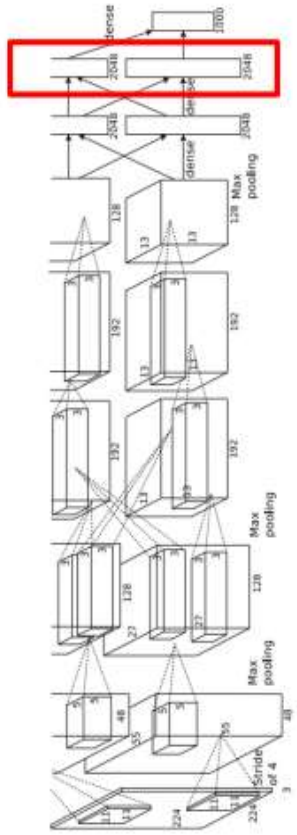


4096-dimensional feature vector for an image (layer immediately before the classifier)

Run the network on many images, collect the feature vectors

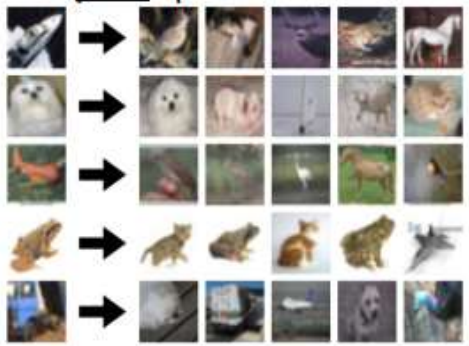
What does the last layer learn?

4096-dim vector



Test image L2 Nearest neighbors in feature space

Recall: Nearest neighbors in pixel space



AlexNet re-implementations

```
net = nn.Sequential(
    # Here, we use a larger 11 x 11 window to capture objects. At the same
    # time, we use a stride of 4 to greatly reduce the height and width of the
    # output. Here, the number of output channels is much larger than that in
    # LeNet
    nn.Conv2d(1, 96, kernel_size=11, stride=4, padding=1), nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    # Make the convolution window smaller, set padding to 2 for consistent
    # height and width across the input and output, and increase the number of
    # output channels
    nn.Conv2d(96, 256, kernel_size=5, padding=2), nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    # Use three successive convolutional layers and a smaller convolution
    # window. Except for the final convolutional layer, the number of output
    # channels is further increased. Pooling layers are not used to reduce the
    # height and width of input after the first two convolutional layers
    nn.Conv2d(256, 384, kernel_size=3, padding=1), nn.ReLU(),
    nn.Conv2d(384, 384, kernel_size=3, padding=1), nn.ReLU(),
    nn.Conv2d(384, 256, kernel_size=3, padding=1), nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Flatten(),
    # Here, the number of outputs of the fully-connected layer is several
    # times larger than that in LeNet. Use the dropout layer to mitigate
    # overfitting
    nn.Linear(6400, 4096), nn.ReLU(),
    nn.Dropout(p=0.5),
    nn.Linear(4096, 4096), nn.ReLU(),
    nn.Dropout(p=0.5),
    # Output layer. Since we are using Fashion-MNIST, the number of classes is
    # 10, instead of 1000 as in the paper
    nn.Linear(4096, 10))
```

[here]

Famous models can be built directly in PyTorch:

```
import torchvision.models as models
alexnet = models.alexnet() # constructs the model with random weights
alexnet = models.alexnet(pretrained=True)
```

ZFNet (2013) – Winner of ILSVRC 2013

- A variant of **AlexNet** enhanced based on visualizing convolutional layers

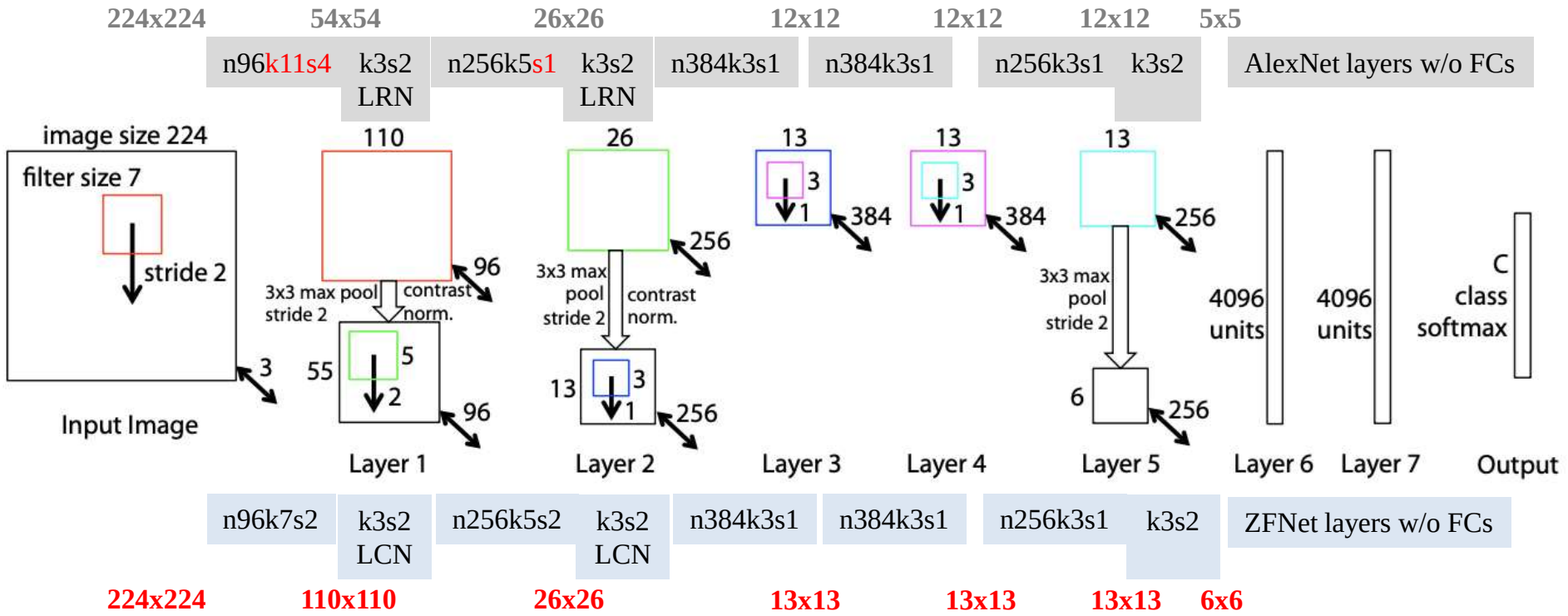


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through 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.

Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." In ECCV, pp. 818-833. Springer, Cham, 2014.

ZFNet (2013) – Winner of ILSVRC 2013

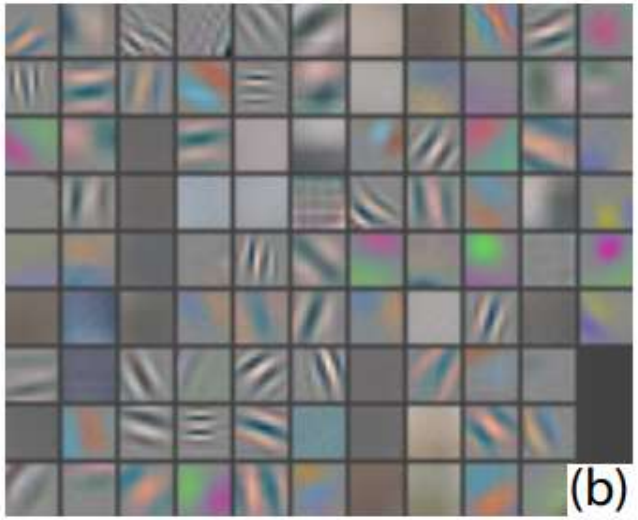
Error %	Val Top-1	Val Top-5	Test Top-5
(Gunji et al., 2012)	-	-	26.2
(Krizhevsky et al., 2012), 1 convnet	40.7	18.2	– –
(Krizhevsky et al., 2012), 5 convnets	38.1	16.4	16.4
(Krizhevsky et al., 2012)*, 1 convnets	39.0	16.6	– –
(Krizhevsky et al., 2012)*, 7 convnets	36.7	15.4	15.3
Our replication of (Krizhevsky et al., 2012), 1 convnet AlexNet	40.5	18.1	– –
1 convnet as per Fig. 3 ZFNet	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**

Table 2. ImageNet 2012 classification error rates. The $*$ indicates models that were trained on both ImageNet 2011 and 2012 training sets.

Layer 1 Filters

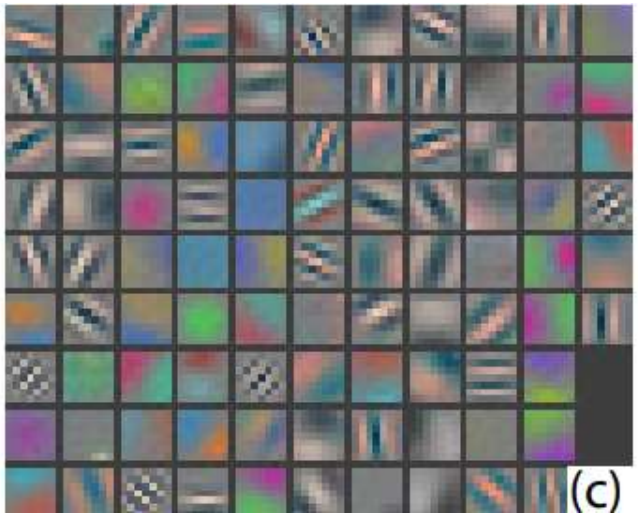
❖ (b) The 1st layer features from AlexNet

*"The first layer filters are **a mix of extremely high and low frequency** information, with little coverage of the mid frequencies."*



❖ (c) The 1st layer features from ZFNet

*"The smaller stride (2 vs 4) and filter size (7x7 vs 11x11) results in **more distinctive features** and **fewer "dead" features**. "*

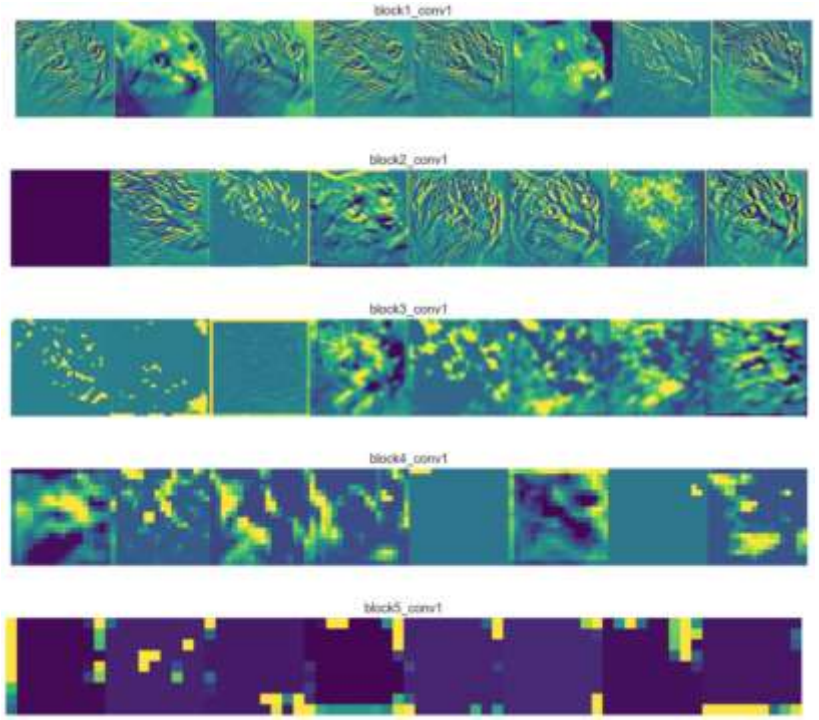


Toward Understanding CNNs

- Filters are usually small (e.g., 3x3, 5x5)
  - visualizing them are not helpful, esp. in deeper layers.

- We can visualize feature maps for each input image:

Low-level features



High-level features (semantic)

Given an input image, some feature maps from 5 layers of a trained VGG network are displayed.

For example, “block1_conv1” layer has 64 feature maps. But, here only the first 8 of them are shown.

[here]

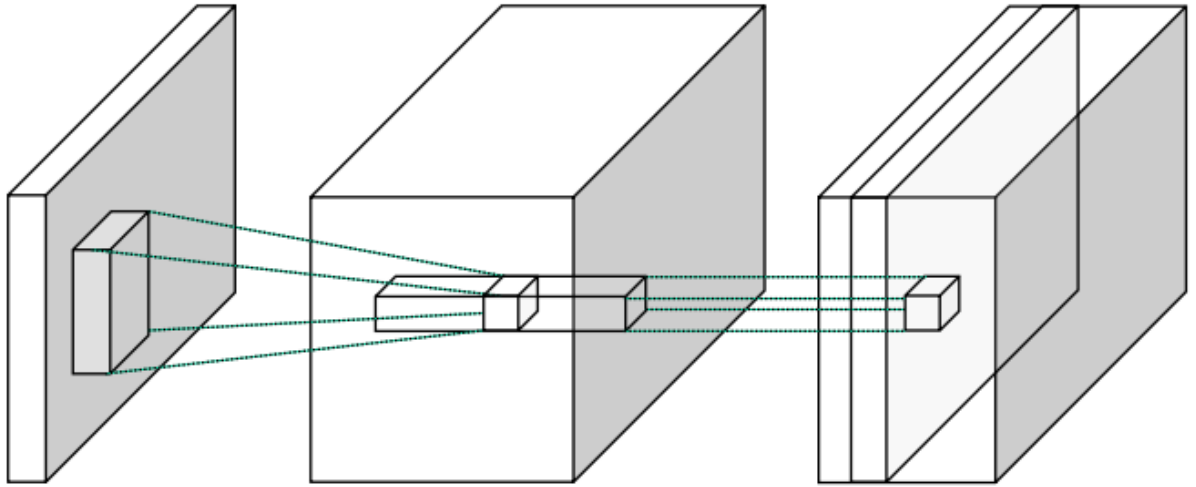
- These are very simple visualization / explanation techniques
- Can’t shed much light on what the model has learned!

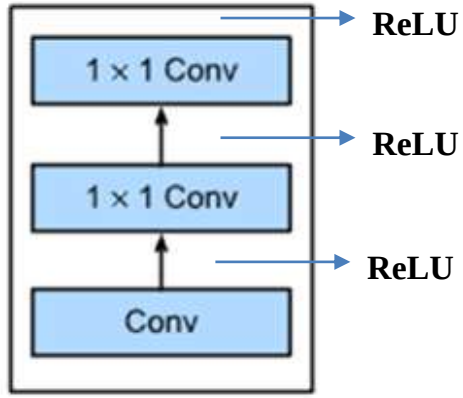
Network in Network (ICLR 2014)

- LeNet, AlexNet, and ZFNet all share a common design pattern:
  - Extract features via a sequence of convolution and pooling
  - Post-process the representation via FC layers to make a global decision.

- Question:
  - **Is there a way to exploit fully connected layers in early layers?**

- *Use an MLP on the channels for each pixel separately.* ⇔ 1x1 Convolution.
  - *Think that each pixel is an example and the corresponding channel tube is the feature.*



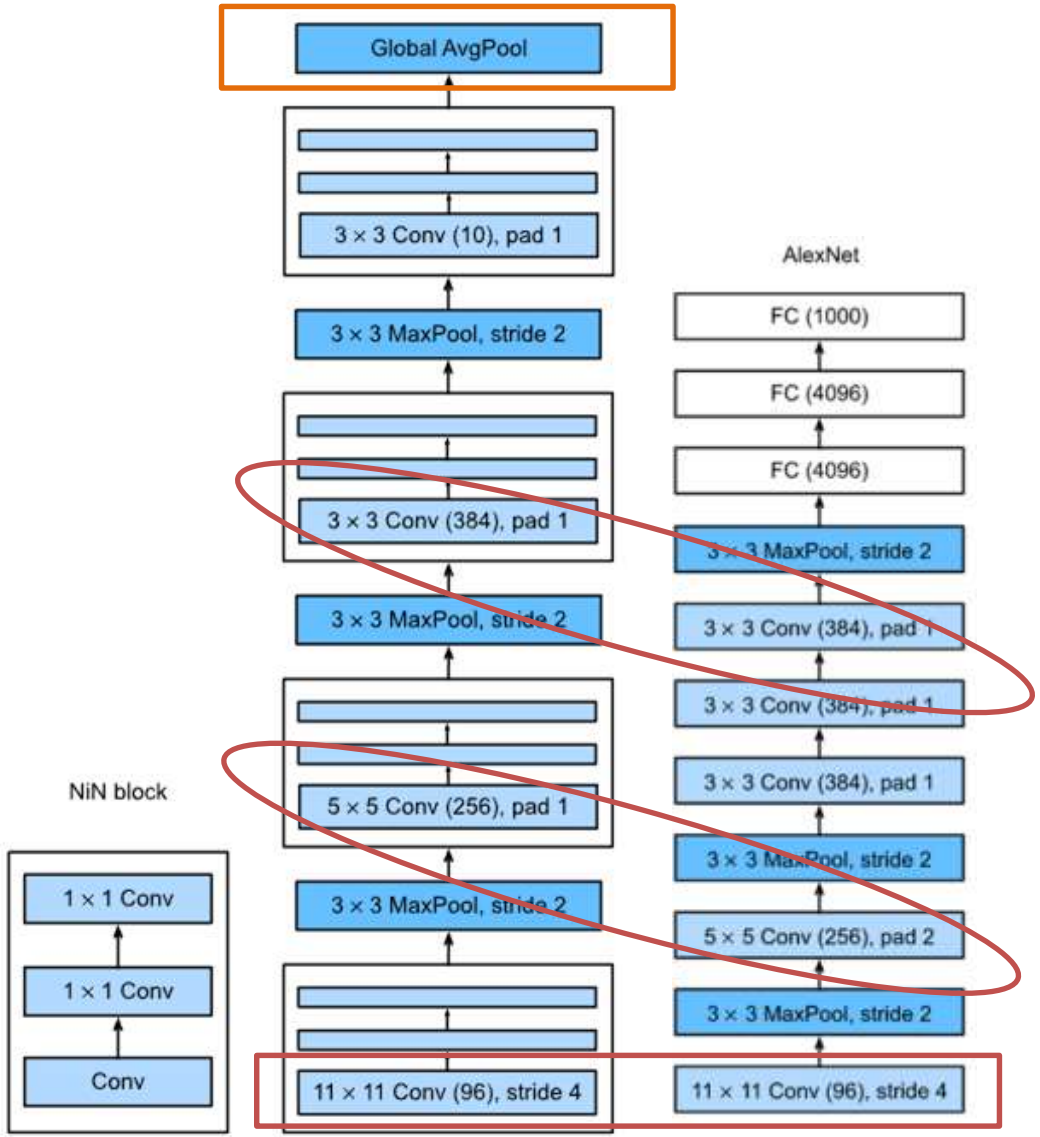


NiN Block / MLPconv layer

[Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." arXiv preprint arXiv:1312.4400 (2013)]

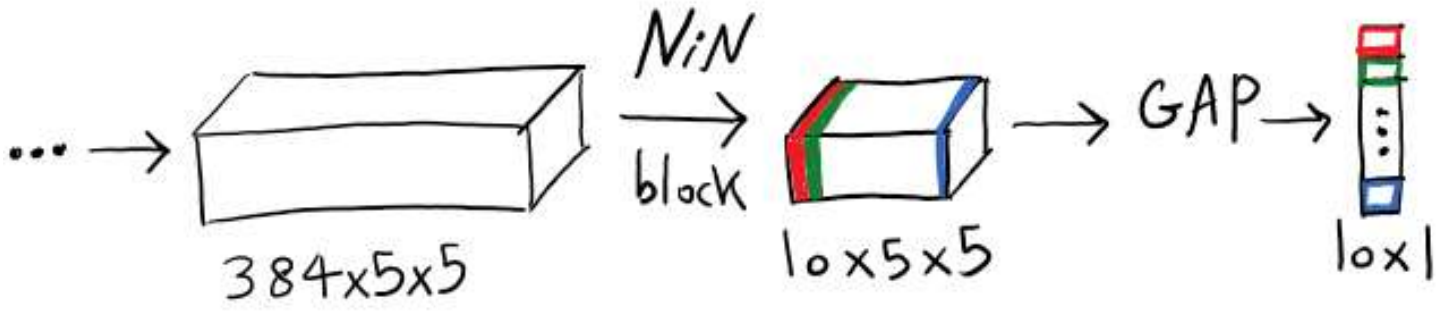
NiN

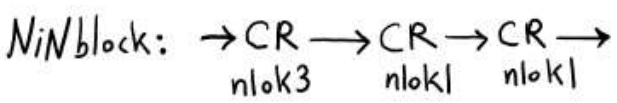
- NiN depth = 12
- Non 1x1 conv. layers are similar to AlexNet w/ the same # of FMs
- NiN blocks increase the power of **nonlinear transformation at pixel level.** (compared w/ linear convolution layer).
- Using 1x1 conv. few parameters are shared across channels
  - Allow for learning inter-channel correlations.
- GAP instead of FC layer at the end
  - Less prone to overfitting

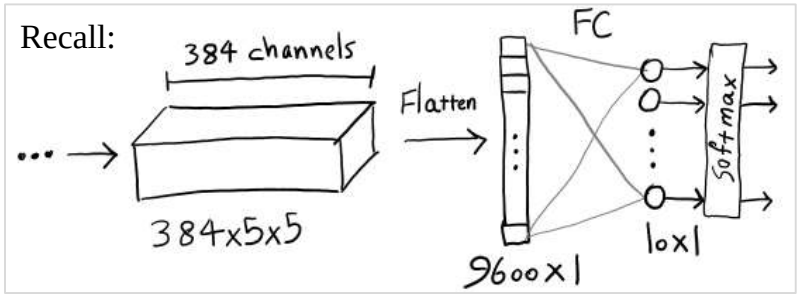


Global Average Pooling (GAP) instead of FC layer(s)

- Generate one feature map for <u>each corresponding category</u> of the classification task







- + No parameter to optimize => reduces the risk of overfitting
- + Because it sums out the spatial information, it is more robust to spatial translations of the input.
- - Each feature map summarized into one number. => too much loss of information!
- + Images with different sizes can be fed into the network.
  - The network relies on the number of channels instead of the input image spatial size.

NiN re-implementation

```
def nin_block(in_channels, out_channels, kernel_size, strides, padding):
    return nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size, strides, padding),
        nn.ReLU(),
        nn.Conv2d(out_channels, out_channels, kernel_size=1), nn.ReLU(),
        nn.Conv2d(out_channels, out_channels, kernel_size=1), nn.ReLU())
```

Training procedure was like AlexNet. [See NiN paper]

```
net = nn.Sequential(
    nin_block(1, 96, kernel_size=11, strides=4, padding=0),
    nn.MaxPool2d(3, stride=2),
    nin_block(96, 256, kernel_size=5, strides=1, padding=2),
    nn.MaxPool2d(3, stride=2),
    nin_block(256, 384, kernel_size=3, strides=1, padding=1),
    nn.MaxPool2d(3, stride=2),
    nn.Dropout(0.5),
    # There are 10 label classes
    nin_block(384, 10, kernel_size=3, strides=1, padding=1),
    nn.AdaptiveAvgPool2d((1, 1)),
    # Transform the four-dimensional output into two-dimensional output with a
    # shape of (batch size, 10)
    nn.Flatten())
```

Dropout [in the original model]

Dropout

FC layer is replaced with a GAP.

NiN Results At a Glance

- NiN mainly tested on small datasets:
  - CIFAR-10, CIFAR-100, Street View House Numbers, and MNIST

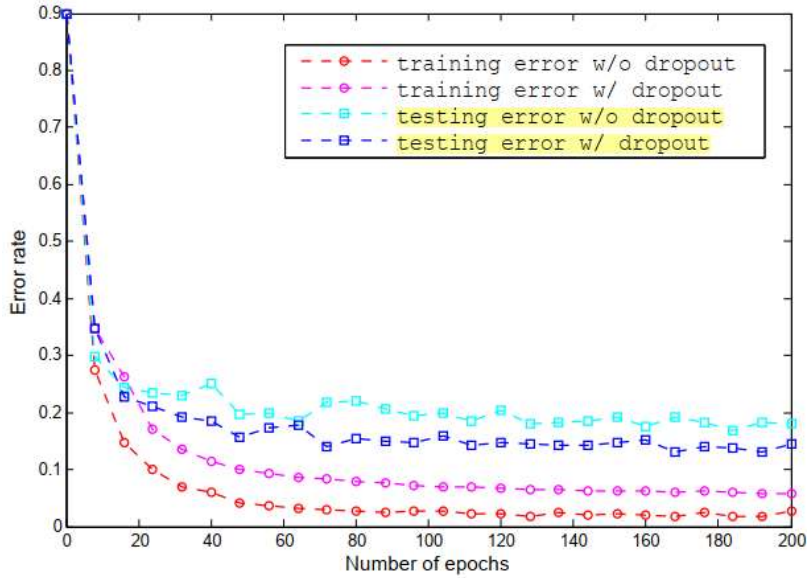
Method	Test Error
Stochastic Pooling [11]	15.13%
CNN + Spearmint [14]	14.98%
Conv. maxout + Dropout [8]	11.68%
NIN + Dropout	**10.41%**
CNN + Spearmint + Data Augmentation [14]	9.50%
Conv. maxout + Dropout + Data Augmentation [8]	9.38%
DropConnect + 12 networks + Data Augmentation [15]	9.32%
NIN + Dropout + Data Augmentation	**8.81%**

Table 1: Test set error rates for CIFAR-10 of various methods

- Regularization effects of Dropout and GAP

Table 5: GAP compared to FC layer.

Method	Testing Error
mlpconv + Fully Connected	11.59%
mlpconv + Fully Connected + Dropout	10.88%
mlpconv + Global Average Pooling	10.41%



Global Average Pooling (GAP) instead of FC layer(s)

- Enforces correspondences between feature maps and categories

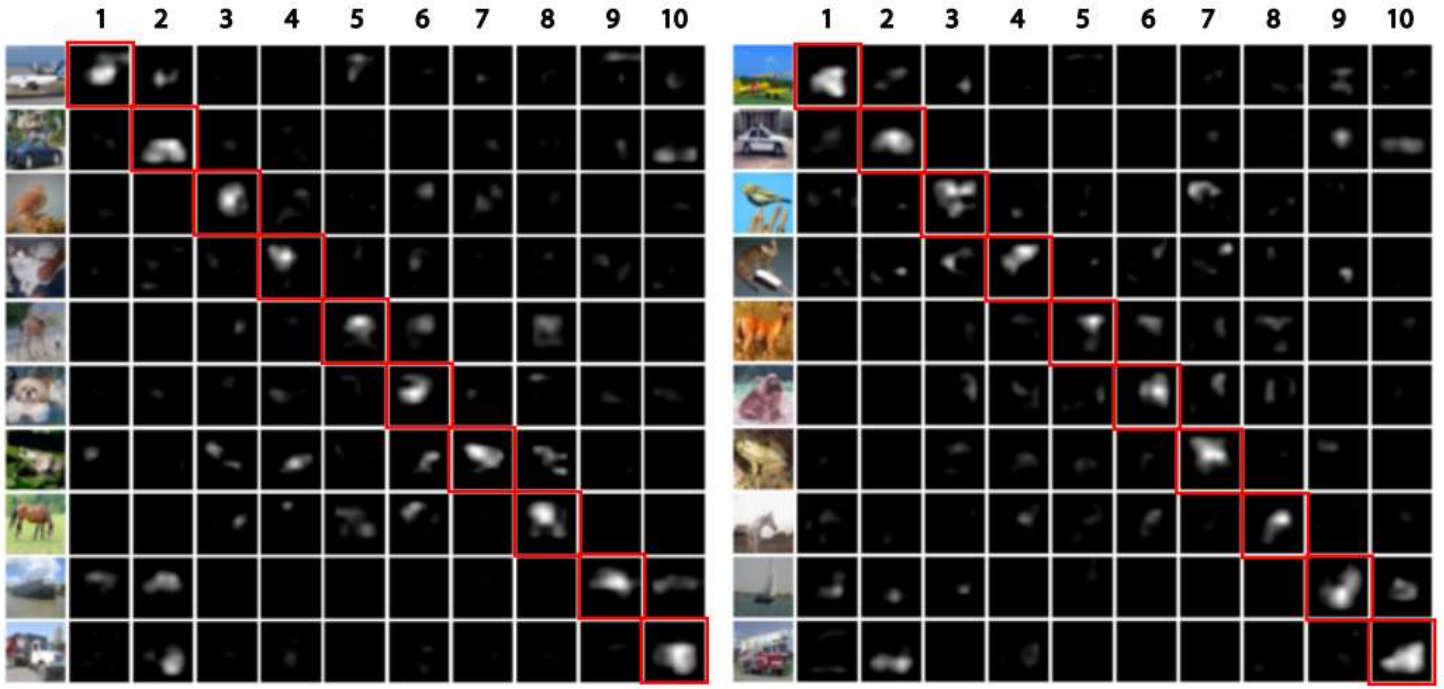


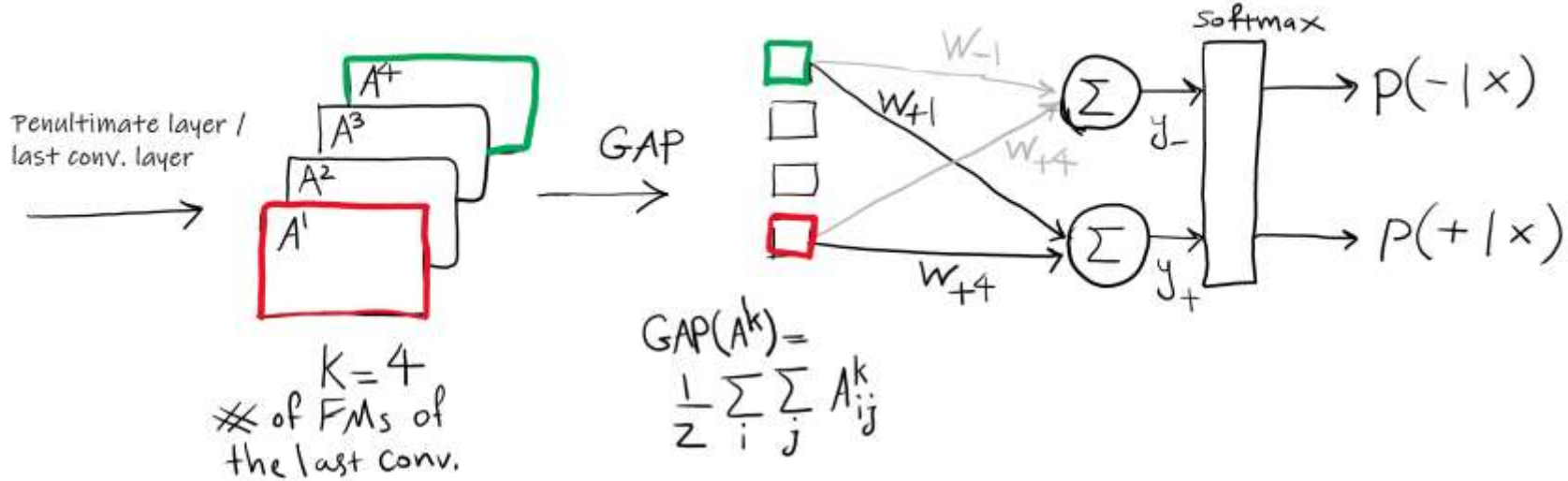
Figure 4: Visualization of the feature maps from the last mlpconv layer. Only top 10% activations in the feature maps are shown. The categories corresponding to the feature maps are: 1. airplane, 2. automobile, 3. bird, 4. cat, 5. deer, 6. dog, 7. frog, 8. horse, 9. ship, 10. truck. Feature maps corresponding to the ground truth of the input images are highlighted. The left panel and right panel are just different examplars.

"Some examplar images and their corresponding feature maps for each of the ten categories selected from CIFAR-10 test set."

=> Saliency Map: an image that highlights the region on which people's eyes focus first.

Extra: **Class Activation Mapping (CAM)**

- Using GAP, CAM was proposed by MIT.
- Object **localization** and image **classification** are performed in a single-forward pass.



① prediction: $p(c|x) = \text{softmax}(y_c) = \text{softmax}\left(\sum_{k=1}^{K=4} w_{ck}\, GAP(A^k) + b_c\right)$

② Heatmap: $L_c = \sum_{k=1}^{K} w_{ck} A^k$

Zhou, Bolei, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. "Learning deep features for discriminative localization." In Proceedings of the IEEE **CVPR**, pp. 2921-2929. **2016**.

Extra: Class Activation Mapping (CAM)

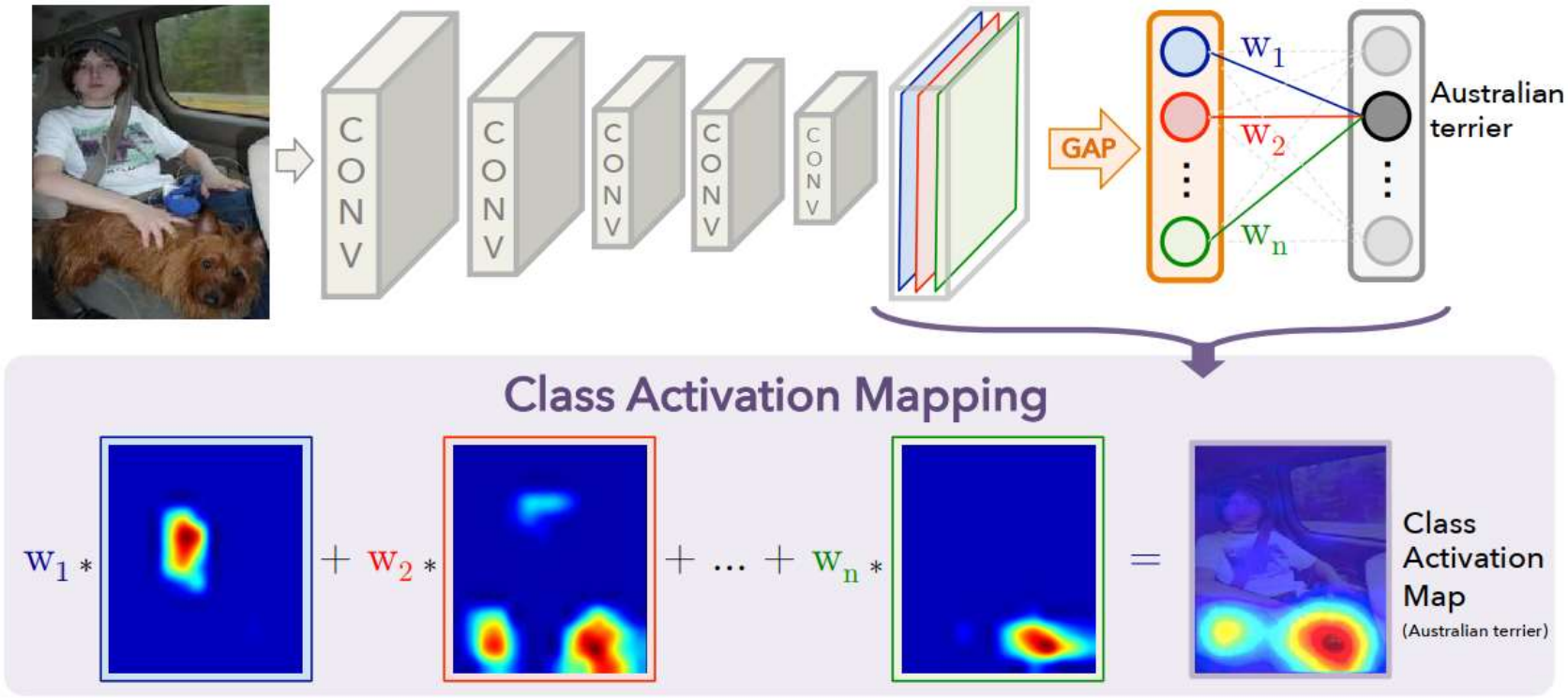
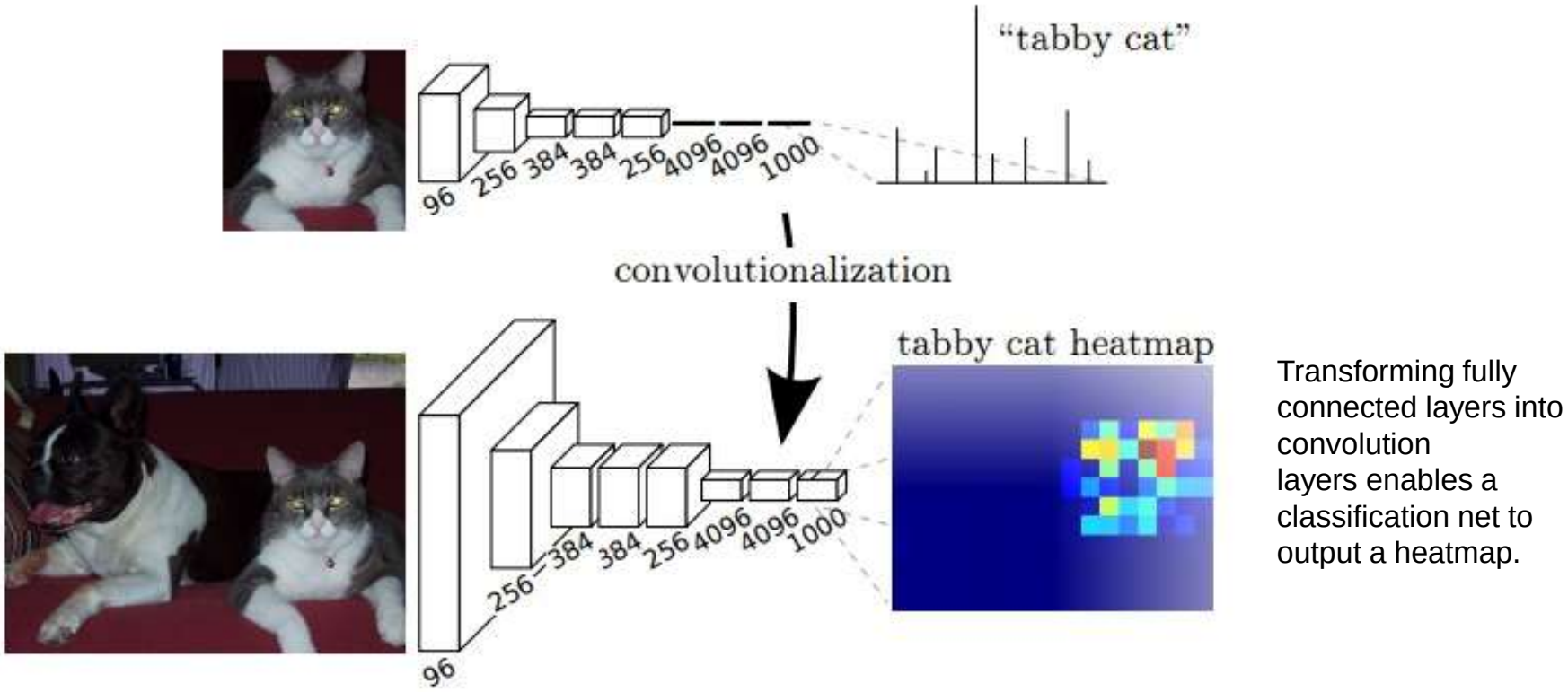


Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

- Also, called, *weakly-supervised object localization*.
- Other techniques: GradCAM, Grad-CAM+, etc.

Extra: The Birth of **Fully Convolutional Networks (FCNs)**

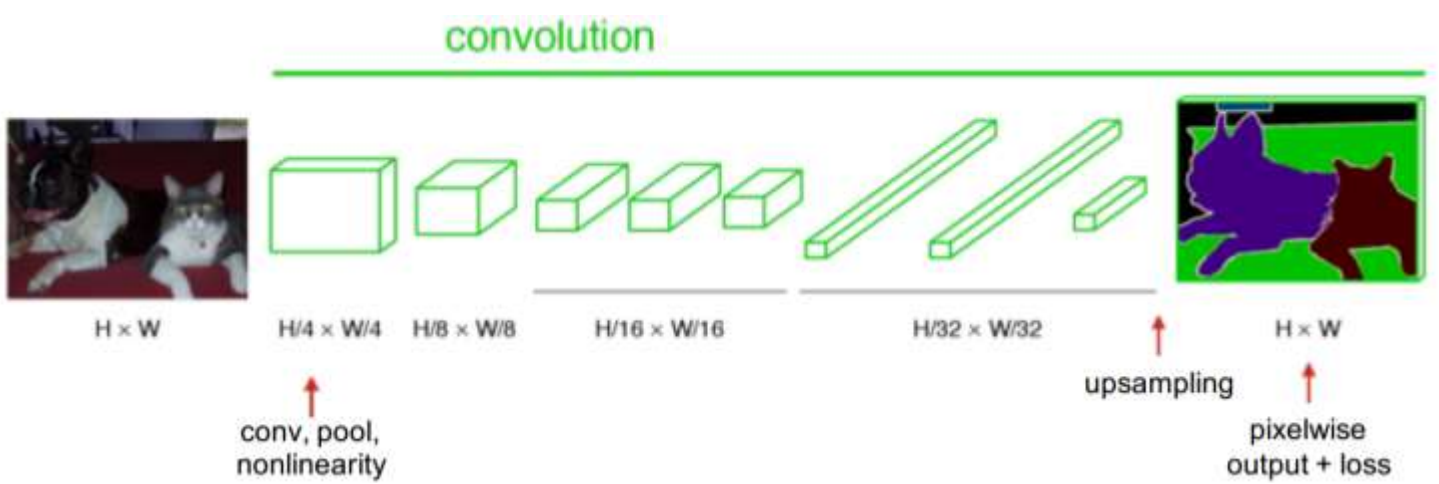
- Because fully connected layers can be implemented in terms of 1x1 convolutions, it enables us to develop **Fully Convolutional Networks** (FCNs).

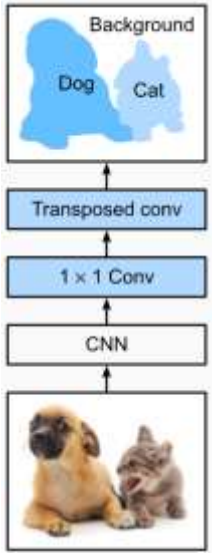


Transforming fully connected layers into convolution layers enables a classification net to output a heatmap.

Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." In Proceedings of the IEEE **CVPR**, pp. 3431-3440. **2015**.

Extra: The Birth of Fully Convolutional Networks (FCNs)





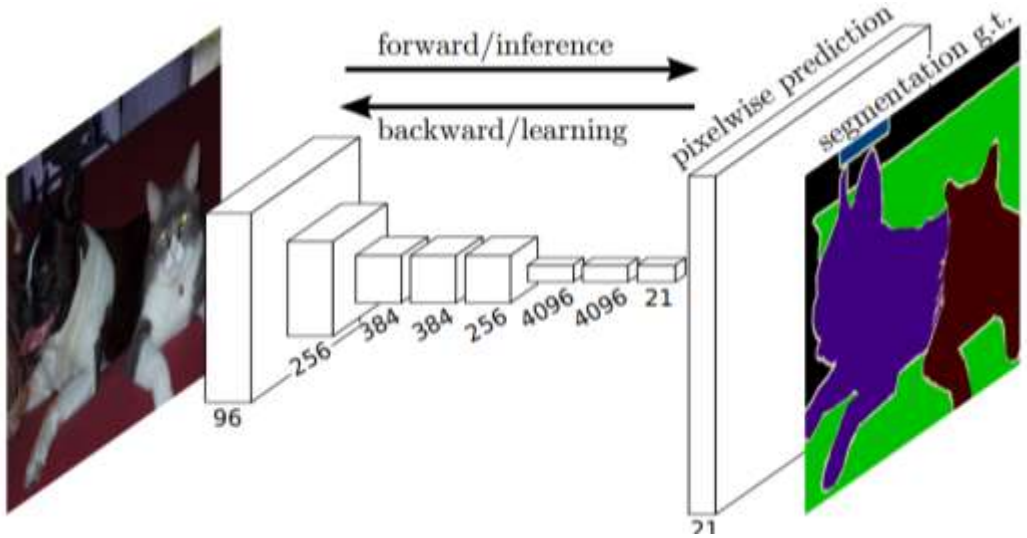


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

Shortcoming of this approach:

Predictions are not fine due to multiple down sampling of the feature maps. => skip connections can help [refer to the paper]

Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." In Proceedings of the IEEE **CVPR**, pp. 3431-3440. **2015**.