MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

CENG 403 Introduction to Deep Learning

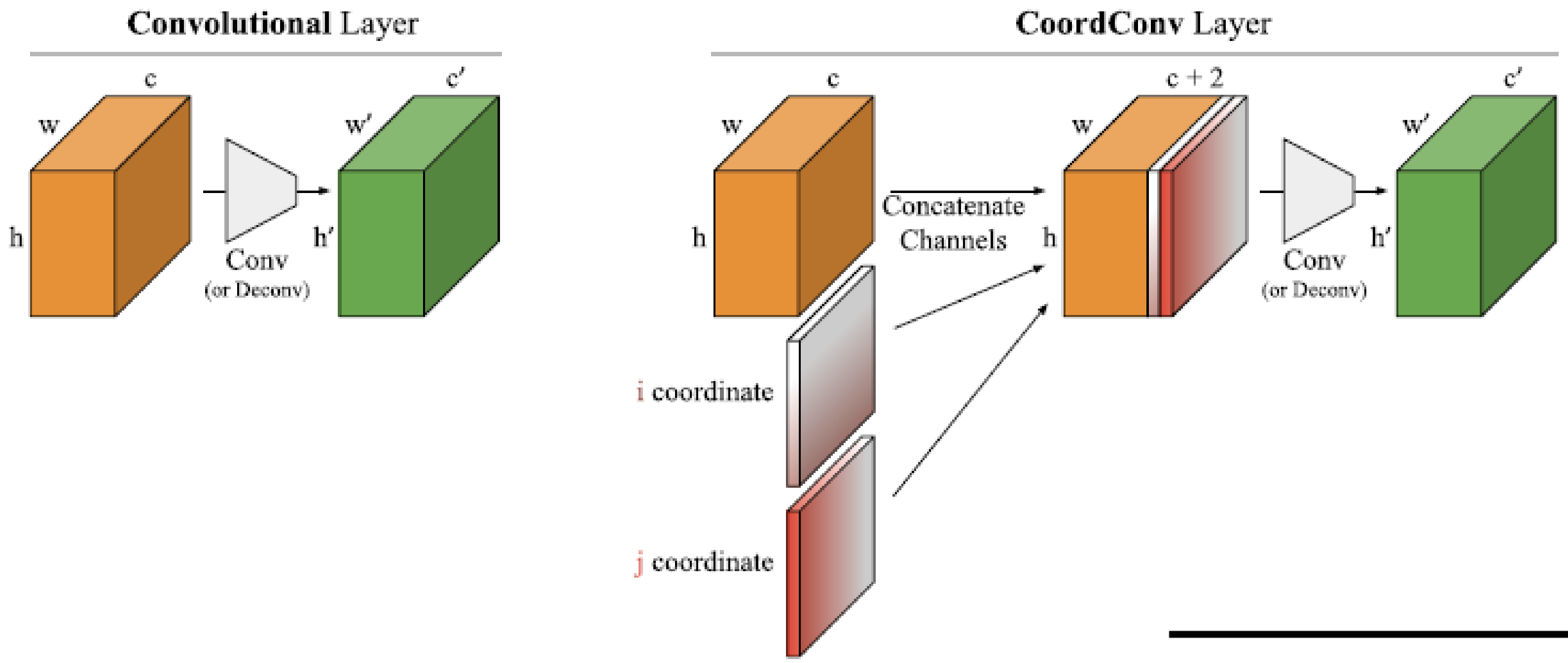
Week 10b

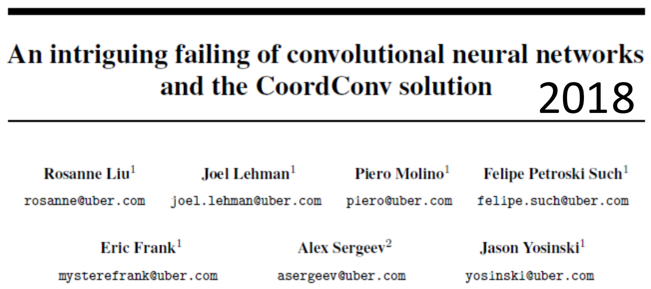
Sinan Kalkan

Previously on CENG403

Types of Convolution: Position-sensitive convolution

- Learn to use position information when necessary




An intriguing failing of convolutional neural networks and the CoordConv solution

2018

Rosanne Liu[1] rosanne@uber.com
Joel Lehman[1] joel.lehman@uber.com
Piero Molino[1] piero@uber.com
Felipe Petroski Such[1] felipe.such@uber.com
Eric Frank[1] mysterefrank@uber.com
Alex Sergeev[2] asergeev@uber.com
Jason Yosinski[1] yosinski@uber.com

Previously on CENG403

Pooling

- Example
  - Pooling layer with filters of size 2x2
  - With stride = 2
  - Discards 75% of the activations
  - Depth dimension remains unchanged
- Max pooling with F=3, S=2 or F=2, S=2 are quite common.
  - Pooling with bigger receptive field sizes can be destructive
- Avg pooling is an obsolete choice. Max pooling is shown to work better in practice.

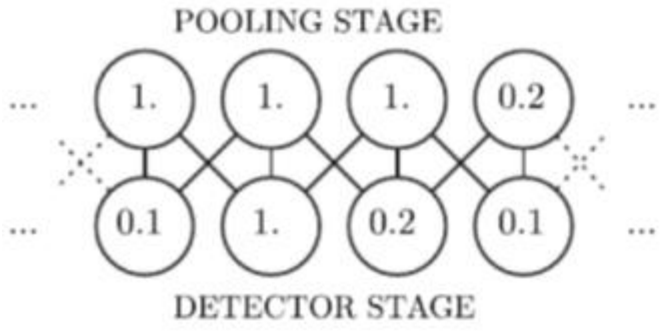
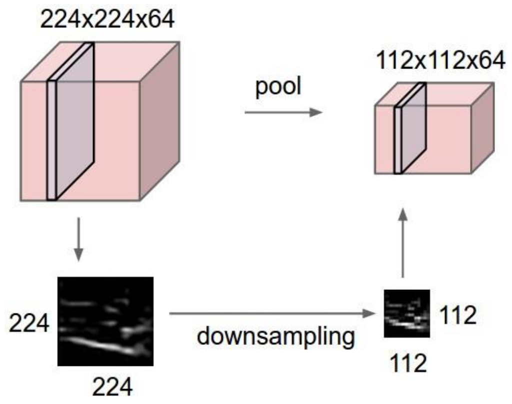
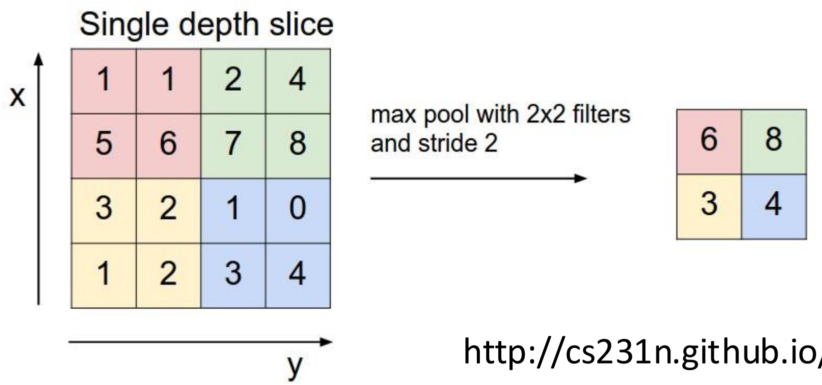


Figure: Goodfellow et al., "Deep Learning", MIT Press, 2016.





1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

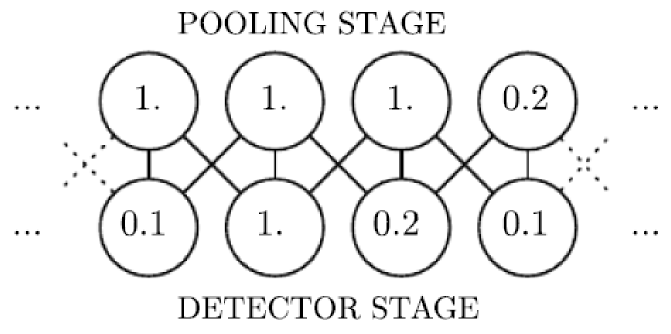
6	8
3	4

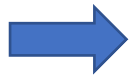
http://cs231n.github.io/convolutional-networks/

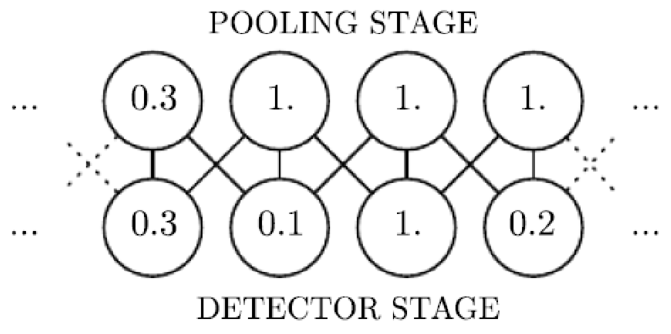
Pooling

Previously on CENG403

- Pooling provides invariance to small translation.




Shifted to right



Figures: Goodfellow et al., "Deep Learning", MIT Press, 2016.

- If you pool over different convolution operators, you can gain invariance to different transformations.

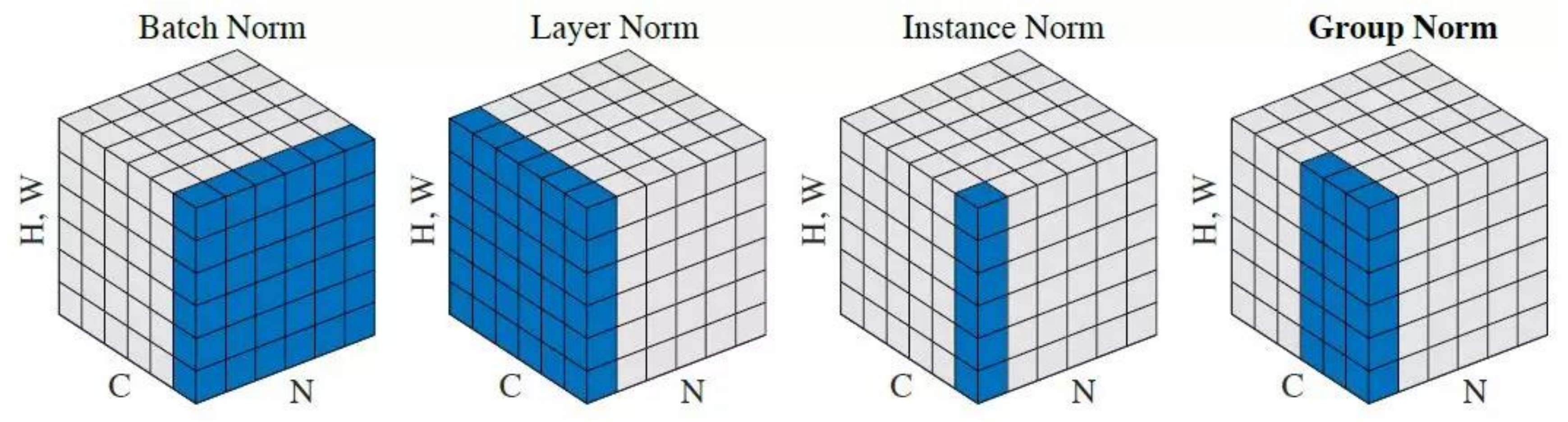
Previously on CENG403

Non-linearity

- Sigmoid
- Tanh
- ReLU and its variants
  - The common choice
  - Faster
  - Easier (in backpropagation etc.)
  - Avoids saturation issues
- …

Previously on CENG403

Normalization



$$\mu_c = \frac{1}{N \cdot H \cdot W} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw}$$

$$\sigma_c^2 = \frac{1}{N \cdot H \cdot W} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_c)^2$$

$$\hat{x}_{nchw} = \frac{x_{nchw} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}$$

$$\mu_n = \frac{1}{C \cdot H \cdot W} \sum_{c=1}^{C} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw}$$

$$\sigma_n^2 = \frac{1}{C \cdot H \cdot W} \sum_{c=1}^{C} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_n)^2$$

$$\hat{x}_{nchw} = \frac{x_{nchw} - \mu_n}{\sqrt{\sigma_n^2 + \epsilon}}$$

$$\mu_{nc} = \frac{1}{H \cdot W} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw}$$

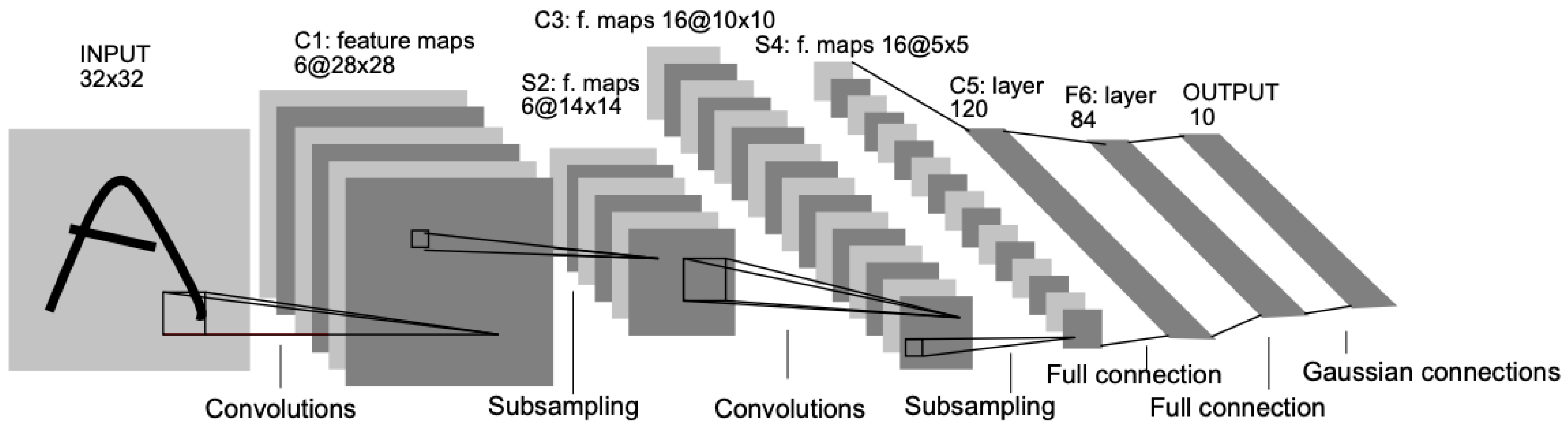
$$\sigma_{nc}^2 = \frac{1}{H \cdot W} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_{nc})^2$$

$$\hat{x}_{nchw} = \frac{x_{nchw} - \mu_{nc}}{\sqrt{\sigma_{nc}^2 + \epsilon}}$$

Figure: https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffae7

Fully-connected layer

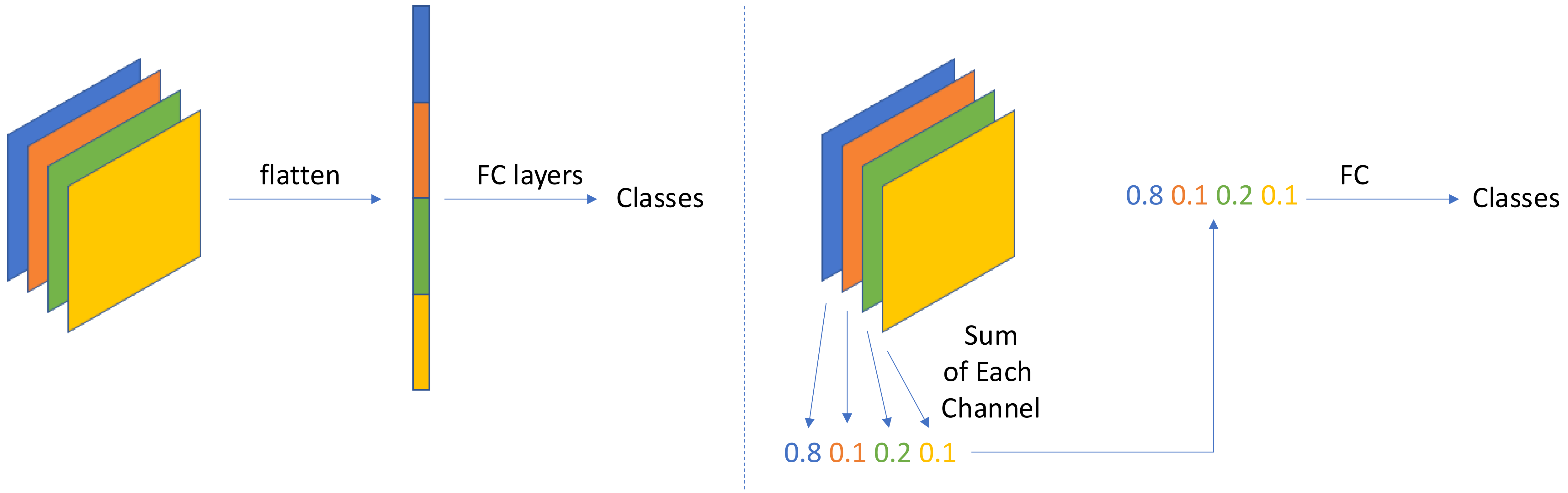
- At the top of the network for mapping the feature responses to output labels
- Full connectivity
- Can be many layers
- Various activation functions can be used



Previously on CENG403

Alternative to FC: Global Average Pooling

"Network In Network", https://arxiv.org/pdf/1312.4400.pdf



Fully Convolutional Networks (FCNs)

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. CVPR.

- Fully-connected layers limit the input size
- We can convert FC layers to convolution

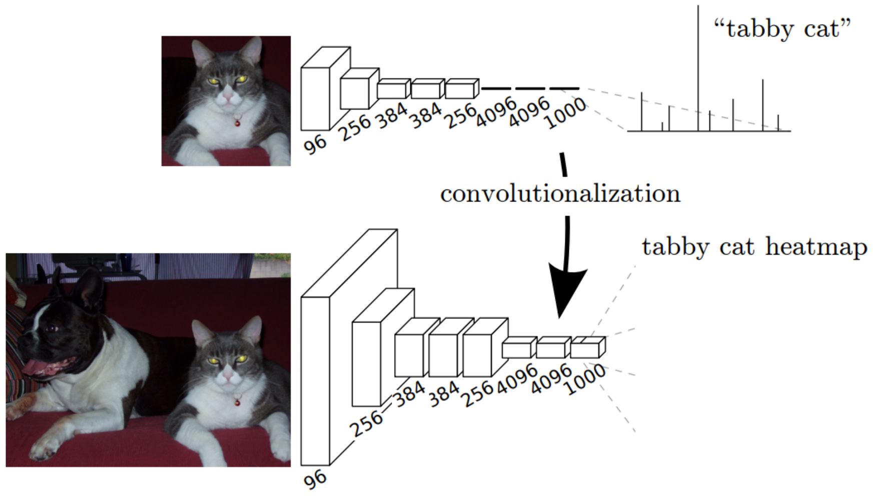


Figure 2. Transforming fully connected layers into convolution layers enables a classification net to output a heatmap. Adding layers and a spatial loss (as in Figure 1) produces an efficient machine for end-to-end dense learning.

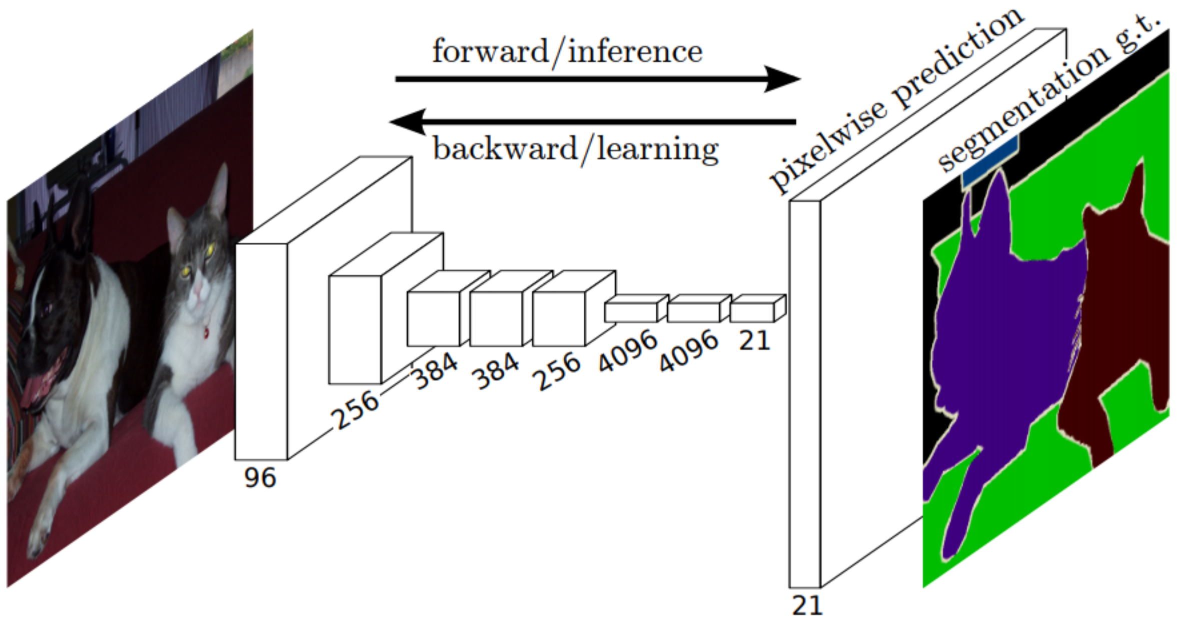


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

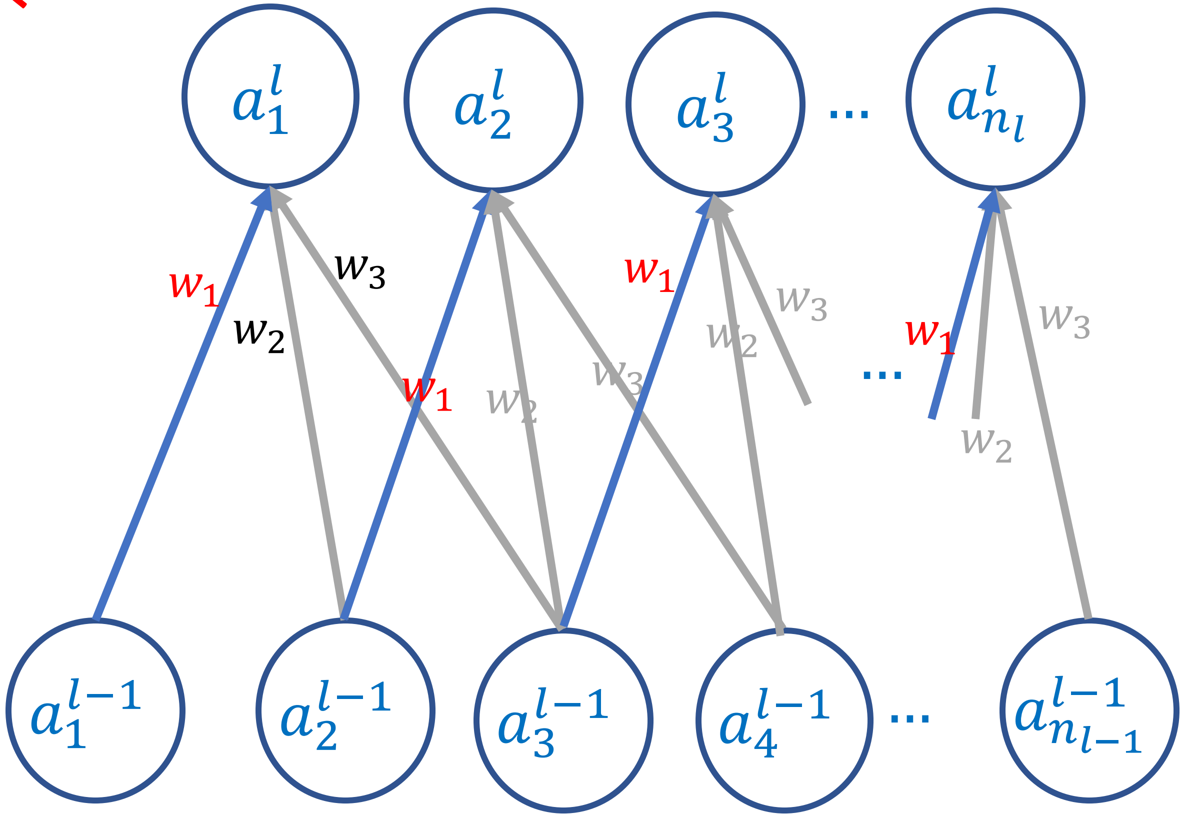
Backpropagation through convolution

Previously on CENG403

Feedforward:

$$a_i^l = \sigma(t_i^l)$$

$$t_i^l = \sum_{j=1}^{F} w_j \cdot a_{i+j-1}^{l-1}$$



Gradient wrt. weights:

$$\frac{\partial L}{\partial w_k} = ?$$

$$= \frac{\partial L}{\partial a_1^l}\frac{\partial a_1^l}{\partial w_k} + \frac{\partial L}{\partial a_2^l}\frac{\partial a_2^l}{\partial w_k} \dots$$

$$= \sum_i \frac{\partial L}{\partial a_i^l}\frac{\partial a_i^l}{\partial w_k}$$

$$= \sum_i \frac{\partial L}{\partial a_i^l}\frac{\partial a_i^l}{\partial t_i^l}\frac{\partial t_i^l}{\partial w_k}$$

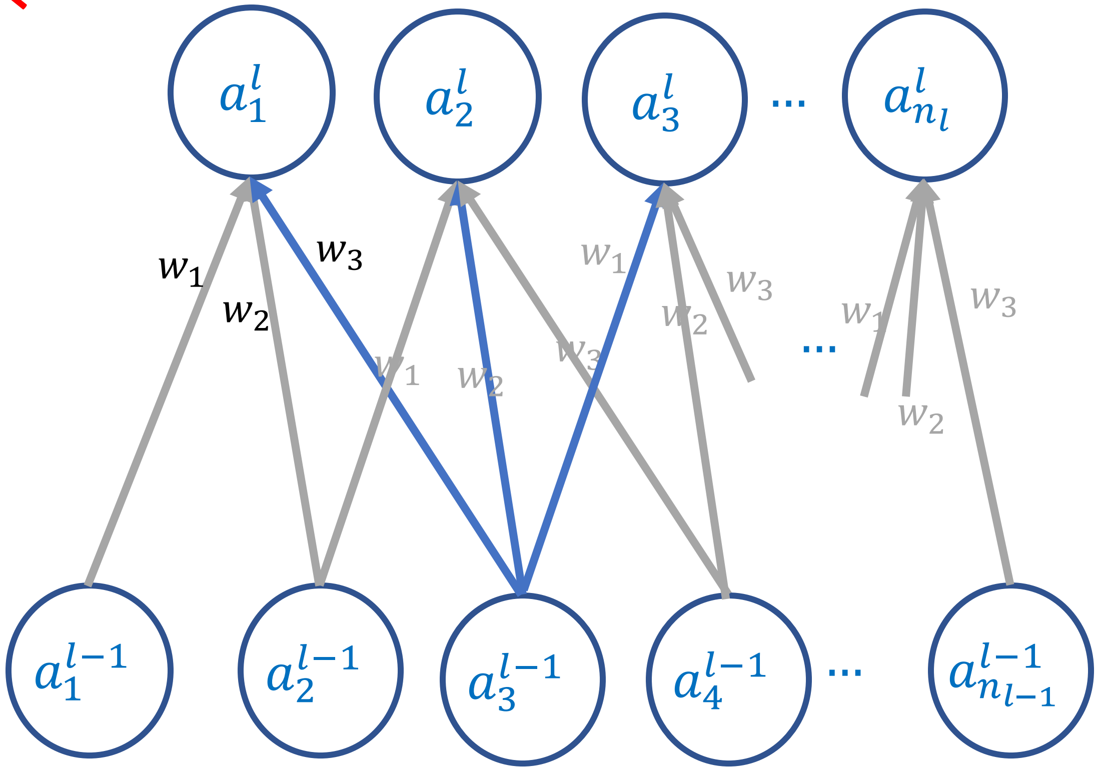
Backpropagation through convolution

Previously on CENG403

Feedforward:

$$a_i^l = \sigma(t_i^l)$$

$$t_i^l = \sum_{j=1}^{F} w_j \cdot a_{i+j-1}^{l-1}$$



Gradient wrt. input layer:

$$\frac{\partial L}{\partial a_3^{l-1}} = ?$$

$$= \frac{\partial L}{\partial a_1^l}\frac{\partial a_1^l}{\partial t_1^l}\frac{\partial t_1^l}{\partial a_3^{l-1}} + \frac{\partial L}{\partial a_2^l}\frac{\partial a_2^l}{\partial t_2^l}\frac{\partial t_2^l}{\partial a_3^{l-1}} + \frac{\partial L}{\partial a_3^l}\frac{\partial a_3^l}{\partial t_3^l}\frac{\partial t_3^l}{\partial a_3^{l-1}}$$

$$= \frac{\partial L}{\partial t_1^l} w_3 + \frac{\partial L}{\partial t_2^l} w_2 + \frac{\partial L}{\partial t_3^l} w_1$$

This is also convolution!

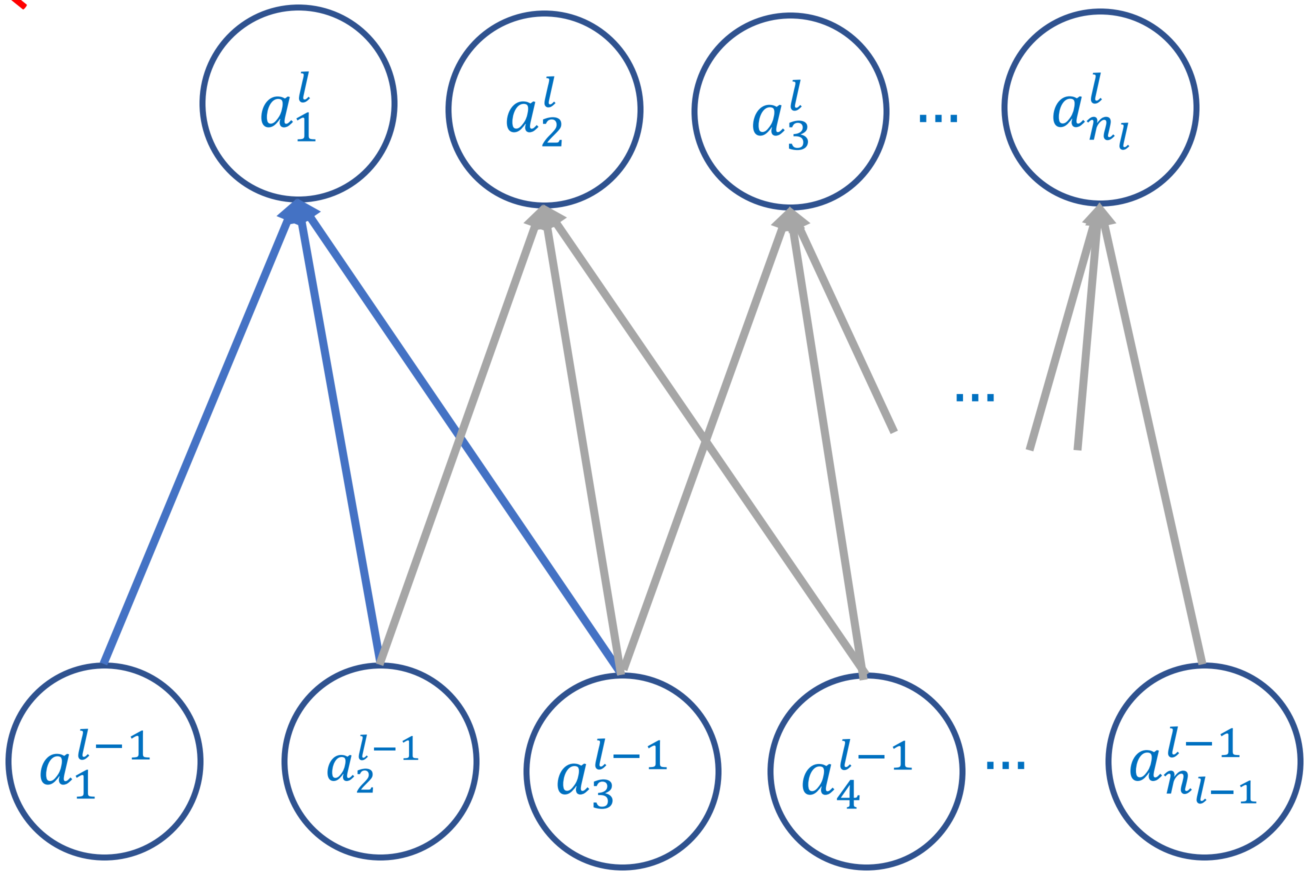
In general:

$$\frac{\partial L}{\partial a_i^{l-1}} = \sum_{j=1} \frac{\partial L}{\partial t_{i-j+1}^l} w_j$$

Backpropagation through pooling

Feedforward:

$$a_i^l = \max\{a_{i+j-1}^{l-1}\}_{j=1}^{F}$$



Using derivative of max:

$$\frac{\partial L}{\partial a_i^{l-1}} = \frac{\partial L}{\partial t_k^l}\frac{\partial t_k^l}{\partial a_i^{l-1}}$$
$$= \begin{cases} \frac{\partial L}{\partial t_k^l}, & a_i^{l-1} \text{ is max} \\ 0, & \text{otherwise} \end{cases}$$

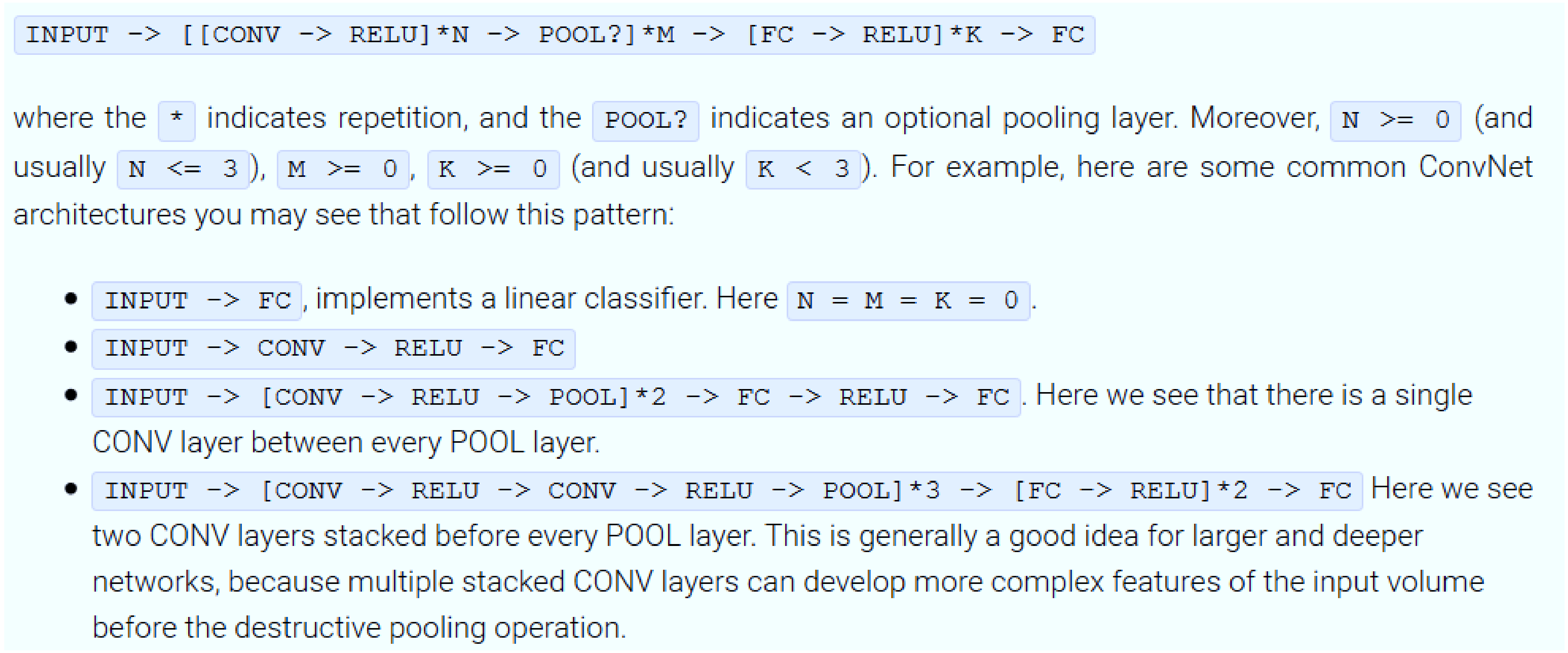
This requires that we save the index of the max activation (sometimes also called *the switches*) so that gradient “routing” is handled efficiently during backpropagation.

Today

- Convolutional Neural Networks (CNNs)
  - Designing CNN architectures
  - Transfer learning
  - Visualizing/understanding CNNs

Designing CNN Architectures

A Blueprint for CNNs



```
INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
```

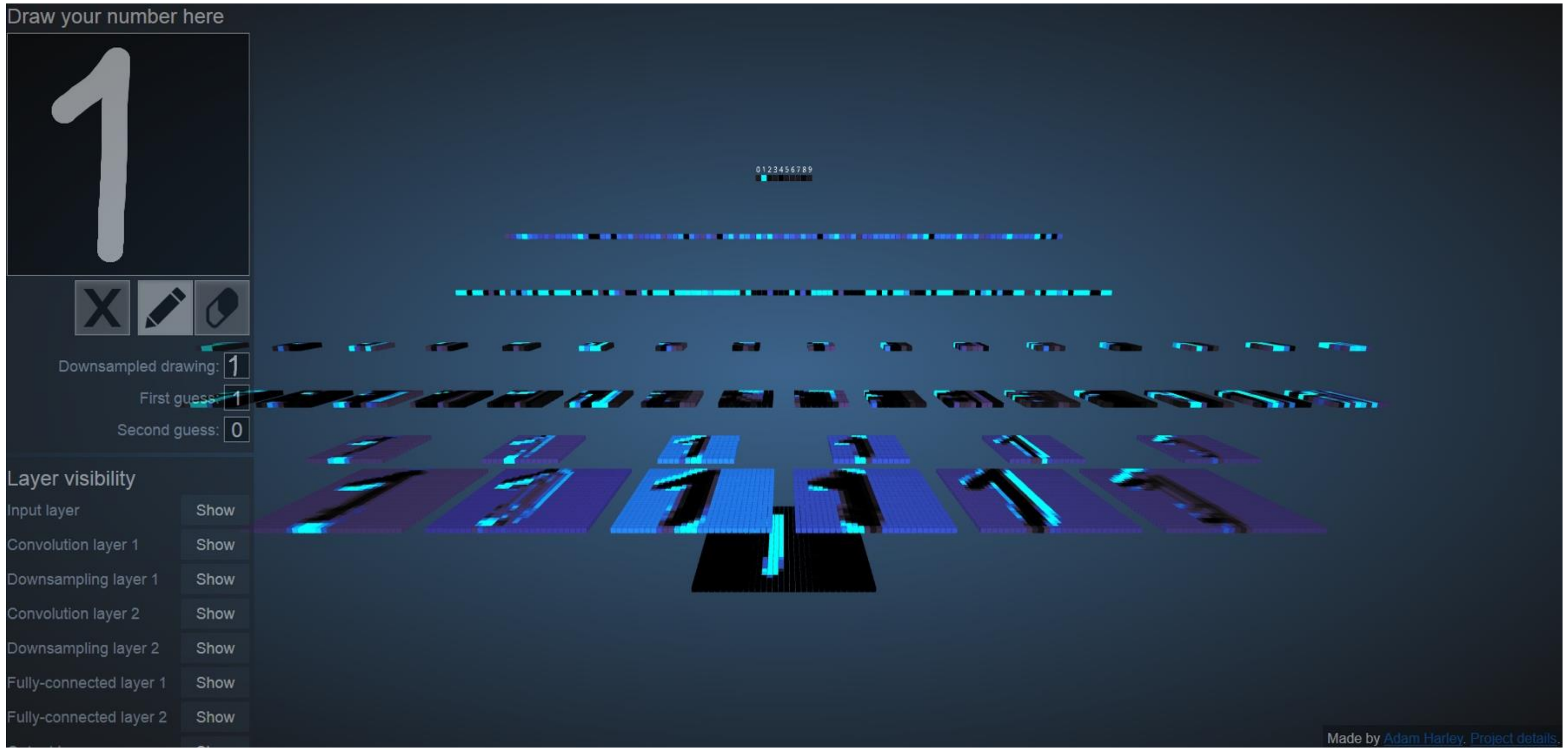
where the `*` indicates repetition, and the `POOL?` indicates an optional pooling layer. Moreover, `N >= 0` (and usually `N <= 3`), `M >= 0`, `K >= 0` (and usually `K < 3`). For example, here are some common ConvNet architectures you may see that follow this pattern:

- `INPUT -> FC`, implements a linear classifier. Here `N = M = K = 0`.
- `INPUT -> CONV -> RELU -> FC`
- `INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC`. Here we see that there is a single CONV layer between every POOL layer.
- `INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*3 -> [FC -> RELU]*2 -> FC` Here we see two CONV layers stacked before every POOL layer. This is generally a good idea for larger and deeper networks, because multiple stacked CONV layers can develop more complex features of the input volume before the destructive pooling operation.

http://cs231n.github.io/convolutional-networks/

Demo

https://adamharley.com/nn_vis/cnn/3d.html



Fully Convolutional Networks (FCNs)

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. CVPR.

- Fully-connected layers limit the input size
- We can convert FC layers to convolution

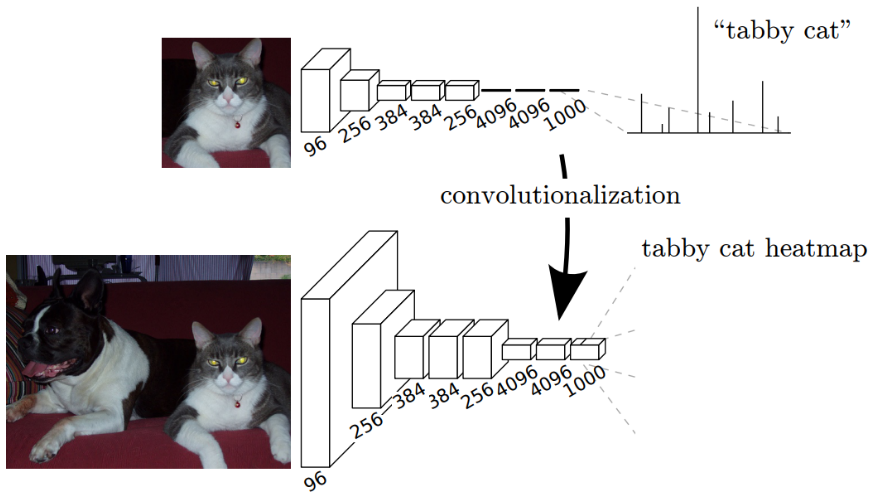


Figure 2. Transforming fully connected layers into convolution layers enables a classification net to output a heatmap. Adding layers and a spatial loss (as in Figure 1) produces an efficient machine for end-to-end dense learning.

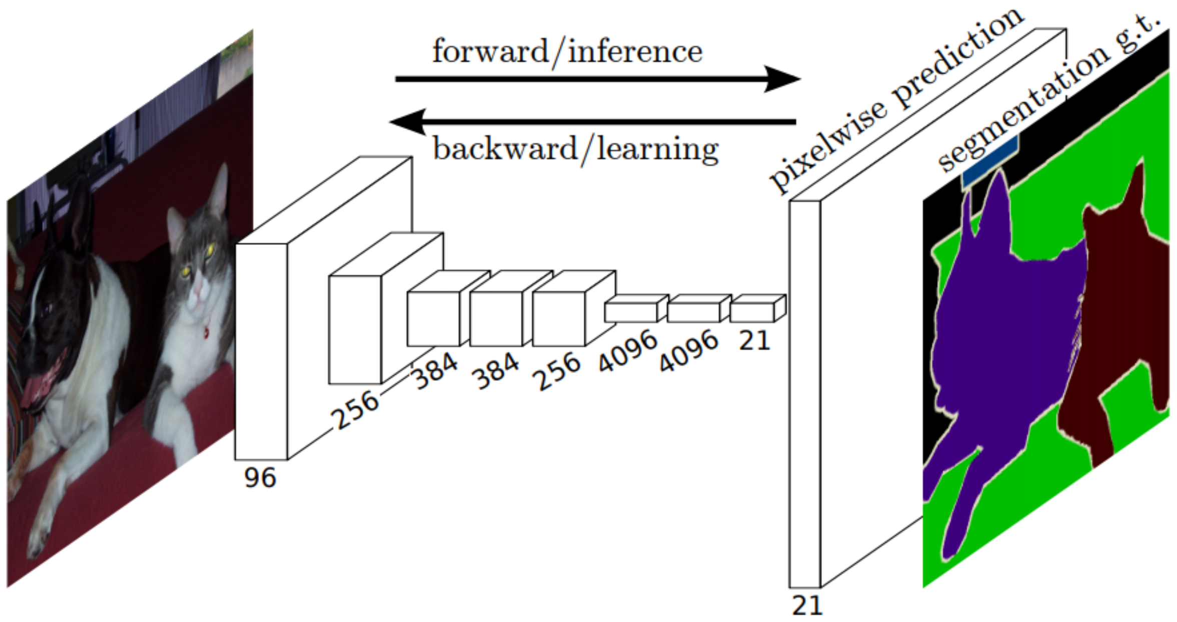


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

General rules of thumb: The input layer

- The size of the input layer should be divisible by 2 many times
  - Hopefully a power of 2
- E.g.,
  - 32 (e.g. CIFAR-10),
  - 64,
  - 96 (e.g. STL-10), or
  - 224 (e.g. common ImageNet ConvNets),
  - 384, and 512 etc.

General rules of thumb: The conv layer

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

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

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

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

- Number of filters:
    - A convolution channel is slower compared to fully-connected layer.
    - This can be a limiting factor in determining the number of filters.

General rules of thumb: The pooling layer

- Commonly,
  - F=2 with S=2
  - Or: F=3 with S=2

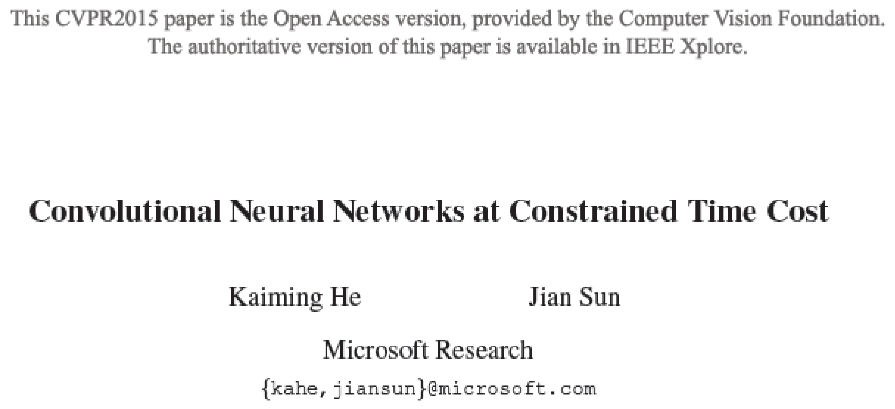
- Bigger F or S is very destructive

Taking care of downsampling

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

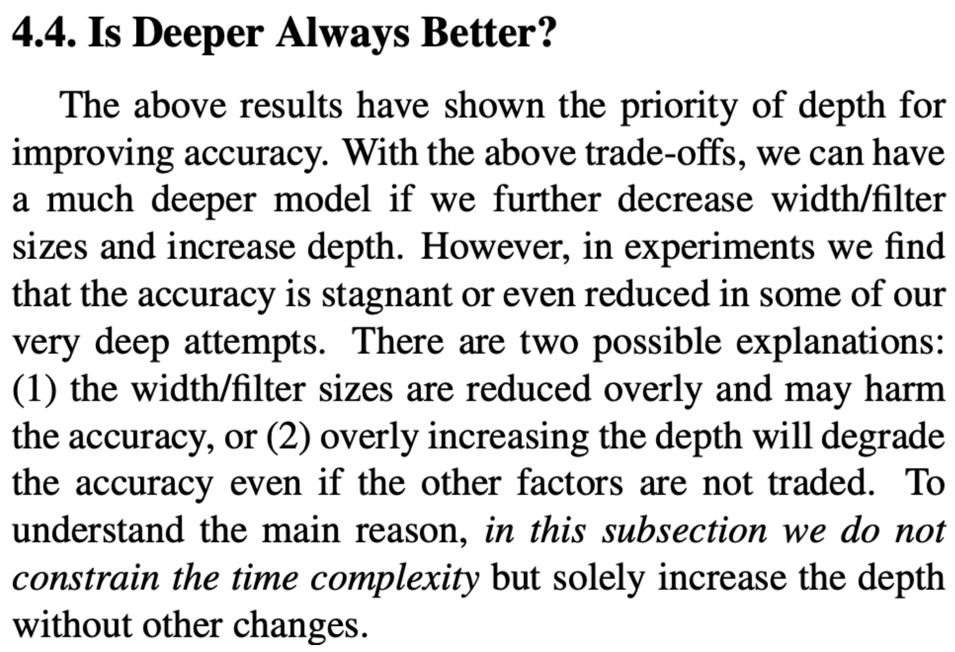
Trade-offs in architecture

- Between filter size and number of layers (depth)
  - Keep the layer widths fixed.
  - *"When the time complexity is roughly the same, the deeper networks with smaller filters show better results than the shallower networks with larger filters."*
- Between layer width and number of layers (depth)
  - Keep the size of the filters fixed.
  - *"We find that increasing the depth leads to considerable gains..."*
- Between filter size and layer width
  - Keep the number of layers (depth) fixed.
  - No significant difference



Convolutional Neural Networks at Constrained Time Cost

Kaiming He Jian Sun
Microsoft Research
{kahe, jiansun}@microsoft.com



4.4. Is Deeper Always Better?

The above results have shown the priority of depth for improving accuracy. With the above trade-offs, we can have a much deeper model if we further decrease width/filter sizes and increase depth. However, in experiments we find that the accuracy is stagnant or even reduced in some of our very deep attempts. There are two possible explanations: (1) the width/filter sizes are reduced overly and may harm the accuracy, or (2) overly increasing the depth will degrade the accuracy even if the other factors are not traded. To understand the main reason, *in this subsection we do not constrain the time complexity* but solely increase the depth without other changes.

Memory

Main sources of memory load:

- Activation maps:
  - Training: They need to be kept during training so that backpropagation can be performed
  - Testing: No need to keep the activations of earlier layers
- Parameters:
  - The weights, their gradients and also another copy if momentum is used
- Data:
  - The originals + their augmentations
- If all these don’t fit into memory,
  - Load your data batch by batch from disk
  - Decrease the size of your batches

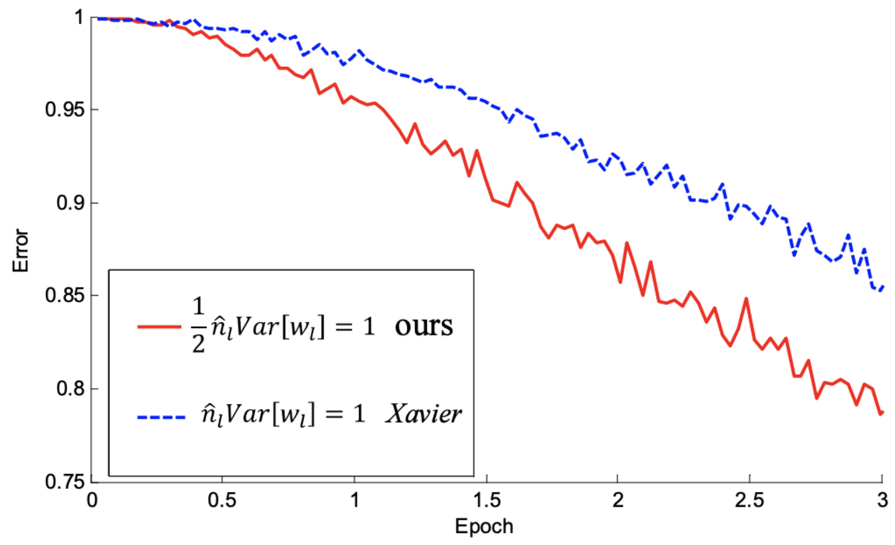
Memory constraints

- Using smaller RFs with more layers means more memory since you need to store more activation maps
- In such memory-scarce cases,
  - the first layer may use bigger RFs with S>1
  - information loss from the input volume may be less critical than the following layers
- E.g., AlexNet uses RFs of 11x11 and S = 4 for the first layer.

How to initialize the weights?

- Option 1: randomly
  - E.g. using He initialization
  - This has been shown to work well in the literature

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



He et al., "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", 2015.

Convolutional Clustering for Unsupervised Learning

Aysegul Dundar, Jonghoon Jin, and Eugenio Culurciello
Purdue University, West Lafayette, IN 47907, USA
{adundar, jhjin, euge}@purdue.edu

Table 3: Classification error on MNIST.

(a) Algorithms that learn the filters unsupervised.

Algorithm	600	1000	3000	All
Zhao et al. (2015) (auto-encoder)	8.4%	6.40%	4.76%	-
Rifai et al. (2011) (constractive auto-encoder)	6.3%	4.77%	3.22%	1.14%
This work (2 layers + multi dict.)	2.8%	2.5%	1.4%	0.5%

(b) Supervised and semi-supervised algorithms.

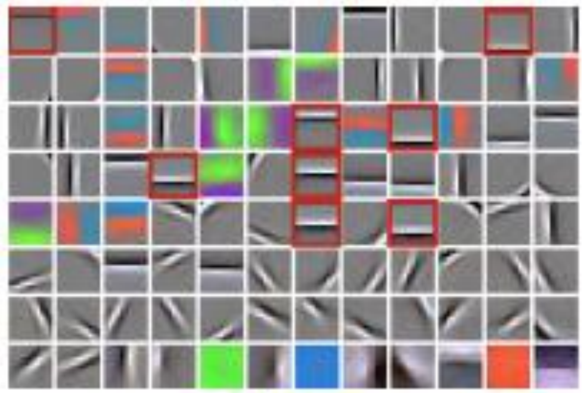
Algorithm	600	1000	3000	All
LeCun et al. (1998) (convnet)	7.68%	6.45%	3.35%	
Lee (2013) (psuedo-label)	5.03%	3.46%	2.69%	-
Zhao et al. (2015) (semi-supervised auto-encoder)	3.31%	2.83%	2.10%	0.71%
Kingma et al. (2014) (generative models)	2.59%	2.40%	2.18%	0.96%
Rasmus et al. (2015) (semi-supervised ladder)	-	1.0%	-	-

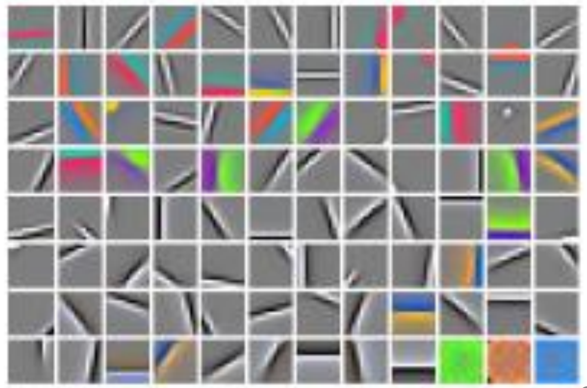
3.1 Learning Filters with K-Means

Our method for learning filters is based on the k-means algorithm. The classic k-means algorithm finds cluster centroids that minimize the distance between points in the Euclidean space. In this context, the points are randomly extracted image patches and the centroids are the filters that will be used to encode images. From this perspective, k-means algorithm learns a dictionary $D \in \mathbb{R}^{n\times k}$ from the data vector $w^{(i)} \in \mathbb{R}^n$ for $i = 1, 2, ..., m$. The algorithm finds the dictionary as follows:

$$\begin{aligned} s_j^{(i)} &:= \begin{cases} {D^{(j)}}^T w^{(i)} & \text{if } j = \underset{l}{\operatorname{argmax}} \left| {D^{(l)}}^T w^{(i)} \right|, \\ 0 & \text{otherwise,} \end{cases} \\ D &:= WS^T + D, \\ D^{(j)} &:= \frac{D^{(j)}}{||D^{(j)}||_2}, \end{aligned} \tag{1}$$

where $s^{(i)} \in \mathbb{R}^k$ is the code vector associated with the input $w^{(i)}$, and $D^{(j)}$ is the j'th column of the dictionary D. The matrices $W \in \mathbb{R}^{n\times m}$ and $S \in \mathbb{R}^{k\times m}$ have the columns $w^{(i)}$ and $s^{(i)}$, respectively. $w^{(i)}$'s are randomly extracted patches from input images that have the same dimension as the dictionary vectors, $D^{(j)}$.


(a) k-means


(b) convolutional k-means

Transfer learning:
Using a trained CNN & fine-tuning

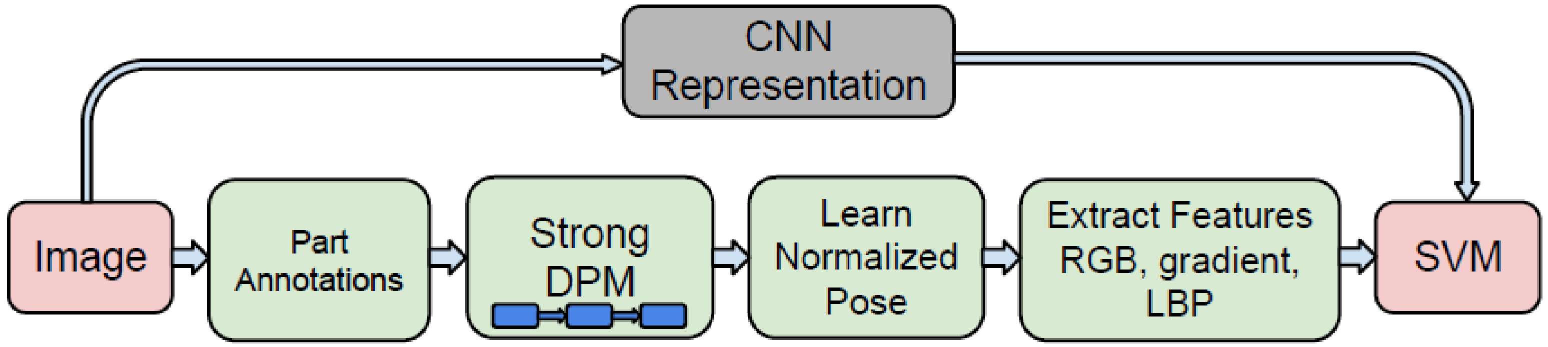
Using trained CNN

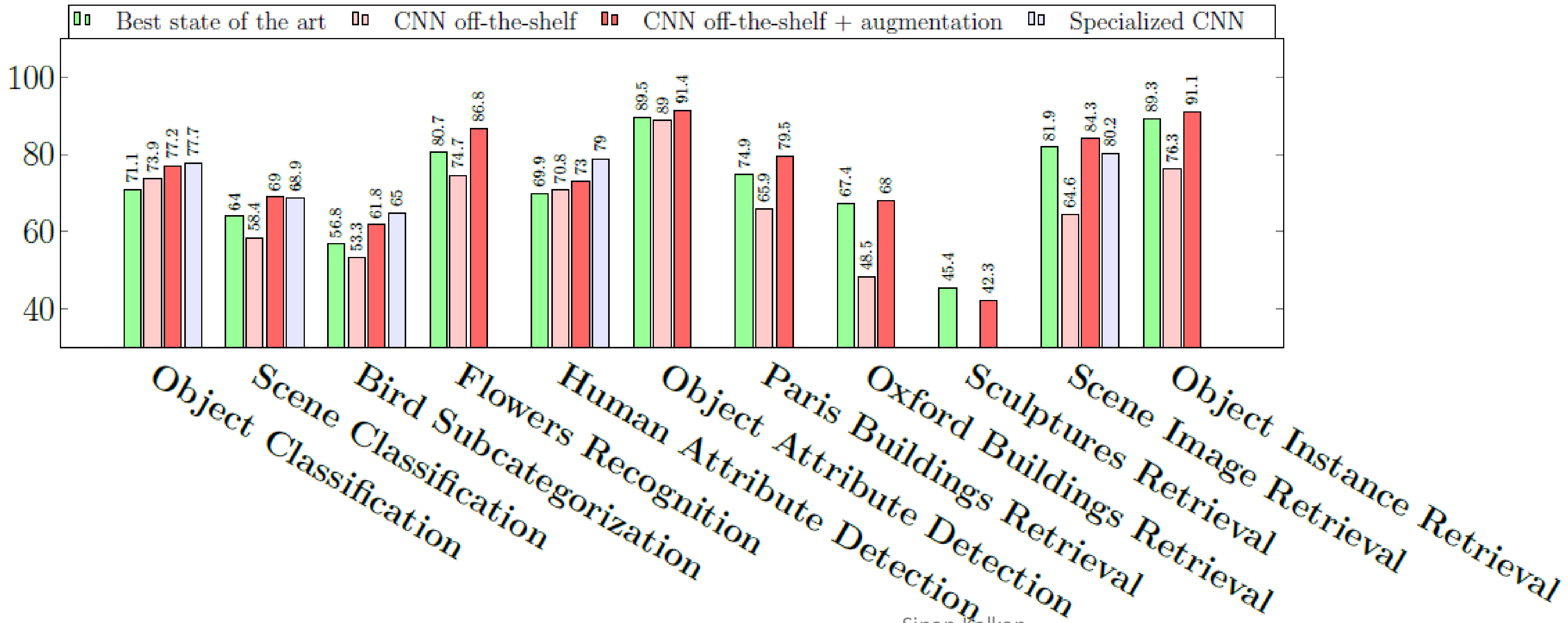
- Also called transfer learning
  - Rare to design and train a CNN from scratch!
- Take a trained CNN, e.g., AlexNet
  - Use a trained CNN as a feature detector:
    - Remove the last fully-connected layer
    - The activations of the remaining layer are called CNN codes
    - This yields a 4096 dimensional feature vector for AlexNet
    - Now, add a fully-connected layer for your problem and train a linear classifier on your dataset.
  - Alternatively, fine-tune the whole network with your new layer and outputs
    - You may limit updating only to the last layers because earlier layers are generic, and quite dataset independent
- Pre-trained CNNs

CNN Features off-the-shelf: an Astounding Baseline for Recognition

Ali Sharif Razavian Hossein Azizpour Josephine Sullivan Stefan Carlsson
CVAP, KTH (Royal Institute of Technology)
Stockholm, Sweden
{razavian, azizpour, sullivan, stefanc}@csc.kth.se

2014





Finetuning

1. If the new dataset is small and similar to the original dataset used to train the CNN:
   - Finetuning the whole network may lead to overfitting
   - Just train the newly added layer

2. If the new dataset is big and similar to the original dataset:
   - The more, the merrier: go ahead and train the whole network

3. If the new dataset is small and different from the original dataset:
   - Not a good idea to train the whole network
   - However, add your new layer not to the top of the network, since those parts are very dataset (problem) specific
   - Add your layer to earlier parts of the network

4. If the new dataset is big and different from the original dataset:
   - We can "finetune" the whole network
   - This amounts to a new training problem by initializing the weights with those of another network

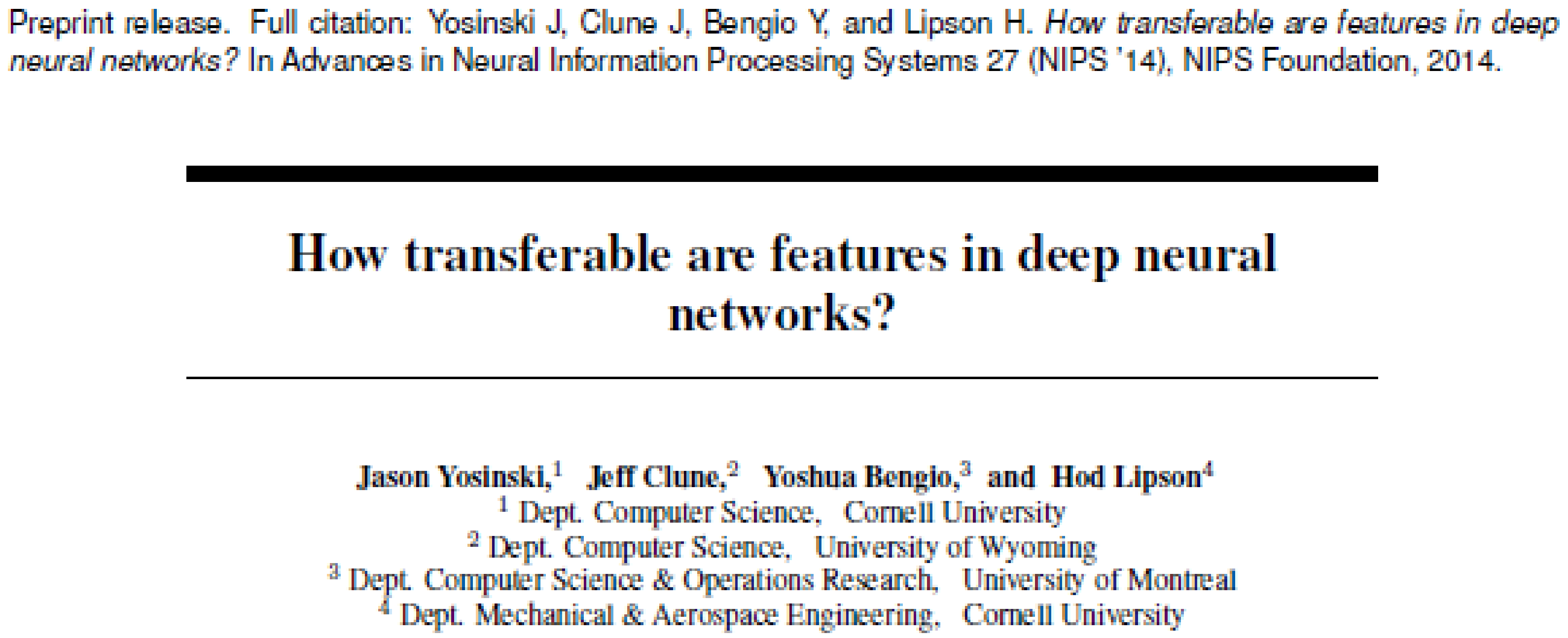
More on finetuning

- You cannot change the architecture of the trained network (e.g., remove layers) arbitrarily

- The sizes of the layers can be varied
  - For convolution & pooling layers, this is straightforward
  - For the fully-connected layers: you can convert the fully-connected layers to convolution layers, which makes it size-independent.

- You should use small learning rates while fine-tuning

See also:



Preprint release. Full citation: Yosinski J, Clune J, Bengio Y, and Lipson H. *How transferable are features in deep neural networks?* In Advances in Neural Information Processing Systems 27 (NIPS '14), NIPS Foundation, 2014.

How transferable are features in deep neural networks?

Jason Yosinski,[1] **Jeff Clune,**[2] **Yoshua Bengio,**[3] **and Hod Lipson**[4]
[1] Dept. Computer Science, Cornell University
[2] Dept. Computer Science, University of Wyoming
[3] Dept. Computer Science & Operations Research, University of Montreal
[4] Dept. Mechanical & Aerospace Engineering, Cornell University

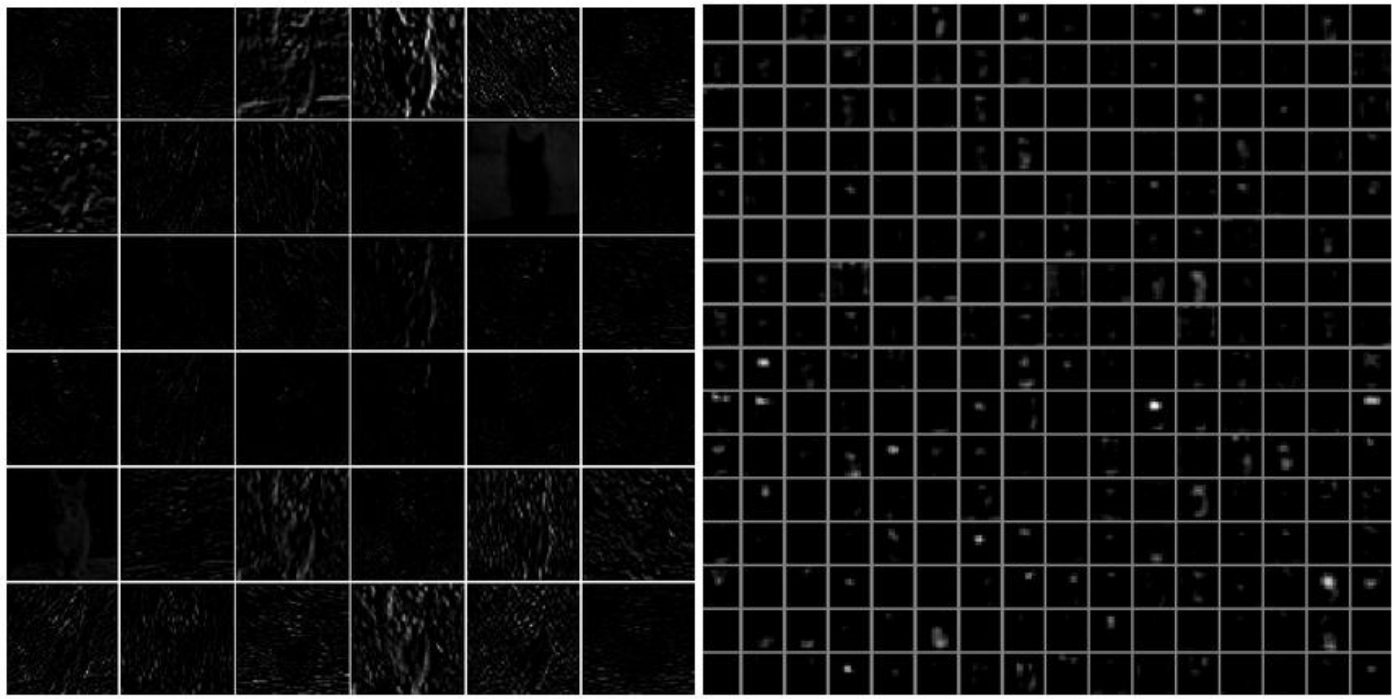
Visualizing and Understanding CNNs

Many different mechanisms

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

Visualize activations during training

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

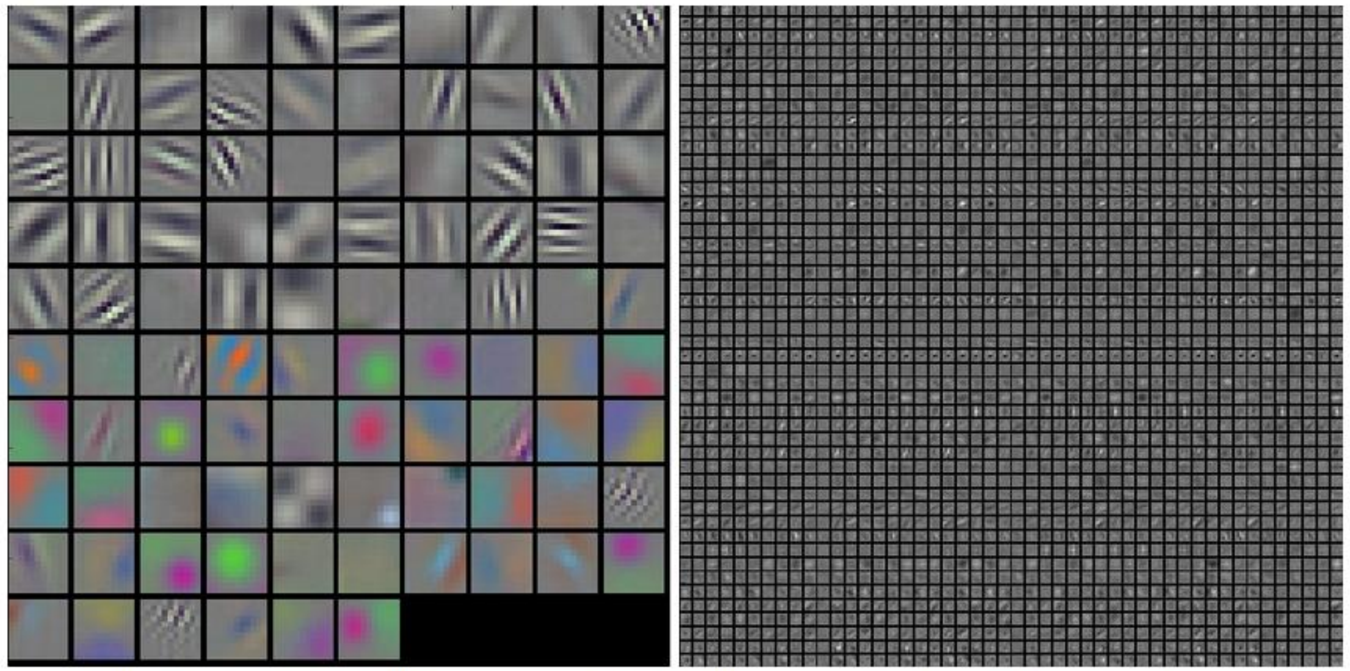


Typical-looking activations on the first CONV layer (left), and the 5th CONV layer (right) of a trained AlexNet looking at a picture of a cat. Every box shows an activation map corresponding to some filter. Notice that the activations are sparse (most values are zero, in this visualization shown in black)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

http://cs231n.github.io/convolutional-networks/

Visualize the weights

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



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

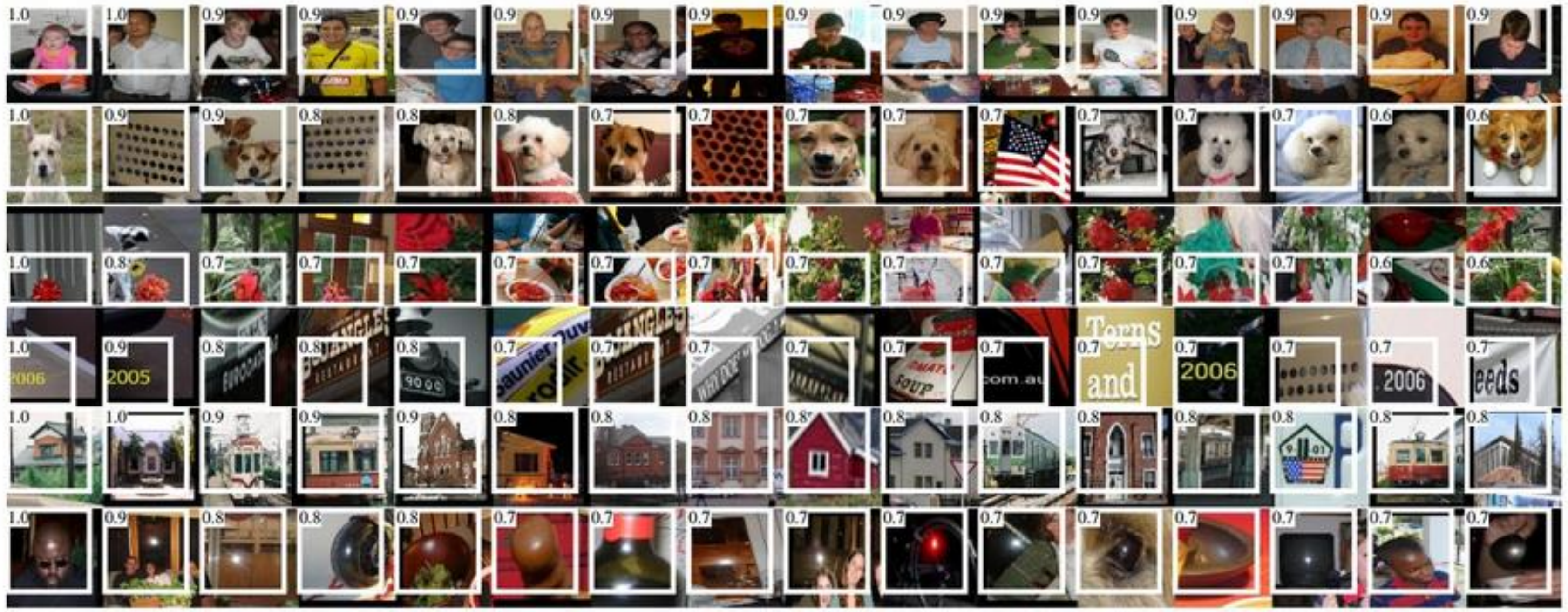
http://cs231n.github.io/convolutional-networks/

Visualize the inputs that maximally activate a neuron

- Keep track of which images activate a neuron most

Rich feature hierarchies for accurate object detection and semantic segmentation
Tech report (v5)

Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik
UC Berkeley
{rbg, jdonahue, trevor, malik}@eecs.berkeley.edu

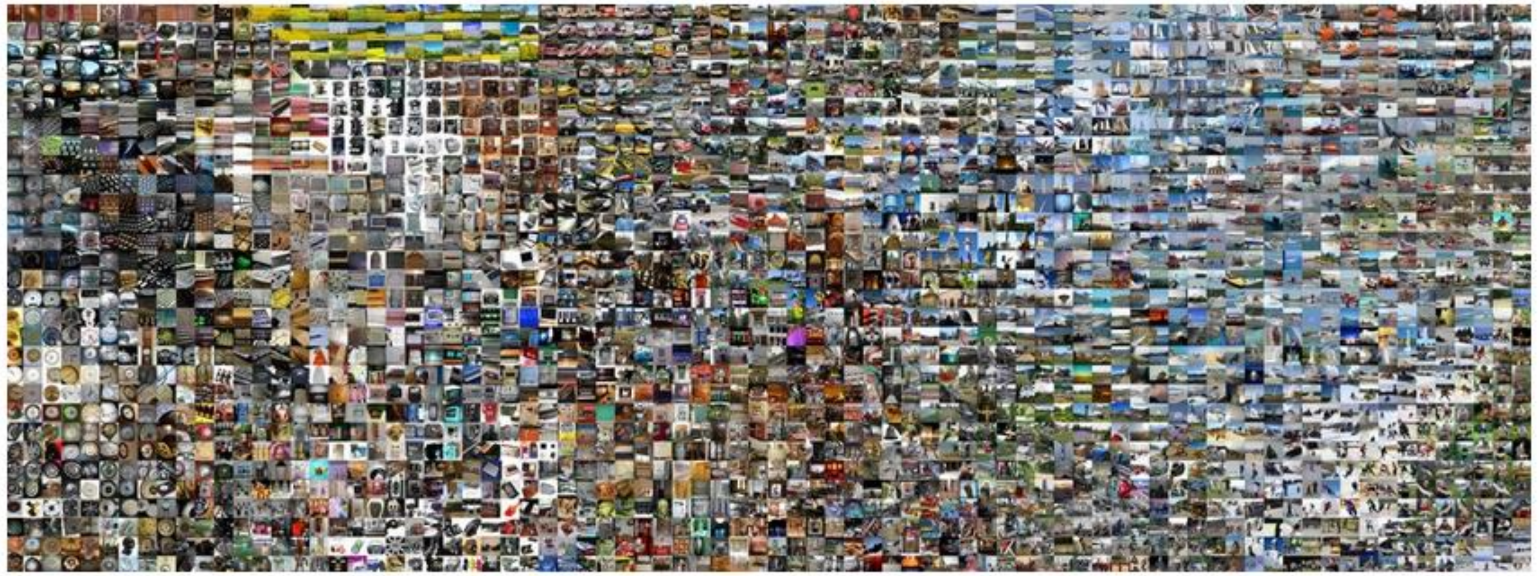


Maximally activating images for some POOL5 (5th pool layer) neurons of an AlexNet. The activation values and the receptive field of the particular neuron are shown in white. (In particular, note that the POOL5 neurons are a function of a relatively large portion of the input image!) It can be seen that some neurons are responsive to upper bodies, text, or specular highlights.

http://cs231n.github.io/convolutional-networks/

Embed the codes in a lower-dimensional space

- Place images into a 2D space such that images which produce similar CNN codes are placed close.
- You can use, e.g., t-Distributed Stochastic Neighbor Embedding (t-SNE)



t-SNE embedding of a set of images based on their CNN codes. Images that are nearby each other are also close in the CNN representation space, which implies that the CNN "sees" them as being very similar. Notice that the similarities are more often class-based and semantic rather than pixel and color-based. For more details on how this visualization was produced the associated code, and more related visualizations at different scales refer to t-SNE visualization of CNN codes.

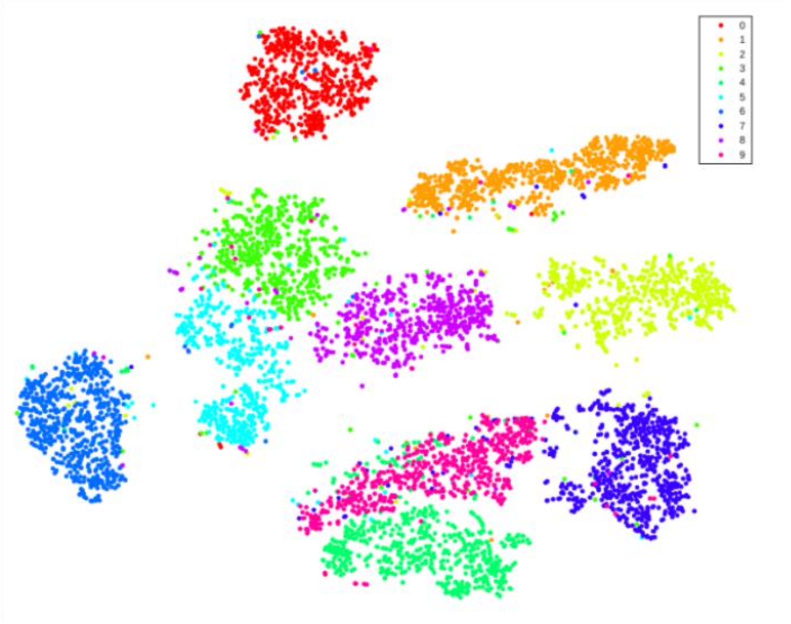


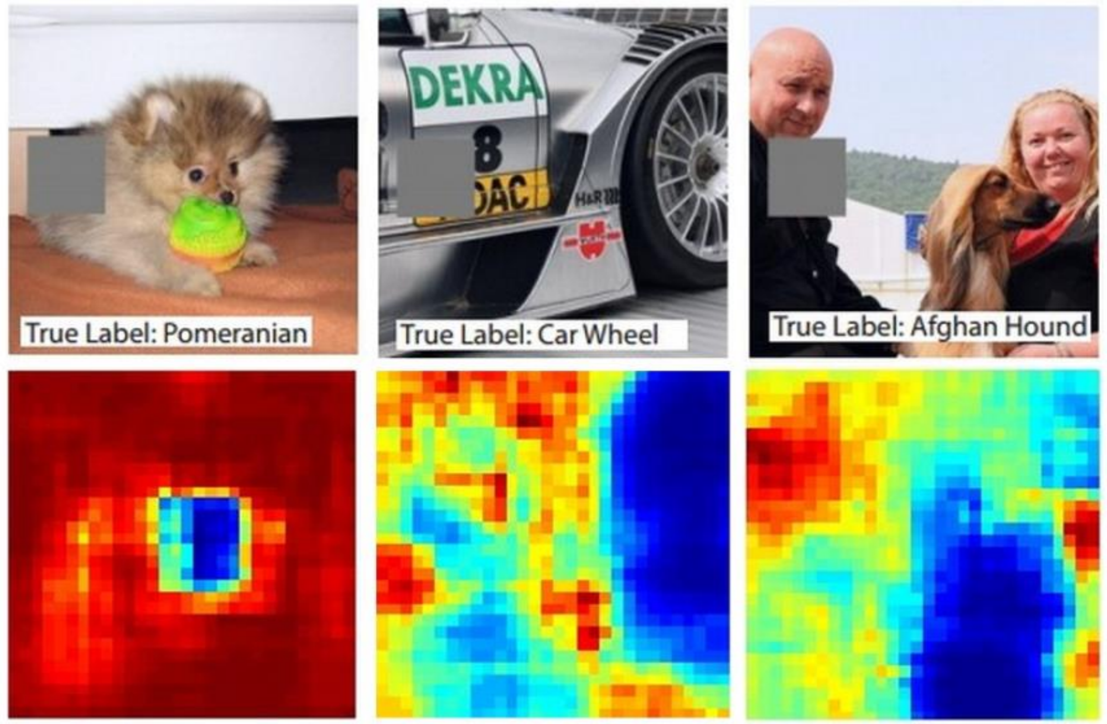
Figure 1 : Illustration of t-SNE on MNIST dataset

Figure: Laurens van der Maaten and Geoffrey Hinton

http://cs231n.github.io/convolutional-networks/

Occlude parts of the image

- Slide an "occlusion window" over the image
- For each occluded image, determine the class prediction confidence/probability.

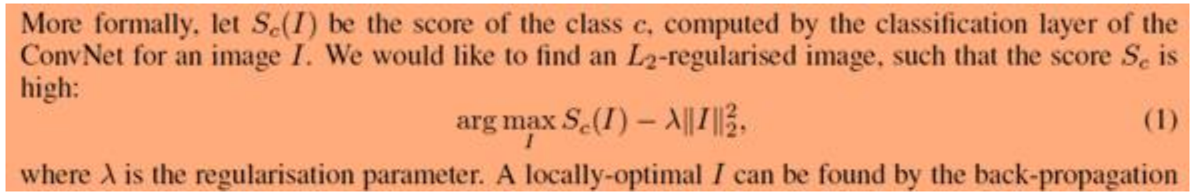


Three input images (top). Notice that the occluder region is shown in grey. As we slide the occluder over the image we record the probability of the correct class and then visualize it as a heatmap (shown below each image). For instance, in the left-most image we see that the probability of Pomeranian plummets when the occluder covers the face of the dog, giving us some level of confidence that the dog's face is primarily responsible for the high classification score. Conversely, zeroing out other parts of the image is seen to have relatively negligible impact.

http://cs231n.github.io/convolutional-networks/

Data gradients

- Generate an image that maximizes the class score.

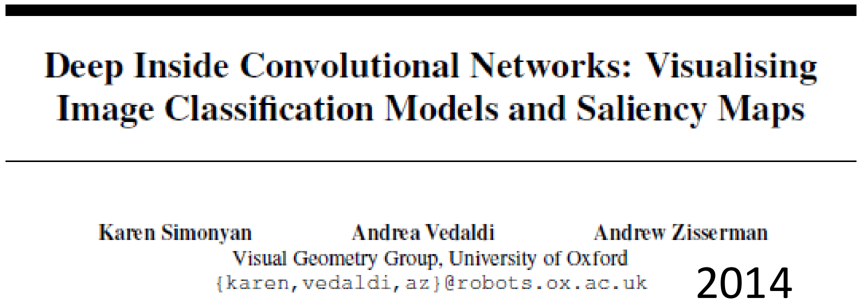


More formally, let $S_c(I)$ be the score of the class c, computed by the classification layer of the ConvNet for an image I. We would like to find an L_2-regularised image, such that the score S_c is high:

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2, \qquad (1)$$

where λ is the regularisation parameter. A locally-optimal I can be found by the back-propagation

- Use: Gradient ascent!



Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps

Karen Simonyan Andrea Vedaldi Andrew Zisserman
Visual Geometry Group, University of Oxford
{karen,vedaldi,az}@robots.ox.ac.uk

2014

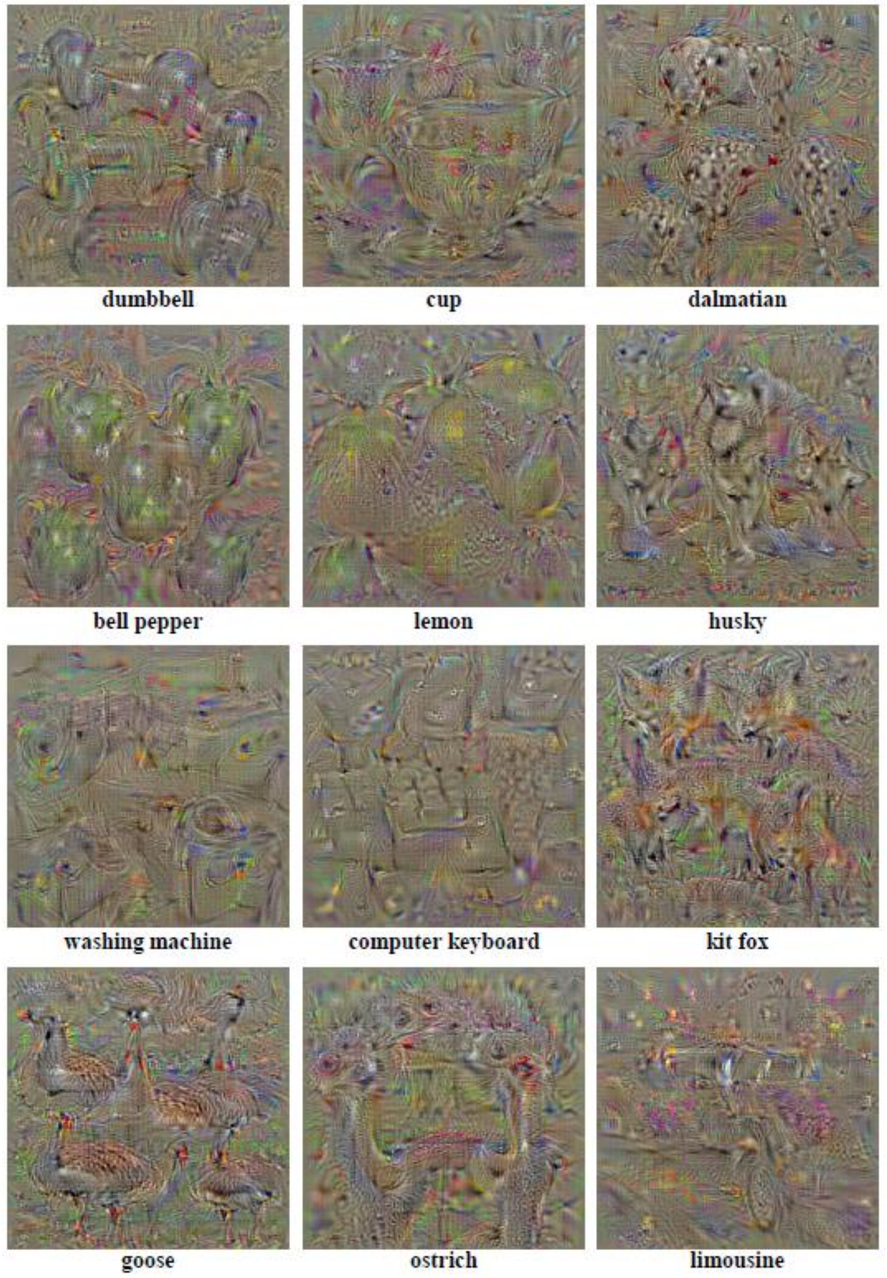
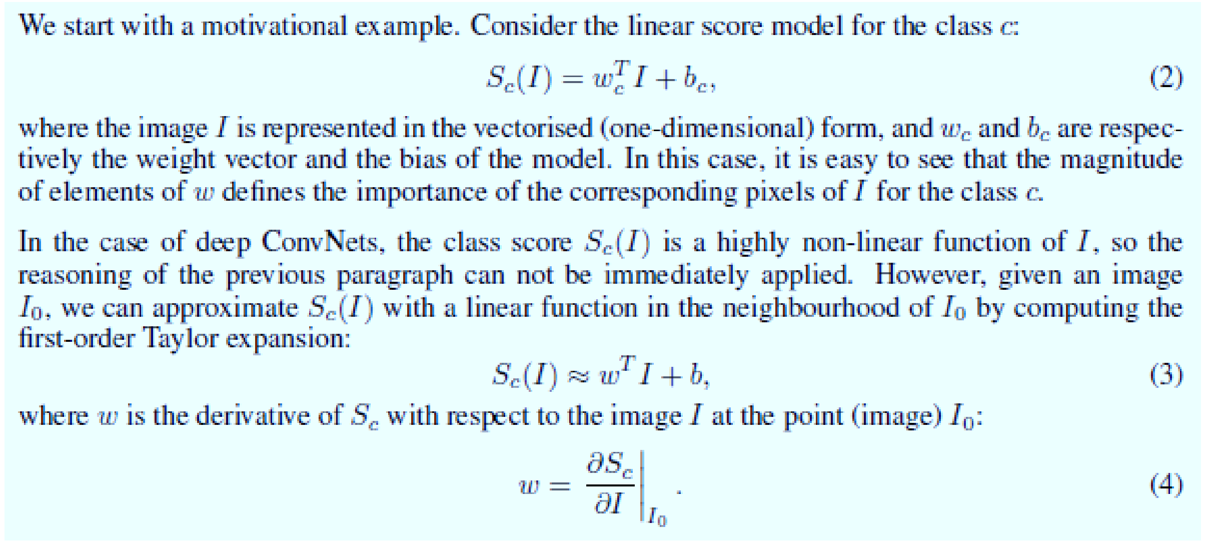


Figure 1: Numerically computed images, illustrating the class appearance models, learnt by a ConvNet, trained on ILSVRC-2013. Note how different aspects of class appearance are captured in a single image. Better viewed in colour.

Data gradients

- The gradient with respect to the input is high for pixels which are on the object



We start with a motivational example. Consider the linear score model for the class c:

$$S_c(I) = w_c^T I + b_c, \tag{2}$$

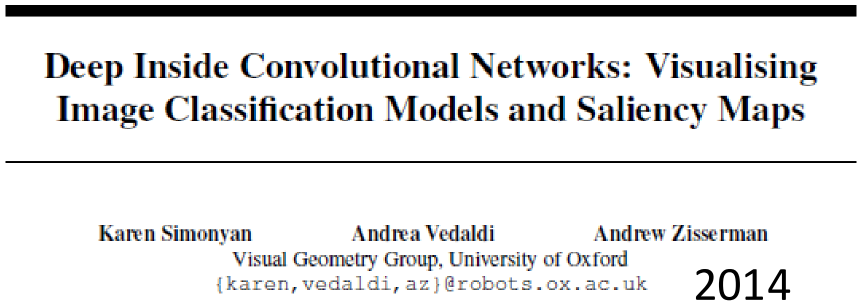
where the image I is represented in the vectorised (one-dimensional) form, and w_c and b_c are respectively the weight vector and the bias of the model. In this case, it is easy to see that the magnitude of elements of w defines the importance of the corresponding pixels of I for the class c.

In the case of deep ConvNets, the class score $S_c(I)$ is a highly non-linear function of I, so the reasoning of the previous paragraph can not be immediately applied. However, given an image I_0, we can approximate $S_c(I)$ with a linear function in the neighbourhood of I_0 by computing the first-order Taylor expansion:

$$S_c(I) \approx w^T I + b, \tag{3}$$

where w is the derivative of S_c with respect to the image I at the point (image) I_0:

$$w = \left. \frac{\partial S_c}{\partial I} \right|_{I_0} . \tag{4}$$



Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps

Karen Simonyan Andrea Vedaldi Andrew Zisserman
Visual Geometry Group, University of Oxford
{karen,vedaldi,az}@robots.ox.ac.uk

2014