Lecture 7: Convolutional Neural Networks

André Martins, Francisco Melo, Mário Figueiredo

Deep Learning Course, Winter 2022-2023

Outline

Convolutional Neural Networks

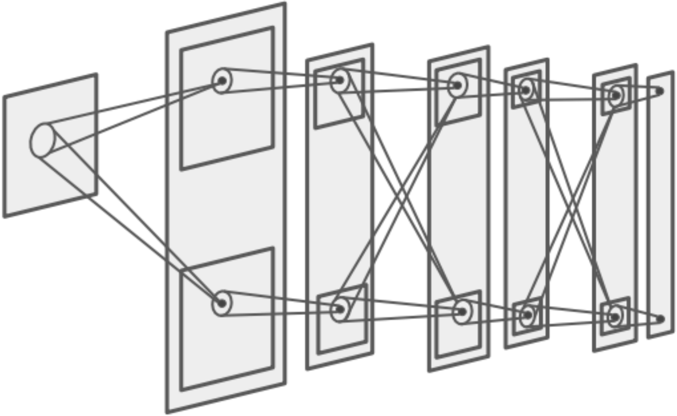
What is a convolutional neural network (CNN)?

...just a NN with a special connectivity structure.

Roadmap:

- Parameter tying/sharing
- 2D CNNs for object recognition
- Pooling layers
- Classical CNNs: ImageNet, AlexNet, GoogLeNet
- One-dimensional CNNs for NLP

Neocognitron (Fukushima and Miyake, 1982)



(Credits: Fei-Fei Li, Johnson, Yeung)

- "Sandwich" architecture, alternating between simple cells with modifiable parameters and complex cells which perform pooling

Neocognitron (Fukushima and Miyake, 1982)

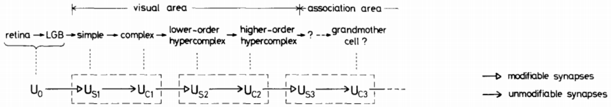


Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron

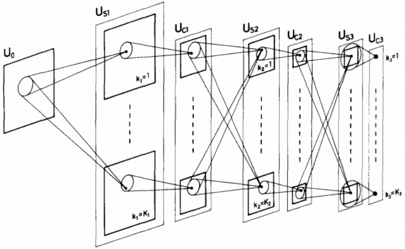
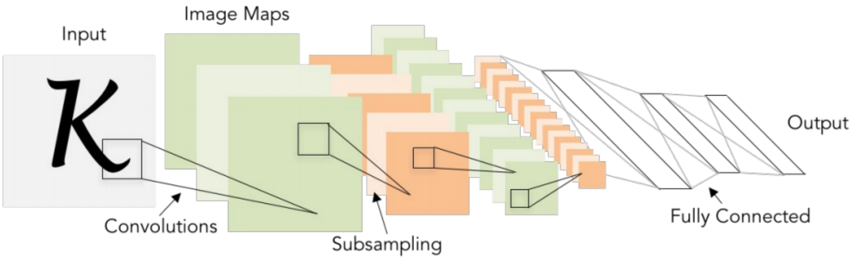


Fig. 2. Schematic diagram illustrating the interconnections between layers in the neocognitron

- Inspired by the multi-stage hierarchy model of the visual nervous system (Hubel and Wiesel, 1965)

ConvNet (LeNet-5) (LeCun et al., 1998)

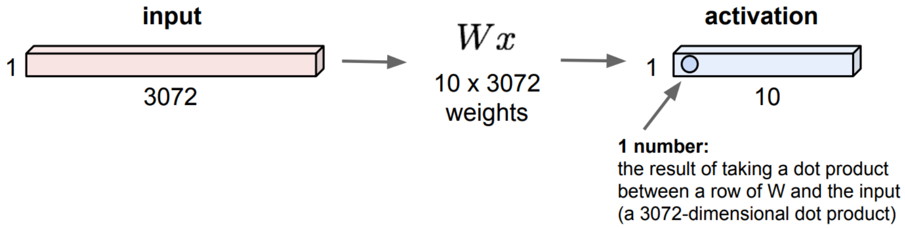


Convolutional Networks

- How is a CNN different from a standard feedforward NN?
- What is a convolutional layer?
- How is it different from a fully connected layer?

Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

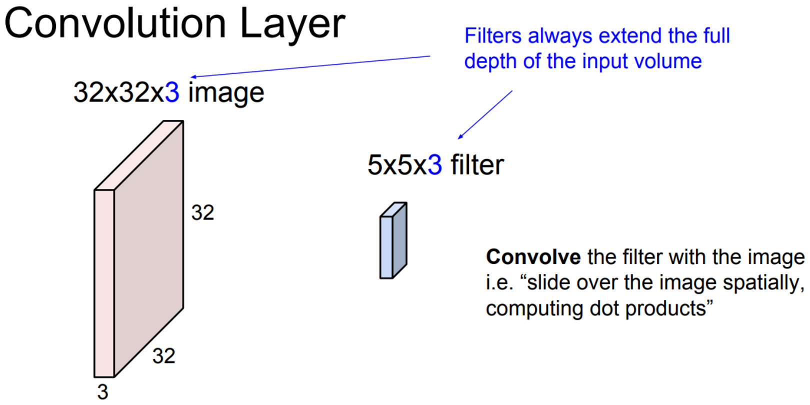


(Credits: Fei-Fei Li, Johnson, Yeung)

All activations depend on all inputs.

Convolutional Layer

Don't stretch/reshape: preserve the spacial structure!

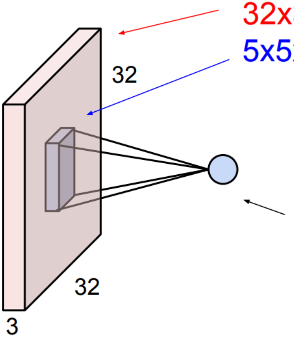


(Credits: Fei-Fei Li, Johnson, Yeung)

Convolutional Layer

32x32x3 image

5x5x3 filter w



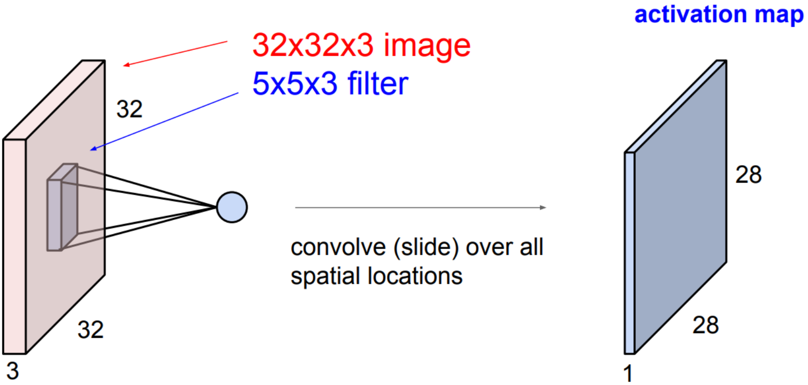
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

(Credits: Fei-Fei Li, Johnson, Yeung)

Convolutional Layer

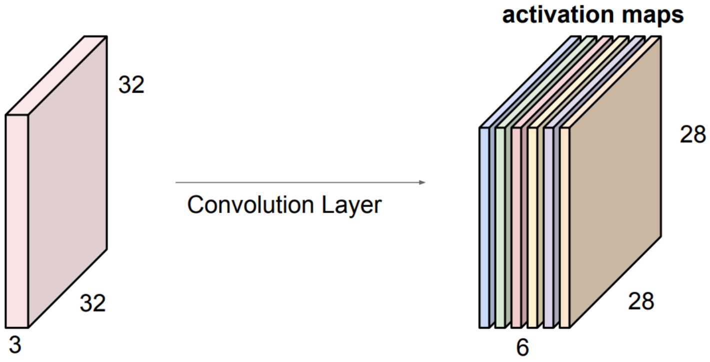
Apply the same filter to all spatial locations (28x28 times, why?):



(Credits: Fei-Fei Li, Johnson, Yeung)

Convolutional Layer

- For example, if we have **6** 5x5x3 filters, we get **6** activation maps:



(Credits: Fei-Fei Li, Johnson, Yeung)

- We stack these up to get a new "image" of size 28x28x6!

Image Size, Filter Size, Stride, Channels

Stride: shift in pixels between two consecutive windows.

In the previous illustrations: stride $= 1$.

Image Size, Filter Size, Stride, Channels

Stride: shift in pixels between two consecutive windows.

In the previous illustrations: stride $= 1$.

Number of channels: number of filters used in each layer.

Image Size, Filter Size, Stride, Channels

Stride: shift in pixels between two consecutive windows.

In the previous illustrations: stride $= 1$.

Number of channels: number of filters used in each layer.

Given an $N \times N \times D$ image, $F \times F \times D$ filters, K channels, and stride S, the resulting output will be of size $M \times M \times K$, where

$$M = (N - F)/S + 1$$

Image Size, Filter Size, Stride, Channels

Stride: shift in pixels between two consecutive windows.

In the previous illustrations: stride $= 1$.

Number of channels: number of filters used in each layer.

Given an $N \times N \times D$ image, $F \times F \times D$ filters, K channels, and stride S, the resulting output will be of size $M \times M \times K$, where

$$M = (N - F)/S + 1$$

Examples:

- $N = 32$, $D = 3$, $F = 5$, $K = 6$, $S = 1$ results in an $28 \times 28 \times 6$ output
- $N = 32$, $D = 3$, $F = 5$, $K = 6$, $S = 3$ results in an $10 \times 10 \times 6$ output

Image Size, Filter Size, Stride, Channels

Stride: shift in pixels between two consecutive windows.

In the previous illustrations: stride $= 1$.

Number of channels: number of filters used in each layer.

Given an $N \times N \times D$ image, $F \times F \times D$ filters, K channels, and stride S, the resulting output will be of size $M \times M \times K$, where

$$M = (N - F)/S + 1$$

Examples:

- $N = 32$, $D = 3$, $F = 5$, $K = 6$, $S = 1$ results in an $28 \times 28 \times 6$ output
- $N = 32$, $D = 3$, $F = 5$, $K = 6$, $S = 3$ results in an $10 \times 10 \times 6$ output

Padding: append zeros around the images.

Common padding size: $(F - 1)/2$, which preserves spatial size: $M = N$.

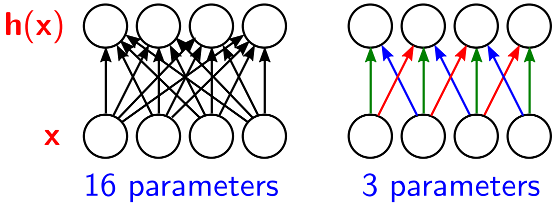
CNNs and Convolutions

Why is this called "convolutional"?

The convolution of a signal x and a filter w, denoted $x * w$, is:

$$h[t] = (x * w)[t] = \sum_{a=-\infty}^{\infty} x[t-a]w[a].$$

Basic idea: sparse/local connectivity and parameter tying/sharing.



Convolutions with Padding

Expression above is for infinite-support signal x and filter w.

Convolutions with Padding

Expression above is for infinite-support signal x and filter w.

Finite support: $x = (x[0], ..., x[N-1])$; $w = (w[-D], ..., w[D])$ $(F = 2D + 1)$

$$h[t] = (x * w)[t] = \sum_{a=-D}^{D} w[a]x[t-a], \text{ for } t = D, ..., N - D - 1$$

The result has support of size $N - D - 1 - D + 1 = N - 2D$.

Convolutions with Padding

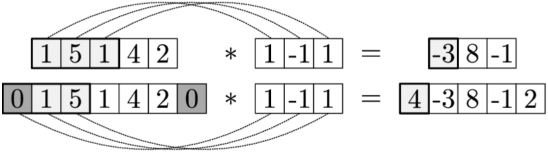
Expression above is for infinite-support signal x and filter w.

Finite support: $x = (x[0], ..., x[N-1])$; $w = (w[-D], ..., w[D])$ $(F = 2D + 1)$

$$h[t] = (x * w)[t] = \sum_{a=-D}^{D} w[a]x[t-a], \text{ for } t = D, ..., N - D - 1$$

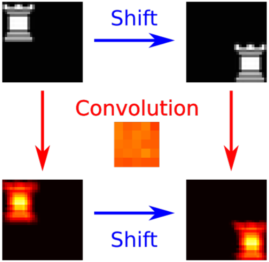
The result has support of size $N - D - 1 - D + 1 = N - 2D$.

Padding: append $D = (F - 1)/2$ zeros at each side of x.



Convolutions and Parameter Tying

Leads to translation/shift equivariance



Why do we want to tie (share) parameters?

- Reduce the number of parameters to be learned
- Deal with arbitrary long, variable-length, sequences: rather than shifting the filters, shift the input

Can also be done in 1D (e.g., text data, signals, ...)

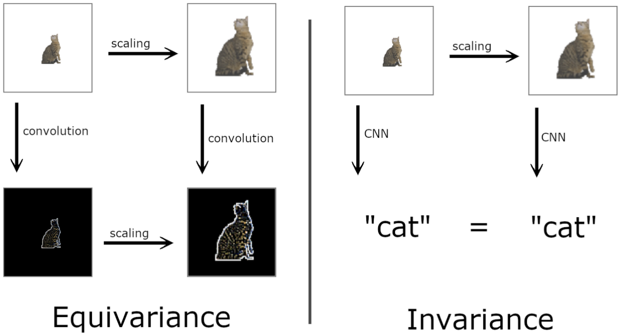
Convolutions and Pooling

The second component of CNNs is pooling

Common CNNs alternate convolutional layers and pooling layers.

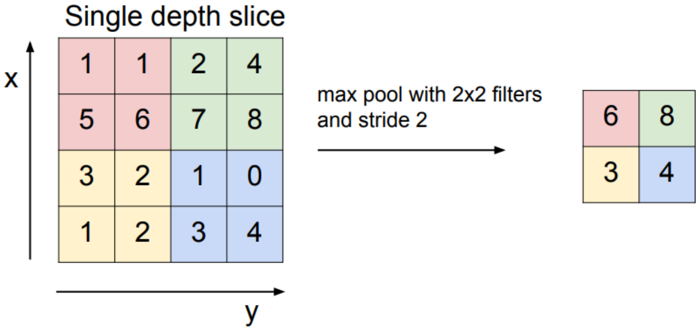
Pooling layers provide invariance.

Equivariance vs Invariance



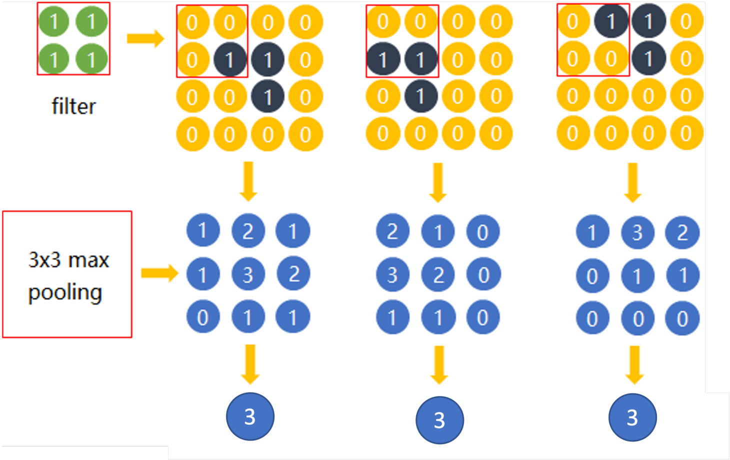
Pooling Layer

- Makes the representations smaller, more manageable.
- Operates over each activation map (each channel) independently
- Max-pooling:

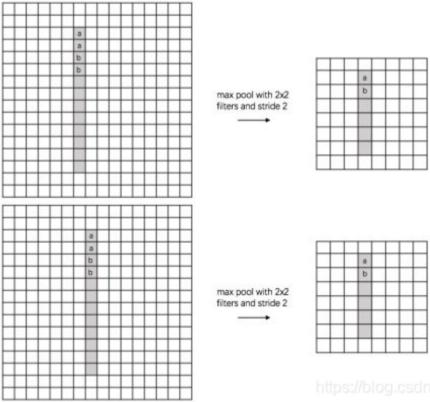


(Credits: Fei-Fei Li, Johnson, Yeung)

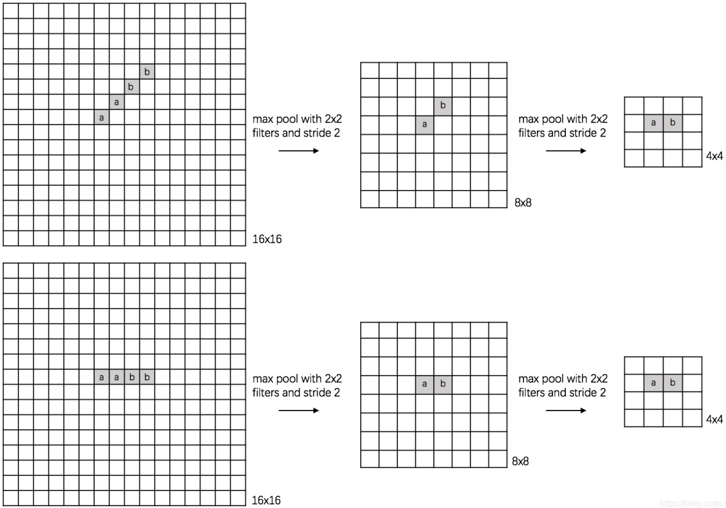
Max Pooling: Shift Invariance



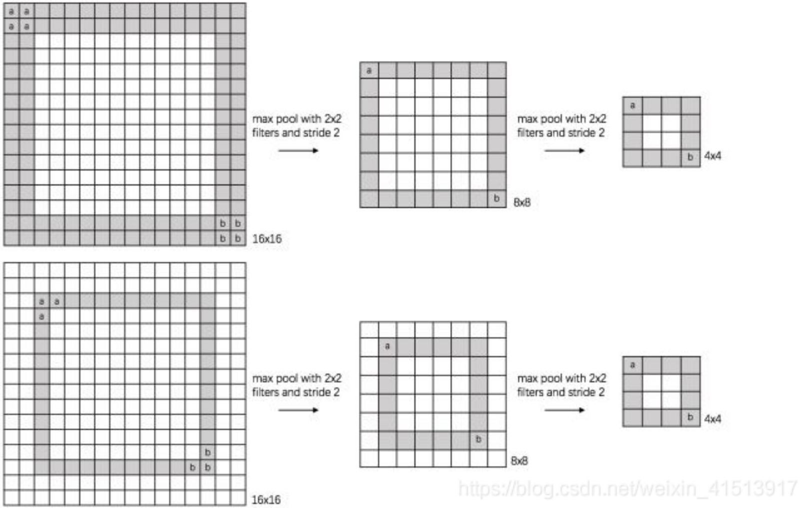
Max Pooling: Shift Invariance (II)



Max Pooling: Rotation Invariance

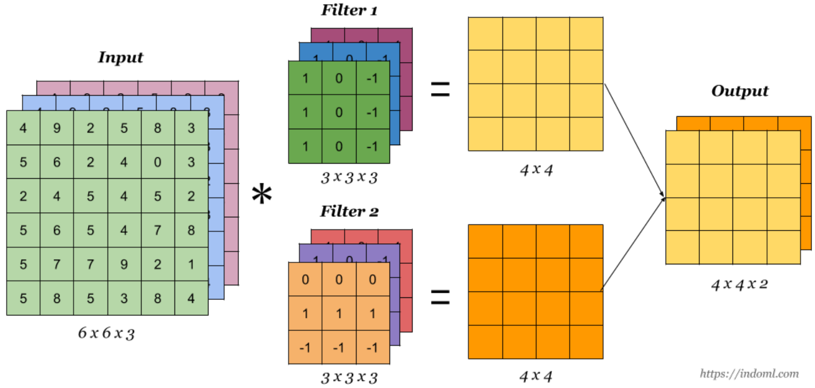


Max Pooling: Scale Invariance



Multiple Convolution Filters: Feature Maps

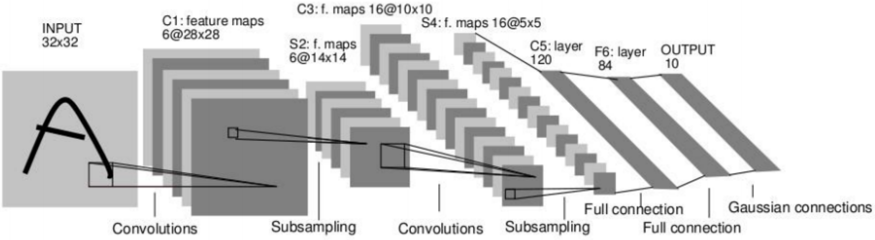
- Different filter for each channel, but keeping spatial invariance:



(Figure credit: Andrew Ng)

2D Convolutional Nets (LeCun et al., 1989)

- Inspired by "Neocognitron" (Fukushima, 1980)
- 2D Convolutions: the same filter (e.g. 3x3) is applied to each location of the image
- The filter weights are learned (as tied parameters)
- Multiple filters
- Alternates convolutional and pooling layers.



ConvNet Successes: MNIST

Handwritten text/digits:

- MNIST (0.35% error (Ciresan et al., 2011b))
- Arabic and Chinese (Ciresan et al., 2011a)

ConvNet Successes: CIFAR-10, Traffic Signs

Simpler recognition benchmarks:

- CIFAR-10 (9.3% error (Wan et al., 2013))
- Traffic signs: 0.56% error vs 1.16% for humans (Cireşan et al., 2011)

Less good at more complex tasks, e.g., Caltech-101/256 (few training samples).

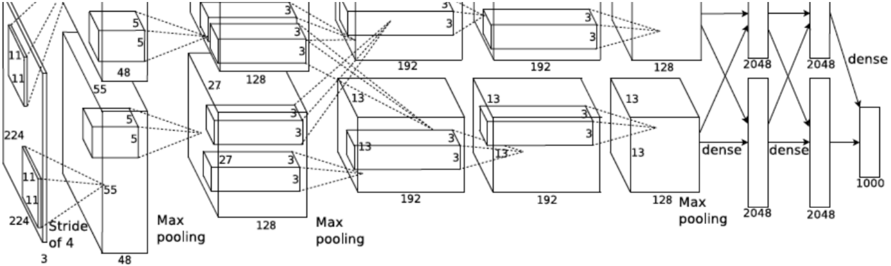
ImageNet Dataset

- 14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon Turk

(Slide credit to Rob Fergus)

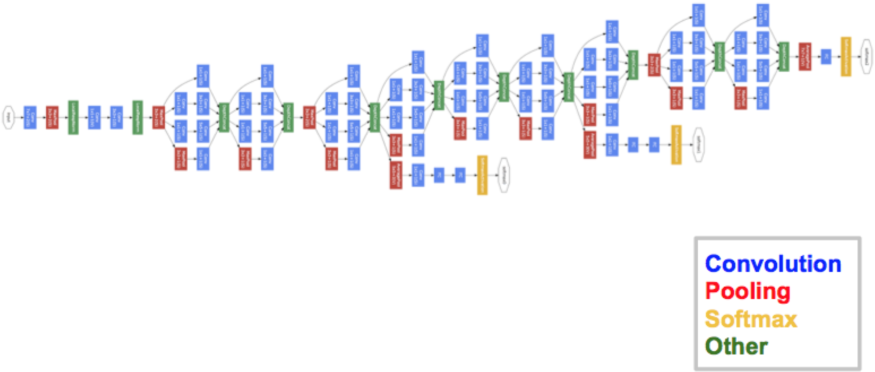
AlexNet (Krizhevsky et al., 2012)

- 54M parameters; 8 layers (5 conv, 3 fully-connected)
- Trained on 1.4M ImageNet images
- Trained on 2 GPUs for a week (50x speed-up over CPU)
- Dropout regularization
- Test error: 16.4% (second best team was 26.2%)



GoogLeNet (Szegedy et al., 2015)

- GoogLeNet inception module: very deep convolutional network, fewer (5M) parameters



Residual Networks (ResNets)

- Add skip-connections; tends to lead to more stable learning.

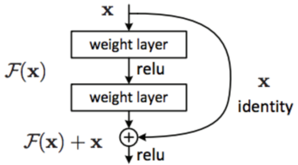


Figure 2. Residual learning: a building block.

(He et al., 2016)

Residual Networks (ResNets)

- Add skip-connections; tends to lead to more stable learning.

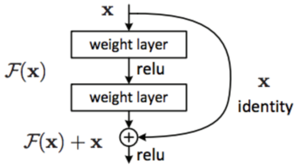


Figure 2. Residual learning: a building block.

(He et al., 2016)

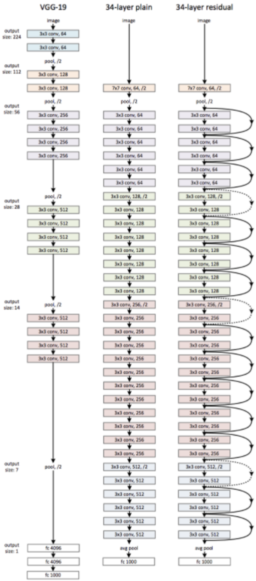
Key (not unique) motivation: mitigate the vanishing gradient problem

With $H(\boldsymbol{x}) = \mathcal{F}(\boldsymbol{x}) + \lambda \boldsymbol{x}$, the gradient back-propagation becomes

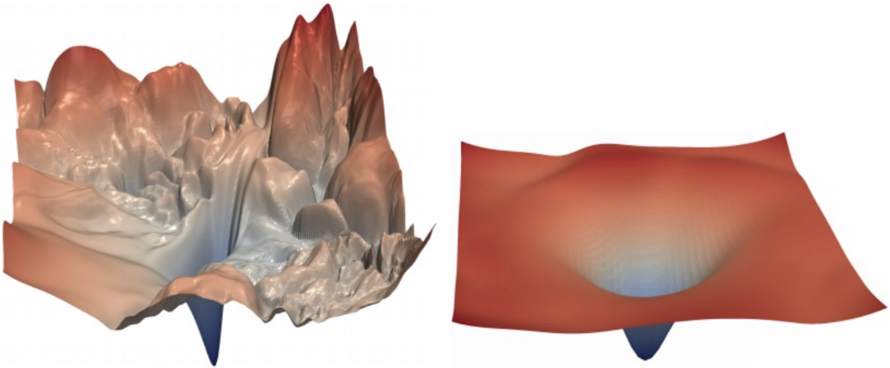
$$\frac{\partial L}{\partial \boldsymbol{x}} = \frac{\partial L}{\partial H}\frac{\partial H}{\partial \boldsymbol{x}} = \frac{\partial L}{\partial H}\left(\frac{\partial \mathcal{F}}{\partial \boldsymbol{x}} + \lambda\right)$$

Residual Networks (ResNets)

- Very deep network (34 layers here, but up to 152 layers!)
- VGG-19 ("Visual Geometry Group") by Simonyan and Zisserman (2014) (19 layers, but more FLOPs)



Residual Networks (ResNets)



(a) without skip connections   (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

(Li et al., 2018)

Convolutional Nets in NLP

- So far, we focused on CNNs for images.

Convolutional Nets in NLP

- So far, we focused on CNNs for images.
- Are CNNs also used in NLP? Not as much, but...

Convolutional Nets in NLP

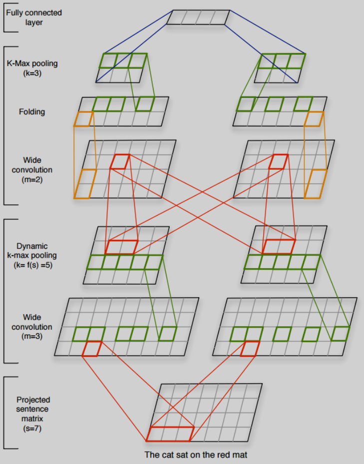
- So far, we focused on CNNs for images.
- Are CNNs also used in NLP? Not as much, but...
- Yoav Goldberg in the Representation Learning Workshop (ACL 2018):

  "NLP's ImageNet moment has arrived."

  (not referring to CNNs, in particular, but to big NNs for NLP.

Convolutional Nets in NLP

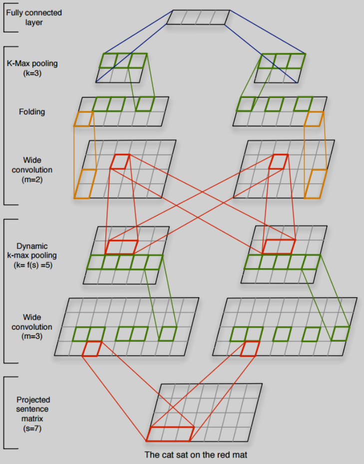
- 1D convolutions (text is a sequence)



Kalchbrenner et al. (2014)

Convolutional Nets in NLP

- 1D convolutions (text is a sequence)
- Filters are applied to local windows around each word



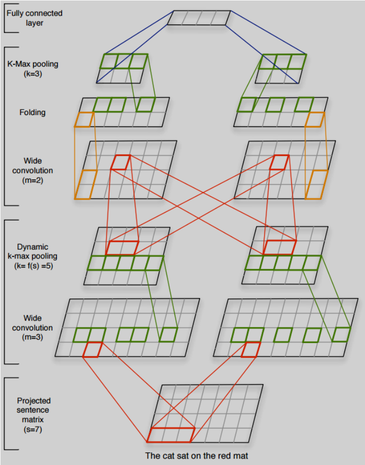
Kalchbrenner et al. (2014)

Convolutional Nets in NLP

- 1D convolutions (text is a sequence)
- Filters are applied to local windows around each word
- For word embeddings $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L$, the filter response for word i is:

$$\boldsymbol{h}_i = \boldsymbol{g}(\boldsymbol{W}[\boldsymbol{x}_{i-h} \oplus \ldots \oplus \boldsymbol{x}_i \oplus \ldots \boldsymbol{x}_{i+h}] + \boldsymbol{b}),$$

where $\oplus$ denotes vector concatenation and $\boldsymbol{W}$ are shared parameters



Kalchbrenner et al. (2014)

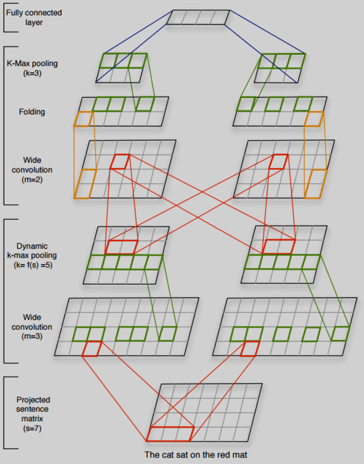
Convolutional Nets in NLP

- 1D convolutions (text is a sequence)
- Filters are applied to local windows around each word
- For word embeddings $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L$, the filter response for word i is:

$$\boldsymbol{h}_i = \boldsymbol{g}(\boldsymbol{W}[\boldsymbol{x}_{i-h} \oplus \ldots \oplus \boldsymbol{x}_i \oplus \ldots \boldsymbol{x}_{i+h}] + \boldsymbol{b}),$$

  where $\oplus$ denotes vector concatenation and $\boldsymbol{W}$ are shared parameters

- Can pad left and right with special symbols if needed.



Kalchbrenner et al. (2014)

Variable Input Length

- Most computation in CNNs can be done in parallel.

Variable Input Length

- Most computation in CNNs can be done in parallel.

- GPUs can leverage this and achieve great speed-ups!

Variable Input Length

- Most computation in CNNs can be done in parallel.
- GPUs can leverage this and achieve great speed-ups!
- But, unlike images, which have fixed size, sentences have different lengths, which makes batching a bit trickier!

Mini-Batching, Padding, and Masking

Mini-batching is necessary to speed up training in GPUs

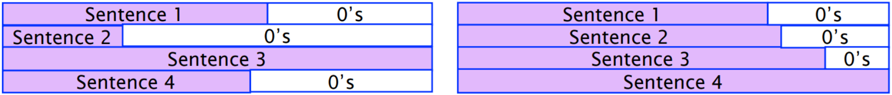
How to cope with different sentence lengths?

Mini-Batching, Padding, and Masking

Mini-batching is necessary to speed up training in GPUs

How to cope with different sentence lengths?

Solution: minimize waste by sorting by sentence length before forming mini-batches, then padding:



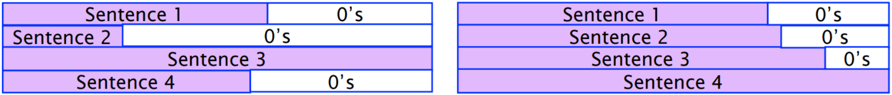
(Image credit: Thang Luong, Kyunghyun Cho, Chris Manning)

Mini-Batching, Padding, and Masking

Mini-batching is necessary to speed up training in GPUs

How to cope with different sentence lengths?

Solution: minimize waste by sorting by sentence length before forming mini-batches, then padding:



(Image credit: Thang Luong, Kyunghyun Cho, Chris Manning)

Masking is needed to ensure the padding is not affecting the results.

Beyond Convolutions

- Other architectures have been proposed which offer alternatives to convolutions
- For example: transformers.
- This is somewhat similar to "dynamic convolutions".
- Covered in another lecture.

Outline

1. Convolutional Neural Networks
2. Visualizing Representations
3. Conclusions

What Representations Are We Learning?

- Which neurons fire for recognizing a particular object?
- What parts of the network are activated?

What Representations Are We Learning?

- Which neurons fire for recognizing a particular object?
- What parts of the network are activated?
- To answer this, visualize what is happening inside the network.

Visualization

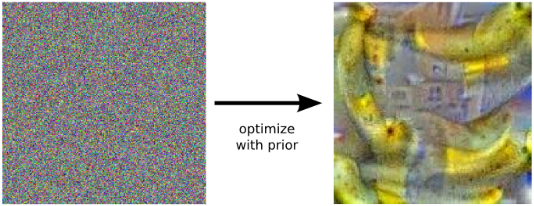
- **Idea:** Optimize input to maximize particular output

Visualization

- **Idea:** Optimize input to maximize particular output
- Depends on the initialization

Visualization

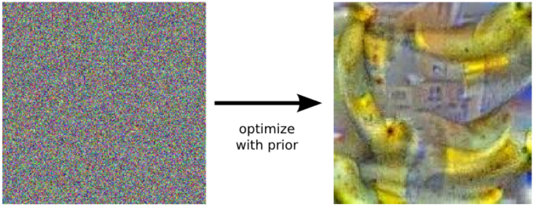
- **Idea:** Optimize input to maximize particular output
- Depends on the initialization
- Google DeepDream, maximizing "banana" output:



(from `https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`)

Visualization

- **Idea:** Optimize input to maximize particular output
- Depends on the initialization
- Google DeepDream, maximizing "banana" output:

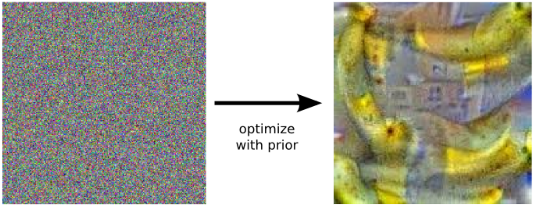


(from https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html)

- Can also specify an inner layer and tune the input to maximize its activations: useful to see what kind of features it is representing.

Visualization

- **Idea:** Optimize input to maximize particular output
- Depends on the initialization
- Google DeepDream, maximizing "banana" output:



(from `https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`)

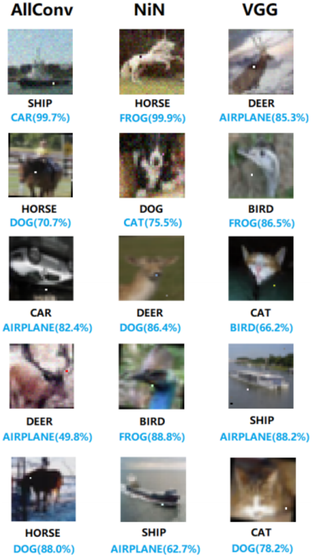
- Can also specify an inner layer and tune the input to maximize its activations: useful to see what kind of features it is representing.
- Specifying a higher layer produces more complex representations...

Google DeepDream

(from https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html)

Adversarial Attacks

- How can we perturb an input slightly to fool a classifier?
- For example: 1-pixel attacks
- Glass-box model: assumes access to the model
- Backpropagate to the inputs to find pixels which maximize the gradient
- There's also work for black-box adversarial attacks (don't have access to the model, but can query it).



(Credits: Su, Vargas, Sakurai (2018))

Even Worse: Perturb Object, Not Image

- Print the model of a turtle in a 3D printer.
- Perturbing the texture fools the model into thinking it's a rifle, regardless of the pose of the object!

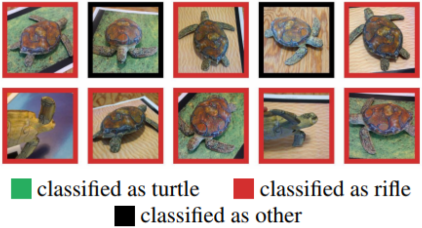


Figure 1. Randomly sampled poses of a 3D-printed turtle adversarially perturbed to classify as a rifle at every viewpoint[2]. An unperturbed model is classified correctly as a turtle nearly 100% of the time.

(Credits: Athalye, Engstrom, Ilyas, Kwok (2018))

Neural networks are still very brittle!

Outline

1. Convolutional Neural Networks
2. Visualizing Representations
3. Conclusions

Conclusions

- CNNs are a very powerful architecture for computer vision
- CNNs take advantage of parameter sharing and sparse connectivity
- They are extremely useful to capture translational invariances in images
- Typically, convolution layers are alternated with max-pooling layers
- Lower layers capture more low-level representations (edges, corners)
- Higher layers have more "semantic" representations (objects, scenes)

Thank you!

Questions?

References I

Cireşan, D., Meier, U., Masci, J., and Schmidhuber, J. (2011). A committee of neural networks for traffic sign classification. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1918–1921. IEEE.

Ciresan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2011a). Convolutional neural network committees for handwritten character classification. In *2011 International Conference on Document Analysis and Recognition*, pages 1135–1139. IEEE.

Ciresan, D. C., Meier, U., Masci, J., Maria Gambardella, L., and Schmidhuber, J. (2011b). Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202.

Fukushima, K. and Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hubel, D. H. and Wiesel, T. N. (1965). Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat. *Journal of neurophysiology*, 28(2):229–289.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2018). Visualizing the loss landscape of neural nets. *Advances in Neural Information Processing Systems*, 31.

References II

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9.

Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In *Proc. of the International Conference on Machine Learning*, pages 1058–1066.