CS-667 Advanced Machine Learning

Nazar Khan

PUCIT

Convolutional Neural Networks

A Neuron as a Detector

- A neuron can be viewed as a detector.
- When it fires, the input must have been similar to its weights. (**Why?**)
- So neuron firing indicates detection of something similar to its weights.

Convolutional Neural Networks

- For recognition of hand-written digits, we have seen that inputs are images and outputs are posteriors probabilities $p(C_k|\mathbf{x})$ for $k = 1, \dots 10$.
- The digits true identity is invariant under
  - translation, scaling, (small) rotation, and
  - small elastic deformations (multiple writings of the same digit by the same person will have subtle differences).
- The output of the neural network should also be invariant to such changes.
- A traditional fully connected neural network can, in principle, learn these invariances using lots of examples.

Convolutional Neural Networks

- However, it totally ignores the *local correlation* property of images.
  - Nearby pixels are more strongly correlated than pixels that are far apart.
- Modern computer vision exploits local correlation by extracting features from local patches and combines this information to detect higher-order features.
  - Example: Gradients $\longrightarrow$ Edges $\longrightarrow$ Lines $\longrightarrow \ldots$.
- Local features useful in one sub-region can be useful in other sub-regions.
  - Example: same object appearing at different locations.
- This weakness of standard neural nets is overcome by Convolutional Neural Networks (CNNs) also known as ConvNets.

NN vs. CNN

NN

- Global receptive fields due to being fully connected.
- Separate weights for each neuron.

CNN

- *Local receptive fields* due to being sparsely connected.
- *Shared weights* among different neurons.
- *Subsampling* of each layer's outputs.

- Receptive field of a neuron consists of previous layer neurons that it is connected to (or looking at).
- A CNN consists of two kinds of layers
  - Convolutional layer
  - Subsampling layer

Convolutional layer

- Consists of multiple arrays of neurons. Each such array is called a *slice* or more accurately *feature map*.
- Each neuron in a feature map
  - is connected to only few neurons in the previous layer, but
  - uses the same weight values as all other neurons in that feature map.
- So within a feature map, we have both
  - local receptive fields, and
  - shared weights.

Convolutional layer

- Example: A feature map may have
  - 100 neurons placed in a 10×10 array, with
  - each neuron getting input from a 5×5 patch of neurons in the previous layer (receptive field), and
  - the same $26 (= 5 \times 5 + 1)$ weights shared between these 100 neurons.
- **Viewed as detectors, all 100 neurons detect the same 5×5 pattern but at different locations of the previous layer**.
- Different feature maps will learn[1] to detect different kinds of patterns.
  - For example, one feature map might learn to detect horizontal edges while others might learn to detect vertical or diagonal edges and so on.

[1] based on their learned weights

Convolutional layer

- To compute activations of the 100 neurons, a dot-product is computed between the same shared weights and different 5×5 patches of previous layer neurons.
- This is equivalent to **sliding a window of weights over the previous layer and computing the dot-product at each location of the window**.
- Therefore, activations of the feature map neurons are computed via *convolution* of the previous layer with a *kernel* comprising the shared weights. Hence the name of this layer.

Invariance in convolutional layer

- If the previous layer is shifted, the activations of the feature map will also be shifted the same way and otherwise remain unchanged.
- **This is why ConvNet outputs achieve some invariance to translations and distortions of inputs.**

Subsampling layer

- Reduces the spatial dimensions of the previous layer by downsampling. Also called *pooling* layer.
- Example: downsampling previous layer of $n \times n$ neurons by factor 2 yields a pooled layer of $\frac{n}{2} \times \frac{n}{2}$ neurons.
- No adjustable weights. Just a fixed downsampling procedure.
- Reduces computations and weights in subsequent layers. Leads to lesser overfitting. (Why?)

Subsampling

- Options: From non-overlapping 2×2 patches
  - pick top-left (standard downsampling by factor 2)
  - pick average (*mean-pooling*)
  - pick maximum (*max-pooling*)
  - pick randomly (*stochastic-pooling*)
- Fractional max-pooling: pick pooling region randomly.

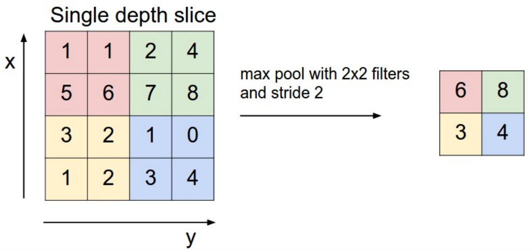

Figure: Max-pooling with 2×2 receptive fields, and stride of 2 neurons.
Source: `http://cs231n.github.io/convolutional-networks/`

Subsampling

- The options in the last slide discard 75% of the data.
- They correspond to
  - neurons with 2×2 receptive fields, and
  - *stride* of 2 neurons.
- This is the most commonly used configuration. Other options exist but note that pooling with larger receptive fields discards too much data.
- Note that since maximum does not change when input is slightly translated, *max-pooling leads to some translation invariance*.
- Subsampling layer can be skipped if convolution layers uses stride>1 since it also produces a subsampled output.

Subsampling

A pooling layer

- with $F \times F$ receptive field and stride S,
- "looking at" a $W_1 \times H_1 \times D_1$ input volume,
- produces a $W_2 \times H_2 \times D_2$ output volume, where
  - $W_2 = \frac{W_1 - F}{S} + 1$
  - $H_2 = \frac{H_1 - F}{S} + 1$
  - $D_2 = D_1$.

Fully Connected Layer

- The last layer is chosen to be fully connected (F)
  - with neurons equal to the desired output size, and
  - activation functions based on the problem to be solved.
- The *output of the second-last layer can therefore be viewed as the transformation ϕ* for which the optimal output layer weights are to be learned.
- Similarly, outputs of earlier layers are *intermediate representations* of the input.

Intermediate Representations

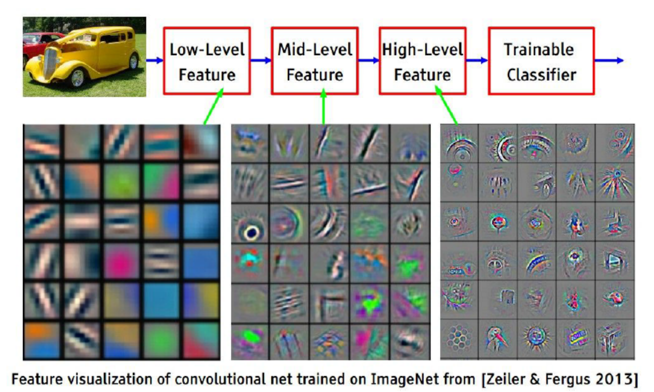

Figure: Intermediate feature representations. Early layers form simple, low-level representations of the input. They are used to incrementally form more complex, high-level representations. Source: `http://cs231n.stanford.edu/slides/winter1516_lecture7.pdf`

Putting it all together

- A standard CNN architecture takes the form $I \longrightarrow \{C - S\} \longrightarrow \{C - S\} \longrightarrow \cdots \longrightarrow F$.
- A particular CNN can be of the form $I_{28\times 28} \longrightarrow 6C_{5\times 5} - 6S_{2\times 2,2} \longrightarrow 12C_{3\times 3} - 12S_{2\times 2,2} \longrightarrow \cdots \longrightarrow F$. Corresponds to
  - Input array of size 28×28.
  - 6 feature-maps/slices in the first convolution layer with each feature-map's neurons looking at 5×5 patches in all 3 input slices.
  - 12 feature-maps/slices in the second convolution layer with each feature-map's neurons looking at 3×3 patches in all 6 slices of previous subsampling layer.
  - Both subsampling layers have neurons with non-overlapping, 2×2 receptive fields.
- To work with colour images, input volume will be changed to $I_{28\times 28\times 3}$.

Putting it all together

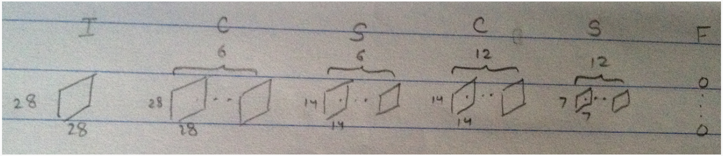

Figure: A CNN architecture for 2D input images of size 28×28 followed by a 6 slice CONV layer, 6 slice POOL layer, 12 slice CONV layer, 12 slice POOL layer and finally a fully connected output layer of K neurons. The POOL layers are performing a downsampling by factor of 2. For nD images, input layers will be $28 \times 28 \times n$ and first CONV layer's receptive fields will be $F \times F \times n$.

Take-home quiz 2

- **How many neurons?**
- **How many weights (including biases)? Assume 3×3 receptive field for CONV layer 1 and $3 \times 3 \times 6$ for CONV layer 2.**

Putting it all together

- There are LOTS of variations.
  - 1×1 convolutions.
  - Fully convolutional networks.
  - Residual blocks
  - Inception modules
  - SqueezeNet
- We have covered only the architectural details of CNNs.
- Implementation details will be covered in the project and tutorial(s).
  - Handling borders in convolution.
  - Backpropagation *with shared weights*.
  - Mathematical derivation.
  - *Efficient* software implementation.
  - `http://cs231n.github.io/neural-networks-3/`

Assignment 4

CNN Implementation

- Implement a Convolutional Neural Network for classification and train it to recognise categories from the Fashion-MNIST dataset.
- Use ReLU activations for hidden layers.
- Due Tuesday, April 1st, 2019 before 5:30 pm on \\printsrv.
- Report classification accuracy and confusion matrix on the training and testing data.
- **Do not submit the dataset**.
- Submit your_roll_number_CNN.zip.
- Resources
  - `http://cs231n.github.io/convolutional-networks/` for forward propagation.

Assignment 4

CNN Implementation

- http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/ for backward propagation.
- http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf for general CNN details.
- Fashion-MNIST can be obtained from https://github.com/zalandoresearch/fashion-mnist. It also shows how to load the data into Matlab.
- Consult your seniors in the CVML lab.