02_Convolutional_Neural_Networks

January 17, 2024

1 Convolutional Neural Networks

- A convolutional neural network is a type of artificial neural network. Therefore, it is a parameterized function whose parameters may be adapted to minimize a loss using gradient descent.
- Convolutional neural networks are suitable to receive images as inputs, since they are naturally able to detect patterns by relying on the spatial relationships between pixels.
- Convolutional neural networks are an example of architecture that exploits domain knowledge to improve performance.
- Convolutional neural networks and their variants can also be applied to audio, text, and time series data.

1.1 Parameter sharing

- Consider a classification task where grayscale images with 256×256 pixels must be classified as containing a *cat* or a *dog*.
- In order to present these images directly to a multilayer percetron, each image would have to be flattened into a vector with $256 \cdot 256 = 65536$ elements.
- If the first hidden layer of such a multilayer perceptron had 128 units, it would have \$65536 128 = 8388608 \$ weights!
- Models with many parameters typically need to be trained in large datasets (or using severe regularization) to prevent overfitting.
- Note that a randomly initialized multilayer perceptron is (almost certainly) unaware of spatial relationships between pixels. For example, any consistent reordering of the pixels in the input vector would lead to the same expected performance after training.
- Convolutional neural networks *share* parameters under the following assumption: if *detecting a pattern* somewhere in an image is useful (for example, a long snout in the center of the image), then detecting the same pattern somewhere else is also useful (for example, a long snout in the top left of the image).

2 Convolutional Layer

- Convolutional layers are the defining component of convolutional neural networks.

- A convolutional layer receives an image as input and produces another image as output.
- The input image is typically divided into *windows* of size $k \times k$. The same neuron will be applied to each of these windows.
- The figure below illustrates most of the computation carried out by a convolutional layer with a single neuron that receives a 3×3 grayscale image as input and outputs a 2×2 grayscale image. The parameter vector $[0, 1, 2, 3]^T$ of the neuron is organized into an image that matches the size of the windows.
- In order to compute the first output image element, the neuron is positioned at the top-left corner of the input image. A dot product is computed between the window (flattened into a vector) and the parameters of the neuron, and the result is placed in the top-left corner of the output image.
- In the example above, this involves computing $[0, 1, 3, 4] \cdot [0, 1, 2, 3]^T = 0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19$.
- In order to compute the remaining output image elements, the convolutional layer *slides* the neuron across the input image (from left to right, top to bottom). As before, a dot product is computed between the window (flattened into a vector) and the parameters of the neuron, and the result is placed in the corresponding position of the output image.
- In the example above, this corresponds to computing

$$\begin{aligned}
[1, 2, 4, 5] \cdot [0, 1, 2, 3]^T &= 1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25, \\
[3, 4, 6, 7] \cdot [0, 1, 2, 3]^T &= 3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37, \\
[4, 5, 7, 8] \cdot [0, 1, 2, 3]^T &= 4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.
\end{aligned}$$

- Recall that the dot product between two vectors divided by the product of their norms gives the cosine of the angle between them, which is maximum when the corresponding unit vectors are perfectly aligned.
- Intuitively, the same parameters are used to detect features in different positions of the input image.
- The "convolution" operation (see note at the end of this section) described above is typically followed by adding a bias and applying an activation function (such as ReLU) to the intermediate output image, elementwise.
- The parameters of a neuron can be organized into a (typically very small) image. These images are called (convolutional) *kernels*. In the example above, the kernel is a 2×2 image.
- A neuron in a convolutional layer has $k^2 + 1$ parameters ($k \times k$ from the kernel and one from the bias).
- The term *kernel* comes from the signal processing literature, where the operation between the input image and the *neuron image* (kernel) described above would be called *cross-correlation*.
- The term convolution also comes from the signal processing literature, where it refers to a different but highly related operation. Unfortunately, the term convolutional neural network is a misnomer.

2.1 Padding

- In the previous example, note how the output image is smaller than the output image.
- In general, the input image may be divided into windows of size $k_h \times k_w$, which are not necessarily square.
- For an input image of size $n_h \times n_w$, the output image will have size

$$(n_h - k_h + 1) \times (n_w - k_w + 1),$$

so that the output image is smaller than the input image whenever $k_h > 1$ or $k_w > 1$.

- This happens because the kernel must fit entirely inside the image before the dot product is computed.
- Convolutional neural networks typically compose several convolutional layers, so that the output of a convolutional layer becomes the input to another convolutional layer.
- In order to avoid having smaller and smaller images, convolutional layers typically add padding (filler pixels, typically set to zero) around the borders of their input image, so that the size of the output image matches the size of the original input image.
- The figure below illustrates how a 3×3 input image is padded into a 5×5 input image before a 2×2 convolutional kernel is applied so that the output image becomes a 4×4 image.
- Convolutional layers typically employ square windows of size $k \times k$, where k is an odd number. This allows the original image to be padded with $\lfloor k/2 \rfloor$ columns on the left/right, and $\lfloor k/2 \rfloor$ rows on the top/bottom (since $2\lfloor k/2 \rfloor = k - 1$).

2.2 Stride

- In our previous examples, a neuron in a convolutional layer moves across the input image from left to right, top to bottom. In other words, the corresponding kernel slides one column to the right after each use until it no longer fits inside the input image. It is then placed in the leftmost position of the next row, and the process continues.
- In general, a kernel may slide more than one column/row to the right/bottom. These hyperparameters of the convolutional layer are called horizontal/vertical stride. Our previous examples implicitly used a horizontal/vertical stride of 1, which is very common.
- The figure below illustrates a "convolution" operation with a horizontal stride 2 and vertical stride 3.
- Note that the output image will be smaller than the input image if either stride is larger than 1 (in this case, padding does not make sense). If the input image is large, this may be desirable for computational efficiency.

2.3 Multiple Input Channels

- For the sake of simplicity, we described the computations performed by a convolutional layer when it receives a *grayscale* image.

- A color image with height h and width w can be represented by a $3 \times h \times w$ tensor. This tensor can be interpreted as a list of 3 matrices, each of which may contain intensity information about a distinct primary color (red, green, or blue).
- We would say that such a color image has 3 channels. The first dimension of the corresponding tensor is called the *channel dimension*.
- The concept of image generalizes to any number of channels c, so that any $c \times h \times w$ tensor can be called an image. In this case, the content of each channel may no longer have an intuitive interpretation.
- If a convolutional layer receives an image with c channels, then a convolutional kernel must also have c channels.
- If the input image with c channels is divided into windows of size $k_h \times k_w$, a convolutional kernel would be a $c \times k_h \times k_w$ image.
- In this case, a neuron in a convolutional layer would have $ck_hk_w + 1$ parameters.
- If there is a single convolutional kernel, the output image of the convolutional layer has a single channel.
- The operations computed by the convolutional layer remain essentially the same.
- In order to compute the first output image element, a neuron is positioned at the top-left corner of the input image. A dot product is computed between the window (flattened into a vector, including all channels) and the parameters of the neuron, and the result is placed in the top-left corner of the output image.
- In the image above, the first element of the output image can be obtained by computing the dot product $[0, 1, 3, 4, 1, 2, 4, 5] \cdot [0, 1, 2, 3, 1, 2, 3, 4]^T = 56$.
- In order to compute the remaining output image elements, the convolutional layer slides the neuron across the input image (from left to right, top to bottom). As before, a dot product is computed between the window (flattened into a vector, including all channels) and the parameters of the neuron, and the result is placed in the corresponding position of the output image.
- The multichannel "convolution" operation described above is typically followed by adding a bias and applying an activation function (such as ReLU) to the intermediate output image, elementwise.

2.4 Multiple Output Channels

- For the sake of simplicity, we described the computations performed by a convolutional layer with a single neuron.
- Intuitively, the role of a neuron in a convolutional layer is to detect a particular *feature* in every single window that fits into the input image, where the window size is a hyperparameter.
- Because we considered convolutional layers with a single neuron, the output image could be interpreted as a so-called *feature map*, which indicates the presence of features in different locations.

- When a convolutional layer has c neurons, independent feature maps are computed for each of these neurons and stacked into c independent channels.
- In more detail, if a convolutional layer receives an image with c_i channels and has c_o kernels of size $k_h \times k_w$, then the output image will have c_o channels. Each channel of the output image results from sliding one of the kernels across the input image and performing the computations described earlier (including adding a bias term and computing an activation function, elementwise)
- The c_o different kernels can be represented by $c_o \times c_i \times k_h \times k_w$ tensor containing all the weights of the convolutional layer.
- Returning to a previous example, if a convolutional layer with a window size 3×3 receives a 256×256 grayscale image and has 128 neurons (units), then it would have $128 \times 1 \times 3 \times 3 = 1152$ weights.

3 Receptive field

- Because a convolutional layer receives an image and outputs an image, convolutional layers can be composed.
- Convolutional neural networks are typically built by composing several convolutional layers, so that they become deep neural networks.
- The receptive field of a neuron j is the set of all neurons (including input neurons) whose output affect the output of the neuron j.
- The set of input neurons in the receptive field of neurons in later convolutional layers is typically larger than the set of input neurons in the receptive field of neurons in earlier layers.
- Therefore, neurons in later layers are capable of detecting features that take larger patches of the input image into account.
- Deep convolutional neural networks can perform hierarchical feature detection, which motivates their usage.

4 Pooling layer

- A pooling layer receives an input image with c channels and outputs an image with c channels.
- There are two well known types of pooling layers: maximum pooling (widely used) and average pooling (rarely used).
- For each channel, the pooling layer slides a window across the corresponding matrix (from left to right, top to bottom). For each window, the maximum/average value is computed, and the result is placed in the corresponding channel and position of the output image
- The image above illustrates maximum pooling (max-pooling) with a window of size 2×2 for an input image with a single channel. The four elements are derived from computing the following:

$$\max([0, 1, 3, 4]^T) = 4, \ \max([1, 2, 4, 5]^T) = 5, \ \max([3, 4, 6, 7]^T) = 7, \ \max([4, 5, 7, 8]^T) = 8.$$

- If there were c channels instead of 1 channel, the result of the pooling operation for each of the c channels would be stacked into an output image with c channels.
- Pooling layers also have horizontal/vertical strides, which typically match the horizontal/vertical sizes of the windows to make the output image smaller than the input image.
- Intuitively, the exact position in which a feature is detected is often unimportant.
- Including pooling layers between convolutional layers enables achieving similar results to using comparatively larger convolutional kernels in the next layers, since pooling increases the receptive fields of neurons in the next layers with no additional parameters.
- Ideally, the receptive field of neurons in the last convolutional layer should contain all input neurons.

5 Fully connected layer

- A fully connected layer receives an image (or a vector) and outputs a vector. If it receives a $c \times h \times w$ image, the image is first flattened into a vector with chw elements.
- A fully connected layer is analogous to a layer in a multilayer perceptron.
- A fully connected layer is typically only followed by other fully connected layers.

6 Example: the LeNet architecture

- In 1998, **LeNet** was among the first published convolutional neural networks to gather wide attention for its performance on computer vision tasks. We will detail LeNet (with minor adaptations) as an example of convolutional neural network architecture.
- The architecture a receives a $1 \times 28 \times 28$ (grayscale) image and outputs a vector with 10 elements (since it was employed in a classification task with 10 classes). The following layers are employed:
  - A convolutional layer with 6 kernels, each a $1 \times 5 \times 5$ tensor, padding 2, stride 1, and a sigmoid activation function. **Output**: a $6 \times 28 \times 28$ tensor.
  - An average pooling layer with windows of size 2×2 and stride 2. **Output**: a $6 \times 14 \times 14$ tensor.
  - A convolutional layer with 16 kernels, each a $6 \times 5 \times 5$ tensor, padding 0, stride 1, and a sigmoid activation function. **Output**: a $16 \times 10 \times 10$ tensor.
  - An average pooling layer with windows of size 2×2 and stride 2. **Output**: a $16 \times 5 \times 5$ tensor.
  - A fully connected layer wih 120 units and a sigmoid activation function. The input is flatenned into a vector with $16 \cdot 5 \cdot 5 = 400$ elements. **Output**: a vector with 120 elements.
  - A fully connected layer with 84 units and a sigmoid activation function. **Output**: a vector with 84 elements.
  - A fully connected layer with 10 units and a softmax activation function **Output**: a vector with 10 elements. *Note: the original LeNet used a so-called Gaussian activation layer, which is currently rarely used.*

7 Recommended reading

- Dive into Deep Learning: Chapter 7.

8 [Storing this notebook as a pdf]

- In order to store this notebook as a pdf, you will need to hide the images included in the previous cells using the following syntax:
  - `<!--- ![Image caption.](https://link.to.image) --->`

[14]:
```
%%capture
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)

!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic

# Set the path to this notebook below (add \ before spaces). The output `pdf`␣
↪will be stored in the corresponding folder.
!jupyter nbconvert --to pdf /content/gdrive/My\ Drive/Colab\ Notebooks/nndl/
↪week_06/lecture/02_Convolutional_Neural_Networks.ipynb

# If having issues, save this notebook (File > Save) and restart the session␣
↪(Runtime > Restart session) before running this cell. To debug, remove the␣
↪first line (`%%capture`).
```