Image Classification

Juan Elenter, Ignacio Hounie, and Alejandro Ribeiro[*]

April 2, 2023

1 Convolutional Neural Networks in Space

To process images we use spatial Convolutional Neural Networks (CNNs). Spatial CNNs are defined as compositions of layers, which are themselves compositions of spatial convolutional filters with pointwise nonlinearities. Thus, for a network with L layers we consider L different filters. The filter at Layer ℓ has coefficients $h_{\ell,uv}$ that are entries of the matrix $\mathbf{H}_\ell$. We then define the input output relationship of the CNN through the recursion

$$\mathbf{X}_\ell = \sigma\Big(\mathbf{Z}_\ell\Big) = \sigma\left(\sum_{u=-K_\ell}^{K_\ell}\ \sum_{v=-K_\ell}^{K_\ell} h_{\ell,un}\, \mathcal{S}_{\mathrm{V}}^{u}\, \mathcal{S}_{\mathrm{H}}^{v}\, \mathbf{X}_{\ell-1}\right), \tag{1}$$

which we initialize with $\mathbf{X}_0 = \mathbf{X}$ and terminate at Layer L. We further define $\mathcal{H}$ as a tensor that groups all of the filters of all of the layers and write the output of the CNN as,

$$\mathbf{\Phi}(\mathbf{X};\mathcal{H}) = \mathbf{X}_L. \tag{2}$$

This is the same architecture that we used to define CNNs for time signals, which is also the same architecture we used to define GNNs; see Figure 1. The differences between these three different types of CNNs is the use of convolutions that are adapted to the specific structure of the input. We used one dimensional convolutions to process signals in time,

[*]In alphabetical order.

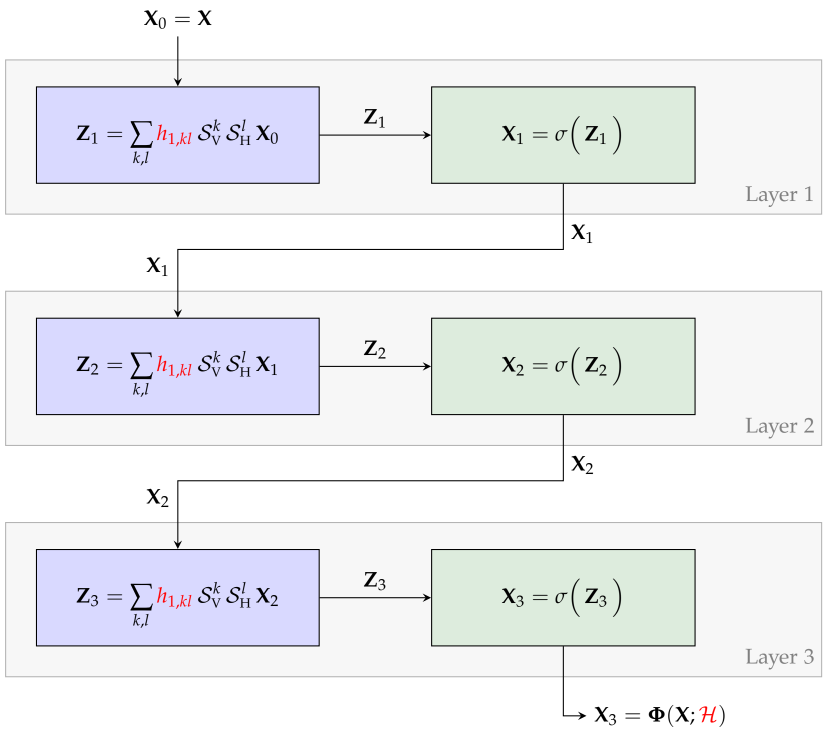
$X_0 = X$
$Z_1 = \sum_{k,l} h_{1,kl}\, \mathcal{S}_V^k\, \mathcal{S}_H^l\, X_0$
Z_1
$X_1 = \sigma\big(Z_1\big)$
Layer 1
X_1
X_1
$Z_2 = \sum_{k,l} h_{1,kl}\, \mathcal{S}_V^k\, \mathcal{S}_H^l\, X_1$
Z_2
$X_2 = \sigma\big(Z_2\big)$
Layer 2
X_2
X_2
$Z_3 = \sum_{k,l} h_{1,kl}\, \mathcal{S}_V^k\, \mathcal{S}_H^l\, X_2$
Z_3
$X_3 = \sigma\big(Z_3\big)$
Layer 3
$X_3 = \Phi(X; \mathcal{H})$

Figure 1. A Convolutional Neural Network (CNN) with three layers. To process images we use spatial CNNs. Spatial CNNs are compositions of layers, which are themselves compositions of a spatial convolutions with pointwise nonlinearities. [cf. (2)]. This computation structure is shared with time CNNs and GNNs. They differ in the adaptation of the filter to the specific domain.

graph convolutions to process graph signals, and we are now using two dimensional convolutions to process images.

In case it is worth repeating a third time, the recursion in (2) begins at Layer 1 where the input image $\mathbf{X} = \mathbf{X}_0$ is processed by the convolutional filter $\mathbf{H}_\ell$. This produces the intermediate output $\mathbf{Z}_1$ that is passed through the pointwise nonlinearity $\sigma(\cdot)$ to produce the Layer 1 output $\mathbf{X}_1$. This image is now passed onto Layer 2 where it is processed with the convolutional filter $\mathbf{H}_2$ and the pointwise nonlinearity $\sigma(\cdot)$. The output of Layer 2 is passed on to Layer 3 where we repeat the process of applying a convolutional filter, $\mathbf{H}_3$ in this case, and a pointwise nonlinearity. If, as is the case of Figure 1, the CNN has $L = 3$ layers, the output of Layer 3 is declared to be the output of the CNN.

1.1 Layers with Multiple Features

As we did for one dimensional CNNS and GNNs, we increase the representation power of two dimensional CNNs with the addition of multiple features per layer. To explain this, we introduce the notation $\mathbf{H} \star \mathbf{X}$ to denote the spatial convolution of filter $\mathbf{H}$ with image $\mathbf{X}$,

$$\mathbf{Z} = \mathbf{H} \star \mathbf{X} = \sum_{k=-K}^{K} \sum_{l=-K}^{K} h_{kl}\, \mathcal{S}_{\text{V}}^{k}\, \mathcal{S}_{\text{H}}^{l}\, \mathbf{X}. \tag{3}$$

With this notation the CNN recursion in (2) can be written as

$$\mathbf{X}_\ell = \sigma\Big(\mathbf{Z}_\ell \Big) = \sigma\Big(\mathbf{H}_\ell \star \mathbf{X}_{\ell-1} \Big). \tag{4}$$

In a CNN with multiple features, each layer processes multiple images in parallel to produce a number of images at the output. To write this formally we let each layer produce as an output a collection of F_ℓ features $\mathbf{X}_\ell^g$. Each of these features is an image. These features are produced by processing the $F_{\ell-1}$ features $\mathbf{X}_{\ell-1}^f$ that are output by Layer $\ell - 1$.

The mapping from features $\mathbf{X}_{\ell-1}^f$ into features $\mathbf{X}_\ell^g$ is determined by a collection of convolutional filters $\mathbf{H}^{fg}$. The specific relationship is

$$\mathbf{X}_\ell^g = \sigma \left(\sum_{f=1}^{F} \mathbf{H}^{fg} \star \mathbf{X}^f \right) . \tag{5}$$

The expression in (5) is such that all input features $\mathbf{X}^f$ can affect all output features $\mathbf{X}^g$. The influence of input $\mathbf{X}^f$ on output g is the convolution $\mathbf{H}^{fg} \star \mathbf{X}^f$. All of the convolutions for a fixed output feature index g are summed and then passed through a pointwise nonlinearity.

1.2 Spatial Convolutional Neural Network Specification

To specify a CNN we need to specify the number of layers L and the characteristics of the filters that are used at each layer. The latter are the number of filter taps K_ℓ and the number of features F_ℓ at the output of the layer. The number of features F_0 must match the number of features at the input and the number of features F_L must match the number of features at the output. Observe that the number of features at the output of Layer $(\ell - 1)$ determines the number of features at the input of Layer ℓ.

Task 1 Program a class that implements a spatial CNN with L layers. This class receives as initialization parameters a CNN specification consisting of the number of layers L and vectors $[K_1, \dots, K_L]$ and $[F_0, F_1, \dots, F_L]$ containing the number of taps and the number of features of each layer. Use ReLU activations.

Endow the class with a method that takes an input feature $\mathbf{X}$ and produces the corresponding output feature $\mathbf{\Phi}(\mathbf{X}; \mathcal{H})$. ■

We have provided two solutions of Task 1. In the first solution we use our own implementation of the convolutional filter. In the second implementation we just call the Pytorch implementation. We do that because the Pytorch function is a wrapper that calls a C implementation of convolutions with multiple features. This is numerically more efficient.

Task 2 Instantiate a CNN with 1 layer. In this CNN we have $K_1 = 1$, $F_0 = 3$, and $F_1 = 3$. Set the filters to

$$\mathbf{H}^{f1} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \mathbf{H}^{f2} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \mathbf{H}^{f3} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}. \tag{6}$$

Create an input signal made up of three *different* images that correspond to the *same* digit. Process these input with the CNN and plot the output. Give some interesting observations. ■

2 Pooling

As we did for signals in time, we define a pooling operator to reduce dimensionality. The procedures are analogous except that when we implement pooling in images we need to average in the horizontal and vertical direction. For instance average pooling is an average over a square window with Δ pixels in each direction,

$$x(u,v) \;=\; \frac{1}{\Delta^2}\left(\sum_{u=m\Delta}^{m\Delta+(\Delta-1)}\;\sum_{v=n\Delta}^{n\Delta+(\Delta-1)} w(m,n)\right) \,. \tag{7}$$

Likewise, max pooling chooses the maximum value over a square window with Δ pixels in each direction,

$$x(u,v) \;=\; \max_{u\in[m\Delta,\; m\Delta+(\Delta-1)]}\;\max_{v\in[n\Delta,\; n\Delta+(\Delta-1)]} w(m,n) \,. \tag{8}$$

Max pooling is the most common choice in practice but average pooling is also popular. As is the case of signals in time, pooling is effective when the elements that are being pooled are similar. In such case, there is not much difference between using max or average pooling.

Task 3 Modify the CNN of Task 1 to incorporate pooling. This can be done by modifying the method that implements convolutional layers to incorporate the pooling operation.

As in any CNN the initialization parameters include the number of layers L along with vectors $[K_1, \ldots, K_{L-1}]$ and $[F_0, F_1, \ldots, F_{L-1}]$. These vectors contain the number of taps K_ℓ of the filters used at each layer and the number of features F_ℓ at the output of each layer.

Since we are incorporating pooling the initialization parameters must also include the vector $[N_0, N_1, \ldots, N_L]$ containing the dimension N_ℓ of the features at the output of each layer. Notice that N_0 matches the dimension of the input signal.

The forward method of this class takes a tensor $\mathbf{X}$ in which each slice is an image with $N_0 \times N_0$ pixels. The total number of slices in this tensor is F_0.

Use relu nonlinearities in all layers. Use average pooling in all convolutional layers. ■

Task 4 Modify the CNN of Task 3 to incorporate a readout layer. This readout layer is the same as the readout layer that we implemented for time convolutions in Lab 2C.

■

Task 5 Instantiate the CNN of Task 4 with 2 convolutional layers, taps$K_0 = K_1 = 1$, features $[1, 4, 8]$, and feature dimensions $[28, 14, 7]$.

Use this CNN to train a classifier for digit classification. Remember that this requires splitting the dataset into train and test sets. Use a learning rate of 0.01, batch size 128, and train for 5 epochs. Evaluate the train loss, the test loss, and the classification accuracy. ■

3 Report

Do not take much time to prepare a lab report. We do not want you to report your code and we don't want you to report your work. Just give us answers to questions we ask. Specifically give us the following:

Question	Report deliverable
Task 1	Do not report
Task 2	One paragraph with observations
Task 3	Do not report
Task 4	Do not report
Task 5	Report training loss
Task 5	Report test loss
Task 5	Report relative classification error

We will check that your answers are correct. If they are not, we will get back to you and ask you to correct them. As long as you submit responses, you get an A for the assignment. It counts for 10% of your lab grade.