Lab 2B: Convolutional Neural Networks

Ignacio Boero, Juan Cerviño, Juan Elenter,
Ignacio Hounie and Alejandro Ribeiro*

February 18, 2024

1 Convolutional Neural Networks

A convolutional neural network (CNN) is a neural network in which the linear maps that are used in each layer are convolutional filters; see Figure 4. Using $h_{\ell k}$ to denote the filter coefficients used at Layer ℓ, a CNN is defined by the recursion

$$\mathbf{x}_0 = \mathbf{x}, \quad \mathbf{x}_\ell = \sigma\Big(\mathbf{z}_\ell \Big) = \sigma \left(\sum_{k=0}^{K} h_{lk}\, \mathcal{S}^k \mathbf{x} \right), \quad \mathbf{x}_L = \mathbf{\Phi}(\mathbf{x}; \mathcal{H}). \tag{1}$$

In (1) we use $\mathbf{\Phi}(\mathbf{x}; \mathcal{H})$ to denote the output of the CNN, with $\mathcal{H} = [h_{11}, \ldots, h_{KL}]$ being a tensor that groups all of the filter coefficients that make up the layers of the CNN.

Notice that since convolutions are linear operations, a CNN is a particular case of a standard neural network. This being true, a CNN is still a composition of layers each of which is itself the composition of a linear map with a pointwise nonlinearity. The only difference is that instead of generic linear maps, layers use (linear) convolutions.n When we want to highlight that a neural network is *not* convolutional, we call it a fully connected neural network (FCNN).

*In alphabetical order.

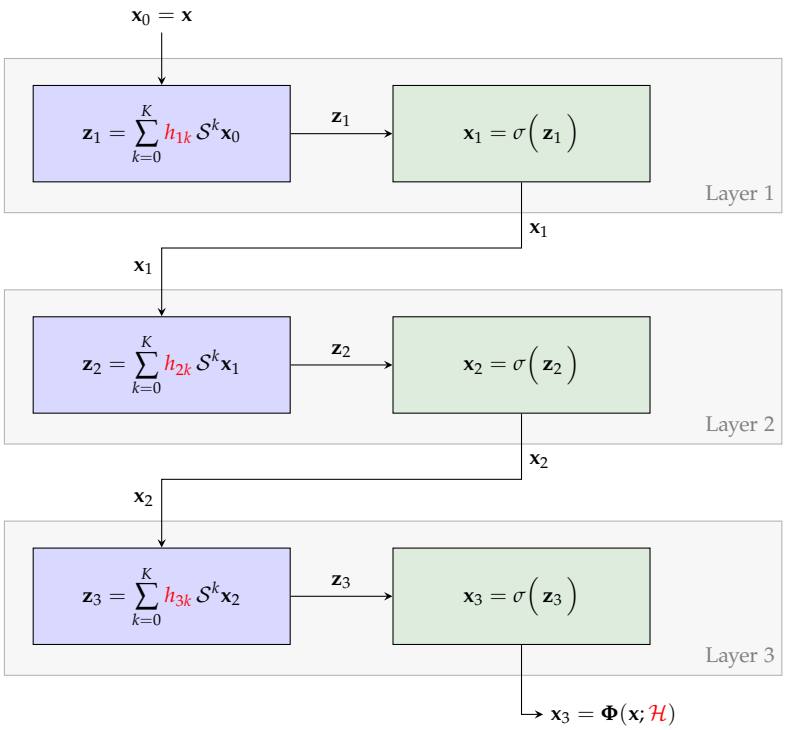

Figure 1. A convolutional neural network (CNN) with three layers. A CNN is a neural network in which the linear maps used in each layer are convolutional filters.

An important observation to make is that since the nonlinear operations that are used at each layer are pointwise, the CNN inherits the shift invariance of convolutions. This is ready to show but worth highlighting as a proposition.

Proposition 1 *Given a CNN with filter tensor $\mathcal{H}$ and an input signal $\mathbf{x}$ let $\mathbf{y} = \mathbf{\Phi}(\mathbf{x}; \mathcal{H})$ be the output of the CNN [cf. (1)]. If $\mathbf{x}_s = \mathcal{S}\mathbf{x}$ is a shifted version of $\mathbf{x}$ the CNN output $\mathbf{y}_s = \mathbf{\Phi}(\mathbf{x}_s; \mathcal{H})$ is the corresponding shifted version of $\mathbf{y}$,*

$$\mathbf{y}_s = \mathbf{\Phi}(\mathbf{x}_s; \mathcal{H}) = \mathbf{\Phi}(\mathcal{S}\mathbf{x}; \mathcal{H}) = \mathcal{S}\mathbf{\Phi}(\mathbf{x}; \mathcal{H}) = \mathcal{S}\mathbf{y} \tag{2}$$

Proof: In each layer of the CNN we know that convolutions are equivariant to shifts. Thus if the intermediate output of Layer ℓ is $\mathbf{z}_\ell = \mathbf{h}_\ell * \mathbf{x}_{\ell-1}$

we known that

$$\mathcal{S}\mathbf{z}_\ell = \mathbf{h} * (\mathcal{S}\mathbf{x}_{\ell-1}). \tag{3}$$

We further know that the nonlinear operation σ is pointwise and that it has the same effect in a component irrespectively of the position of this component in the vector on which σ is acting. Thus,

$$\sigma(\mathcal{S}\mathbf{z}_\ell) \;=\; \mathcal{S}\sigma(\mathbf{z}_\ell). \tag{4}$$

Combining (3) with (4) we conclude that each layer is shift equivariant,

$$\mathcal{S}\mathbf{x}_\ell = \sigma\Big(\mathbf{h} * (\mathcal{S}\mathbf{x}_{\ell-1})\Big). \tag{5}$$

The result follows because the composition of equivariant layers yields an equivariant output. Indeed, suppose that we feed input $\mathbf{x}_0 = \mathbf{x}$ to the CNN. This input results in the observation of $\mathbf{x}_1$ at the output of Layer 1. It follows from (5) applied to $\ell = 1$ that if the input to the CNN is $\mathcal{S}\mathbf{x}_0$ the output of Layer 1 is $\mathcal{S}\mathbf{x}_1$. We can no repeat the same argument to conclude that if the output of Layer 2 when the input to the CNN is $\mathbf{x}_0$ is $\mathbf{x}_2$, the output of Layer 2 when the input to the CNN is $\mathcal{S}\mathbf{x}_0$ is $\mathcal{S}\mathbf{x}_2$. Repeating this argument L times shows that if $\mathbf{x}_L = \mathbf{\Phi}(\mathbf{x}; \mathcal{H})$ is the output of the CNN when the input is $\mathbf{x}$, the shifted input $\mathcal{S}\mathbf{x}$ yields the output $\mathcal{S}\mathbf{x}_L$. This is what we wanted to prove. ■

Locality and shift invariance are the motivations for introducing convolutions. CNNs inherit these two properties. We can then think of CNNs as generalizations of convolutions. We can, in fact, argue that CNNs are minor variations of convolutions. They are nonlinear operators designed to stay as close as possible to linear convolutional filters. We just add pointwise nonlinearities.

Do notice that we are not saying that this minor architectural variation is irrelevant. Quite the contrary, it has a major effect on the practical performance of CNNs as we explore in this lab.

Task 1 Program a class to implement CNNs. This class receives as initialization parameters the number of layers L and the number of taps of the filters of each layer. The number of taps per layer is given as a vector with L entries. Endow the class with a forward methods that takes a vector $\mathbf{x}$ as an input and works through the recursion in (1) to return the CNN output $\mathbf{\Phi}(\mathbf{x}; \mathcal{H})$. Use tanh nonlinearities in all layers. ■

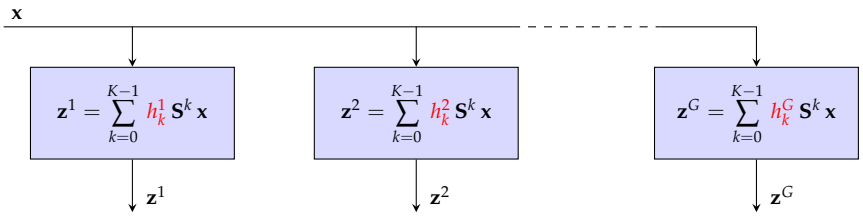

Figure 2. Filterbanks. A filterbank is a collection of convolutional filters that process the input signal $\mathbf{x}$ in paralell. The outputs of each of these filters are called features and are intended to capture different properties of the input signal.

2 Filterbanks

To make CNNs more effective we need to make them more complex. We do that by incorporating multiple channels. That is, in each of the layers we process the input using multiple convolutional filters in parallel. This is illustrated in Figure 2 where the signal $\mathbf{x}$ is sent through a collection of G filters. If we denote as h_k^g the coefficients of the gth filter, Channel g implements the convolution

$$\mathbf{y}^g = \sum_{k=0}^{K} h_k^g \mathcal{S}^k \mathbf{x}, \tag{6}$$

The output $\mathbf{y}^g$ us called a feature and the collection of F features is called a filterbank.

For implementation purposes it is convenient to group the output of the filterbank in a matrix $\mathbf{Y}$ in which each of the columns of $\mathbf{Y}$ represents a different feature $\mathbf{y}^g$. Using this definition we can write the filerbank as

$$\mathbf{Y} = \left[\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^G\right] = \left[\sum_{k=0}^{K} h_k^1 \mathcal{S}^k \mathbf{x}, \sum_{k=0}^{K} h_k^2 \mathcal{S}^k \mathbf{x}, \ldots, \sum_{k=0}^{K} h_k^G \mathcal{S}^k \mathbf{x}\right]. \tag{7}$$

If we further define the filterbank coefficients $\mathbf{h}_k = [h_k^1, h_k^2, \ldots h_k^G]$ the filterbank in (7) can be rewritten as

$$\mathbf{Y} = \sum_{k=0}^{K} \mathcal{S}^k \mathbf{x} \mathbf{h}_k. \tag{8}$$

In (8) the input is a column vector $\mathbf{x}$ of dimension $N \times 1$. The filterbank coefficients $\mathbf{h}_k$ are row vectors of dimension $1 \times G$. It follows that the

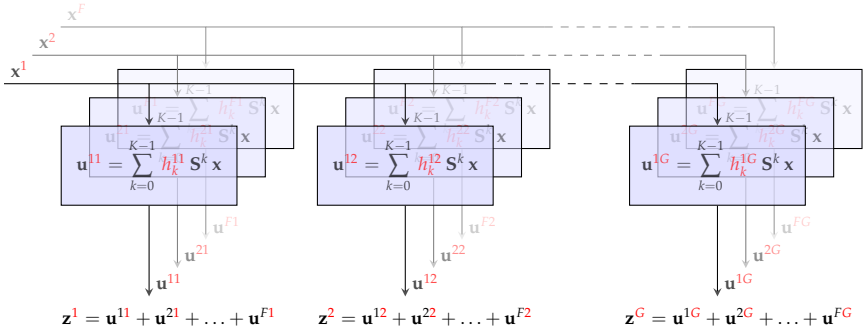

Figure 3. Multiple-Input-Multiple-Output (MIMO) Filters.

output $\mathbf{Y}$ is a matrix of dimension $N \times G$. In this matrix each separate column is a different feature. Each different feature is the result of processing the input signal in a different way. The intention is that each of this different ways of processing $\mathbf{x}$ results in the extraction of different pieces of information from $\mathbf{x}$. This is why we call them features.

2.1 Multiple-Input-Multiple-Output (MIMO) Filters

Multiple input (MI) features can be processed with separate filterbanks to produce multiple output (MO) features. In these MIMO filters the input is a matrix $\mathbf{X}$ and the output is another matrix matrix $\mathbf{Y}$. The MIMO filter coefficients are matrices $\mathbf{H}_k$ and the MIMO filter itself is a generalization of (8) in which the matrix $\mathbf{H}_k$ replaces the vector $\mathbf{h}_k$,

$$\mathbf{Y} = \sum_{k=0}^{K} \mathcal{S}^k \mathbf{X} \mathbf{H}_k. \tag{9}$$

In (9), the input feature matrix $\mathbf{X}$ has dimension $N \times F$ and the output feature matrix $\mathbf{Y}$ has dimension $N \times G$. This means that each of the F columns of $\mathbf{X}$ represents a separate input feature whereas each of the G columns of $\mathbf{Y}$ represents an output feature. To match dimensions, the filter coefficient matrices $\mathbf{H}_k$ must be of dimension $F \times G$.

Task 2 Program a class that implements a MIMO filter. This class has as attributes the length of the filter K and the dimensions F and G of the

filter coefficients. The filter coefficients themselves are also an attribute of the class. Endow the class with a forward method that takes an input feature $\mathbf{X}$ and produces the corresponding output feature $\mathbf{Y}$. ■

To gain more insight on MIMO filters write the input feature as $\mathbf{X} = [\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^F]$ and the output feature as $\mathbf{Y} = [\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^G]$. We can then unpack (9) into the expression

$$\mathbf{y}^g = \sum_{f=1}^{F} \left(\sum_{k=0}^{K} h_k^{fg} \mathcal{S}^k \mathbf{x}^f \right) := \sum_{f=1}^{F} \mathbf{u}^{fg}. \tag{10}$$

We then see that (9) represents two different operations: (i) A collection of $F \times G$ filterbanks – the inner sum. (ii) A linear aggregation to combine outputs of some of these filters – the outer sum.

This is represented in Figure 3. Each of the input features $\mathbf{x}^f$ is processed by a filterbank made up of G filters. The coefficients of each of the filters in this bank are h_k^{fg} and their action on $\mathbf{x}^f$ results in the creation of feature $\mathbf{u}^{fg}$. We then proceed to sum all the features $\mathbf{u}^{fg}$ associated with a given g. This is the output feature $\mathbf{y}^g$.

Adding a sum of features to a MIMO filter is an ad-hoc solution to avoid excessive growth in the number of features at the output of each layer. After processing the input features with separate filterbanks we have a total of $F \times G$ features $\mathbf{u}^{fg}$. Without a step to reduce the number of features we may end up with exponential growth in the number of features in a layered architecture. Summing the features $\mathbf{u}^{fg}$ reduces the number of features to G. It gives explicit control on the number of features at the output of each layer.

3 Real Convolutional Neural Networks

The MIMO filters of Section 2.1 can be used to define MIMO CNNs. A MIMO CNN is a neural network in which each of the layers is the composition of a MIMO convolutional filter with a pointwise nonlinearity. If at each layer we denote filter coefficients as $\mathbf{H}_{\ell k}$, the MIMO CNN is given

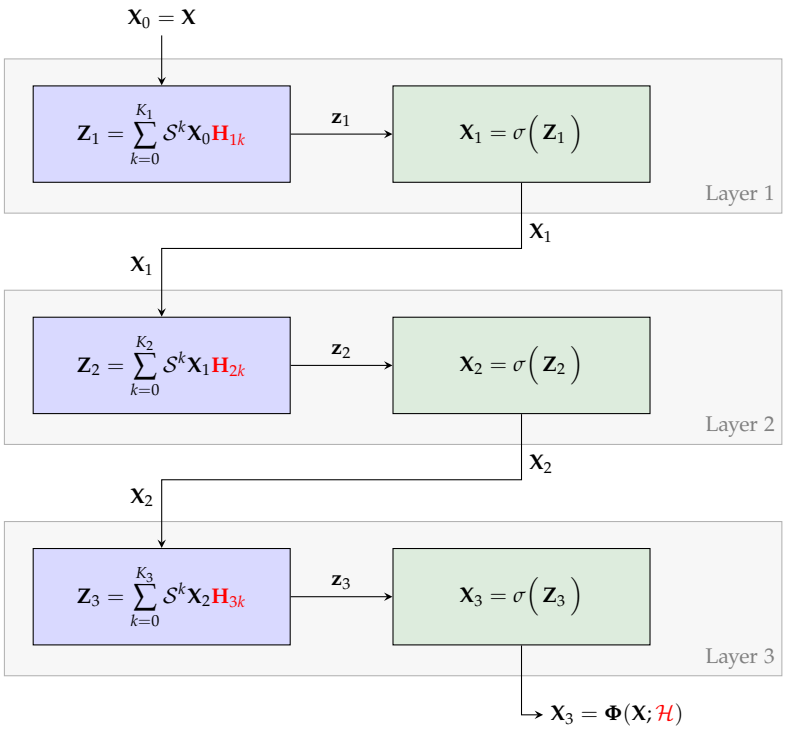

Figure 4. Multiple-Input-Multiple-Output (MIMO) Convolutional Neural Networks (CNNs). CNNs that are used in practice contain several parallel convolutional filters at each layer. This is captured here with the use of multiple features per layer and the corresponding use of MIMO convolutional filters [cf. (9)].

by the recursion

$$\mathbf{X}_0 = \mathbf{X}, \quad \mathbf{X}_\ell = \sigma\Big(\mathbf{z}_\ell\Big) = \sigma\left(\sum_{k=0}^{K_\ell} \mathcal{S}^k \mathbf{X} \mathbf{H}_{\ell k}\right), \quad \mathbf{x}_L = \mathbf{\Phi}(\mathbf{x}; \mathcal{H}). \tag{11}$$

Thus, we have a composition of layers each of which is itself the composition of a MIMO convolutional filter with a pointwise nonlinearity.

The CNN of Section 1 was introduced for didactic purposes because almost all CNNs used in practice are MIMO CNNs – hence, the title of this section. For that reason we never call them MIMO CNNs, we just call them CNNs.

3.1 Convolutional Neural Network Specification

To specify a CNN we need to specify the number of layers L and the characteristics of the filers 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. Then, the filter coefficients at Layer ℓ are of dimension $F_{\ell-1} \times F_\ell$.

Task 3 Program a class that implements a CNN with L layers. This class receives as initialization parameters a CNN specification consisting of the number of layers L and vectors $[K_1, \ldots, K_L]$ and $[F_0, F_1, \ldots, F_L]$ containing the number of taps and the number of features of each layer.

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

3.2 Convolutions and Convolutional Neural Networks

Convolutions and CNNs are at the same time very similar and very dissimilar.

Convolutions and CNNs are similar because at a fundamental level CNNs are just as simple as convolutions. They are, in the end, just a collection of convolutional filters composed with pointwise nonlinearities. These are careful choices to maintain the locality and shift equivariance of convolutions.

Convolutions and CNNs are different because the complexity of a CNN is at some length of the complexity of a convolutional filter. Suppose that we consider a CNN with 20 features per layer and 5 layers. This CNN contains a grand total of $(20 \times 20) \times 5 = 2,000$ convolutional filters. This makes CNNs more *expressive* than filters. It makes them more capable of identifying relevant features that we can leverage in the processing of the input.

It bears repeating that the vast complexity increase of CNNs does not

come at the cost of giving up on the fundamental properties of convolutions. Fully connected neural networks are also more complex than convolutions. They are also more complex than CNNs. But, as we have seen, they are unworkable.

We close by pointing out that the use of layers is known to make CNNs more stable to perturbations. This is the reason why deep architectures with several layers are preferred relative to shallow architectures with a small number of layers. The reasons why this happen are also known but their explanation requires the introduction of Fourier transforms and frequency representations of convolutional filters.

3.3 Audio Processing with Convolutional Neural Networks

Using the same dataset prepared for Lab 2A, we ask that you process it with a CNN.

Task 4 Load audio data from the link provided in the course site, and unzip it. This file contains a pytorch tensor with Q = 1500 pairs of audio recordings $(\mathbf{x}_q, \mathbf{y}_q)$. Split the dataset into training and testing sets, keep 100 samples for testing. Train a CNN to remove the background noise. This CNN has 2 layers, the numbers of features per layer are $[1, 5, 1]$ and the filter lengths are $[40, 40]$. Use a learning rate of 0.5, batch size of 32 and train for 20 epochs. Evaluate the train and test loss (use L1 loss as in Lab 2A). ■

4 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	Do not report
Task 3	Do not report
Task 4	Report training loss
Task 4	Report test loss

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.