CS 419M Introduction to Machine Learning **Spring 2021-22**

Lecture 18: Introduction to CNN

Lecturer: Abir De *Scribe: Group 18A*

Participants: Mitalee Oza, Parth Dange, Harshit Shrivastava, Varad Mahashabde

18.1 Flow of the notes

1. Convolution Operators
2. Pooling Operators
3. Example for a basic CNN
4. Transfer Learning Model

18.1.1 Why Convolution?

We know from the Universal Approximation Theorem that any function can be modelled using Feed-Forward Neural Networks (FFNNs). However the number of parameters can be unmanageably large, requiring a large number of examples and computational resources. However, we can reduce the number of parameters needed by leveraging existing structure in the input.
Convolutional Neural Networks (CNNs) are commonly used to process image and video data. They improve on FFNNs by respecting the spatial and temporal structure of the image, where convolution operations combine data in a space-time neighbourhood, and pooling combines data in chunks representing larger blocks of data. This leads to huge reduction in number of parameters.

18.1.2 Convolution Operators

A convolution is a linear operation that involves the multiplication of a set of weights with the input. Element wise multiplication is performed between an array of input data and an array of weights called a filter or a kernel. These can be thought of as neurons most of whose weights are artificially held to be 0.

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$	$x_{1,5}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$	$x_{2,5}$
$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$	$x_{3,5}$
$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$	$x_{4,5}$
$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$	$x_{5,5}$

Table 18.1: Input to Convolution Layer

$w_{1,1}$	$w_{1,2}$	$w_{1,3}$
$w_{2,1}$	$w_{2,2}$	$w_{2,3}$
$w_{3,1}$	$w_{3,2}$	$w_{3,3}$

Table 18.2: Kernel

The operation: You start with a kernel (matrix of weights). Starting at a corner of the input, the kernel is multiplied element wise to the array of an input. The kernel slides over the 2D input performing such element wise operations. The matrix formed by these outputs is the activation map. Each element wise multiplication may further be operated on by a non linearity to form the activation map.

x1 * w1	x2 * w2	x3* w3	x4	x5
x6* w4	x7* w5	x8* w6	x9	x10
x11* w7	x12* w8	x13* w9	x14	x15
x16	x17	x18	x19	x20
x21	x22	x23	x24	x25

Figure 18.1: Element-wise multiplication for first convolution output

$$
\begin{aligned}
z_{1,1} = b \quad & + w_{1,1} \cdot x_{1,1} && + w_{1,2} \cdot x_{1,2} && + w_{1,3} \cdot x_{1,3} \\
& + w_{2,1} \cdot x_{2,1} && + w_{2,2} \cdot x_{2,2} && + w_{2,3} \cdot x_{2,3} \\
& + w_{3,1} \cdot x_{3,1} && + w_{3,2} \cdot x_{3,2} && + w_{3,3} \cdot x_{3,3}
\end{aligned}
$$

$$o_{1,1} = g(z_{1,1})\,\mathrm{ReLU}(z_{1,1})$$

x1	x2*w1	x3*w2	x4*w3	x5
x6	x7*w4	x8*w5	x9*w6	x10
x11	x12*w7	x13*w8	x14*w9	x15
x16	x17	x18	x19	x20
x21	x22	x23	x24	x25

Figure 18.2: Element-wise multiplication for second convolution output

$$
\begin{aligned}
z_{1,2} = b \quad & + w_{1,1} \cdot x_{1,2} && + w_{1,2} \cdot x_{1,3} && + w_{1,3} \cdot x_{1,4} \\
& + w_{2,1} \cdot x_{2,2} && + w_{2,2} \cdot x_{2,3} && + w_{2,3} \cdot x_{2,4} \\
& + w_{3,1} \cdot x_{3,2} && + w_{3,2} \cdot x_{3,3} && + w_{3,3} \cdot x_{3,4}
\end{aligned}
$$

$$o_{1,2} = g(z_{1,2}) = \mathrm{ReLU}(z_{1,2})$$

$o_{2,1}$	$o_{2,2}$	$o_{2,3}$
$o_{3,1}$	$o_{3,2}$	$o_{3,3}$
$o_{1,1}$	$o_{1,2}$	$o_{1,3}$

Table 18.3: Kernel

K filters would result in K different activation maps. Further convolution operation can also be used for 3 dimensional images. The filters for this case would be 3 dimensional with the same depth as the image.

18.1.3 Pooling Operators

In pooling, the output from convolution is taken and a filter is applied whose position is such that there is no overlap. The stride will be equal to the length of the filter. Sliding is also used in pooling.

o1	o2	o3	o4
o5	o6	o7	o8
o9	o10	o11	o12
o13	o14	o15	o16

Figure 18.3: Filter in pooling

There are two operations in pooling:

- **Max pooling**: Each cell will be the maximum of all the operations in the filter

max(o1, o2, o5, o6)	max(o3, o4, o7, o8)
max(o9, o10, o13, o14)	max(o11, o12, o15,16)

Figure 18.4: Max pooling

- **Avg pooling** Each cell will be the average of all the operations in the filter

avg(o1, o2, o5, o6)	avg(o3, o4, o7, o8)
avg(o9, o10, o13, o14)	avg(o11, o12, o15,16)

Figure 18.5: Average pooling

18.1.4 Example for a Basic CNN

Let us consider a basic CNN with the following architecture, where the values in the indices represent the relevant tensor dimensions of the filter.

$$\begin{aligned}\text{Input}_{28\times 28} &\rightarrow \text{Convolution}_{16}^{3\times 3} \rightarrow \text{Pooling}^{2\times 2} \rightarrow \text{Convolution}_{32}^{16,3\times 3} \rightarrow \text{Pooling}^{2\times 2} \\ &\rightarrow \text{Flatten / Vectorize} \rightarrow \text{FFNN} \rightarrow \text{Output}\end{aligned}$$

Layer type and Filter Shape	Input Shape	Output Shape	Parameters
Input Image		(28×28)	
Convolution Layer with 16 filters of size 3×3 and stride=1 $\implies$ Filter size $= \mathcal{F}_{16}^{3\times 3}$	(28×28)	$(16, 26 \times 26)$	$16 \cdot (3 \cdot 3 + 1) = 160$
Pooling Layer with size 2×2 and stride=2 on each filter	$(16, 26 \times 26)$	$(16, 13 \times 13)$	0 (has only hyper-parameters)
Convolution Layer with 32 filters of size 3×3 and stride=1 $\implies$ Filter size $= \mathcal{F}_{32}^{16,3\times 3}$	$(16, 13 \times 13)$	$(32, 11 \times 11)$	$32 \cdot (16 \cdot 3 \cdot 3 + 1) = 4640$
Pooling Layer with size 2×2 and stride=2 on each filter	$(32, 11 \times 11)$	$(32, 5 \times 5)$	0
Flatten/Vectorize	$(32, 5 \times 5)$	$32 \cdot 5 \cdot 5 = 800$	

Table 18.4: Shape and Parameter counts of the various layers

We thus observe that we have obtained 800 outputs from 784 inputs from the initial image. However, the new outputs necessarily respect the spatial structure of the image and contain higher-order information about the image. Thus we can have a much smaller FFNN to produce the final output. The reduction in parameters is much more drastic for inputs with larger dimensions.
We also note that some information was lost in the second Pooling step as the image could not be neatly divided into pooling regions. We can pad the input image with a few extra empty rows and columns to avoid this.

18.1.5 Transfer Learning

The basic idea behind Transfer Learning is to re-use the learned feature maps of a different model and then fine tune them to the specific model we are trying to develop. This proves advantageous as we utilise these learned feature maps without having to start from scratch by training a large model on a large dataset.

There are two ways we can use this Pre-trained model:

- **Feature Extraction:** Here we use the representations learnt by the pre-trained model and add a new classifier. This new classifier will be learnt from scratch to repurpose the feature maps learned previously for the dataset. We do not need to re-train the entire model.
- **Fine Tuning:** In this method, we freeze all the base layers of the models except the few at the top. Then we add new classifier layers and train these layers along with the unfrozen top layers of the pre-trained model. This way, the higher-order feature representations can be "fine-tuned" for our specific task

18.2 Group Details and Individual Contribution

- Mitalee Oza: Convolution Operators

- Harshit Shrivastava: Transfer Learning Model
- Varad Mahashabde: Example CNN
- Parth Dange: Pooling Operators