EECS 182 Deep Neural Networks **Spring 2023**

Lecture 6: February 13

Lecturer: Anant Sahai *Scribe: Rahul Khorana*

6.1 Agenda and Viewpoint

Last time we were working on adaptive methods, today we will shift focus.
Agenda:

- Survey of Topics
- Conv-Net Basics
- Normalization
- Dropout

6.1.1 Architectures Viewpoint

Architectures:	MLP	CNN	RNN	GraphNN	Transformer-Based	Problem Domains
Regression Classification Generation						Vision NLP Control+Robotics Vision
Recommender Systems						

6.1.1.1 Note:

This diagram represents a sequence of ideas that we will introduce with respect to the different problem categories (Regression, Classification, Generation), and the various problem domains. It should be noted that problem domains is a third dimension orthogonal to the other two dimensions (Architecture, Problem Categories). We do not discuss recommender systems, hence the separation.

6.2 Convolutional Neural Networks

6.2.1 Key Ideas

- Respect Locality in Sub-Function Kernel $\implies$ Convolutional Structure with small filters.
- Respect invariances/equivariances within data for the problem domain $\implies$ Weight Sharing and Data Augmentations.
- Support Hierarchical Structure & a Multi-Resolution perspective

6.2.1.1 Why is it called a Conv-Net?

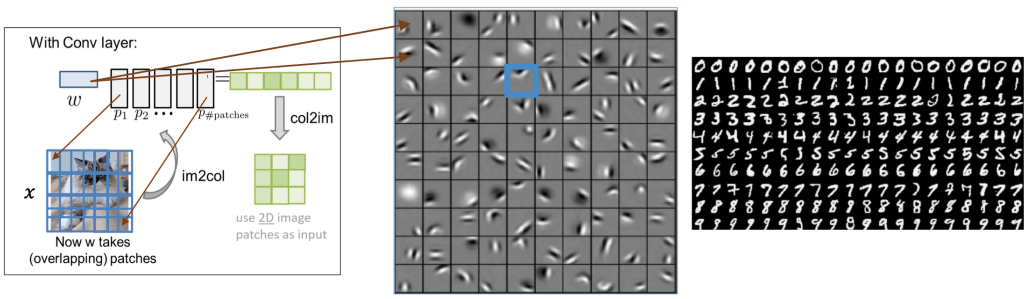
Convolutional neural nets have learnable filters.
In particular for functions u, h defined on $\mathbb{Z}$, where h is the kernel, the $1-$D discrete convolution follows the formula:

$$o(t) = (u * h)[t] = \sum_{\tau=-\infty}^{\infty} u[\tau]h[t-\tau] \tag{6.1}$$

In essence we slide the filter along the input. The formula is a kind of weighted average. When discussing momentum we care about smoothing and want a compact representation of that smoothing in terms of operations in memory. In the context of CNNs we want the filter to be learnable. Hence, they are called convolutional neural networks, because they have learnable convolutional filters in them.

6.2.2 Local Structure & Respecting Locality

We assume locality exists in CNN. More specifically, we want to respect locality in sub-functions that are learned. In the learned functions, in the context of images, the pixels that are near each other are important to figure out the relationship of the learned function. This idea is expressed inherently through the convolution structure with "small" filters. In essence, we want to learn local feature filters and combine them. We use a visual from [1] to capture this idea:

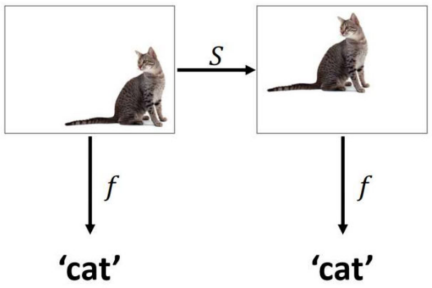


6.2.3 Invariance and Equivariances

In the case of images, the local structure is a 2-D image. Moreover, there exists the notion that things in the real world remain interesting even if you move them around. Suppose one wanted to answer the question "Is there a cat in this image or not?", when answering this question the position of the cat does not matter. Whether the cat is shifted around in the image should not change the answer to the question: "Is there a cat in this image?" This idea can be simply explained as the learned function should not be affected by the translation of objects within the dataset. For example, in the case of a classifier, it should give the same prediction even if the object has moved within the input image. This idea is manifested by weight sharing and data augmentation during the training process.

6.2.3.1 Weight Sharing and Invariances

We know that in CNNs the filter scans across the input and produces a feature map. As a result, every neighborhood of pixels in the image is processed by the same kernel/filter. Meaning, the filters at different positions in the image share the same weights. We don't assign weight to each individual pixel.
If we have different blocks with different weights for every patch of the image, that doesn't capture the idea of invariance. If we use the same patch to cover the whole image by shifting the patch by a pixel at a time, it is much more tractable to lift it to the 64 channels. Therefore it is a much more reasonable number of parameters. This (translational invariance) can be visually depicted using a graphic from [1]:



6.2.3.2 Data Augmentations and Invariances

Altering your data, by introducing noise, has a regularizing effect. The philosophy behind doing this, is ensuring that the outputs are less dependent on small neighborhoods of adding noise. Data Augmentations are not particular to convolutional networks. One classic example of data augmentations in Computer Vision is augmenting a dataset of images by introducing rotated copies of already present images. This augmentation can lead a network to becoming approximately rotationally invariant [2].

6.2.4 Support Hierarchial Structure

As we zoom out, in an image, different structures may exist at coarser levels. This idea can be explained as: "The patterns that we are trying to learn are visible not locally, but rather when we see the whole image." For example, the edges in an image can be identified at the local level, but the segments of the objects we are considering are only visible at another level, and the object itself is only visible at the highest level. This notion manifests as the combination of: depth of the network, downsampling with depth (stride and pooling), and lifting from a local pixel space to a more abstract level by increasing the number of channels with network/layer depth. We can equivalently explain support hierarchical structure as: by increasing the number of layers we increase the receptive field linearly [4]. Additionally noting that, in earlier layers we see edges or very narrow segments of images, and as we increase in depth we see more of the image [3]. Downsampling, in this context, refers to reducing the spatial dimensions of an image based on a mathematical operation such as pooling or strided convolution. In essence as we proceed in a Convolutional Network we look at smaller and smaller image like objects.
In convolutional neural networks the notion of downsampling shows up in two ways:
1. Stride
2. Pooling
As we proceed from fine to coarse, a CNN may receive as input a grey-scale image, however as we increasingly process the input we get an increasing number of channels, meaning we add more information. Essentially, we are lifting. Therefore, we understand that lifting is synonymous with the notion of increasing the number of channels as we go deeper in the network. Applying lifting provides us with expressive flexibility. We also note that optimization can get quite constrained (local minima, getting stuck) if we do not lift sufficiently. Suppose our weights are randomly initialized, we know that what we wish to learn may be diverse. Given that we start with a random initialization, we will only explore local neighborhoods. However by lifting, we increase the chance of getting lucky (lottery-ticket hypothesis) [1].

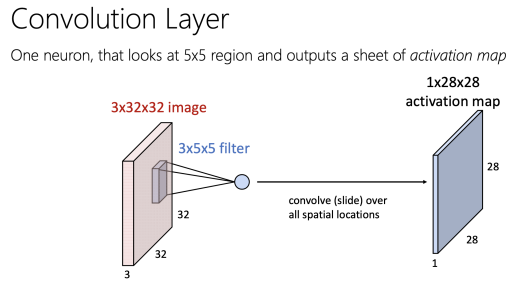
6.3 ConvNet Basics

An image is a set of real numbers arranged in a grid. There is a height and width. In the event that one is working with color images, we additionally have channel information / depth associated with each pixel (RGB). We now split the explanation from lecture [46:06, 1:18:00] into five separate subsections and provide the corresponding context, and image for each part.

[1]The lottery ticket hypothesis posits that, "dense, randomly-initialized, feed-forward networks contain sub-networks ("winning tickets") that - when trained in isolation - reach test accuracy comparable to the original network in a similar number of iterations" [5].

6.3.0.1 Convolutional Layer

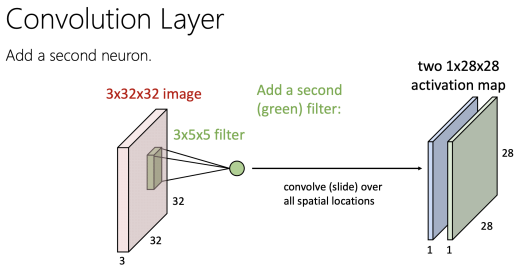
In ConvNets we generally have as input an image. The operation we are performing is a convolution. We do this via a Convolutional Layer. We use a graphic from [1] depicted below to illustrate:



One may naturally wonder: What is the most important thing about a Conv-Layer? The answer to this question is the number of filters I have.

6.3.0.2 Values per Pixel

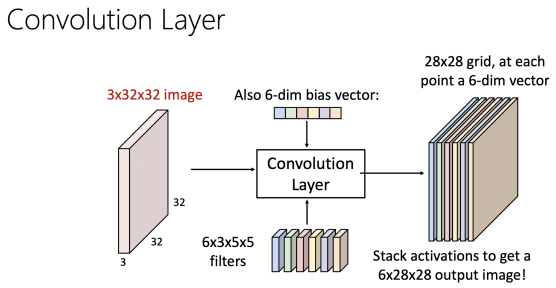
The output of a Convolutional Layer will have a number of values associated with every pixel. For example, a Conv-Layer with 2 filters will have 2 numbers associated with every pixel. In essence the number of filters determines the number of channels at the output of a convolutional layer. We use a graphic from [1] depicted below to illustrate:



One may naturally ask: What is the size of the filter? In general we use a square filter with dimension $k \times k$. For the sake of illustration, that our input is a color image and that $k = 3$. Therefore for each channel of the input we have 9 real numbers associated in the output, (3×3 for each of the channels R, G, and B).

6.3.0.3 Bias and Activation

We then take the outputs from the convolutional layer, add one bias term per output channel and can pipe them through an activation to get the output, which we use as input to the next layer in the network. We use a graphic from [1] depicted below to illustrate:



6.3.0.4 Technicalities

There are technicalities however. The image fed into the network is of fixed dimension. Therefore it has an edge/boundary. When we are sliding the filter across the image we may encounter the edge. The question of what to do when encountering the boundary arises naturally. In practice we have three choices:

(1) Ignore / avoid crossing the boundary (no padding)

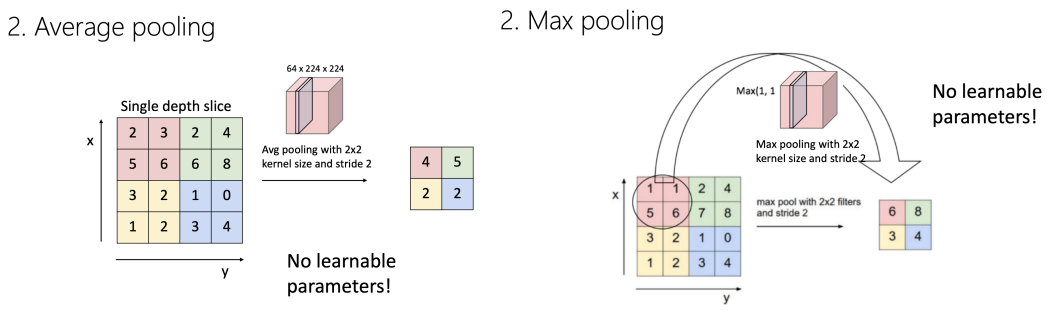
(2) Padding: pretend that in the places where nothing exists, we have zeros instead. In essence add a layer of zeros around the image (zero padding).

(3) Extend the image s.t. it's a torus (circular/mirror padding)

One additional technicality occurs when we do not want to "mix" channels in convolutions. In this scenario we use a depth-wise convolution which is essentially convolution applied independently over each channel of the input tensor.

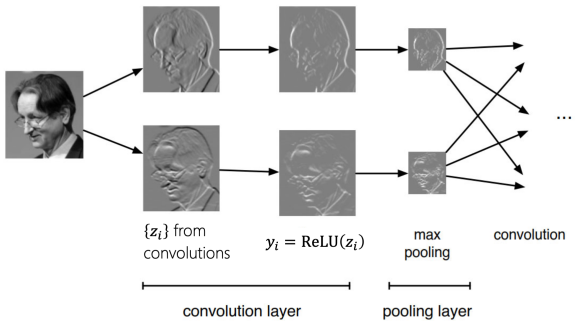
6.3.1 Striding, and Pooling

Given the technicalities presented in the previous section one may wonder: By how much should we be striding? Inherently, we are ticking time in the filter by 1. In principle we can increase the stride to shift the filter by a larger amount. Such a decision would result in downsampling.
When discussing the topic of downsampling, we should also discuss pooling. Pooling layers have no learnable parameters by default. It is defined by the idea that you take in a segment of the input and output a single value. Pooling however, wishes to achieve the same thing as striding: to make the input smaller. Average Pooling: take average of numbers in segment. Max Pooling: take the max of numbers in segment. We use graphics from [1] depicted below to illustrate:



6.3.1.1 The CNN:

We combine all the previous ideas from this section to get the CNN. We use a graphic from [1] depicted below to illustrate:



6.3.1.2 Standardization

You may want standardization throughout the network. Suppose we normalize the input so it has zero mean and variance one. Even if we choose to standardize in this way initially, it may not be the case that "inputs" to later layers in the network are standardized. This is discussed more in the next lecture.

6.3.1.3 Batch Normalization

In some sense Batch Normalization is equivalent to "performing standard ML style normalization except just use the SGD batch size to do it." In essence for all values of x over a batch $B := \{x_1, \ldots, x_n\}$, we compute the mean μ_B, standard deviation σ_B^2 and standardize x via scale and shift. More concisely one can say that we normalize over all entries in a batch. Even though it has no learnable parameters Batch Normalization in practice still affects the gradients which pass through it. This is discussed more in the next lecture.

6.4 What we wish this lecture also had to make things clearer?

We wish this lecture included a formal definition of a receptive field, walked through the manual computation of a convolution applied to input, and discussed the notions of invariance and equivariance more formally. One can see the first reference to address some of these topics.

References

[1] Jennifer Listgarten, Jitendra Malik, (2022) *EECS 189 Lecture 17, CNNs & Residual Networks*, University of California, Berkeley. Note: `https://www.eecs189.org`

[2] Quiroga, F., Ronchetti, F., Lanzarini, L., Bariviera, A.F. (2020). *Revisiting Data Augmentation for Rotational Invariance in Convolutional Neural Networks.* In: Ferrer-Comalat, J., Linares-Mustarós, S., Merigó, J., Kacprzyk, J. (eds) Modelling and Simulation in Management Sciences. MS-18 2018. Advances in Intelligent Systems and Computing, vol 894. Springer, Cham. `https://doi.org/10.1007/978-3-030-15413-4_10`

[3] Ranzato, M., *Image Classification with Deep Learning*. CVPR 2014. `https://www.cs.toronto.edu/~ranzato/files/ranzato_CNN_stanford2015.pdf`

[4] Luo, W., Li, Y., Urtasun, R., & Zemel, R. (2016). *Understanding the effective receptive field in deep convolutional neural networks.* In Advances in Neural Information Processing Systems (pp. 4898-4906).

[5] Frankle, J., Carbin, M. (2019) *The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks*. International Conference on Learning Representations. `https://arxiv.org/pdf/1803.03635.pdf`