Deep Learning

MInDS @ Mines

Convolutional neural networks (CNNs) are capable of achieving very impressive results in supervised learning when applied to images as well as other applications. In this lecture we go over how CNNs work by discussing how convolutions work in general as well as with an application to computer vision, then discussing how we can train a CNN to learn its weights and perform effectively.

Convolutions

With regards to a convolutional neural network, a convolution on a matrix X in $n \times m$ dimensions, by another matrix K in $d \times e$ dimensions, is the result of applying the matrix K to X's elements, to produce a new matrix by calculating the dot product of K over patches of X. We usually apply a filter matrix K to our data matrix X in a convolution. The filter is also often a square matrix that is significantly smaller than the data matrix. We apply the dot product of K and a similarly sized patch in X to determine the resulting value in a new matrix at the coordinate of the central point of the path in X. We call the matrix K the filter because it acts like one; we pass it along the larger matrix X and apply it to the values then save them. The main factors involved in determining the result of the convolution are the following:

- The filter dimensions.
- The filter values.
- The stride between patches.
- The padding of the input.

The filter dimensions are usually square and we use a small filter size in comparison to the original data. The stride is the number of spaces along the input matrix that we move between applying the filter and the next iteration. Padding is the number of values to pad the input matrix along its edges. We usually pad the matrix with either the same values as the edge values or we use zeros. The main reason to pad the input matrix is to ensure the resulting matrix is of the same dimensions. All of these values are hyperparameters and they result in the output matrix, Y, having the following dimensions,

We generally use filters that are 3×3 but 5×5 and 7×7 are also common.

A good example showing how to process a convolution is shown at section 2.3 of https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

$$\mathsf{Y}_w = \frac{\mathsf{X}_w - \mathsf{K}_w + 2p}{s} + 1, \tag{1}$$

$$\mathsf{Y}_h = \frac{\mathsf{X}_h - \mathsf{K}_h + 2p}{s} + 1, \tag{2}$$

where the subscripts of a matrix A_w and A_h denote its width and height dimensions, p denotes the padding dimension and s denotes the stride dimension. With a 3×3 filter, we can see that we get output dimensions similar to the input X if we set both padding and stride equal to 1.

Image Filters

Images are a great area to explain convolutions and there are many filters that we can apply to images. In the last lecture on computer vision we discussed one application in computer vision of edge detection. We can apply some filters to an image's intensity matrix to determine the edges in the image. We can also apply various image effects using these filters. Some example filters follow:

$$\mathsf{K}_{\text{Identity}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \mathsf{K}_{\text{Gaussian Blur}} = \begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}, \tag{3}$$

$$\mathsf{K}_{\text{Box Blur}} = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}, \mathsf{K}_{\text{Sharpen}} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \tag{4}$$

$$\mathsf{K}_{\text{Horizontal Lines}} = \begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}, \mathsf{K}_{\text{Vertical Lines}} = \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}, \tag{5}$$

$$\mathsf{K}_{45^\circ \text{ lines}} = \begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix}, \mathsf{K}_{\text{Outline}} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \tag{6}$$

$$\mathsf{K}_{\text{Horizontal Sobel}} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \mathsf{K}_{\text{Vertical Sobel}} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}. \tag{7}$$

- The identity filter results in the output being equivalent to the input.
- The Gaussian blur filter applies a blurring effect to the values of the image based on a gaussian distribution.
- The box blur filter applies a blurring effect to the values equally across all points in the patch.
- The sharpen filter applies a sharpening effect whereby the values of the center value are emphasized with respect to its neighbors.
- The lines filters detect lines in an an image matrix at a particular angle. We can see that based on a 3×3 filter we can detect horizontal, vertical and 45° angled lines.
- The outline filter detects the outline of any objects in the image by overweighting the central value minus the sum of all its neighbors.
- The Sobel filters detect the gradient of the image along the respective axes.

A good demo of some of these filters is available at http://setosa.io/ev/image-kernels/.

We previously discussed calculating the gradient of an image by applying the following convolutions,

$$\mathsf{K}_{x \text{ gradient}} = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}, \mathsf{K}_{y \text{ gradient}} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}, \tag{8}$$

where the idea is that we subtract the value one step backwards from the value one step forwards in the appropriate direction. We can calculate the Sobel matrixes as the 3×3 versions of the above kernels weighted with the following transformation,

$$\mathsf{K}_{\text{Horizontal Sobel}} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \mathsf{K}_{x \text{ gradient}}, \mathsf{K}_{\text{Vertical Sobel}} = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \times \mathsf{K}_{y \text{ gradient}} \tag{9}$$

Pooling

Another important convolutional filter that is often used is called pooling. Pooling in combination with manipulating the stride hyperparameter can

be used as a somewhat effective dimensionality reduction method in some applications[1]. The idea of pooling is to group the values in a patch into a single representative value. There are two main approaches to pooling values; average pooling and max pooling. Average pooling produces the average of all the values in the patch. Max pooling produces the maximum value of all the values in the patch.

An example of average and max pooling of a matrix $\mathbf{X}$ with a filter size of 2×2, no padding and a stride of 2, results in,

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 5 & 7 \\ 8 & 9 & 6 & 2 \\ 1 & 1 & 4 & 7 \\ 2 & 0 & 3 & 2 \end{bmatrix}, \tag{10}$$

$$\mathbf{P}_{\text{average}} = \begin{bmatrix} 5 & 5 \\ 1 & 4 \end{bmatrix}, \mathbf{P}_{\text{max}} = \begin{bmatrix} 9 & 7 \\ 2 & 7 \end{bmatrix}. \tag{11}$$

[1] Both average and max pooling are considered somewhat "hacky" ways of approximating the resulting matrix. Both methods contribute little in terms of theory but can produce very effective results in practice.

Convolutional Neural Networks

A convolutional neural network (CNN) is a type of feed forward neural network (FFNN) that utilizes convolutional layers. A typical CNN is used for supervised learning tasks such as classification and contains two parts. First, the network possesses a feature learning component that focuses on learning an effective feature representation for the data. Following the feature learning, the network applies a typical multi-layer fully connected FFNN. The combination of the two portions results in superior results when compared to traditional methods.

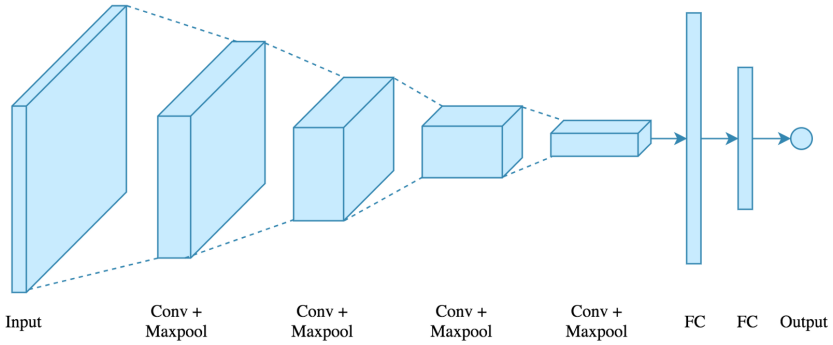

Figure 1: A visualization of a convolutional neural network CNN with the feature learning component of convolutional layers followed by the supervised learning component of fully connected layers.

CNN Feature Learning

The feature learning component of the CNN is made up of convolutional and pooling layers that result in an output representation in reduced dimensions.

A convolutional layer is a neural network layer with a selected number of filters and convolution parameters (filter size, stride, and padding). The output of the convolutional layer has dimensions as computed by the convolution parameters on the input and with an extra dimension determined by the number of filters in the layer. The filter values are the learned parameters (weights) of the convolutional layer. Each filter we add in the layer, will learn a set of weights that when applied to all the patches of the previous input, results in a useful value for predicting the target. If we represent the input as having a width w, height h and depth 1, after going through the convolutional layer, the output will have dimensions $w \times h \times f$ where f is the number of filters in the layer. One key value in the convolutional layer's filters is that the weights are shared across the image. This allows an image to learn the existence of an object in the image and be able to predict it regardless of the location of the object in another image.

The next portion of the feature learning component is the pooling layer. The pooling layer applies a pooling kernel such as max pooling or average pooling to the provided input. The result will decrease in dimensionality with respect to the original

CNNs are robust to translation because of how the weights are shared across the filter for the entire image. If the training set of images contains an image with a cat in the bottom right corner of the image, the model can learn the pattern a cat exhibits in an image and will detect a cat in an image from the test set even if the cat is in a completely different location there.

Training

Training the model is similar to a simple feed forward neural network. We apply the forward pass to predict a value and compare that to our target result, measure a loss and then apply backpropogation with the chain rule to update each of the previous weights. Using backpropogation we can update the filter weights.

Overfitting

As always, overfitting is a key focus when training any machine learning model and CNNs are no exception. As we did with simple feed forward neural networks, we can add a dropout rate to remove some of the connections between layers at random. This will force the model to generalize about the data.

Another overfitting mitigation approach that is CNN-specific is to use a lower number of filters in the first convolutional layers before adding more. When we have a large number of filters initially, the model will overfit to the data's patterns. If we wait to add more filters, the model will have already lowered the dimensionality to a point where the additional filters won't overfit as much.