Building a Convolutional Neural Netwrork

Maya Gonzalez
mgonzalez3@oxy.edu
Occidental College

1 Introduction

The tutorial began by building a CNN from scratch with no special packages. In the second part, we build a CNN using keras, which is a special ML package from tensorflow that takes care of all the layers. The mathematical foundations of CNNs were taught in the first section, my own takeaways will be outlined in the next section.

2 Tutorial

2.1 Overview

Compared to other neural network architecture, Convolutional Neural Networks (CNNs) have demonstrated success in image processing primarily due to the use of convolutions and translational invariance refers to the idea of recognizing objects regardless of relative positions. This is possible with a CNN due to the nature of assessing sections of pixels together by applying a convolution on each section of pixels. CNNs take advantage of the intrinsic properties of images. It is based on the idea that adjacent pixels share similar properties and can be used to inform/influence the context of the surrounding pixels.

The goal of a CNN in image processing is to reduce the image dimension in such a way that avoids losing important information but reduces the pixel amount to reduce computational power. The basic process of CNNs first starts with an input image of size nxn. Data flows from layer to layer in order, so that the output of one layer is fed into the next layer as input. Once the end of the network is reached, the error is calculated by comparing the output to the desired output. This iterative process begins with the input layer in what is referred to as forward propagation. With each output the error is calculated by comparing the output to the desired output. The derivative of the error is then calculated with respect to the given parameter. To adjust the parameter, the calculated derivative is subtracted from the parameter itself. The goal of parameter adjustment is to minimize the error via gradient descent. In order to update the parameters, it is necessary to compute the derivative with respect to the cost.

2.2 Filter

A filter, or kernel, of size fxf is utilized and a stride length, s, is determined. Odd numbered filters are the most common out of convention, since they allow us to refer to the output matrices in terms of a central pixel. In what is referred to as convolution, the kernel moves over the image the distance of the stride length. Each time, it takes the dot product of the filter and the portion of the input image that the filter is hovering over and calculates the sum. This sum becomes a value in the respective cell of the output matrix, or the feature map. Assuming a stride length of 1, the size of the output matrix is (n-f +1) since there are only a certain number of possible positions for the filter to traverse the input image. If a stride length is chosen that is greater than 1 the output size will be (n+2p-s)/s +1. To determine the next value in the output matrix, the filter moves to the right and performs the same convolution over a new portion of the input image until it traverses the entire image width. Then, it will jump to the opposite side of the next row, determined by the stride length and repeat this process until it has traversed the entire image. Assuming only one filter was used, the results from each matrix multiplication are summed up with the bias, a non-linearity is applied, and the result is a one-depth channel Convoluted Feature Output. The goal of this step is to extract high-level features such as edges from the input image. The dimensionality of the each layer is determined by the pixel size and channel depth. For example, if you are using a RGB input image of 1000x1000, there will be 3 million input features. Instead of hand picking weights for the filter, it is possible to use each weight as a parameter and have the computer 'learn' the best weights through backpropagation. It has been found that this method can have better feature detection than researcher's chosen weights.

2.3 Padding

There are a few issues with this standard method of using the input exactly as it is, the first being that every time a convolution is performed the output gets smaller, which means that in a network with many layers, the image will be very small. The next issue is that corner pixels from the input image are utilized less in the convolutions than middle pixels, which is a problem since the edges of the input image may contain important information. To fix the problems of shrinking output and the disregard for edge information, one method is to 'pad' the image before any convolutions are calculated. This adds additional pixels around the image border and therefore makes the image dimensions larger. The padding value is chosen based on the goal, the two most common choices are valid and same convolutions. Valid colvolutions mean that no padding is to be added. Same convolutions means that the output size is the same as the input size. The padding value varies depending on filter size and can be calculated by (f-1)/2.

2.4 Forward Propagation

The above mentioned describes one layer of a CNN. CNNs typically have more than one layer and must have a way to take the activations of one layer and map those to the activations of the next layer. The way this is done in CNNs is through forward propagation and backpropagation which uses a convolution operation. The forward pass involves derivation of the gradients moving from left to right. At the end of the circuit, the loss is computed by using the loss function. Relating back to the filter, the convolution between the filter and the input derives the output. Backpropagation is used to derive the error between the expected output and the network's output. It measures the influence of every intermediary value in the graph on the final loss function. After the forward path, the backward path is derived by calculating the gradient of all the intermaries in the circuit from right to left until we are left with the gradients of the input. Three derivatives need to be computed for the backward pass: dA, dW, and db. The derivative dA refers to the derivative with respect to the cost for a specific filter Wc and dZhw refers to the gradient of the cost with respect to the convolution layer's output Z at the hth row and wth column. Basically we are just adding all the gradients from all the slices. The next derivative to calculate is dW which is the derivative of one filter with respect to the loss. In this step, we are essentially adding up all the gradients of W with respect to each slice. The final derivative is db which is computed by summing all the gradients of the convolution output (Z) with respect to the cost.

Since this is a recursive function, in the base case we consider the identity function, the gradient of f with respect to f, which has a gradient of 1. To compute the gradient of the input layer, the chain rule must be applied. Each intermediate value is positively influencing the loss if it is a positive gradient, whereas it is negatively influencing the loss if it is a negative gradient. The general backpropagation procedure calculates the error gap, changes the weights, and stops once the gap is no longer updated or when the differential value becomes 0. The least squares method can be used to calculate the error value. A gradient descent function is used to calculate the weights of each layer. Training under gradient descent forces the network to learn to extract the most important features or features that will minimize the loss.

2.5 Improvements

Although a general gradient descent function can calculate the weights accurately, it is computationally expensive since the data needs to be differentiated at each update step. There are improvements that can be made to increase efficiency such as SGD and Adaptive Moment Estimation. SGD only uses a random subset of the data to make faster and more frequent updates. CNNs include convolutional and pooling layers that reduce the number of model parameters and ultimately reduce the complexity. Pooling layers takes an input image, breaks it into sections, and takes the maximum or average value of each section as the respective section's value in the output.

2.6 AlexNet Architecture

AlexNet is a deep CNN that has outperformed other top models and achieved record error rates, the model also won the ImageNet classification challenge in 2012. This was the first published architecture to utilize consecutive convolutional layers. Their CNN has an overall architecture of 8 layers which include 5 fully connected layers and 3 overlapping layers. Next, I will analyze some features of their model's architecture that are unique. The researchers identified the most important feature of AlexNet as the implementation of ReLU nonlinearity. The standard way to model the output as a function of its input is through saturating linearity. Since gradient descent is used to train the CNN, the updates to the weights at each iteration are dependent on the gradient. The tanh(x) function plateaus anywhere beyond ±1, which means that the gradient of this function is basically 0 anywhere beyond this middle range. The weights will stop changing once a neuron's output reaches this range since the gradient will be in close proximity to 0. The other option is non-saturating linearity which has the function of f(x) = max(0,x). This function only plateaus for negative values. Non-saturating linearity is computationally much faster than that of saturating linearity. Neurons with this nonlinearity are referred to as Rectified Linear Units (ReLU). Overall, networks with ReLUs train several times faster than those with tanh units. It wouldn't be possible to work with such a large network if a ReLU was not implemented. In AlexNet, ReLU was applied to the output of every convolutional and fully-connected layer. It is important to note that saturating nonlinearities may work well for other smaller models trained on smaller datasets. Normalization was applied after the ReLU in the first and second layer. AlexNet also includes overlapping pooling, which is not a traditional method but was found to further reduce the error rate. The authors found that it is more difficult to overfit in methods with overlapping implemented. Moreover, two methods were implemented to reduce overfitting: data augmentation and dropout. Data augmentation included mage translations and horizontal reflections. Although this method makes the training set larger by an exponent of 2048, it dramatically reduces overfitting to allow for the use of a large network.