Convolution Neural Networks

Gary Guzzo

October 2020
Cornwell
Math 686 - Deep Learning

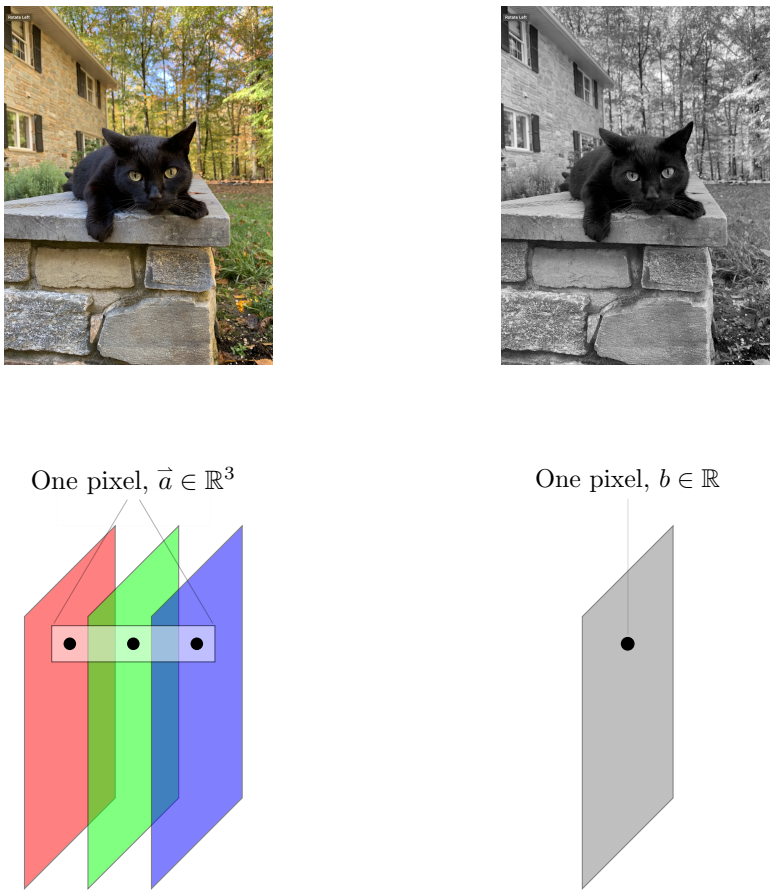


Figure 1: RGB image A and gray-scale image B (top) represented as tensors $\mathbf{A}$ and $\mathbf{B}$ (bottom)

1 Introducton to Convolutional Neural Networks

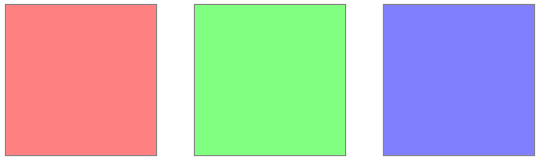
In the previous paper, *Feed-Forward Neural Networks*, we studied the behaviors of a neural network used for binary classification, a process where the network decides whether a particular data point belongs to a class, or does not. The data points were essentially just lists of various characteristics, where each characteristic was a single number in $\mathbb{R}$. If there were n characteristics, then the data points were vectors in $\mathbb{R}^n$, or *tensors of rank one*. What if our data points are no longer tensors of rank one, but tensors of rank two or three?

1.1 Images and Tensors

Figure 1 shows an RGB image A and a gray-scale image B. Image A is associated with a tensor $\mathbf{A} = [a_{i,j,k}]_{i=j=k=1}^{m_1,m_2,3}$ of rank 3, and image B is associated with a tensor $\mathbf{B} = [b_{i,j}]_{i=j=1}^{m_1,m_2}$ of rank 2. $\mathbf{A}$ has length, width, and *depth*, whereas $\mathbf{B}$ has only length and width. The depth of $\mathbf{A}$ is 3, since there are three color channels (one for each of the red, green, and blue components of the image A). This means that, for a specific pixel (α, β) of A, there is an associated *vector*

$$\vec{a}_{(\alpha,\beta)} = (a_R, a_G, a_B) = [a_{\alpha,\beta,k}]_{k=1}^{3} \in \mathbb{R}^3$$

where $a_R, a_G, a_B \in [0, 255]$ are the red, green, and blue color intensities (see Figure 1). The vector $\vec{a}_{(\alpha,\beta)}$ gives the pixel its color. In the case of a gray-scale image (rank two tensor), the vector associated with pixel (α, β) of B could be written as $\vec{b} = (b) = [b_{\alpha,\beta}] \in \mathbb{R}$, where $b \in [0, 255]$, but that notation is a bit overkill for a single number in $\mathbb{R}$.



$$\mathbf{A}_1 = \begin{pmatrix} a_{1,1,1} & a_{1,2,1} & \cdots & a_{1,m_2,1} \\ a_{2,1,1} & a_{2,2,1} & \cdots & a_{2,m_2,1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m_1,1,1} & a_{m_1,2,1} & \cdots & a_{m_1,m_2,1} \end{pmatrix}, \cdots, \mathbf{A}_3 = \begin{pmatrix} a_{1,1,3} & a_{1,2,3} & \cdots & a_{1,m_2,3} \\ a_{2,1,3} & a_{2,2,3} & \cdots & a_{2,m_2,3} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m_1,1,3} & a_{m_1,2,3} & \cdots & a_{m_1,m_2,3} \end{pmatrix}$$

Figure 2: The red, green, and blue color channels from an RGB image (rank three tensor)

In the figure above, we see each of the three color channels from an RGB image (rank three tensor). Below it, we see the color channels expressed as matrices. It is these matrices that we will perform the *discrete convolution* operation on.

1.2 Discrete Convolution

In this section, we will examine a way for computers to *see* vertical edges, which will extrapolate into different features, such as corners, circular arcs, textures, and more. The way this is done is looking at the image in *windows*, or small blocks of the image, one window at a time. At each window, a *filter* applied against it, transforming the image in various ways. It is these filters that are the *trainable weights* of the network.

1.2.1 The definition

Let $\mathbf{A} = [a_{i,j,\gamma}]$, with $\gamma \in \{1, 2, 3\}$ fixed, be one color channel from an RGB image. Each color channel is of size $m_1 \times m_2$. Let W be a *filter tensor* of size $k \times k$, where $k \leq m_1 \wedge k \leq m_2$. Then for any pixel $\mathbf{A}_{(\alpha,\beta)}$, with $0 \leq \alpha \leq m_1 - k$ and $0 \leq \beta \leq m_2 - k$, the entry in position $(\alpha + 1, \beta + 1)$ of the convolution of $\mathbf{A}$ with filter W (or simply $\mathbf{A} * W$) is:

$$\sum_{j=1}^{k} \sum_{l=1}^{k} \mathbf{A}_{(\alpha+i,\beta+j)} W_{(k+1-i,k+1-j)}$$

The inputs and output of this operation are rank two tensors. The definition extrapolates nicely into rank three tensors, or RGB images, by essentially taking the convolution on each color channel and summing the three resulting values in $\mathbb{R}$. The convolution operation on an entire image can be thought of as sliding a window across the image, and computing something similar to a dot product between the current window of the image and the filter tensor, at each window position. The outputs (of images) tend to be altered versions of the image, and they depend on the choice of filter W. This section uses (1).

1.3 Using Convolution to Detect Edges

Consider the Figure below; a simple image $\mathbf{A}$ where the left half of the pixels are black, and the right half are white. The red rectangular window A' (of pixels) is examined. It is a $1x2$ matrix, where the left entry is 0 and the right entry is 255. Next to this window is another $1x2$ matrix W, and is called a *filter*. The convolution between window A' and filter W is computed.

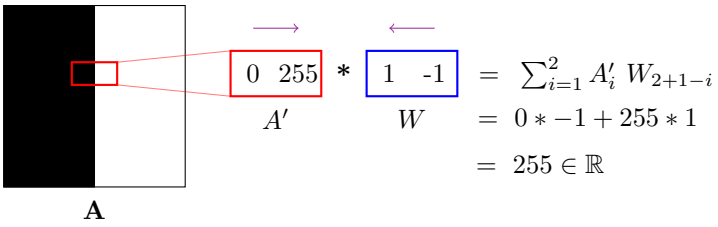


Figure 3: Simple image **A** with the computation of convolution at a specific window position (a vertical edge)

Notice that the output 255 is a single number in $\mathbb{R}$. This number is the intensity or brightness of the output from one *window position* of convolution between **A** and W. The pixel output is *white*, and an edge was detected. This computation can be made between W and any possible window A of **A**. The pixel window we chose in Figure 3 was precisely on a vertical edge. Let us compute the outputs from two other window positions, which are not on an edge.

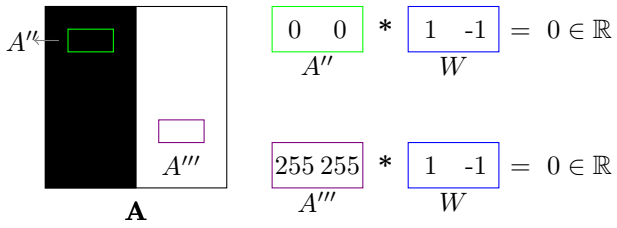


Figure 4: Convolution outputs from two more window positions on **A**

Computing convolution with filter $W = (1, -1)$ on the entire image **A** gives us the following output in figure HI. Notice that every pixel is black, except for the vertical line down the center. This vertical line is essentially the outline of a vertical edge, and occurred based on the choice of W, as its purpose was to detect vertical edges. Using the transpose W^T would instead attempt to find horizontal edges. Increasing the size of the filters allows for the detection of more complicated features.

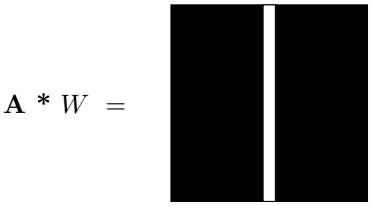


Figure 5: The result of performing convolution on simple image **A** with an edge-detecting filter

Figure 6: An image of the neighbor's cat (left), its output from convolution with Sobel edge detecting filters (right)

1.4 Weights and Network Architecture

In the previous paper, Feed-Forward Neural Networks, we saw the architecture of a basic neural network. The input was a rank one tensor, and the input layer was viewed as a column of nodes. Each node in the input layer was connected to each lode in the next layer, with a unique weight $w_{i,j}$ on the edge connecting the j^{th} node in the input layer to the i^{th} node in the next layer.

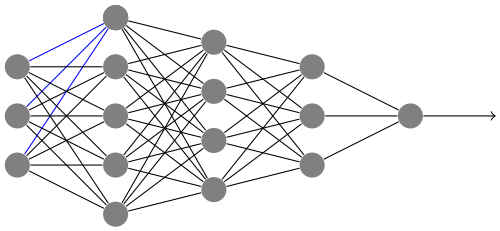


Figure 7: Visualization of a traditional feed-forward neural network

In convolutional neural networks, where the inputs are rank two or three tensors, the network cannot be viewed in this way. It is much harder to visualize the network, as layers cannot be represented as a single column of nodes. One advantage, however, is the idea of shared weights. Recall the *filter tensors* W introduced in Section 1.2. These filter $k \times k \times 3$ filter tensors (or the components of the tensors) are the new *trainable weights*.

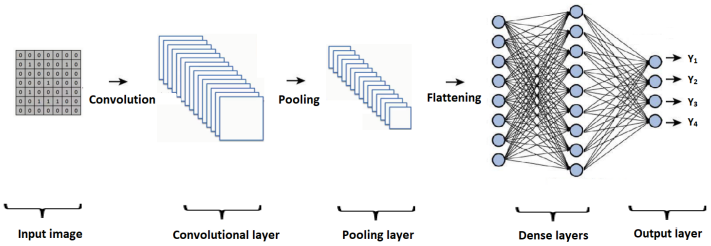


Figure 8: Visualization of convolutional neural network architecture (4)

Typically, several filter tensors W_i, say n of them, are applied to the input layer (raw images). Each of the n convolution products are computed (for each image in the training set), and are the outputs in the first hidden layer (called the Convolution Layer in Figure 8). Although the input images and the filter tensors W_i are rank three tensors, the outputs are rank two, as seen in Section 1.2.1.

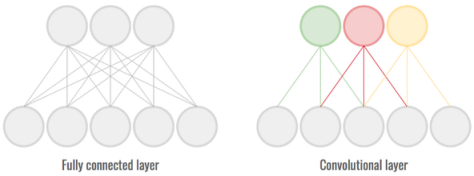


Figure 9: Parameter sharing versus fully-connected. The green, red, and yellow nodes and edges represent the outputs from three different filter tensors (3).

An important feature about convolutional neural networks is the idea of *sharing weights*. Unlike traditional feed-forward neural networks, where each connection between nodes in sequential layers has a *unique* weight attached to it, convolutional neural networks use the *same filter tensor* across the entire image, and on the entire image data set. So, for each of the n filter tensors W_i used on the input layer, there are only $3k^2 + 1$ trainable weights. Below we see a collection of these rank 3 filter tensors from (5). The tensors were learned through training, and resemble different features.

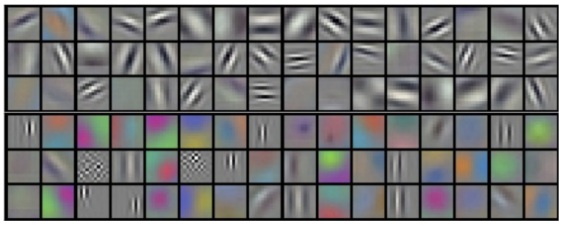


Figure 10: Filter tensors learned from (5)

2 Convolutional Neural Networks in Practice

2.1 Our data set

We will study the CADDY Underwater Gesture Dataset(2), a set of images of divers making 16 different gestures. In total, there are over 18 thousand images, where each class (gesture) is split into two groups: *dark and cloudy* and *crystal clear*.

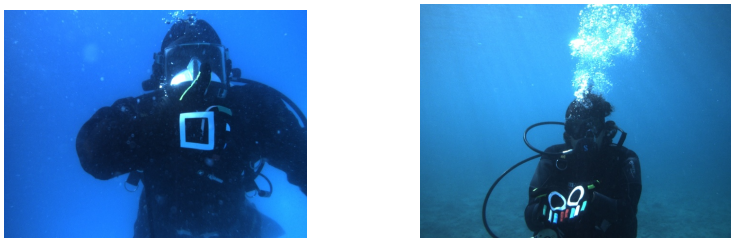


Figure 11: The "up" and "boat" classes from (2)

2.2 Memory usage

With over 18 thousand images of size $640 \times 480 \times 3$, extreme RAM usage is expected. Opening all of the the images from every class and subclass of the directory using the PIL Image.open command consistently took over 55 minutes with a GPU. Also, conversion of the image set to an n-dimensional numpy array and/or resizing of the images using Image.resize consistently drove the RAM up to crash the code. After some attempts to avoid this obstacle using batches, gc.collect() and *del* *, and even using Colab Pro, the RAM consistently surpassed the available memory. Because of this, a decision was made to only use the first eight classes of images.

2.3 Baseline accuracy

After a few Google searches, an accuracy between 97% and 98% tends to be the highest achieved on the data set (6). Three computer science graduate students achieved this accuracy using what they called a 'vanilla CNN with a ResNet-50 backbokne' (6). This means that there are 50 hidden layers, and only convolution, max-pool, flatten, and dense layers are used. In this (6) study, mild data augmentaion was used, and the Adam optimizer was chosen.

2.4 Speculations about the data

In the figure below 12, we see that the distribution of images is certainly not uniform. In fact, there are more than 1200 images in the "start" class, and less than 100 in the "five" class.

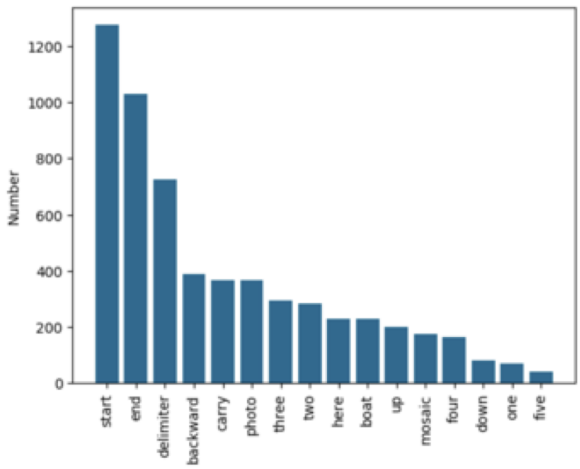


Figure 12: Histogram from (6)

This likely is not much of an issue for the following reasons: The "start" and "end" gestures look rather similar (Figure 13). Both gestures display a backward facing hand (shown as a full square on the glove), and either one or two fingers held up. It makes sense to have a lot of practice on these classes.

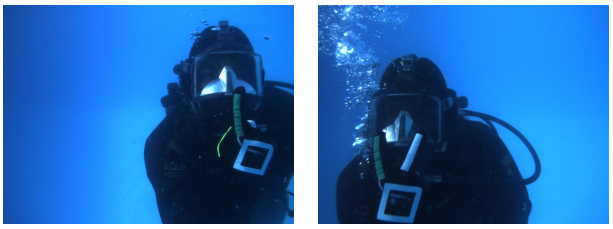


Figure 13: Images from "start communication" and "end communication"

On the other hand, the "five" gesture, which has less than 100 images, is very distinctive. It likely does not require much to learn filters which recognize this gesture.

2.5 Speculations about the network

Figure 14 shows an image from the class "two" (left) and the result of convolution using an edge-detection filter (middle). On the right, a *suppression* is performed, which has the purpose of reducing edges to be only one pixel wide. This was done using numpy.

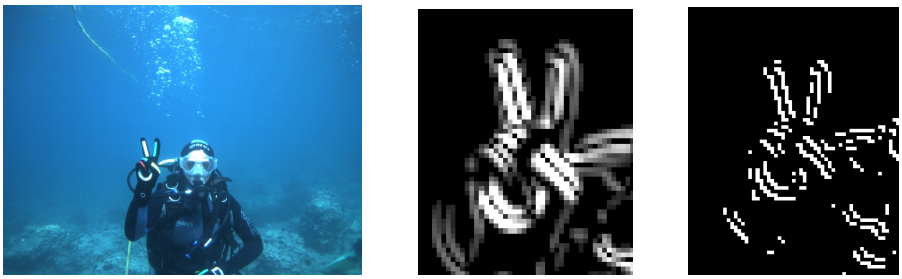


Figure 14: Two images next to their output of convolution with edge-detecting filters. Bottom is from BIBLIOGRAPHY

We can also recognize a few dominating features, such as a rather straight line along the diver's index and middle fingers, and two circular arcs on the palm. In practice, it is likely that separate trained filter tensors would recognize these features.

2.6 Architectural choices

As we know from (6), a 50-layer deep network was used, and high accuracy was achieved. A similar architecture was attempted, but RAM was a substantial obstacle, and led to reduction of the model's complexity to prevent crashing. One observation is that it is important to have filters large enough to capture the gesture, with little discrepancies. In other words, we want filter window sizes that are able to fit over a considerable amount of the diver's glove.

The choices made for this network are as follows: five convolutional layers, with number of filters 64, 64, 64, 128, 256. Almost every layer had a weight regularizer applied to it, as well as same-padding and a relu activation function. The activation function on the output layer was softmax, and the categorical cross-entropy loss function was used, as the number of classes is greater than two.

2.7 Results

Unfortunately, navigating the obstacles of RAM usage and crashing took up an extremely substantial amount of time, and a well-performing network was not achieved. However, a similar, but larger, network architecture achieved over 99% training accuracy after 40 epochs without data augmentation or regularizing. After that model, 41% training accuracy was not surpassed with any architecture.

References

[1] Cornwell, Christopher. *Math 686: Class 9*. Towson University. 2020, October 26. Towson, Maryland.

[2] Chiarella, David. Gomaz Chavez, Arturo. *CADDY Underwater Gestures Dataset*. http://www.caddian.eu//CADDY-Underwater-Gestures-Dataset.html

[3] Despoise, Julien. *Convolutional Neural Networks*. Quora. 2018, May 22. https://www.quora.com/What-are-the-advantages-of-a-convolutional-neural-network-CNN-compared-to-a-simple-neural-network-from-the-theoretical-and-practical-perspective

[4] Maeta-Gutierrez, Valeria et al. *Comparison of Convolutional Neural Network Architectures for Classification of Tomato Plant Diseases.* 2020, February 12. https://www.mdpi.com/2076-3417/10/4/1245/html

[5] Li, Fei-Fei et al. Convolutional Neural Networks for Visual Recognition. Stanford University. Spring 2020. https://cs231n.github.io/convolutional-networks/

[6] Martija, Mygel et al. *Underwater Gesture Recognition Using Classical Computer Vision and Deep Learning Techniques.* Journal of Image and Graphics. 2020, March. http://www.joig.org/uploadfile/2020/0318/20200318050938621.pdf