Simple Convolutional Neural Network From Scratch

Chris Yogodzinski

Notebook Overview

This notebook is a follow up to src/python/scratch_two_layer.ipynb.
The previous notebook I made a simple neural network from scratch to predict handwritten digits in the mnist dataset. The neural network consisted of one fully connected hidden layer and an output layer.

In this notebook I will be adding a convolutional layer and pooling layer to this simple neural network. The convolutions in this neural network function to key features from the dataset prior to the predictive fully connected layer.

Variables

I matrix: a mnist image of dim $= (m$ by $n)$
K matrix: kernel of dim $= (m_k$ by $n_k)$
s integer: stride, a constant set by user
ϕ_a function: activation function of convolution
F matrix: a feature map of dim $= (m - m_k + 1$ by $n - n_k + 1)$ or $(i$ by $j)$
ϕ_p function: pooling function applied to feature map
P matrix: pooled feature map, condensed image of dim $= (\frac{m-m_k}{s}$ by $\frac{n-n_k}{s})$

Convolutional Layer

The convolutional layer in a cnn is named after a method known as kernel convolution.
Kernel convolution works by passing a filter or kernal iteratively over the input matrix and transform the values of the input based on the values of the kernel.
In this notebook our kernal transformation will be $\sum_i \sum_j I[m-i, n-j] \times K$.
In words this is element-wise multiplication of the kernel-sized subset of I and K followed by the sum of the product.

```
setwd("~/Projects/nn_playground")

read_mnist <- function(label_path, im_path) {
    # read in labels
    f <- file(label_path, "rb")
    meta <- readBin(f, n = 2, "integer", endian = "big")
    labels <- as.integer(readBin(f, n = meta[2], "raw", endian = "big"))
    close(f)

    # read in imgs
    f <- file(im_path, "rb")
    meta <- readBin(f, n = 4, "integer", endian = "big")
    byte_count <- meta[2] * meta[3] * meta[4]
```

```r
    imgs <- readBin(f, n = byte_count, "raw", endian = "big")
    close(f)
    imgs <- array(as.integer(imgs), dim = c(meta[3], meta[4], meta[2]))
    dat <- list(labels = labels, imgs = imgs)
    return(dat)
}

dat_path = "data/mnist"
train_dat <- read_mnist(
    paste(dat_path, "train-labels.idx1-ubyte", sep = "/"),
    paste(dat_path, "train-images.idx3-ubyte", sep = "/")
    )

test_dat <- read_mnist(
    paste(dat_path, "t10k-labels.idx1-ubyte", sep = "/"),
    paste(dat_path, "t10k-images.idx3-ubyte", sep = "/")
    )
```

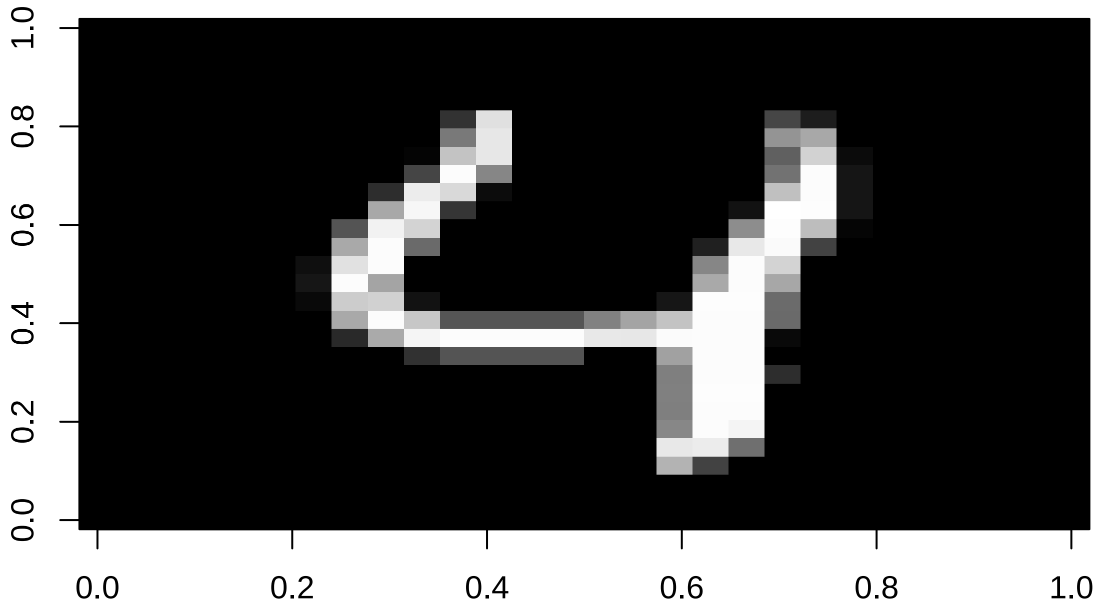
Flipping The Images

The function above reads the images in upside down for some reason.
In the code chunk below I flip the images right side up and visualize them.

```r
flip_all_imgs <- function(array) {

    flip_image <- function(img) {
        flipped <- img[, ncol(img):1]
        return(flipped)
    }
    dims <- dim(array)
    flipped <- array(apply(array, 3, flip_image), dim = dims)
    return(flipped)
}

train_dat[["imgs"]] <- flip_all_imgs(train_dat[["imgs"]])
test_dat[["imgs"]] <- flip_all_imgs(test_dat[["imgs"]])


im_num <- 5

# test image
img <- test_dat[["imgs"]][ , , im_num]
cat(test_dat[["labels"]][im_num])
```

```
## 4
```

```r
image(img, col = gray((0:255) / 255))
```




```
# train image
img <- train_dat[["imgs"]][ , , im_num]
cat(train_dat[["labels"]][im_num])
```

```
## 9
```

```
image(img, col = gray((0:255) / 255))
```