Handwriting Recognition using Neural Networks and Image Processing

Akash Rana, Dhruv Reshamwala, Kishore Saldanha, Riyansh Karani

5th November 2015

1 Image Processing

1.1 Convolution

1.1.1 Introduction

In mathematics and, in particular, functional analysis, convolution is a mathematical operation on two functions, producing a third function that is typically viewed as a modified version of one of the original functions, giving the area overlap between the two functions as a function of the amount that one of the original functions is translated. For continuous functions $f(x, y)$ and $g(x, y)$, the convolution operation is defined as following :

$$(f * g)(x, y) = \int \int f(x - u, y - v)g(u, v)dudv$$

In the case of discrete signals, the integral takes the form of summation. Hence, in the discrete domain, convolution of two discrete signals $f(x, y)$ and $g(x, y)$ is given as :

$$(f * g)(x, y) = \sum_u \sum_v f(x - u, y - v)g(u, v)$$

Convolution is widely used in Image Processing, as an operation of filtering in the spatial domain. In our case, convolution operations will be performed between a 2D Grayscale image, and a convolution kernel.

1.1.2 Steps involved in 2D convolution of an Image with a convolution kernel

1. Generate a convolution kernel, depending on the application for which convolution is used.

2. Rotate the mask by 180° so that the result of convolution does not have to be rotated by 180°.

3. Apply padding to the image, so that the entire convolution kernel fits inside the image. The number of padding pixels outside the image should be $(size\ of\ convolution\ kernel\ - 1)$.

4. For every pixel in the image, apply element wise matrix multiplication of the pixel neighborhood of the appropriate size (depending on the size of convolution kernel) and the convolution. Sum all the values of the resultant matrix, and assign the resultant value to the pixel in consideration.

The following is the result of convolution of an image with the $3*3$ $mean\ filter$ $\frac{1}{9}\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix}$

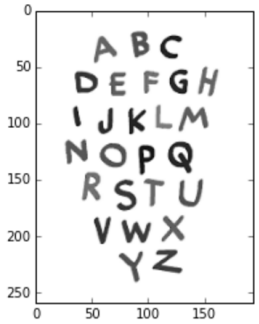

Figure 1: Before convolution

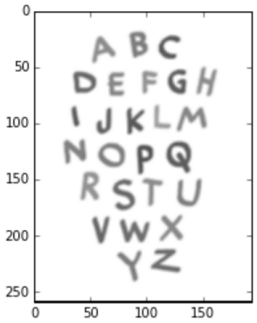

Figure 2: After convolution

1.2 Edge Detection

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. The major importance of edge detection in our project is for the purpose of segmentation of the image into smaller images, for the purpose of object detection.

There are various methods through which edges can be detected. We have tried various approaches, which have been explained below.

1.2.1 Vertical and Horizontal Edge Detection

In this process, two kernels are used to find edges in the vertical and horizontal directions. The corresponding kernels are

$$k_{vertical} = \begin{pmatrix}1 & 0 & -1\\ 1 & 0 & -1\\ 1 & 0 & -1\end{pmatrix} and \quad k_{horizontal} = \begin{pmatrix}1 & 1 & 1\\ 0 & 0 & 0\\ -1 & -1 & -1\end{pmatrix}$$

By convolving the image with the above two kernels separately, we obtain the horizontal and vertical edges as detected in the image.

The following are the results of applying horizontal and vertical edge detection on an image.

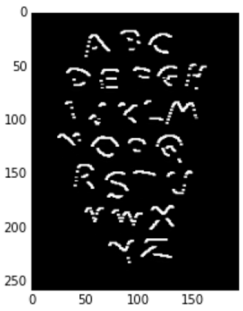

Figure 3: After vertical edge detection

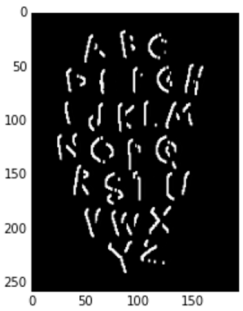

Figure 4: After horizontal edge detection

After this process, the pixel values in the two images can be added to obtain an image that has both horizontal and vertical edges.

Figure 5: Horizontal plus vertical edge detection

The downside of this approach is that using this method, only one side of the edge can be detected (i.e either black to white or white to black). This problem can be solved using the Full Convolution approach.

1.2.2 Full Convolution

This approach was used to overcome the problems of vertical and horizontal convolution approach. In this approach we calculate four separate convolutions, using four different convolution kernels.

The first pair of convolutions are done using the convolution kernels that were used in horizontal and vertical convolutions.

$$k_{vertical} = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} and \quad k_{horizontal} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$$

After the convolution of the image with these two kernels, we convolve the image with two different kernels, shown below :

$$k_{vertical} = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} and \quad k_{horizontal} = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

After all the four convolutions are completed, the pixel values of the resultant images from all the four convolutions are summed and the resultant image looks like as shown below :

ASDGASKDLJGALSDKJFLAKSJFLKASDJF
ASDKLJGLAS;DJGL;ASKJDGL;ASDJGL;
KAJSDGLKASJDGLKASDJG;LKADSJG;LA
ASKLDJGLAKSDJGLAKSDJGL;KADJGLKA

As seen in the resultant image, unlike in vertical and horizontal edge detection, all the edges are detected using this method.

1.2.3 Sobel-Feldman operator

Technically, the Sobel operator is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel-Feldman operator is either the corresponding gradient vector or the norm of this vector. The Sobel-Feldman operator is based on convolving the image with a small, separable, and integer valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations.

The operator uses two 33 kernels which are convolved with the original image to calculate approximations of the derivatives - one for horizontal changes, and one for vertical. If we define A as the source image, and g_x and g_y are two images which at each point contain the horizontal and vertical derivative approximations, the computations are as follows:

$$g_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} * A \qquad and \qquad g_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} * A$$

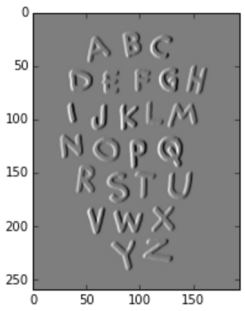

Figure 6: After convolution with sobel X kernel

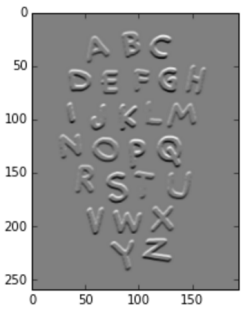

Figure 7: After convolution with sobel Y kernel

At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude and the gradient direction, using:

$$G = \sqrt{g_x^2 + g_y^2} \qquad \text{and} \qquad \theta = tan^{-1}(\frac{g_y}{g_x})$$

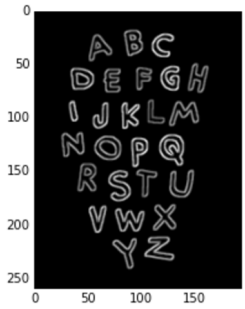

Figure 8: Gradient Magnitude

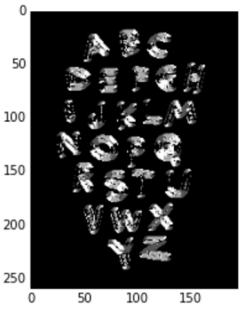

Figure 9: Gradient Direction

1.2.4 Laplacian of Gaussian

The Laplacian of Gaussian (LoG) filter is another filter, commonly used to detect edges in an image.

Gaussian Filter A Gaussian filter is a filter whose impulse response is a Gaussian function (or an approximation to it). Gaussian filters have the properties of having no overshoot to a step function input while minimizing the rise and fall time. This behavior is closely connected to the fact that the Gaussian filter has the minimum possible group delay. It is considered the ideal time domain filter, just as the sinc is the ideal frequency domain filter.
$G(x) = \frac{1}{2\pi\sigma} e^{\frac{-x^2}{2\sigma^2}}$

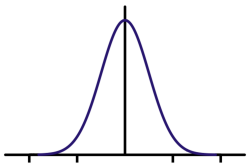

Figure 10: Impulse Response of Gaussian Filter

A two variable gaussian filter ($G(x, y)$) can similarly be represented as :

$G(x, y) = \frac{1}{2\pi\sigma} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$

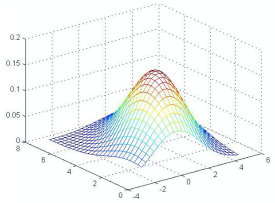

Figure 11: Gaussian Function of
two variables

Second Derivative of Gaussian The second derivative of any function is zero, when the function value reaches a maximum or a minimum. Hence, the second derivative of Gaussian filter will provide us with two Zero crossings on either sides of the gaussian peak in the case of Gaussian filter with two variables,

and a trough in the case of Gaussian filter with two variables. The second derivative of gaussian is also called the Laplacian of Gaussian.

$LoG(x, y) = [\frac{x^2+y^2-2\sigma^2}{\sigma^4}]e^{\frac{-(x^2+y^2)}{2\sigma^2}}$

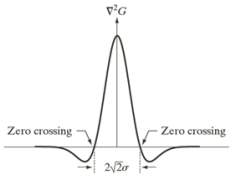

Figure 12: Single variable LoG

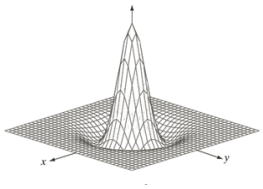

Figure 13: Two variable LoG

The following is the result of applying LoG filter to an image and thresholding the LoG image to show edges in the image :

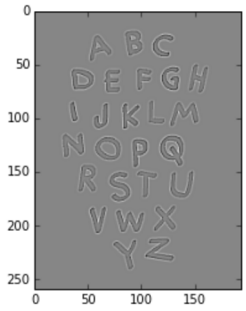

Figure 14: convolution with LoG filter

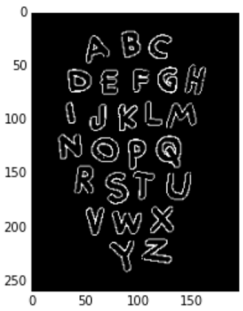

Figure 15: zero crossings in LoG filtered image

1.3 Edeg Thinning

Edge thinning is a technique used to remove the unwanted spurious points on the edges in an image. This technique is employed after the image has been filtered for noise (using median, Gaussian filter etc.), the edge operator has been applied to detect the edges and after the edges have been smoothed using an appropriate threshold value. This removes all the unwanted points and if applied carefully, results in one pixel thick edge elements.

There are various methods of applying edge thinning, two of which are shown below :

1.3.1 Non-Maximum supression

Non-Maximum suppression is applied to thin the edge. After applying gradient calculation, the edge extracted from the gradient value is still quite blurred. There should only be one accurate response to the edge. Thus non-maximum suppression can help to suppress all the gradient values to 0 except the local maximal, which indicates location with the sharpest change of intensity value. So, basically, non-maximum supression finds the local maximum in the direction perpendicular to the edge.

The steps involved in implementing the non-maximum supression algorithm are as follows :

1. Compare the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient directions.

2. If the edge strength of the current pixel is the largest compared to the other pixels in the mask with the same direction(i.e., the pixel that is pointing in the y direction, it will be compared to the pixel above and below it in the vertical axis), the value will be preserved. Otherwise, the value will be suppressed.

In some implementations, the algorithm categorizes the continuous gradient directions into a small set of discrete directions, and then moves a 3x3 filter over the output of the previous step (that is, the edge strength and gradient directions). At every pixel, it suppresses the edge strength of the center pixel (by setting its value to 0) if its magnitude is not greater than the magnitude of the two neighbors in the gradient direction. For example,

1. If the rounded gradient angle is 0° (i.e. the edge is in the northsouth direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the east and west directions.

2. If the rounded gradient angle is 90° (i.e. the edge is in the eastwest direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the north and south directions.

3. If the rounded gradient angle is 135° (i.e. the edge is in the northeast-southwest direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the north west and south east directions.

4. If the rounded gradient angle is 45° (i.e. the edge is in the north westsouth east direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the north east and south west directions.

In more accurate implementations, linear interpolation is used between the two neighbouring pixels that straddle the gradient direction. For example, if the gradient angle is between 45 and 90, interpolation between gradients at the north and north east pixels will give one interpolated value, and interpolation between the south and south west pixels will give the other (using the conventions of last paragraph). The gradient magnitude at the central pixel must be greater than both of these for it to be marked as an edge.

The following are the results, when non-maximum supression algorithm is applied to an image :

The following are the results of applying erosion operation on an image :

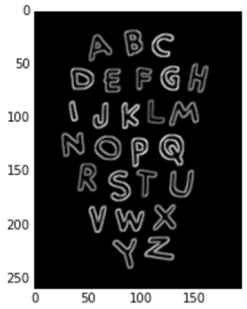

Figure 16: Before Non-Maximum supression

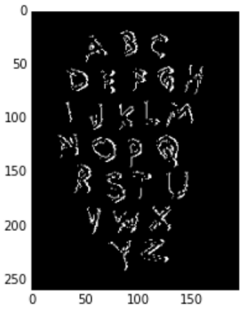

Figure 17: After Non-Maximum supression

1.3.2 Image Erosion

Erosion is one of two fundamental operations in morphological image processing from which all other morphological operations are based. It was originally defined for binary images, later being extended to grayscale images, and subsequently to complete lattices.

The basic idea in binary morphology is to probe an image with a simple, predefined shape, drawing conclusions on how this shape fits or misses the shapes in the image. This simple probe is called structuring element, and is itself a binary image.

The erosion of an image A with a structuring element B is defined as the following :

$A \Theta B = \{z|(B)_z \subseteq A\}$ or $A \Theta B = \{z|(B)_z \bigcap A^c = \phi\}$

The following are the results of applying erosion operation on an image :

Figure 18: Before Erosion

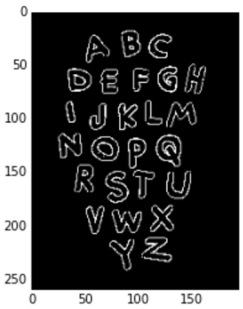

Figure 19: After erosion

1.4 Connected Component Labelling

Connected-component labeling is an algorithmic application of graph theory, where subsets of connected components are uniquely labeled based on a given heuristic. Two most popular connedted component labelling algorithms are :

One component at a time This is a fast and very simple method to implement and understand. It is based on graph traversal methods in graph theory. In short, once the first pixel of a connected component is found, all the connected pixels of that connected component are labelled before going onto the next pixel in the image. In order to do that a linked list is formed that will keep the indexes of the pixels that are connected to each other.

It is assumed that the input image is a binary image, with pixels being either background or foreground and that the connected components in the foreground pixels are desired. The algorithm steps can be written as:

1. Start from the first pixel in the image. Set curlab (short for current label) to 1.
2. If this pixel is a foreground pixel and it is not already labelled, then give it the label curlab and add it as the first element in a queue, then go to (3). If it is a background pixel, then repeat (2) for the next pixel in the image.
3. Pop out an element from the queue, and look at its neighbours (based on any type of connectivity). If a neighbour is a foreground pixel and is not already labelled, give it the curlab label and add it to the queue. Repeat (3) until there are no more elements in the queue.
4. Go to (2) for the next pixel in the image and increment curlab by 1.

Two-pass Relatively simple to implement and understand, the two-pass algorithm iterates through 2-dimensional, binary data. The algorithm makes two passes over the image: the first pass to assign temporary labels and record equivalences and the second pass to replace each temporary label by the smallest label of its equivalence class.

Connectivity checks are carried out by checking neighbor pixels' labels (neighbor elements whose labels are not assigned yet are ignored), or say, the North-East, the North, the North-West and the West of the current pixel (assuming 8-connectivity). 4-connectivity uses only North and West neighbors of the current pixel. The following conditions are checked to determine the value of the label to be assigned to the current pixel (4-connectivity is assumed)

Conditions to check:

1. Does the pixel to the left (West) have the same value as the current pixel?
   (a) Yes − We are in the same region. Assign the same label to the current pixel
   (b) No − Check next condition
2. Do both pixels to the North and West of the current pixel have the same value as the current pixel but not the same label?
   (a) Yes − We know that the North and West pixels belong to the same region and must be merged. Assign the current pixel the minimum of the North and West labels, and record their equivalence relationship
   (b) No − Check next condition
3. Does the pixel to the left (West) have a different value and the one to the North the same value as the current pixel?
   (a) Yes − Assign the label of the North pixel to the current pixel
   (b) No − Check next condition
4. Do the pixel's North and West neighbors have different pixel values than current pixel?
   (a) Yes − Create a new label id and assign it to the current pixel

The algorithm continues this way, and creates new region labels whenever necessary. Once the initial labeling and equivalence recording is completed, the second pass merely replaces each pixel label with its equivalent disjoint-set representative element.