Edge Extraction from Images

Lei Zhao

We can easily differentiate the edges and colors to identify what is in the picture. However, machines store images in the form of a matrix of numbers. The size of this matrix depends on the number of pixels we have in any given image. These numbers, or the pixel values, denote the intensity or brightness of the pixel. And the dimensions of an image are basically the number of pixels in the image. Since almost all colors can be generated from three primary colors, i.e., red, green and blue, the colored image is stored in three Matrices (or channels). Each matrix has values between $[0, 255]$ representing the intensity of the color for that pixel.

We can just using the pixel values to be the feature of one image, however, it will be easily trapped in curse of dimensionality when we are dealing with high-definition images. We need to redesign the features to represent the image with fewer numbers. When we are dealing colored images, it will be intuitive that we generate a new matrix that has the mean value of pixels from all three channels, then we can reduce the feature dimension to $1/3$ of the original size.

However, pixel values themselves are typically not discriminative enough to be useful for machines to identify the objects present in the given images. The pixel values can not distinct different objects or reflect the similarity of the same objects due to the difference in relative size, position in the image, and the contrast of the image itself. One idea to redesign the feature by shifting perspective from the pixels themselves to the edge content at each pixel. By taking edges instead of pixel values we significantly reduce the amount of information we must deal with in an image without destroying its identifying structures [1].

How can we identify edges in an image? Edge is basically where there is a sharp change in color. An image is represented in the form of numbers, we can look for pixels around which there is a drastic change in the pixel values. To identify if a pixel is an edge or not, we can simply compare the values around the pixel and believe that there is an edge in this pixel when the difference among the values is great. **How to compute this difference?**

I. KERNEL VECTORS

Intuitively, we can choose two directions to compare the difference, i.e., the horizontal and vertical directions. The difference in both directions can be quantified by a linear difference equation with given kernels, $\boldsymbol{h_x}$ and $\boldsymbol{h_y}$.

$$\boldsymbol{h_x} = \begin{bmatrix} h_x^1 & \cdots & h_x^{N_1} \end{bmatrix} \quad \boldsymbol{h_y} = \begin{bmatrix} h_y^1 \\ \vdots \\ h_y^{N_2} \end{bmatrix} \tag{1}$$

If we use matrix $\boldsymbol{M} \in R^{K \times L}$ to represent all the pixel values in a gray-scale image. The convolution results of $\boldsymbol{h_x} \in R^{1 \times N_1}$ and $\boldsymbol{x_i} \in R^{1 \times (L+N_1-1)}$ in the horizontal direction is denoted by $\boldsymbol{d_x^i}$

$$\boldsymbol{d_x^i} = \boldsymbol{x_i} \otimes \boldsymbol{h_x} = \begin{bmatrix} \sum_{k=1}^{N_1} h_x^k x_i^{1-k+1}, & \cdots, & \sum_{k=1}^{N_1} h_x^k x_i^{L-k+1}, & \cdots, & \sum_{k=1}^{N_1} h_x^k x_i^{L+N_1-k} \end{bmatrix} \tag{2}$$

where $\boldsymbol{x_i}$ could be regarded as the extended ith row vector of the image, when the index is smaller than 1 or larger than L, the pixel values are set to 0, which is known as **zero padding**.

The convolution results of $\boldsymbol{h_y} \in R^{N_2 \times 1}$ and $\boldsymbol{y_j} \in R^{(K+N_2-1) \times 1}$ in the vertical direction is denoted by $\boldsymbol{d_y^j}$

$$\boldsymbol{d_y^j} = \boldsymbol{y_j} \otimes \boldsymbol{h_y} = \begin{bmatrix} \sum_{k=1}^{N_2} h_y^k y_j^{1-k+1}, & \cdots, & \sum_{k=1}^{N_2} h_y^k y_j^{K-k+1}, & \cdots, & \sum_{k=1}^{N_2} h_y^k y_j^{K+N_2-k} \end{bmatrix}^T \tag{3}$$

where $\boldsymbol{y_j}$ could be regarded as the extended jth column vector of the image, when the index is smaller than 1 or larger than K, the pixel values are set to 0. The computing results that we apply $\boldsymbol{h_x}$ and $\boldsymbol{h_y}$ over one row or column of the extended pixel matrix are $\boldsymbol{d_x^i} \in R^{1 \times (L+N_1-1)}$ and $\boldsymbol{d_y^j} \in R^{(K+N_2-1) \times 1}$, respectively. The extended image matrix after zero padding is denoted by $\boldsymbol{X} \in R^{(K+2(N_2-1)) \times (L+2(N_1-1))}$. This operation is also known as convolution which is performed by multiplying and accumulating the instantaneous values of the overlapping part of the pixel values and the given kernels. As we can see, the kernel vector $\boldsymbol{h_x}$ and $\boldsymbol{h_y}$ both have been flipped to $\hat{\boldsymbol{h}}_{\boldsymbol{x}}$ and $\hat{\boldsymbol{h}}_{\boldsymbol{y}}$,

$$\hat{\boldsymbol{h}}_{\boldsymbol{x}} = \begin{bmatrix} h_x^{N_1} & \cdots & h_x^1 \end{bmatrix} \quad \hat{\boldsymbol{h}}_{\boldsymbol{y}} = \begin{bmatrix} h_y^{N_2} \\ \vdots \\ h_y^1 \end{bmatrix} \tag{4}$$

then we do the piece-wise multiplication of the flipped kernel vector $\hat{\boldsymbol{h}}_{\boldsymbol{x}}$ and $\hat{\boldsymbol{h}}_{\boldsymbol{y}}$ with the corresponding elements in the row or column of the extended pixel matrix $\boldsymbol{X}$, and sum the results up to be one element

in the output vector, i.e., the difference vector.

$$\begin{aligned} d_x^i(n) &= \hat{\boldsymbol{h}}_{\boldsymbol{x}} \boldsymbol{x}_{\boldsymbol{i,n}}^{\boldsymbol{T}} \quad \text{for } n = 1, 2, \cdots, L + N_1 - 1 \\ d_y^j(n) &= \hat{\boldsymbol{h}}_{\boldsymbol{y}}^{\boldsymbol{T}} \boldsymbol{y}_{\boldsymbol{j,n}} \quad \text{for } n = 1, 2, \cdots, K + N_2 - 1 \end{aligned} \tag{5}$$

where $\boldsymbol{x_{i,n}} = [x_i^{n-N_1+1}, \cdots, x_i^n]$ is one piece of ith row with N_1 pixels and $\boldsymbol{y_{j,n}} = [y_j^{n-N_2+1}, \cdots, y_j^n]^T$ is one piece of jth column with N_2 pixels. The size of $\boldsymbol{x_{i,n}}$ and $\boldsymbol{y_{j,n}}$ are determined by the size of the kernel vectors $\boldsymbol{h_x}$ and $\boldsymbol{h_x}$, respectively.

What kind of kernel vectors could bring us the edges in the image?

For the simplest case as shown in [2], we can just set these two kernel vectors as

$$\boldsymbol{h_x} = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad \boldsymbol{h_y} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{6}$$

and the flipped kernel vectors will be

$$\hat{\boldsymbol{h}}_{\boldsymbol{x}} = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \quad \hat{\boldsymbol{h}}_{\boldsymbol{y}} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \tag{7}$$

Then, we can see the output values from the convolution computing,

$$\begin{aligned} d_x^i(n) = \hat{\boldsymbol{h}}_{\boldsymbol{x}} \boldsymbol{x}_{\boldsymbol{i,n}}^{\boldsymbol{T}} &= \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_i^{n-2} \\ x_i^{n-1} \\ x_i^n \end{bmatrix} = x_i^{n-2} - x_i^n \quad \text{for } n = 1, 2, \cdots, L + N_1 - 1 \\ d_y^j(n) = \hat{\boldsymbol{h}}_{\boldsymbol{y}}^{\boldsymbol{T}} \boldsymbol{y}_{\boldsymbol{j,n}} &= \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} y_j^{n-2} \\ y_j^{n-1} \\ y_j^n \end{bmatrix} = y_j^{n-2} - y_j^n \quad \text{for } n = 1, 2, \cdots, K + N_2 - 1 \end{aligned} \tag{8}$$

and we can see actually

$$\begin{aligned} \nabla_x(x_i^n) &\approx -(x_i^{n-2} - x_i^n) = -d_x^i(n) \\ \nabla_y(y_j^n) &\approx -(y_j^{n-2} - y_j^n) = -d_y^j(n) \end{aligned} \tag{9}$$

which is known as digital differentiator.

We do the flip because we want to combining the past samples to get the current output, this

concept is always used to deal with time sequential signals. However, for the pixel values in one image, we just scan the image from left to right and from top to bottom. For the convolution on horizontal direction, we can regard the pixel values on the left as the samples in the past. If we want to understand this convolution process by sliding the kernel vector over the image, the vector that we slide from left to right of the image actually is the flipped kernel vector $\hat{h}_{\boldsymbol{x}}$, the first element in the kernel vector should be multiplied with the latest sample, i.e., the pixel in the right side, thus, the head of the kernel vector is also in the right side.

II. Kernel Matrix

In practice, we always use a kernel matrix to scan the image which will combine more past pixel values, i.e., in two dimensions, to compute the current derivative. For example, with the Prewitt kernel,

$$\boldsymbol{H_x} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \boldsymbol{H_y} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{10}$$

the flipped kernels are

$$\hat{\boldsymbol{H}}_{\boldsymbol{x}} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \qquad \hat{\boldsymbol{H}}_{\boldsymbol{y}} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{11}$$

Actually, these kernel matrices have been flipped twice which achieves a result that the top elements flipped to the bottom and the left elements flipped to the right. This procedure is different from the matrix transpose which only flipped the elements along the diagonal.

First, we give a general formulation for the convolution procedure with kernel matrix. From the above section, we know the size of each row is determined by the size of the horizontal kernel, when the horizontal kernel becomes a matrix $\boldsymbol{H_x}$, it is size of the column dimension that controls the size of each row. Thus, we define $\boldsymbol{H_x} \in R^{p \times N_1}$, we extend the kernel from one row to p rows, and it does the same thing in each row with the kernel vector. The convolution result of pixel (n_1, n_2) in the horizontal direction goes like:

$$D_x(n_1, n_2) = \sum \hat{\boldsymbol{H}}_{\boldsymbol{x}} \odot \boldsymbol{X}_{(\boldsymbol{n_1}, \boldsymbol{n_2})} \quad \text{for } n_1 = 1, 2, \cdots, K + p - 1,\ n_2 = 1, 2, \cdots, L + N_1 - 1 \tag{12}$$

where $\boldsymbol{D_x} \in R^{(K+p-1)\times(L+N_1-1)}$ represent the convolution result matrix by applying $\boldsymbol{H_x}$, (n_1, n_2) is the index of one element in this matrix, and the overlapped pixel matrix $\boldsymbol{X_{(n_1,n_2)}}$ is

$$\boldsymbol{X_{(n_1,n_2)}} = \begin{bmatrix} x_{n_1-p+1}^{n_2-N_1+1} & \cdots & x_{n_1-p+1}^{n_2} \\ \vdots & \ddots & \vdots \\ x_{n_1}^{n_2-N_1+1} & \cdots & x_{n_1}^{n_2} \end{bmatrix} \tag{13}$$

which is a partial of the extended pixel mtrix $\boldsymbol{X} \in R^{(K+2(p-1))\times(L+2(N_1-1))}$. If we use the horizontal Prewitt kernel $\boldsymbol{H_x}$, where $N_1 = 3$ and $p = 3$, the output value

$$D_x(n_1, n_2) = \sum_{i=0}^{2}(x_{n_1-i}^{n_2-2} - x_{n_1-i}^{n_2}) \quad \text{for } n_1 = 1, 2, \cdots, K+2,\ n_2 = 1, 2, \cdots, L+2 \tag{14}$$

which means each value in matrix $\boldsymbol{D_x}$ is the combination of 3 differences between the values in 2 columns. The similar calculation goes with the convolution over the vertical direction. If we define it in a general case, $\boldsymbol{H_y} \in R^{N_2\times q}$, which means we combine q columns and in each column we do the convolution over N_2 pixels. The convolution result of pixel (n_1, n_2) in the vertical direction goes like:

$$D_y(n_1, n_2) = \sum \hat{\boldsymbol{H}}_{\boldsymbol{y}} \odot \boldsymbol{X_{(n_1,n_2)}} \quad \text{for } n_1 = 1, 2, \cdots, K+N_2-1,\ n_2 = 1, 2, \cdots, L+q-1 \tag{15}$$

and the overlapped pixel matrix $\boldsymbol{X_{(n_1,n_2)}}$ will becomes to

$$\boldsymbol{X_{(n_1,n_2)}} = \begin{bmatrix} x_{n_1-N_2+1}^{n_2-q+1} & \cdots & x_{n_1-N_2+1}^{n_2} \\ \vdots & \ddots & \vdots \\ x_{n_1}^{n_2-q+1} & \cdots & x_{n_1}^{n_2} \end{bmatrix} \tag{16}$$

If we use the vertical Prewitt kernel $\boldsymbol{H_y}$, where $N_2 = 3$ and $q = 3$, the output value

$$D_y(n_1, n_2) = \sum_{j=0}^{2}(x_{n_1-2}^{n_2-j} - x_{n_1}^{n_2-j}) \quad \text{for } n_1 = 1, 2, \cdots, K+N_2-1,\ n_2 = 1, 2, \cdots, L+q-1 \tag{17}$$

which means each value in matrix $\boldsymbol{D_y}$ is the combination of 3 differences between the values in 2 rows.

We slid the horizontal kernel $\boldsymbol{H_x}$ column-wise first and followed by advancing along the rows of the image, and we get the convolution result matrix $\boldsymbol{D_x}$. The first overlap between the kernel and the image pixels would result when the pixel at the bottom-right of the kernel falls on the first-pixel value at the top-left of the image matrix, and the convolution result will be $D_x(1, 1)$, i.e., the first value in differentiation matrix $\boldsymbol{D_x}$. After moving the kernel forward pixel-by-pixel until there is none of the pixels in the kernel

overlap with those in the image, we have collected all the values in $\boldsymbol{D_x}$ which has $(K+p-1)\times(L+N_1-1)$ elements. The original image only has $K \times L$ elements. We want to see the edges in each pixel of the original image, thus, we select the center part of $\boldsymbol{D_x}$, i.e. $\hat{\boldsymbol{D}}_{\boldsymbol{x}} \in R^{L\times K}$ to be the representation of edges in the original image. As we know how to compute each element in $\boldsymbol{D_x}$, then what will be the elements in $\hat{\boldsymbol{D}}_{\boldsymbol{x}}$? Since $\hat{\boldsymbol{D}}_{\boldsymbol{x}}$ picks out the elements in the center of $\boldsymbol{D_x}$, then

$$\hat{D}_x(1,1) = D_x(\frac{p-1}{2}+1, \frac{N_1-1}{2}+1) \tag{18}$$

That means when we compute the first value in $\hat{\boldsymbol{D}}_{\boldsymbol{x}}$, the first element of kernel matrix has moved to the $(\frac{p-1}{2}+1, \frac{N_1-1}{2}+1)$ pixel of the image, and we can extend this to every value in $\hat{\boldsymbol{D}}_{\boldsymbol{x}}$:

$$\hat{D}_x(n_1,n_2) = \sum_{i=0}^{N_1-1}(x_{n_1-i}^{n_2-2} - x_{n_1-i}^{n_2}) \tag{19}$$

where

$$n_1 = \frac{p-1}{2}+1, \frac{p-1}{2}+2, \cdots, \frac{p-1}{2}+K,\ n_2 = \frac{N_1-1}{2}+1, \frac{N_1-1}{2}+2, \cdots, \frac{N_1-1}{2}+L \tag{20}$$

Remember, these indexes are based on the pixel matrix $\boldsymbol{X} \in R^{(K+p-1)\times(L+N_1-1)}$ after the zero padding, if we changed our view back to the original image matrix and connect to the index of $\hat{\boldsymbol{D}}_{\boldsymbol{x}}$, and still use the Prewitt Kernel, where $\frac{p-1}{2}=1, \frac{N_1-1}{2}=1$, we can see that

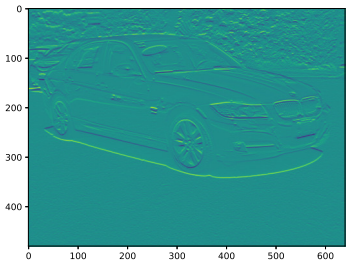
$$\hat{D}_x(k,l) = \sum_{i=0}^{N_1-1}(x_{k+1-i}^{l-1} - x_{k+1-i}^{l+1}) \quad \text{for } k=1,2,\cdots,K,\ l=1,2,\cdots,L \tag{21}$$

the index (k,l) is the same in the original image matrix. Then, we can directly see that the (k,l)th element in $\hat{\boldsymbol{D}}_{\boldsymbol{x}}$ combines the difference among the pixel values $(k,l-1)\&(k,l+1)$ over 3 rows, i.e., $k-1,k,k+1$. And for a given image as shown in Fig. 1, we transform it into 2-D gray-scale image, and apply the Prewitt kernel to extract edges. From matrix $\hat{D}_x$, we can get the edge extraction results shown in Fig 2(a). And following the same idea we can get the matrix $\hat{D}_y$, which is the central part of $\boldsymbol{D_y}$, and the result is shown in Fig 2(b). When we combine the horizontal scanning results and the vertical scanning results as

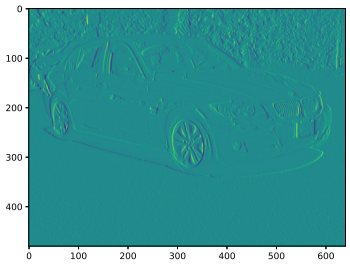
$$\hat{D}(k,l) = \sqrt{\hat{D}_x(k,l)^2 + \hat{D}_y(k,l)^2} \quad \text{for } k=1,2,\cdots,K,\ l=1,2,\cdots,L \tag{22}$$

(a) Original Image

Fig. 1



(a) Horizontal direction.



(b) Vertical Direction

(c) Combined

Fig. 2: Feature extraction with Prewitt kernel.

which you can find the performance in Fig. 2(c). If we change the kernel matrix into

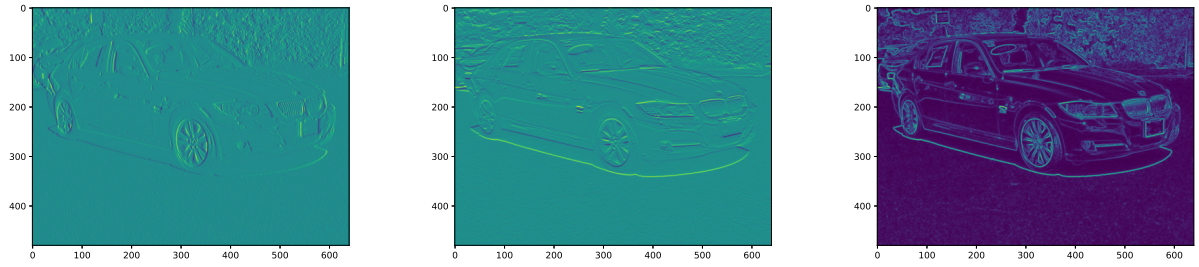
$$\boldsymbol{H_X} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \boldsymbol{H_Y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{23}$$

which is known as Sobel kernel, we can get the results as

REFERENCES

[1] J. Watt, R. Borhani, and A. Katsaggelos, *Machine learning refined: foundations, algorithms, and applications*. Cambridge University Press, 2020.

[2] W.-S. Lu, "Handwritten digits recognition using pca of histogram of oriented gradient," in *2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. IEEE, 2017, pp. 1–5.



(a) Vertical Direction (b) Horizontal Direction (c) Combined

Fig. 3: Edge extraction with Sobel kernel