Face Detection using Neural Network and Various Other Techniques

Graham Burgsma, Eric Kollee

***Abstract*—There are many different ways to do facial detection each similar in some way. Almost every algorithm does a variation of skin detection or contrast comparison to detect locations of skin in the image. Some face detection algorithm are very fast and can do it in real time video previews on smart phones and digital cameras. In this paper face detection is done using a variety of algorithms and techniques.**

***Keywords*—Face detection, Feed Forward Neural Network, Flood Fill, Box Merging, Sobel Edge Detection, Eigenface**

I. Introduction

The purpose of this paper is to explain and explore face detection. This project uses a Neural Network to do skin segmentation in an image. From there it removes noise by removing small spots of black or white in the images. Next the white spots are located using box merging techniques, these boxes are then compared against Eigenfaces which determine if that area contains a face.

A. Skin Segmentation Data Set

The Skin Segmentation Data Set is from the UCI Machine Learning Repository. The data is collected by randomly sampling RGB values from face images of various age groups, race groups and genders obtained from FERET database and PAL database. The data set contains 245057 total samples, 50859 are skin samples and 194198 is non-skin samples. There are three features (red, green, blue) and one classification (skin or non-skin).

II. Problem Definition

Detect the location of all the faces in an image.

III. Algorithms

A. Neural Network

An artificial neural network (ANN) in machine learning is a network that resembles biological neural networks, such as in the human brain. ANNs model the brain in two ways, the first is that knowledge acquired is done through a learning process. Secondly, ANNs use a network of connected neurons with synaptic weights, that adjust with learning, to store information and patterns. An ANN is good at taking a large number of inputs to produce a fairly accurate output. ANNs are very good at pattern recognition, some common uses are hand writing interpretation and image analysis such as finger print recognition or image compression. ANNs can be used for both supervised and unsupervised learning. With supervised learning the outcome is most often looking for a classification where unsupervised learning usually works to organize or cluster input.

1) Feed Forward Network

Feed Forward Neural Networks are the most simple form of an ANN in which the connections between neurons only move in one direction, they do not move backwards or have any cycles. Connections start from the inputs and are connected to one or more hidden layers which are then connected to the output. It is common for feed forward networks to use backpropagation for learning.

2) Backpropagation

Backpropagation calculates error in an neural network and is a popular method used for supervised learning networks. Backpropagation works in a three step process: forward propagation, backward propagation and weight update. The forward propagation or forward pass moves the training input linearly through the network in order to produce the output of the network. The output is then compared to the teacher value (in supervised learning) to produce an error signal. In backward propagation the error is moved backward through the network to each of the neurons. In the final step, weight update, the weights of the network are updated which is the key process in the network learning.

The purpose of the neural network is to classify each pixel as skin or not skin. The neural network trains on 245057 samples from the Skin Segmentation Data Set. The parameters for the network are shown in Table I. The RGB values from the training data and from the image are the input and the output has two classifications of skin and non-skin. When training is complete the image is passed through the network to determine where there is skin in the image. For each pixel, if it is classified as skin its colour is changed to white and if it is non-skin it is changed to black. Figure 1 shows the initial output of the neural network after detecting the skin in the image.

Epochs	5
Cross-Validation	Holdout (2)
Learning Rate	0.005
Momentum Rate	0.0
Activation Function	Logistic
Hidden Layers	1
Hidden Neurons	6
Inputs	3
Output Neurons	2

TABLE I: Neural Network Parameters

Fig. 1: Neural Network Output

B. White and Black Spot Removal

The purpose of the image segmentation is to remove all black and white spots from the image after the skin detection is complete. All black spots will be completely removed (except the background) and only white spots which are less than 300 pixels (set from experimentation).

Removing all black spots is done using an iterative flood fill algorithm which maps out the background of the image by traversing each connected pixel that is black. The background is marked as visited, then all pixels which are black and not visited are changed to white. This removes all the black "islands" or "blobs" from the image such as the location of eyes. Figure 2 shows the image after the black spots have been removed.

Fig. 2: Image After Black Spots Filled

Removing all the white spots less than 300 pixels is done using a recursive flood fill algorithm. The reason it is done recursively is because the algorithm needs to explore each white spot, if it is less than 300 pixels it gets changed to black when reversing up the stack. However if the area is 300 pixels or larger than the recursion stops and it reverses up the stack but does not change the colour of any pixels. This preserves the larger white areas such as faces or large skin areas but removes small areas which have no value. Figure 3 shows the image after white spots smaller than 300 pixels have been filled in.

Fig. 3: Image After White Spots Filled

C. Sobel Edge Detection

Sobel Edge Detection uses the Sobel operator on an image to perform edge detection. The Sobel operator uses two convolution matrices shown in Figure 4 and 5. The first step is converting the image to gray scale. Then each convolution matrix is applied individually to each pixel in the image. The surrounding eight pixels are multiplied by the corresponding number in the convolution matrix. The sum of these values from both the vertical (G_x) and horizontal (G_y) are put into the formula $G = \sqrt{{G_x}^2 + {G_y}^2}$ to get the new pixel value. This new value is known as the gradient magnitude. The output of the Sobel Edge Detection is shown in Figure 6.

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Fig. 4: Vertical Convolution Matrix

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Fig. 5: Horizontal Convolution Matrix

The purpose of the edge detection is to combine with the output from the white and black spot removal. This should help define the faces and outline more; However, it was discovered that this did not help define the faces better and in fact it reduced the accuracy of the box merging algorithm. For these reasons the edge detection was not used, though the option is available in our program.

Fig. 6: Output From Sobel Edge Detection

D. Box Merging

The box merging algorithm is fairly simple. The algorithm loops through all the boxes in a nested loop to see if any of them intersect with the current one. If they intersect and the box being compared is one third the size of the current box, then they are merged. The reason for the box to be one third the size is to prevent all the faces from being combined, instead only small boxes are added to larger ones.

Aside from boxes that are intersecting, boxes that are close together should be merged. For this there is a vertical and horizontal merging threshold. The vertical threshold is larger because facial features and necks are most often vertical and should be combined. The horizontal threshold should be smaller. For this project the vertical threshold was 10 and horizontal threshold was 5.

This algorithm loops until there are no more changes made. This is because when 2 boxes merge, the bounds become larger and so more boxes could then be merged. Figure 7 should the image before box merging. Figure 8 should the image after the box merging algorithm was done.

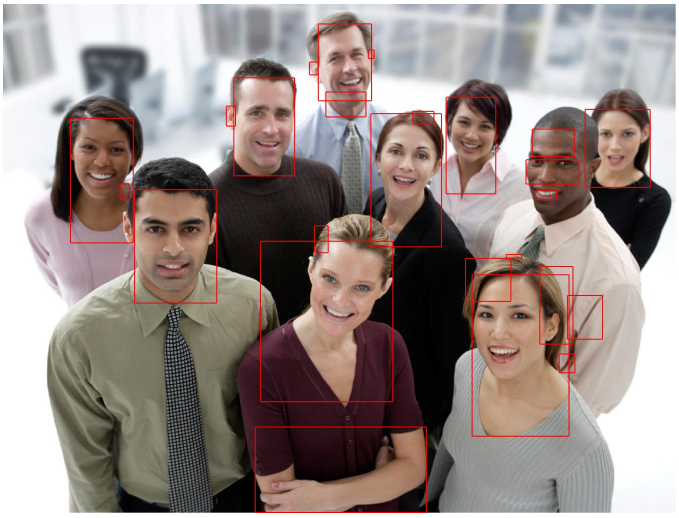

Fig. 7: Image Before Box Merging

Fig. 8: Image After Box Merging

E. Eigenfaces

Eigenfaces are a collection of eigenvectors that represent human faces and are used in facial detection for computer vision applications. Eigenvectors are sometimes known as ”characteristic vectors” and can be used to determine defining features. Simply put: In facial detection they are used to correlate which attributes are most representative and defining of a ”face”.

1) Creation

Creation of eigenfaces is very mathematically complex and therefore computationally expensive. Most applications of facial detection will use already generated databases of eigenfaces for this reason. This paper will aim to give an overview on the creation of eigenfaces for completeness on facial detection.

a) Training Data and Structure

For this implementation of eigenfaces, 100 faces were used from the LFWcrop Face Dataset. These images are 64 pixels tall by 64 pixels wide and are in grey-scale (all training images are in grey-scale for easier representation. With grey-scale images R=G=B). Each image contains exactly one centered face filling the entire frame.

Fig. 9: Example Training Image

The training images are then converted into a single column vector containing the grey-scale value of every pixel in the image.

$$\begin{bmatrix} Image1Pixel1 \\ Image1Pixel2 \\ ... \\ Image1PixelN \end{bmatrix}$$

Fig. 10: Grey-scale Image Column Vector

Each of the single column representations for the training images are placed into a matrix with all of the other training images. The final structure is shown in Figure 10.

$$\begin{bmatrix} Image1Pixel1 & Image2Pixel1 & ... & ImageNPixel1 \\ Image1Pixel2 & Image2Pixel2 & ... & ImageNPixel2 \\ ... & ... & ... & ... \\ Image1PixelN & Image2PixelN & ... & ImageNPixelN \end{bmatrix}$$

Fig. 11: Grey-scale Training Set Matrices

The first mathematical calculation and operation performed is the mean grey-scale pixel subtraction from the image matrix. The mean grey-scale pixel is calculated by adding up each of the columns per row and dividing by the total number of images. This new column vector is then subtracted from the original matrix giving a data set that is centered around the origin. This is required for a later step called Singular Value Decomposition.

b) Covariance

In statistics, covariance measures the amount of change two random variables have on one another. With matrices, covariance is defined as $C = A(A^T)$. This matrix multiplication will result in a new matrix with n^2 elements where n is the number of pixels in a training image. (e.g in the 64x64 case the resulting Covariance matrix was 4096x4096)

c) Singular Value Decomposition

Singular Value Decomposition or SVD is a simple way to achieve Principle Component Analysis or PCA. PCA is a method of reducing dimensional complexity by filtering out uninformative attributes in an input space and keeping only the relevant or contributing attributes. This is important with facial detection or recognition algorithms because images can contain a vast amount of uninformative information known as "noise". In the case of eigenface generation SVD is performed on the covariance matrix in order to generate a set with n^2 u eigen vectors. Each of these vectors represent an eigenface with the most defining eigenface being the first eigenvector. Each proceeding eigenvector is less informative than its predecessor, therefore it is only required to use the top K eigenvectors. The K value can vary, in this project a value of 10 was used however studies have shown that a max K value required is 43 to cover enough variance to amount for every human face.

d) Normalization and Visualization

Now that SVD has been calculated, the eigenvectors need to be normalized in order to be displayable in grey-scale. This process is done by setting the largest value in each eigenvector to the maximum grey-scale value of 255 and the smallest value to 0. The remaining values are then converted to fit within the range by the following formula.

$$newValue_i = 255 * \frac{(eigenVector_i - min)}{(max - min)}$$

Now that the eigenvectors have been normalized to the displayable range (0-255), they can be converted to eiganfaces by the reverse process for creating the column vectors. It is important to note that the RGB colour channels will need to be manually set to the singular value in the eigenvector. Figures 12 and 13 depict our first and seventh eigenfaces and Figure 14 is the mean eigenface.

Fig. 12: Example Eigenface

Fig. 13: Example Eigenface

Fig. 14: Mean Eigenface

IV. Conclusion

In conclusion it is evident there are many different machine learning techniques necessary to do face detection on an image. It is also a very difficult task and many factors affect the accuracy of the detection. With the algorithm used there is no way to detect if faces are separate if they are close together because the algorithm detects skin as a whole and does not differentiate between shades of skin color or edges of a face. It is also a difficult task to differentiate between hands arms or other skin areas from faces.

Face detection is the first step in facial recognition and has many uses in smart phones and digital cameras so it is widely used. There are many different algorithms and methods for face detection, the one used in this project had some flaws. Training the neural network caused the program to run slowly, eigenfaces are difficult and time consuming to generate and it is difficult to accurately detect which is a face and which is not.

References

[1] Inseong Kim, Joon Hyung Shim, and Jinkyu Yang. Face detection. 2003. https://web.stanford.edu/class/ee368/project$_0$3/$project/reports/ee368group02.pd$

[2] Uci machine learning repository: Data set. http://archive.ics.uci.edu/ml/datasets/skin segmentation.

[3] Eigenface tutorial. http://www.pages.drexel.edu/ sis26/eigenface tutorial.htm.

[4] Eigenfaces for dummies. http://jmcspot.com/eigenface/.