Neural Network Based Face Detection

CS 7495 Final Project
Ben Axelrod

This project's goal was to implement a neural network based face detector as outlined in this paper: Rowley, H., Baluja, S., and Kanade, T., "Neural Network-Based Face Detection", Proc. IEEE Conf. on Computer Vision and Pattern Recognition, San Francisco, CA, pp. 203-207. 1996. There is a newer version of the paper with better figures here < http://www.vision.caltech.edu/CNS179/papers/Kanade98.pdf >. The authors of the paper have set up a cool web demo < http://demo.pittpatt.com/ > that allows anyone to upload a picture and try out their detector.

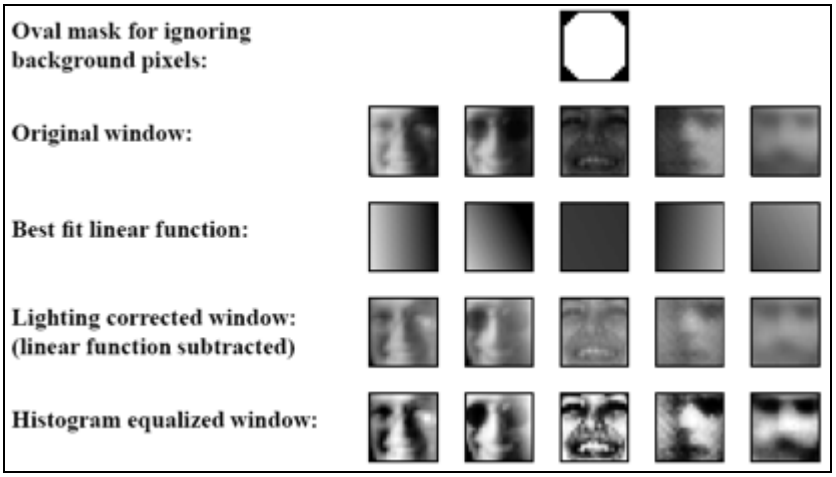
Overview

This project seeks to detect upright human faces in grayscale images with the help of a neural network. The retinally connected neural network (NN) examines small windows in the image and outputs a confidence factor corresponding to whether or not it thinks the window contains a face. Heuristics are then used to clean up the output of the network. This algorithm is not real-time as it can take up to an hour to analyze even a moderately sized an image. I created my NN detector in Matlab using it's NNtoolbox. I followed the paper pretty closely; however my implementation only used 1 NN with only 1 layer of hidden nodes. The paper used multiple networks with multiple hidden layers. My network achieved a face detection rate of 41.3% and a false positive rate of 0.00044%. These percentages are much lower than the paper's, therefore, there are some parameters of the algorithm that can still be optimized.

Training Procedure

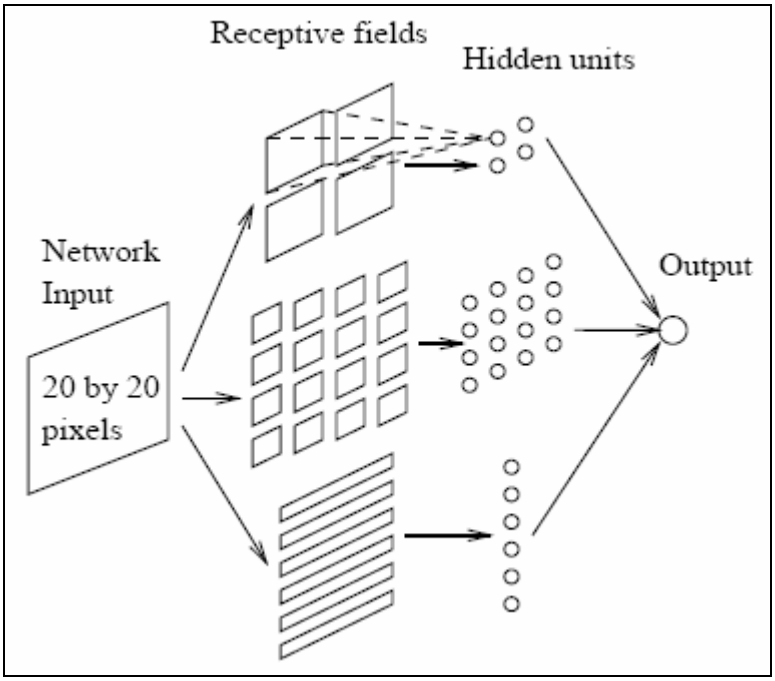
1. Gather a database of face images. I got mine from the Yale < http://cvc.yale.edu/projects/yalefaces/yalefaces.html > and CalTech < http://www.vision.caltech.edu/html-files/archive.html > databases. The faces need to be upright and be 20x20 pixels in size (with approx 12 pixels between the centers of the eyes). The point halfway between the center of the eyes and the upper lip should be in the center of the image. The paper then created 15 images from each face by mirroring, rotating, scaling, and translating. I only mirrored each face. I ended up with 624 face images. 

2. Pre-process the faces to eliminate some lighting variation. This involves masking the corners, fitting a linear function to the data, and equalizing. The process is shown below.



3. Create 1000 images of random noise and apply the same pre-processing to these as well. 

4. Train the network to classify faces as 1 and non-faces as -1. The network architecture can be seen below.



Note that the 4 upper hidden nodes each take only a 10x10 quadrant of the input window as their input. The middle 16 hidden nodes each take only a 5x5 pixel region. And the bottom nodes take a 5x20 pixel horizontal slice of the window. Because Matlab does not allow the inputs of a NN to be different, I had to do an ugly and expensive hack. The entire image was fed to all inputs, then the unwanted parts of the image were masked out of the input weights. This masking was done after each iteration of training. This probably considerably slowed the network down. I was essentially carrying around 9000 network weights for no reason. (Only 1527 weights mattered).

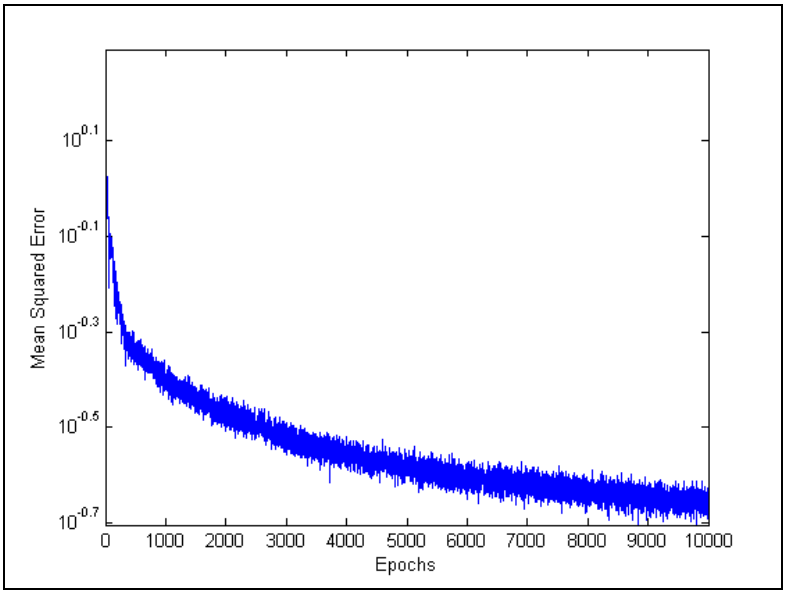
5. Run the network on images that are known to contain no faces. I used CalTech's background database < http://www.vision.caltech.edu/html-files/archive.html > for this. Note that the network must be run on all pixel locations in an image pyramid where the images are scaled by a factor of 1.2. The query window must also be put through the same pre-processing as the training images. Save the false positives to add to the non-faces training set. Go to step 4. This process of retraining with false positives is called bootstrapping.

Notes on my Training

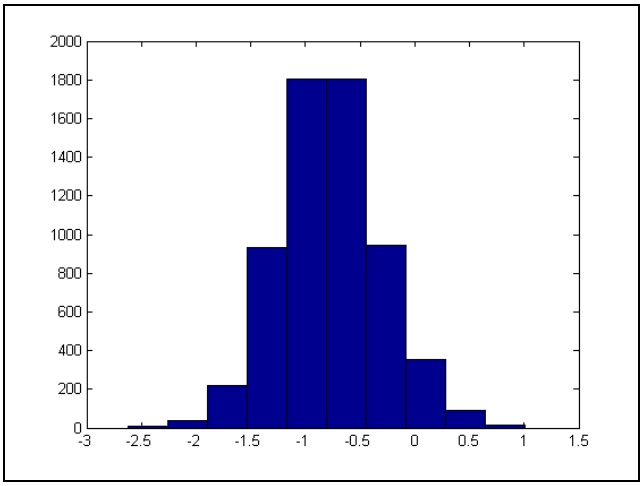
I went through these steps a few times and towards the end, the false positives began looking pretty face-like. I ended up with 6219 non-face images. 

This is one instance where I didn't follow the paper. I observed that if I train with the entire set of non-faces, the network learns to classify everything as a non-face. This is possibly because there are so many more non-faces than faces. The network training finds an easy local minima by classifying everything as -1. It is possible that this may only be an issue for the particular training algorithm I used. I needed to use scaled conjugate gradient backpropagation (trainscg) instead of the usual Levenberg-Marquardt backpropagation (trainlm). This was due to the large memory requirements of LM backprop.

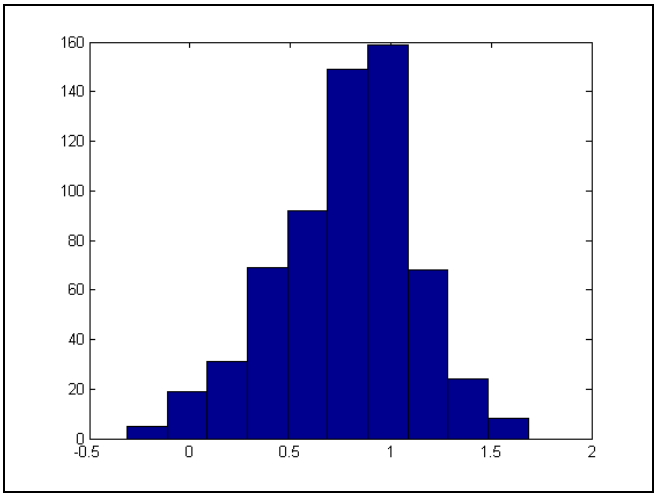
I avoided this sub-optima by choosing a small subset of the non-faces to use every training iteration. I randomly choose non-faces to achieve a 50/50 ratio of faces to non-faces. Once I gathered enough non-face examples, I trained a new network from scratch with 10000 epochs. 10000 epochs did better than 5000, but 50000 seemed to overfit. The only downside I can see to training the network in this manner is slower convergence. Here are some output graphs after training.



Mean squared error during training



Histogram of the classified training images (only non-faces).

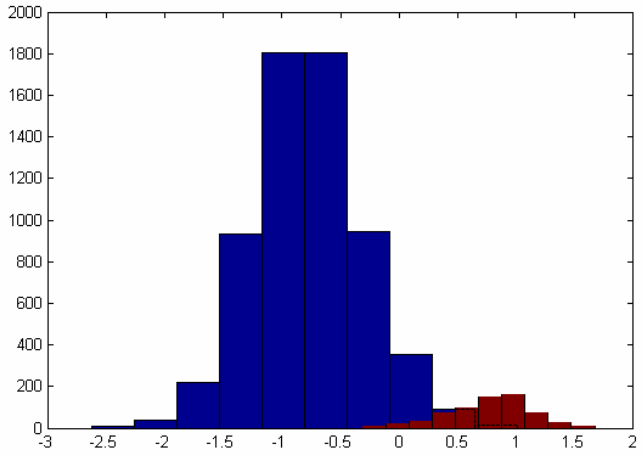


Histogram of the classified training images (only faces).

Testing Procedure

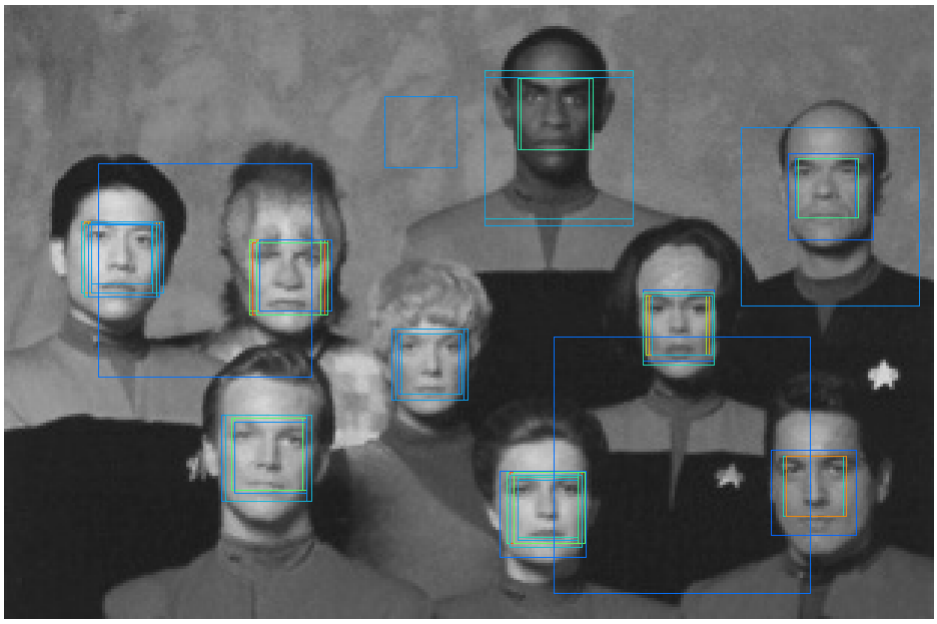
Testing involves running the network on every pixel location of the image pyramid and recording when the output of the network is above a threshold. The paper used 0 as a threshold, but my algorithm finds way too many false positives at this value. I used a threshold of 0.8 for all my test runs which seemed to give good balance between face detection and false positives. From examination of the histogram below, it might seem odd that a value closer to 0.5 was not chosen as the threshold. This is because the number of window evaluations over the entire image is so high that the false positive rate must be very low. However, my results

indicate that a lower threshold would have been beneficial.



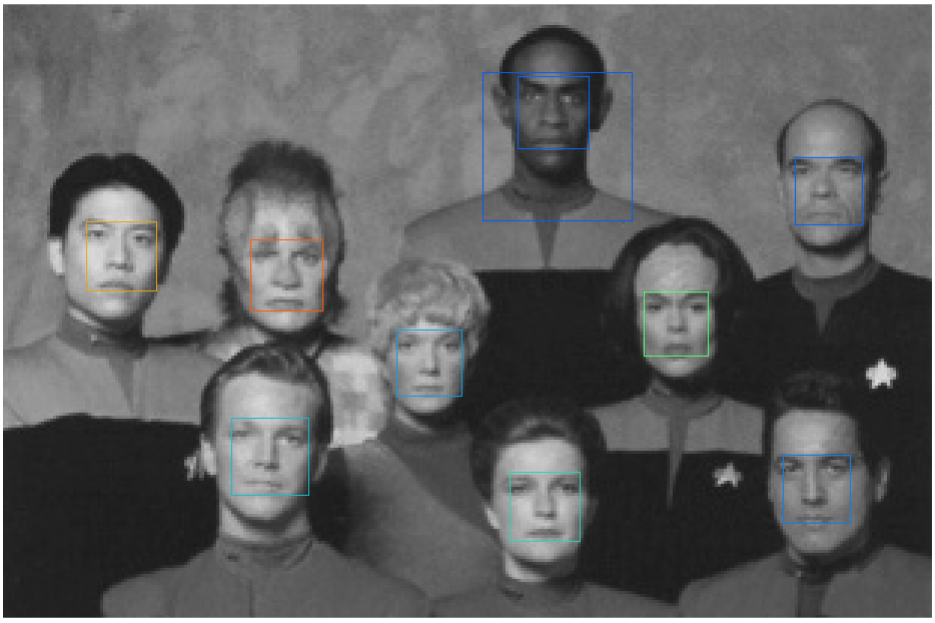
Both face and non-face histograms plotted together for comparison

After the matches in the image have been found, some post-processing can significantly reduce the number of false positives and merge overlapping predictions. This is the output of a typical run:



Note that the colors of the boxes represent the output of the NN. With red being the strongest face prediction at roughly 1.4, and blue being the weakest at 0.8. This image illustrates the 2 heuristics that can be used to clean up the face predictions. The first heuristic is that real faces have multiple matches. To take advantage of this, the locations of the matches are "spread out" and added together yielding an image like this:

The peak of every hill in this image becomes the location of a new face prediction box. This new box will have the average size of the boxes close to it, and a confidence weight corresponding to the height of the hill. Note: the paper computed the centroid of these hills in a window to determine the center, while I used hill climbing. The output of this agglomeration phase is shown below.

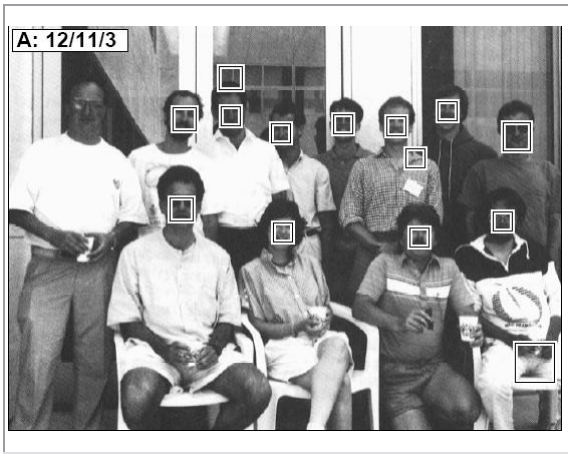


The next heuristic that can be applied is that faces rarely overlap in an image. When overlapping boxes are detected, the higher confidence weight box is kept. This produces the final output:

On a funny side note, the “alien” in this image with perhaps the most un-humanlike appearance actually received the highest face confidence!

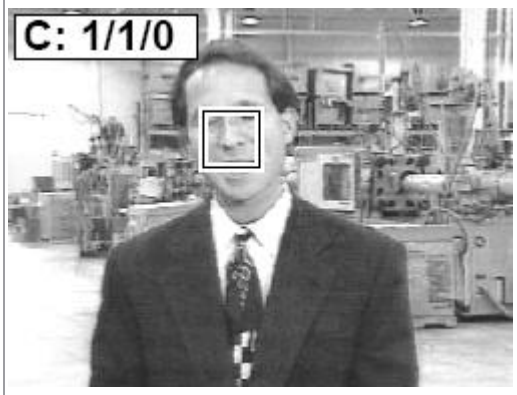
Results

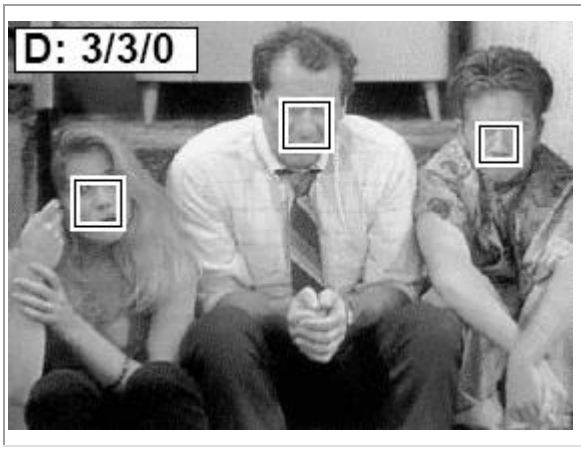
I used a completely different set of images to test my network than was used for training. I used the [CMU-VASC dataset](http://vasc.ri.cmu.edu/idb/html/face/frontal_images/index.html) < http://vasc.ri.cmu.edu/idb/html/face/frontal_images/index.html > just like the paper. Because I didn’t have enough time to run the network on the entire database, I only ran it on the images shown in figure 3 of the paper, plus a few extras. This allows a good qualitative comparison as seen below. My results are in the left column, and the paper's results are in the right column.

My Results	Paper Results
	

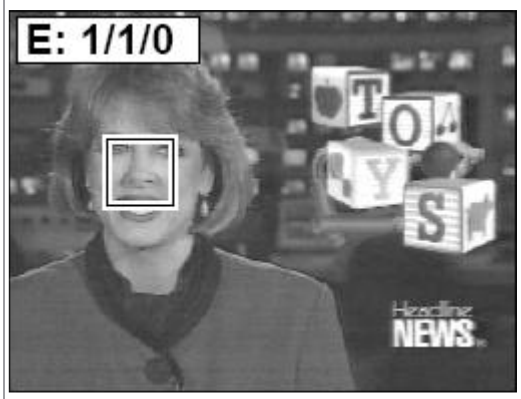

1 THRU 14
ROOF ACCESS
2
STAIR 10

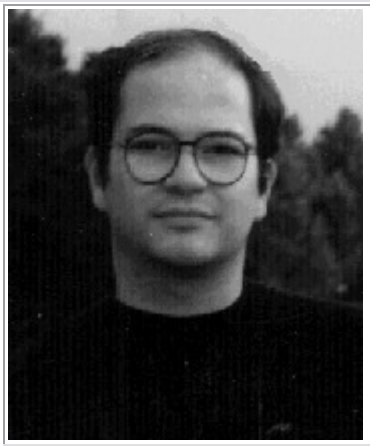

B: 6/5/1
1 THRU 14
ROOF ACCESS
2
STAIR 10

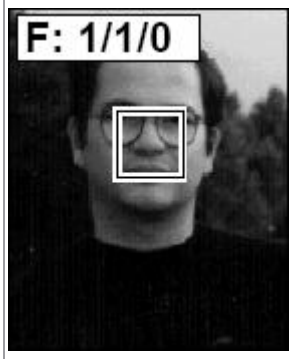

C: 1/1/0


D: 3/3/0


T
O
Y
S
Headline
NEWS

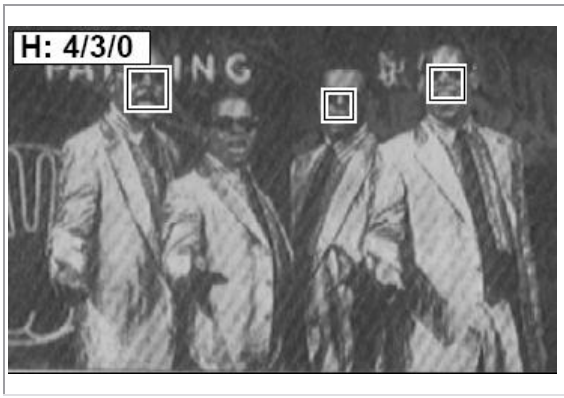

E: 1/1/0
T
O
Y
S
Headline
NEWS

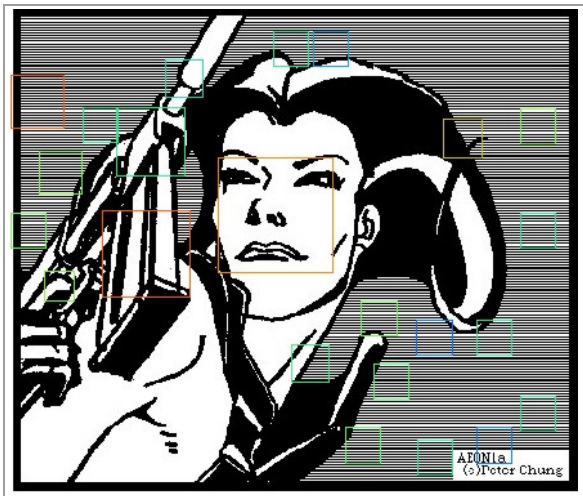


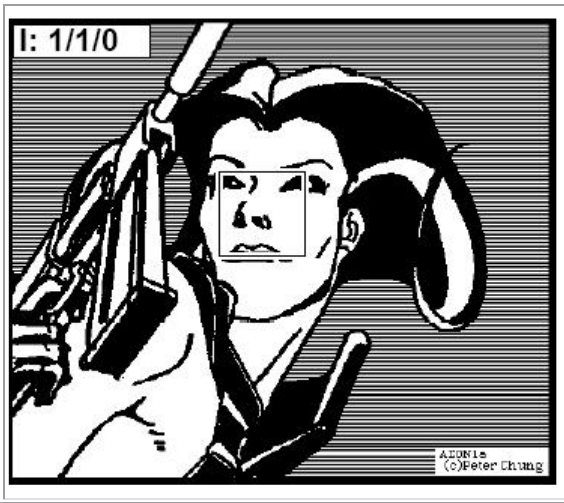

F: 1/1/0


GEORGIA PEACHES


G: 2/2/0


H: 4/3/0

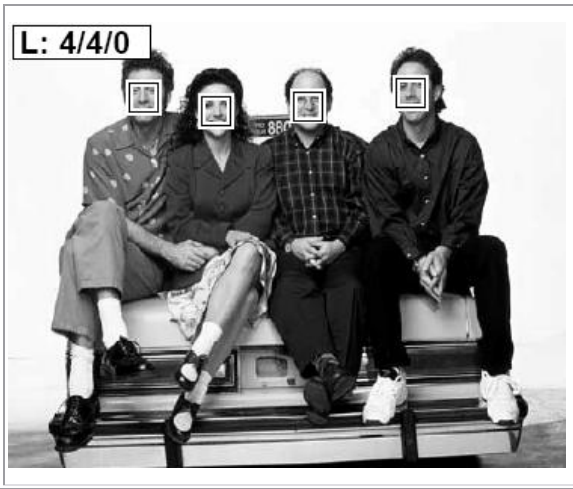

AEON1a
(c)Peter Chung

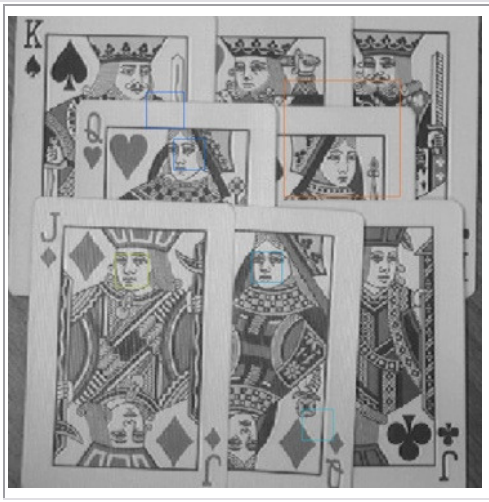

I: 1/1/0
AEON1a
(c)Peter Chung

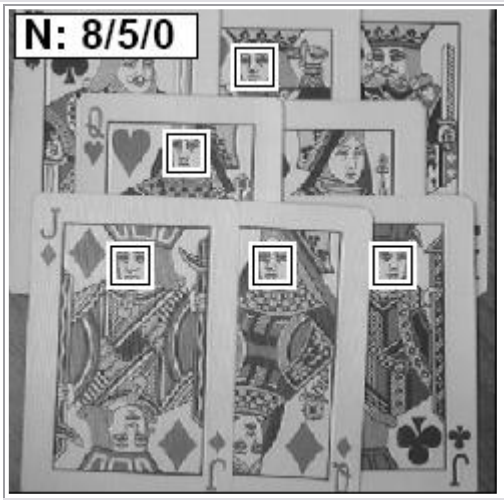

J: 1/1/0

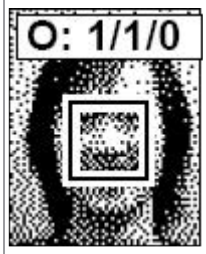

ORIGINAL MASTER RECORDING
BLUE TRAIN
john coltrane

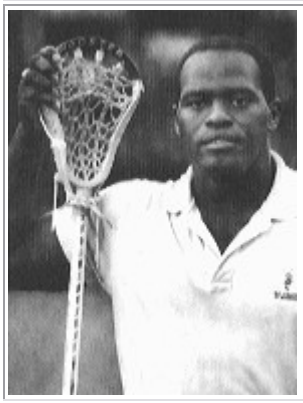

K: 1/1/0
MASTER RECORDING
BLUE TRAIN
john coltrane


L: 4/4/0

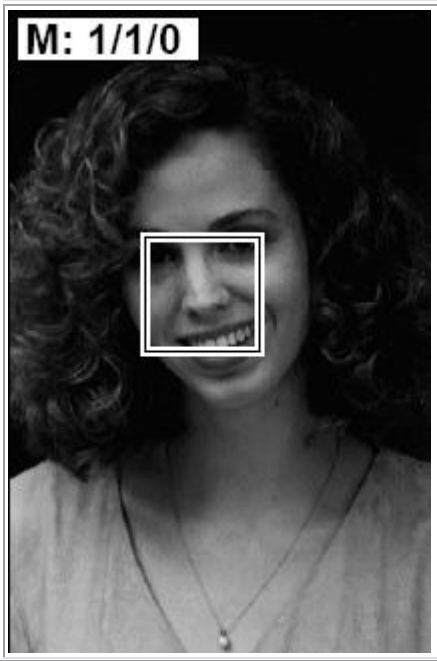


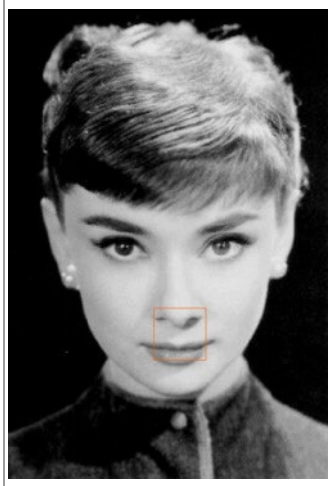

N: 8/5/0

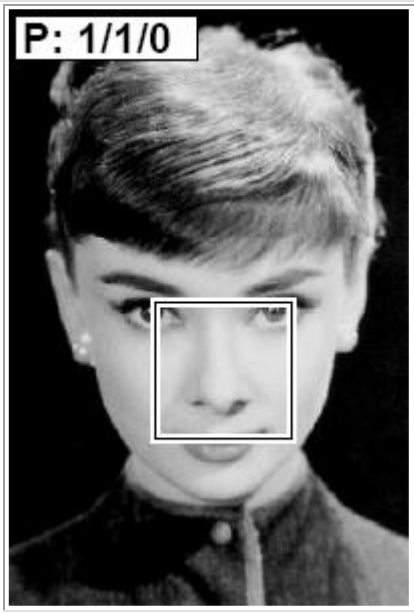

O: 1/1/0




R: 1/1/0

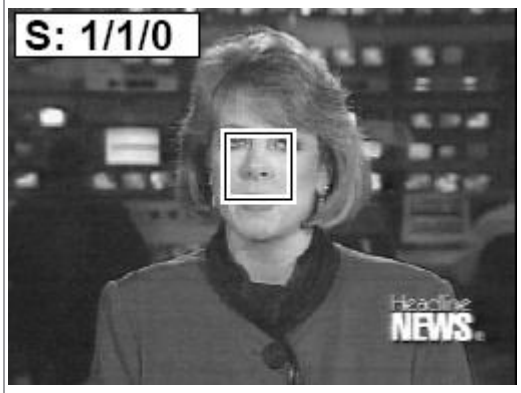

M: 1/1/0

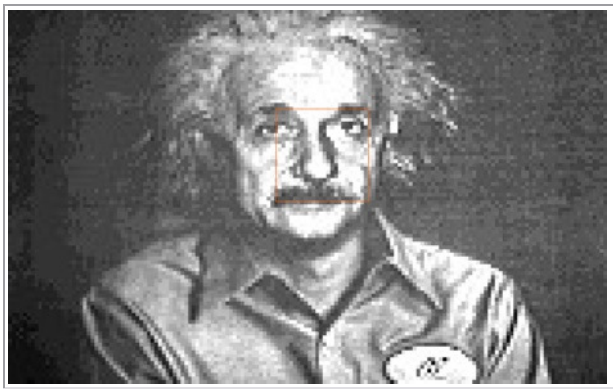


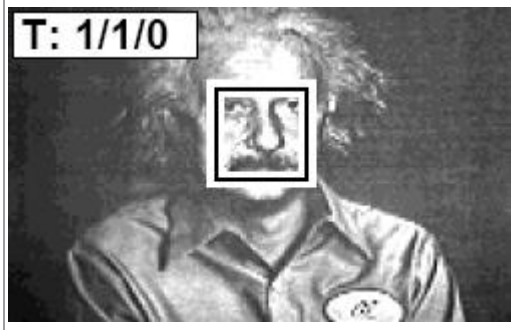

P: 1/1/0


Q: 1/1/0

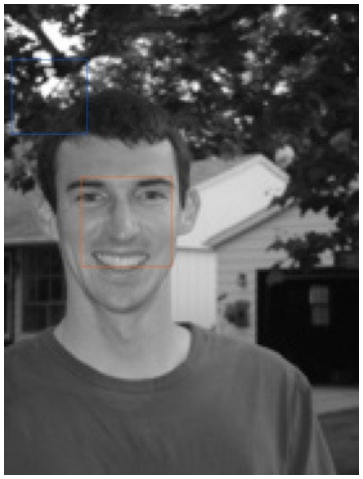

Headline
NEWS.

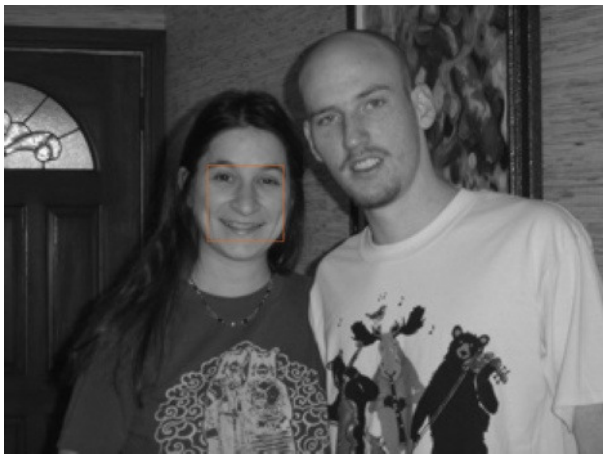

S: 1/1/0
Headline
NEWS.

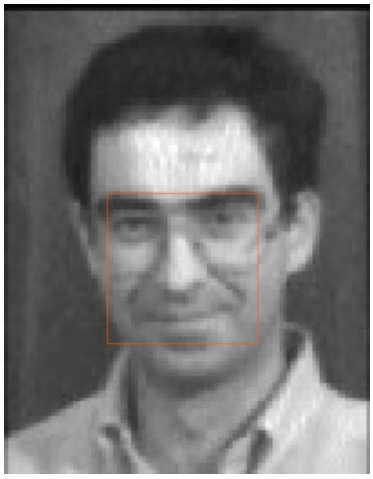



T: 1/1/0

These are the extra images that I tested with. I don't have the paper's results on these images.







I counted the missed faces and false detects in the testing images for comparison with the paper. In the table below, my network is compared against the paper's single networks with heuristics.

System	Detect Rate	False Detect Rate
My network (27 hidden nodes, 1527 connections)	41.3 %	1 / 227580
Paper's network 1 (52 hidden nodes, 2905 connections)	90.9 %	1 / 98459
Paper's network 2 (78 hidden nodes, 4357 connections)	89.5 %	1 / 115576
Paper's network 3 (52 hidden nodes, 2905 connections)	89.5 %	1 / 85230
Paper's network 4 (78 hidden nodes, 4357 connections)	90.7 %	1 / 78992

Analysis and Conclusion

Because both my detect rate and false detect rate are much lower than the paper's, it is safe to say that I ran my network with too high of a threshold. The other factors that probably hurt my performance include my network having fewer hidden nodes and a smaller training set. Taking this into consideration, I think my network compared well to the paper's networks. It seems neural networks can provide good face detectors, however their speed needs to be improved. Matlab's NNtoolbox made it easy to work with images and the network, except for its input limitations that forced me to use more inputs than needed. This severely slowed down the network.