

CS536  
Book Exercise 4.11  
Using ANNs for face recognition

2. The code you have been given is currently set up to recognize the person with the userid glickman. Modify this code to implement a "sunglasses" recognizer.

This command

```
./facetrain -n glickman.net -t straightrnd_train.list -1 straightrnd_test1.list  
-2 straightrnd_test2.list -e 75 > /mnt/floppy/glickman.out
```

parses as follows:

```
./facetrain          program name  
-n glickman.net      save network in this file  
-t straightrnd_train.list  training data file (70 images)  
-1 straightrnd_test1.list  test data file 1 (34 images)  
-2 straightrnd_test2.list  test data file 2 (52 images)  
-e 75                75 epochs  
> /mnt/floppy/glickman.out  save screen output in this file on a floppy
```

glickman.out was loaded into a spreadsheet.

The first 156 lines describe loading the 156 training and test files.

There are 20 userid's each with 7-8 (avg. 7.8) files (all straight).

glickman has 7 files:

```
sad      open  
happy   sunglasses  
sad      sunglasses  
happy   open  
neutral open  
neutral sunglasses  
angry   sunglasses
```

Considering that these 7 files have to be split three ways (train + 2test) it's amazing if any learning can occur at all!

sunglasses/open, however, is binary, with 78 cases of each.

The output says the network (glickman.net) is a 960x4x1 network.

Then follows output for each of the 75 epochs. See page 8 for definitions.

epoch	delta	trainperf	trainerr	tlperf	tlerr	t2perf	t2err
0	0.0	14.2857	0.109234	8.82353	0.11411	13.4615	0.116624
1	3.32521	94.2857	0.0139558	97.0588	0.00873864	96.1538	0.0114627
2	1.49389	94.2857	0.0123471	97.0588	0.00759696	96.1538	0.00992205
3	1.44712	94.2857	0.0109007	97.0588	0.00731984	96.1538	0.00903667
4	1.5127	94.2857	0.00893908	97.0588	0.00717316	96.1538	0.00796931
5	1.52463	97.1429	0.00706015	97.0588	0.00655391	100	0.00687427
6	1.37468	100	0.00571391	97.0588	0.00561064	100	0.00589527
7	1.19803	100	0.00475462	97.0588	0.00485443	100	0.00518543
8	1.06263	100	0.00404678	100	0.00429166	100	0.00468606
9	0.954857	100	0.00350706	100	0.00385882	100	0.004318
10	0.868458	100	0.00308521	100	0.00351311	100	0.00403245
...							
74	0.136010	100	0.000169136	100	0.00112293	100	0.00171147
75	0.134216	100	0.000165598	100	0.00111855	100	0.00170517

Accuracy increases as the error function  $[0.5*\sum(ti-oi)^2]$  decreases monotonically. Except for a small blip at epochs 4-5, the sum of deltas also decreases monotonically. Performance (%correct) reaches 100% after 5 epochs for t2, 8 epochs for t1, and 6 epochs for training data. Running the command repeatedly without deleting glickman.net is the equivalent of doubling, tripling, etc., the number of epochs. After 75 more:

```
75    0.071603 100    0.0000592 100    0.000787 100    0.001213
```

After 75 more:

```
75    0.0528632 100    0.000036347 100    0.000600356 100    0.000927038
```

The target vectors are specified in the program imagenet.c lines 39-43:

```
if (!strcmp(userid, "glickman")) {
    net->target[1] = TARGET_HIGH; /* it's me, set target to HIGH */
} else {
    net->target[1] = TARGET_LOW; /* not me, set it to LOW */
}
```

So the new code would be:

```
if (!strcmp(eyes, "sunglasses")) {
    net->target[1] = TARGET_HIGH;
} else {
    net->target[1] = TARGET_LOW;
}
```

3. Train a network using the default learning parameters (learning rate 0.3, momentum 0.3) for 75 epochs, with the following command:

```
./facetrain -n shades.net -t straightrnd_train.list -l straightrnd_test1.list -
2 straightrnd_test2.list -e 75
```

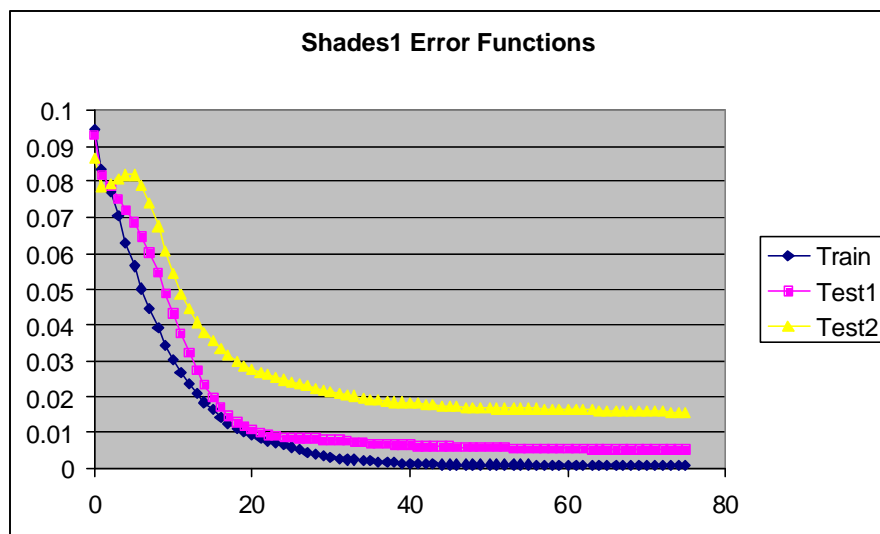
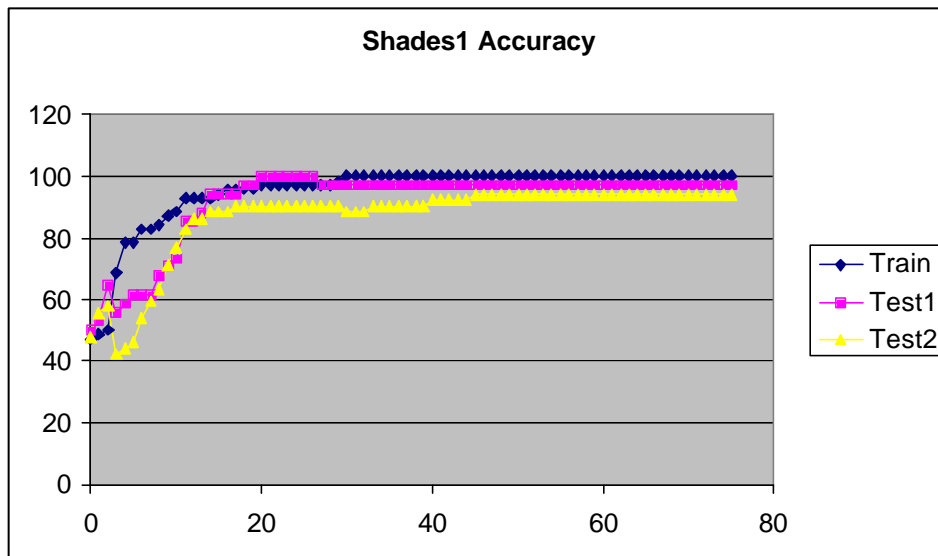
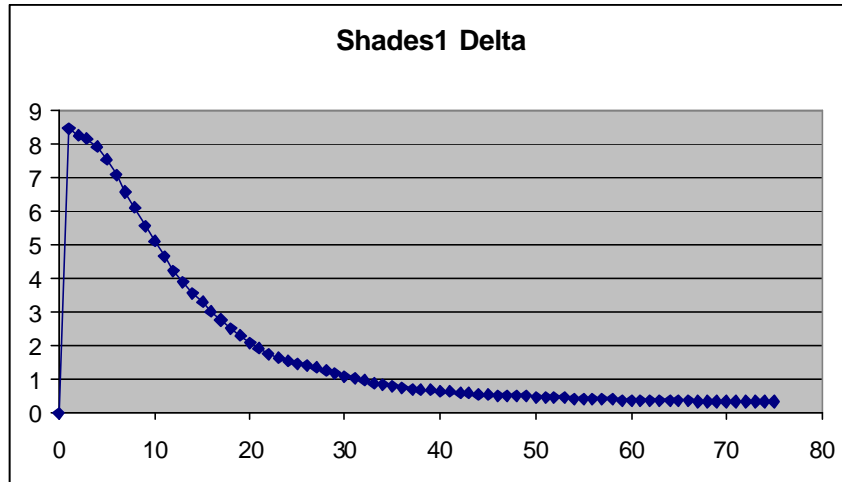
Here is the output.

```
0 0.0      47.1429 0.0943142 50      0.093161 48.0769 0.0863834
1 8.44628 48.5714 0.0832316 52.9412 0.0820533 55.7692 0.0786447
2 8.23015 50      0.0773952 64.7059 0.0787198 57.6923 0.0794553
3 8.14256 68.5714 0.0704867 55.8824 0.0753006 42.3077 0.0808639
4 7.91348 78.5714 0.0632293 58.8235 0.0719708 44.2308 0.0820357
5 7.53522 78.5714 0.0564554 61.7647 0.0685959 46.1538 0.0817391
6 7.07717 82.8571 0.0502622 61.7647 0.0647595 53.8462 0.0791772
7 6.58965 82.8571 0.0445277 61.7647 0.0601591 59.6154 0.07429
8 6.09154 84.2857 0.0392033 67.6471 0.05477   63.4615 0.0677212
9 5.58609 87.1429 0.0343906 70.5882 0.0489123 71.1538 0.0606377
10 5.08724 88.5714 0.0302399 73.5294 0.0430668 76.9231 0.0541998
11 4.62604 92.8571 0.0267367 85.2941 0.0375139 82.6923 0.0488674
12 4.22376 92.8571 0.0237176 85.2941 0.0323313 86.5385 0.0444951
13 3.87473 92.8571 0.0210263 88.2353 0.0275969 86.5385 0.0408728
14 3.57224 92.8571 0.0185766 94.1176 0.0234239 88.4615 0.0379115
15 3.29859 94.2857 0.0163524 94.1176 0.0199    88.4615 0.0355251
16 3.03037 95.7143 0.0143875 94.1176 0.0170322 88.4615 0.0335125
17 2.75656 95.7143 0.0127136 94.1176 0.0147604 90.3846 0.0316925
```

18	2.50424	95.7143	0.011322	97.0588	0.0130169	90.3846	0.0300707
19	2.28627	95.7143	0.0101764	97.0588	0.011714	90.3846	0.0287059
20	2.09716	97.1429	0.00923507	100	0.0107499	90.3846	0.027608
21	1.92755	97.1429	0.008449	100	0.0100326	90.3846	0.026738
22	1.77489	97.1429	0.00776244	100	0.00948975	90.3846	0.0260191
23	1.6449	97.1429	0.00712845	100	0.00907697	90.3846	0.0253764
24	1.54235	97.1429	0.00651256	100	0.00877309	90.3846	0.0247663
25	1.46493	97.1429	0.00588774	100	0.00856594	90.3846	0.0241709
26	1.40369	97.1429	0.00524172	100	0.00843957	90.3846	0.0235853
27	1.34536	97.1429	0.00459652	97.0588	0.00836417	90.3846	0.0230127
28	1.27417	97.1429	0.00400587	97.0588	0.00829419	90.3846	0.0224585
29	1.19195	98.5714	0.00350944	97.0588	0.00818808	90.3846	0.0219233
30	1.1071	100	0.00310684	97.0588	0.0080346	88.4615	0.0214073
31	1.02783	100	0.00277985	97.0588	0.00784922	88.4615	0.0209169
32	0.957725	100	0.00251168	97.0588	0.00765284	88.4615	0.0204602
33	0.896456	100	0.0022899	97.0588	0.00746032	90.3846	0.0200429
34	0.843003	100	0.002105	97.0588	0.0072795	90.3846	0.0196667
35	0.796521	100	0.0019495	97.0588	0.00711347	90.3846	0.0193303
36	0.75597	100	0.00181746	97.0588	0.00696271	90.3846	0.0190306
37	0.720196	100	0.00170422	97.0588	0.00682645	90.3846	0.0187636
38	0.688272	100	0.00160615	97.0588	0.00670344	90.3846	0.0185256
39	0.659791	100	0.00152043	97.0588	0.00659229	90.3846	0.0183128
40	0.633894	100	0.00144486	97.0588	0.00649164	92.3077	0.0181221
41	0.610393	100	0.00137771	97.0588	0.00640025	92.3077	0.0179505
42	0.589284	100	0.00131764	97.0588	0.00631701	92.3077	0.0177957
43	0.5703	100	0.00126354	97.0588	0.00624092	92.3077	0.0176554
44	0.553013	100	0.00121455	97.0588	0.00617111	92.3077	0.0175278
45	0.537013	100	0.00116996	97.0588	0.0061068	94.2308	0.0174113
46	0.522117	100	0.00112919	97.0588	0.00604732	94.2308	0.0173046
...							
74	0.322981	100	0.00062731	97.0588	0.00508542	94.2308	0.0158006
75	0.319317	100	0.000619131	97.0588	0.00505819	94.2308	0.0157641

Here the initial (chance) accuracy is ~50% for the binary variable. The deltas and training data accuracy and error functions behave monotonically like glickman's, but the accuracy takes longer to reach 100% (30 epochs).

The test data performance, however, is very different. Test accuracy rises rapidly to 65% and 58% by epoch 3, then drops to 56% and 42% at epoch 4 and begins a slower increase. Test1 achieves 100% before training at epoch 20, then drops to 97% at epoch 27 and stays there. Test2 rises to 90% at epoch 17 and stays there until it drops to 88% at epoch 30, returns to 90% at epoch 33, rises to 92% at epoch 40 then settles at 94% on epoch 45. The test2 error functions also show biphasic behavior (insert Excel graphs here).



4. What code did you modify?  
 imagenet.c line 39 in load\_target  
 What was the max training set accuracy?  
 100% @ 30 epochs  
 Validation set?  
 100% @ 20 epochs, 97% final  
 Test set?  
 94% @ 45 epochs

5-6. Now implement a 1-of-20 face recognizer that accepts an image as input and outputs the userid. Hint: use 20 hidden units. Later in the doc it also says to change the routines for performance\_on\_imagelist and evaluate\_performance in facetrain.c. Use the straighteven data file series which distributes evenly the 7-8 images of each userid, and 100 epochs.

7. Which parts of the code did you modify?

```
facetrain.c line 135 -- change number of hidden and output units
net = bpnn_create(imgsize, 4, 1); // given
net = bpnn_create(imgsize, 1, 20); // trial 1
net = bpnn_create(imgsize, 4, 20); // trial 2
net = bpnn_create(imgsize, 5, 20); // trial 3
net = bpnn_create(imgsize, 20, 20); // trial 4
net = bpnn_create(imgsize, 40, 20); // trial 5
```

```
facetrain.c -- evaluate_performance routine
for (i=1; i<21; i++) { // loop thru 20 output units
  delta = net->target[i] - net->output_units[i]; // 1 --> i
  ...etc...
```

```
imagenet.c line 39 -- loop thru 20 output units:
for (i=0; i<20; i++) net->target[i+1] = TARGET_LOW;
if (!strcmp(userid, "an2i")) {
  net->target[1] = TARGET_HIGH; // 10000000000000000000
} // but 1=0.9 and 0=0.1
else if (!strcmp(userid, "at33")) {
  net->target[2] = TARGET_HIGH; // 01000000000000000000
}
else if (!strcmp(userid, "boland")) {
  net->target[3] = TARGET_HIGH; // 00100000000000000000
...etc...
```

	# hidden units	1	2	4	5	20	40	80
	time (sec)	2	3	5	5	17	45	90
dataset	N	max%@epoch#						
train	80	95.0 0	95.0 0	95.0 1	100 55	100 14	100 18	100 10
valid	36	94.4 0	94.4 0	94.4 1	94.4 0	97.2 41	97.2 14	100 29
test	40	95.0 0	95.0 0	95.0 1	97.5 54	97.5 16	97.5 15	100 12

note: 5 bits needed to encode 20 words

See graphs at end of report.

8. using 960/20/20 ANN

Failed to classify the following images from the test set 1:  
 an2i\_straight\_neutral\_open\_4.pgm - outputs 0.415 0.093 0.050 0.094 0.067 0.079

0.100 0.062 0.182 0.049 0.114 0.052 0.079 0.072 0.012 0.050 0.022 0.284 0.060  
0.053

Failed to classify the following images from the test set 2:

an2i\_straight\_happy\_open\_4.pgm - outputs 0.384 0.051 0.090 0.111 0.093 0.062  
0.131 0.020 0.203 0.028 0.109 0.039 0.064 0.073 0.019 0.019 0.057 0.169 0.138  
0.095

These images seem to be a little smaller (greater distance from camera) and therefore fuzzier than the equivalent "angry" and "happy" images. Also an2i has bangs which obscure her eyebrows and appears to be Asian (smaller eyes), making her face perhaps more challenging for the ANN.

9-10. Implement a pose recognizer with 6 hidden units. Use all images and 100 epochs.

imagenet.c line 39

```
for (i=0; i<4; i++) net->target[i+1] = TARGET_LOW;
if (!strcmp(head, "straight")) {
    net->target[1] = TARGET_HIGH; // 1000 = 0.9 0.1 0.1 0.1
}
else if (!strcmp(head, "left")) {
    net->target[2] = TARGET_HIGH; // 0100 = 0.1 0.9 0.1 0.1
}
else if (!strcmp(head, "right")) {
    net->target[3] = TARGET_HIGH; // 0010 = 0.1 0.1 0.9 0.1
}
else if (!strcmp(head, "up")) {
    net->target[4] = TARGET_HIGH; // 0001 = 0.1 0.1 0.1 0.9
}
```

facetrain.c line 135:

```
net = bpnn_create(imgsize, 6, 4);
```

line 265:

```
for (i=1; i<5; i++) {
```

11. 960x6x4 -- 25 seconds

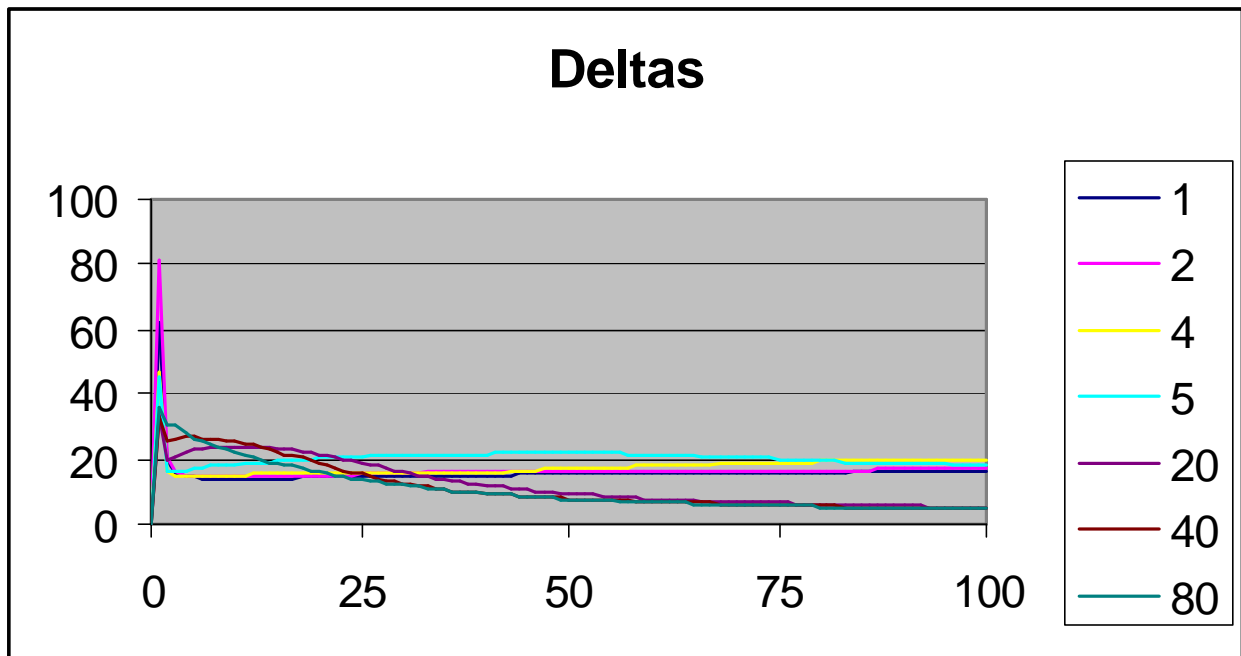
dataset	N	max accuracy	@epoch#
train	277	100 %	36
valid	139	94.24%	48 final 93.53% @ 87
test	208	94.71%	26 final 94.23% @ 90

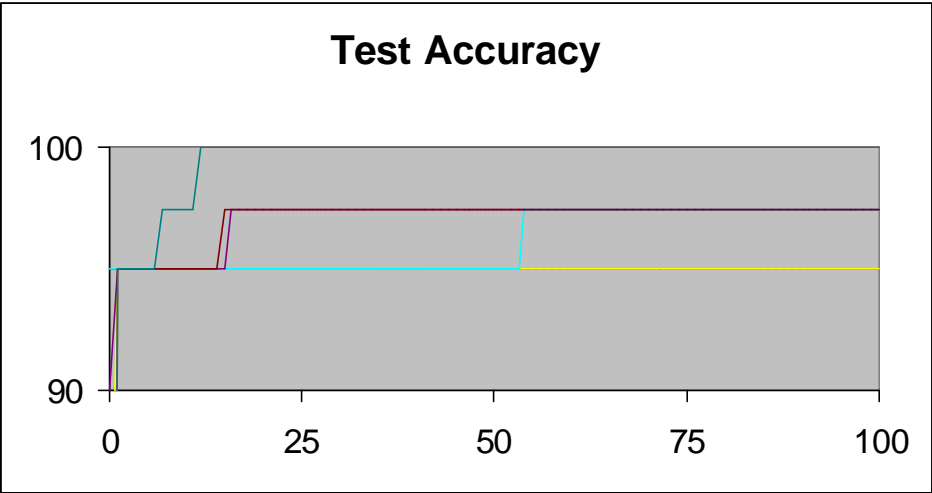
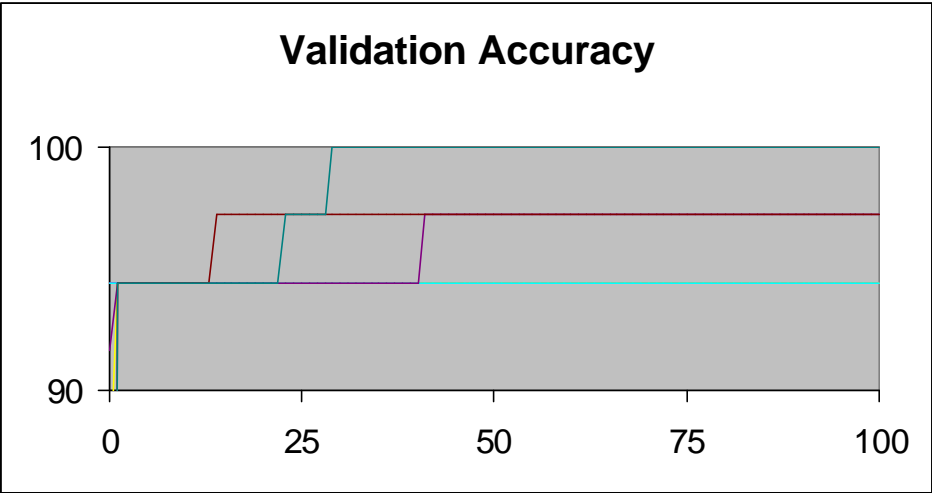
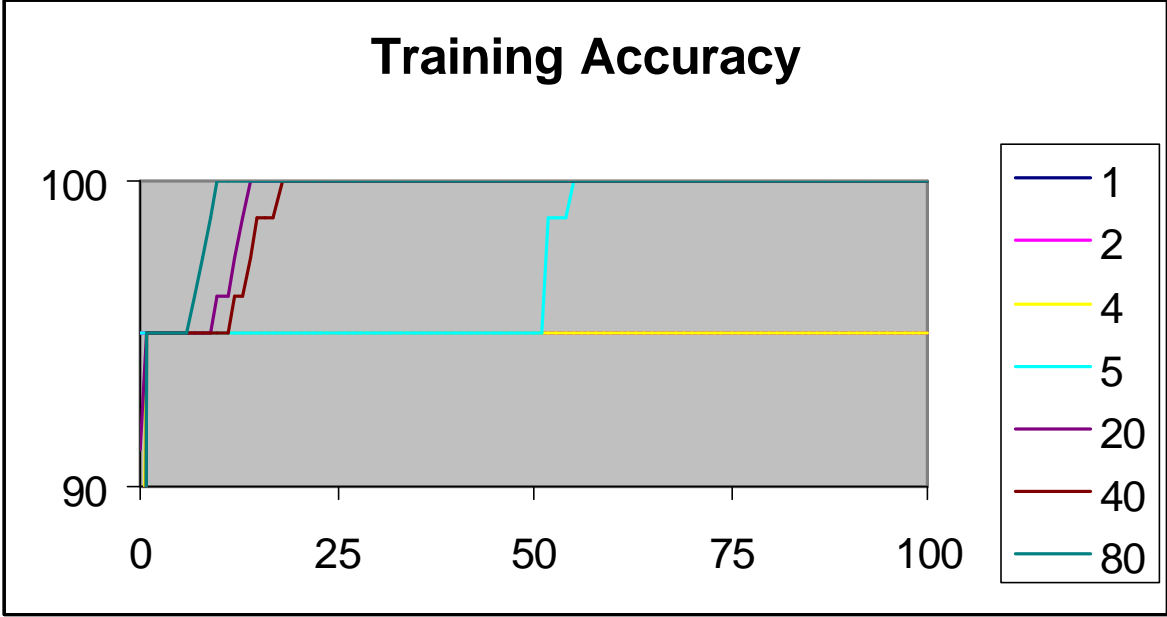
13. Yes, there is more variety (positive and negative) in the weighting of the pixels in the face, head, and throat regions.

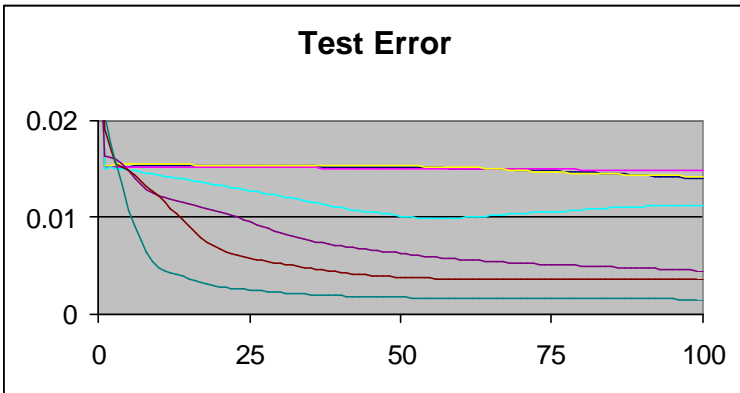
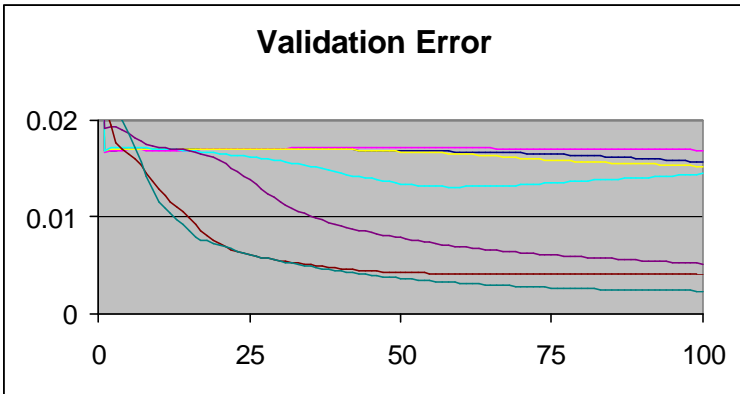
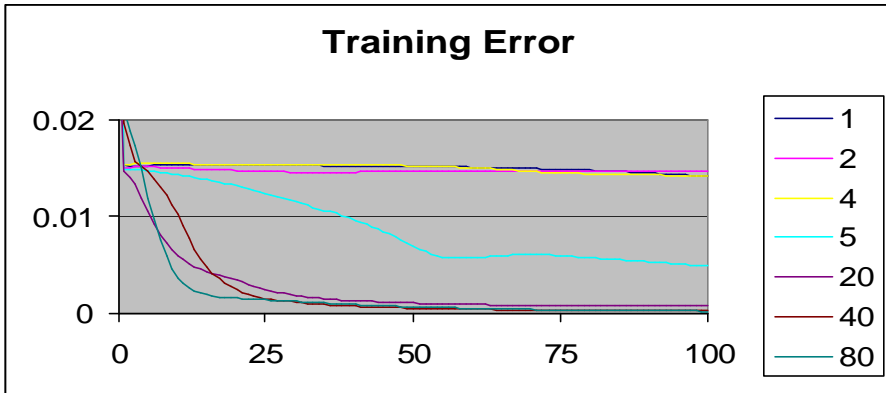
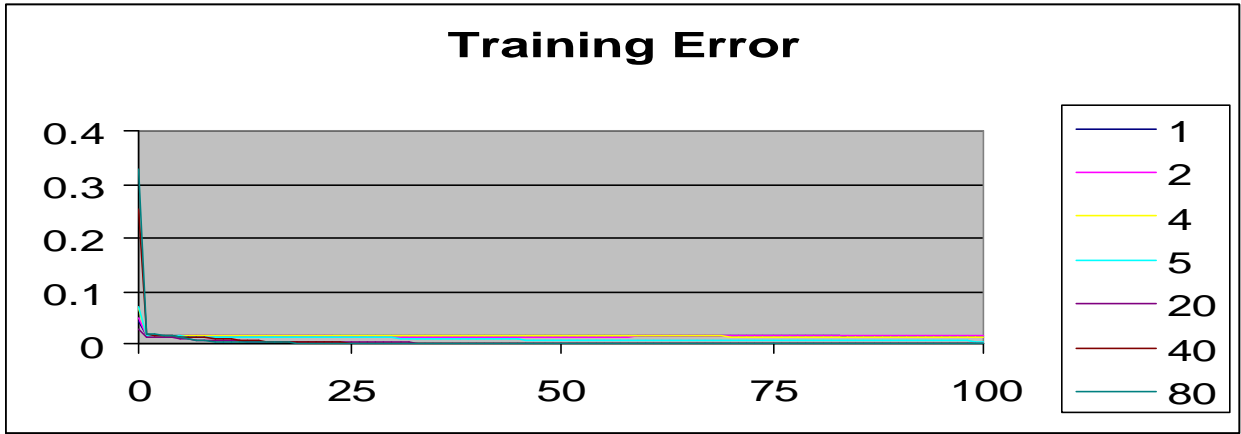
Units 1-4 are bilaterally asymmetric, perhaps corresponding to the "left" and "right" head positions. There is a consistent white spot in the throat area in all units. Unit 6 has a dark spot where the person's right eye should be; it looks like a sunglasses monocle.

Note: printing the weight graphics (page 10) on paper may give better resolution than the screen image.

Face recognizer data (legend = # of hidden units)

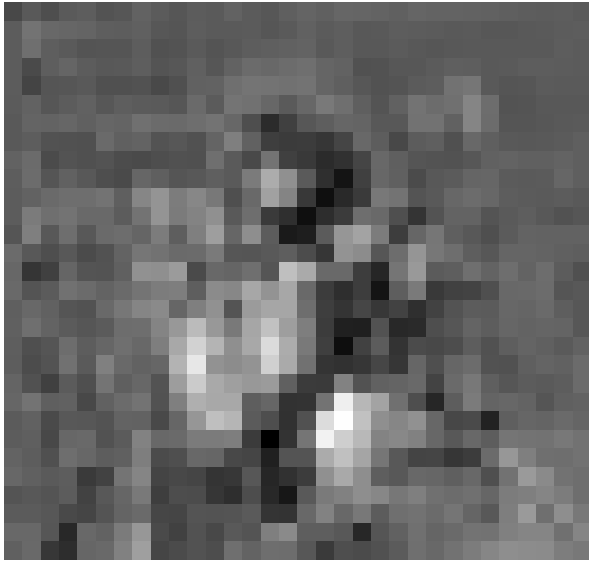




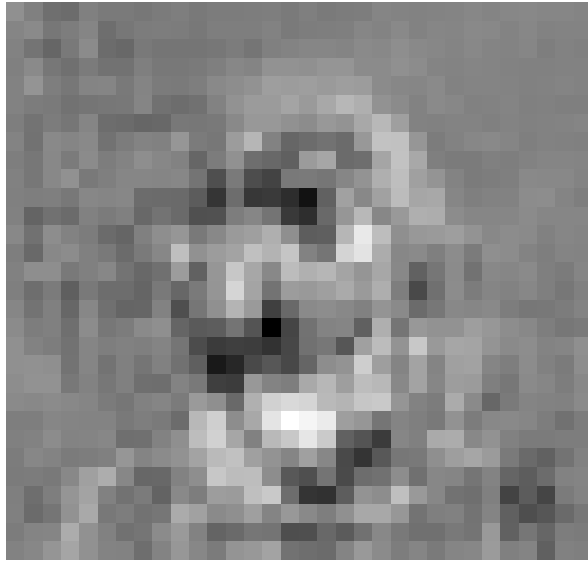


12-13.

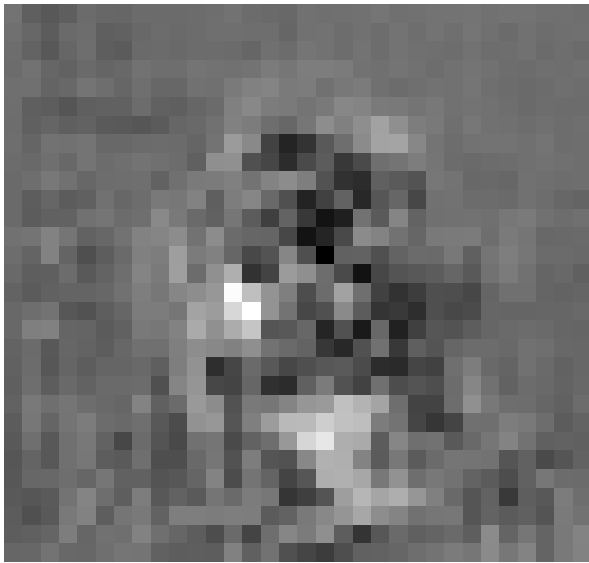
1



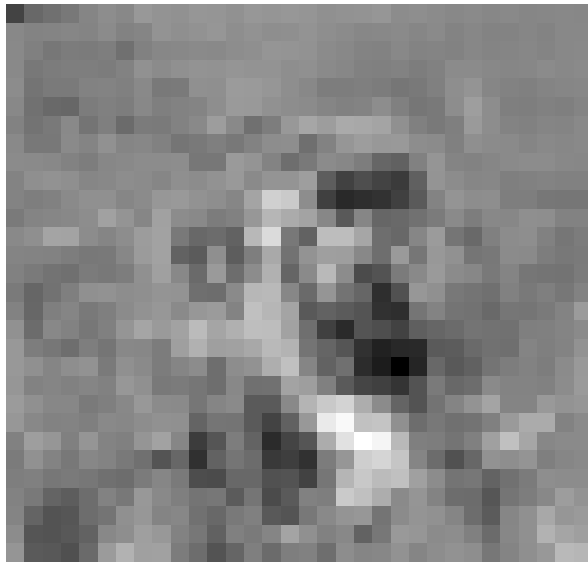
2



3



4



5



6

