CS 543 HW5

Atul Nambudiri (nambudi2)
Collaborators: smishra6, elwu2, pandoh2

April 28, 2017

1 Face Recognition (50%)

a Eigenfaces

(a) This is the pseduocode for the eigenfaces algorithm. It calculates the W_{pca} vector and finds the nearest neighbors for each image

Algorithm 1 Eigenfaces

procedure EIGENFACES(D, SUBSETS)
  $[im, person, number, subset] \leftarrow$ **readFaceImages**$('faces')$
  $im \leftarrow$ **subtractPerImageMeanAndDivideBySTD**(im)
  $im\_training\_subset \leftarrow$ **imagesInSubset**$(im, subset)$
  $X, \mu \leftarrow$ **subtractOverallMean**$(im\_training\_subset)$
  $[U, D] \leftarrow$ **eig**$(X' * X)$
  $U \leftarrow$ **sortedEigenvectors**(U)
  $V \leftarrow X * U$
  $W_{pca} \leftarrow V(:, 1:d)$
  $W_{pca} \leftarrow$ **normalize**(W_{pca})
  $training\_projections \leftarrow W'_{pca} * im\_training\_subset$
  $image\_projections \leftarrow W'_{pca} * im$
  $Matches \leftarrow$ **nearestNeighbor**$(image\_projections, training\_projections)$

The accuracy is shown in a table in the Fisherfaces section.

(b) These are the top 9 eigenfaces for different training subsets

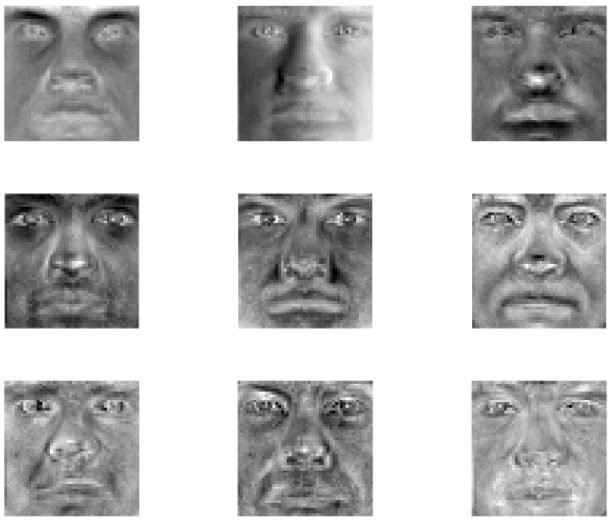

Figure 1: The top 9 eigenfaces from training on subset 1

Figure 2: The top 9 eigenfaces from training on subsets 1 and 5

(c) These are the reconstructions for various values of d and various training subsets.

Figure 3: The original and reconstruction images for the same person across different subsets using d = 9 and training on subset 1

Figure 4: The original and reconstruction images for the same person across different subsets using d = 9 and training on subsets 1 and 5

Figure 5: The original and reconstruction images for the same person across different subsets using d = 30 and training on subset 1

Figure 6: The original and reconstruction images for the same person across different subsets using d = 30 and training on subsets 1-5

b Fisherfaces

(a) This is the pseduocode for the fisherfaces algorithm. It calculates the W_{opt} vector and finds the nearest neighbors for each image

Algorithm 2 Fisherfaces

procedure FISHERFACES(C, SUBSETS)
  $[im, person, number, subset] \leftarrow$ **readFaceImages**$('faces')$
  $im \leftarrow$ **subtractPerImageMeanAndDivideBySTD**(im)
  $im\_training\_subset \leftarrow$ **imagesInSubset**$(im, subset)$
  $X, \mu \leftarrow$ **subtractOverallMean**$(im\_training\_subset)$
  $[U, D] \leftarrow$ **eig**$(X' * X)$
  $U \leftarrow$ **sortedEigenvectors**(U)
  $V \leftarrow X * U$
  $d \leftarrow$ **size**$(X, 1)$
  $W_{pca} \leftarrow V(:, 1 : d)$
  $pca\_projections \leftarrow W'_{opt} * im\_training\_subset$
  $\mu_i s \leftarrow mean(pca\_projections)$
  $Si \leftarrow$ **sum**(**sum**(**pow**$(X_{ki} - \mu_i)$)))
  $Sw \leftarrow$ **sum**(S_i)
  $Si \leftarrow$ **sum**$(N_i *$ **sum**(**pow**$(\mu_i - \mu)$)))
  $[U, D] \leftarrow$ **eig**(S_b, S_w)
  $U \leftarrow$ **sortedEigenvectors**(U)
  $W_{fld} \leftarrow X * U$
  $W_{opt} \leftarrow W_{pca} * W_{fld}$
  $W_{opt} \leftarrow$ **normalize**(W_{opt})
  $W_{opt} \leftarrow W_{opt}(:, 1 : c - 1)$
  $training\_projections \leftarrow W'_{opt} * im\_training\_subset$
  $image\_projections \leftarrow W'_{opt} * im$
  $Matches \leftarrow$ **nearestNeighbor**$(image\_projections, training\_projections)$

(b) Errors

Method (train set)	**Subset 1**	**Subset 2**	**Subset 3**	**Subset 4**	**Subset 5**
PCA d = 9 (S1)	0	0	0.2250	0.6643	0.8579
PCA d = 9 (S1+5)	0	0.1667	0.7250	0.6929	0
PCA d = 30 (S1)	0	0	0.0417	0.5643	0.7737
PCA d = 30 (S1+5)	0	0	0.3417	0.2857	0
FLD c = 10 (S1)	0	0	0.0167	0.4429	0.8684
FLD c = 10 (S1+5)	0	0	0	0.0143	0
FLD c = 31 (S1)	0	0	0	0.4000	0.8368
FLD c = 31 (S1+5)	0	0	0	0.0214	0

2 Object Categorization (50%)

1. Accuracy
   I achieved an overall best accuracy of 0.52060

2. Network Architecture
   This was my network architecture. This architecture is partially based on ConvPool-CNN-C, from this paper: `https://arxiv.org/pdf/1412.6806.pdf`. I tweaked this and set some parameters based on my own previous testing. This is the architecture that I used:

Layer	**Type**	**Input**	**Filter**	**Stride**	**Pad**	**Output**	**dropout/leak**
Layer 1	Conv	32x32x3	3x3x3x96	1	1	32x32x96	-
Layer 2	Relu	32x32x96	-	1	0	32x32x96	0.3
Layer 3	Conv	32x32x96	3x3x96x96	1	1	32x32x96	-
Layer 4	Relu	32x32x96	-	1	0	32x32x96	0.3
Layer 5	Conv	32x32x96	3x3x96x96	1	1	32x32x96	-
Layer 6	Relu	32x32x96	-	1	0	32x32x96	0.3
Layer 7	Dropout	32x32x96	-	1	0	32x32x96	0.2
Layer 8	Pool(Max)	32x32x96	3x3	2	[1 0 1 0]	16x16x96	-
Layer 9	Conv	16x16x96	3x3x96x192	1	1	16x16x192	-
Layer 10	Relu	16x16x192	-	1	0	16x16x192	0.3
Layer 11	Conv	16x16x192	3x3x192x192	1	1	16x16x192	-
Layer 12	Relu	16x16x192	-	1	0	16x16x192	0.3
Layer 13	Conv	16x16x192	3x3x192x192	1	1	16x16x192	-
Layer 14	Relu	16x16x192	-	1	0	16x16x192	0.3
Layer 15	Dropout	16x16x192	-	1	0	16x16x192	0.2
Layer 16	Pool(Max)	16x16x192	3x3	2	0	6x6x192	-
Layer 17	Conv	6x6x192	3x3x192x384	1	1	6x6x384	-
Layer 18	Relu	6x6x384	-	1	0	6x6x384	0.3
Layer 19	Conv	6x6x384	3x3x384x384	1	1	6x6x384	-
Layer 20	Relu	6x6x384	-	1	0	6x6x384	0.3
Layer 21	Conv	6x6x384	3x3x384x384	1	1	6x6x384	-
Layer 22	Relu	6x6x384	-	1	0	6x6x384	0.3
Layer 23	Dropout	6x6x384	-	1	0	6x6x384	0.4
Layer 24	Pool(Avg)	6x6x384	6x6	2	0	1x1x384	-
Layer 25	Softmax	1x1x384	-	1	0	1x100	-

The model I referenced had a similar pattern of $(conv - relu)^3 - pool$. In my previous tests, I found that stacking multiple layers of conv filters together seemed to have a generally positive effect on accuracy. This is partially why I used this model as a basis.

I tried out $(conv - relu)^2 - pool$ as well. This had decent accuracy, but I found I was able to increase it by adding in the additional conv-relu layer.

In addition, I also found that using larger filters generally decreased my accuracy. If padding is not included, larger and larger filters decrease the size of the image each time, which decreases the overall number of layers that it is possible to include. In previous tests, I was able to get an accuracy of around 42% using larger filters, but not much more. By using smaller filters, I was able to get increased accuracy, and was able to have a deeper CNN. I found that having deeper CNNs typically increased the accuracy, as long as the filters were relatively small. When I used larger filters, I had to use more padding, which seemed to decrease accuracy somewhat

I included dropout layers before every pool step. I found that this helped somewhat with over fitting, and also increased my accuracy by around a percent. I tried experimenting with putting the dropout layers before and after the Pool layers, but there wasn't a big difference in either one. I kept by dropout below 0.5. I saw a few other papers that used this figure, but when I tried it out, I found it had negative effects, by around 4% points.

The paper I referenced had layers 17-24 have 192 filters. I upped this to 384. I found that this had a positive effect on accuracy, by about 2%. In previous tests, increasing the number of filters in the early layers helped to a certain extent, but if I increased it too much, the accuracy started falling again. I didn't have time to test this out on this particular architecture, but I surmise that the effects would be similar.

My Relu layers were slightly leaky, with a leak of 0.3. I found that having a small amount of leak improved my implementation, by around 1

I did not use an batch pre-processing in my implementation. I found that both randomly flipping, and randomly rotating my image decreased my accuracy by around 5-10%. I also tried converting my images to HSV. This had an even worse effect, and decreased my accuracy by almost 20%.

3. I used a batch-size of 128. I found that there wasn't a big difference between 100 and 128. 128 was slightly faster. 250 was faster still, but the accuracy dropped significantly when I tried it. The same is true for a batch size of 50, although the drop wasn't as severe.

   I used the default learning rate of $[0.001 * ones(1, 10)0.0001 * ones(1, 15)]$ I found that increasing any of the parameters reduced the accuracy significantly.

   I used 250 epochs. I found that there was only a small improvement in accuracy going from from 150 to 250, around .002. The slope of the error was still negative, so it was still improving, just too slowly to make a real difference. In fact, anything past 110 didn't show any significant improvement.

   In a previous CNN architecture, I found the accuracy plateaued at around 25-20 epochs, at around 39%. It even started to go down when I increased it past 40. This does not seem to be the case for my current architecture.

4. This is my plot of training/validation error vs training iteration/epoch

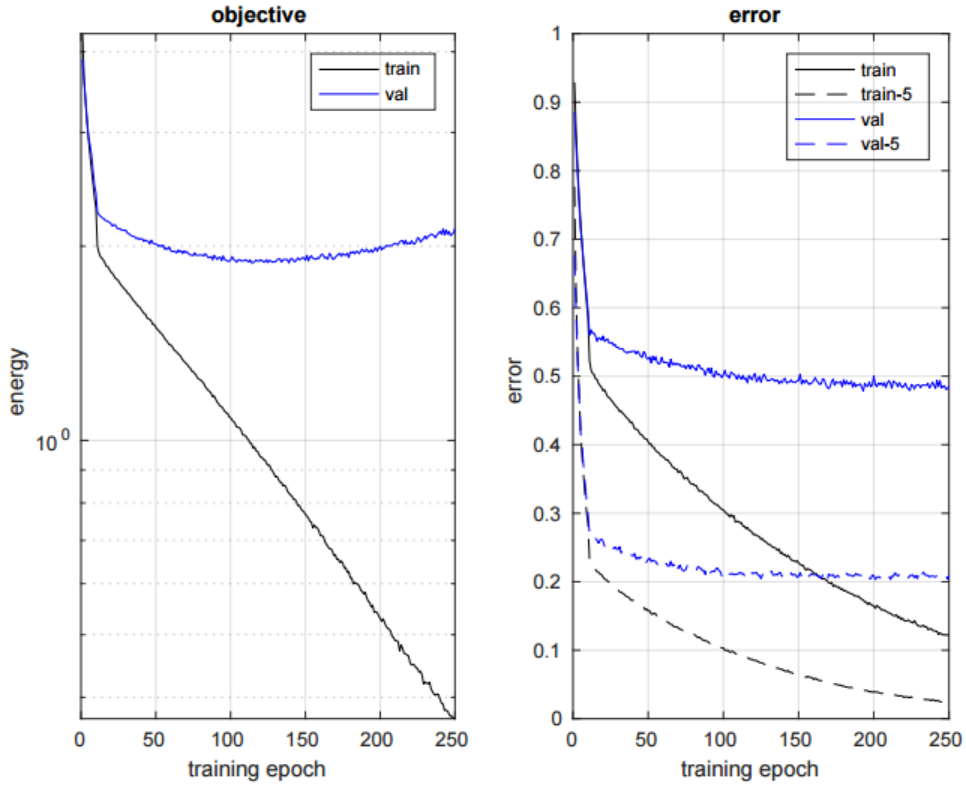

Figure 7: Plot of training/validation error vs training iteration/epoch

5. This is my confusion matrix

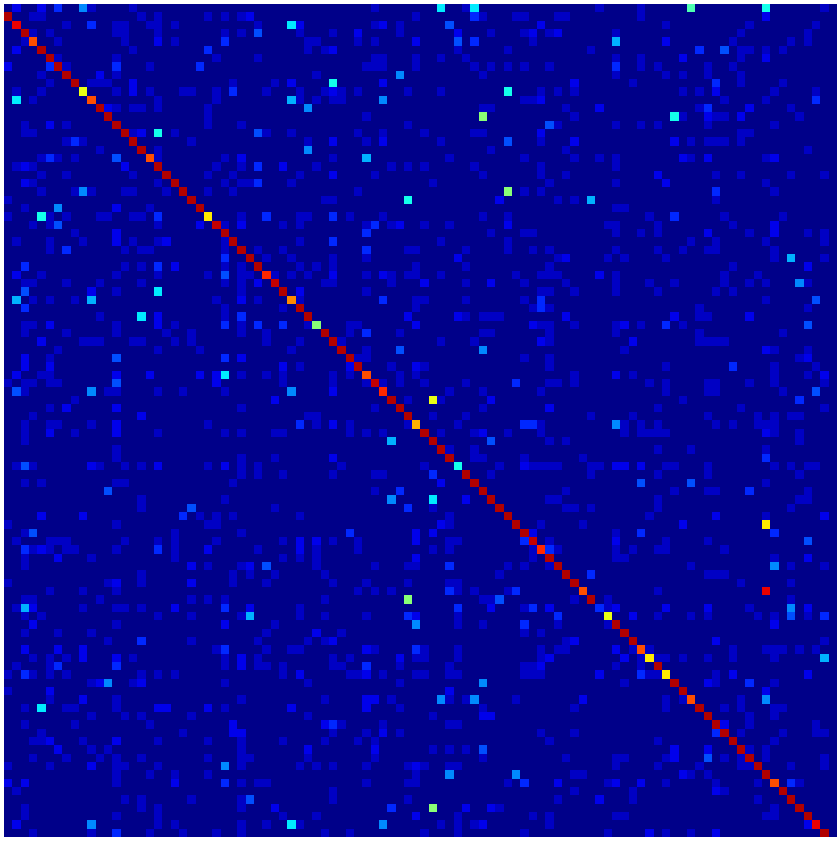

Figure 8: Confusion Matrix

There doesn't appear to be too much confusion between any two classes. The confusion seems to be realtivly spread out among the different classes.