**Assignment 3**

**Overview:**

We were given some tried and tested neural network software, written by Jeff Shufelt and made available at Carnegie Mellon University School of Computer Science. We needed to modify the code and answer some questions.

**Sunglasses:**

To modify the neural network to no longer recognize Glickman and now recognize sunglasses was as simple as changing the load\_target function to check the eyes feature for “sunglasses”.

**imagenet.c**

|  |  |
| --- | --- |
| **Glickman** | **Sunglasses** |
| load\_target(img, net)  IMAGE \*img;  BPNN \*net;  {  int scale;  char userid[40], head[40], expression[40], eyes[40], photo[40];  userid[0] = head[0] = expression[0] = eyes[0] = photo[0] = '\0';  /\*\*\* scan in the image features \*\*\*/  sscanf(NAME(img), "%[^\_]\_%[^\_]\_%[^\_]\_%[^\_]\_%d.%[^\_]",  userid, head, expression, eyes, &scale, photo);  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 \*/  }  } | load\_target(img, net)  IMAGE \*img;  BPNN \*net;  {  int scale;  char userid[40], head[40], expression[40], eyes[40], photo[40];  userid[0] = head[0] = expression[0] = eyes[0] = photo[0] = '\0';  /\*\*\* scan in the image features \*\*\*/  sscanf(NAME(img), "%[^\_]\_%[^\_]\_%[^\_]\_%[^\_]\_%d.%[^\_]",  userid, head, expression, eyes, &scale, photo);  if (!strcmp(eyes, "sunglasses")) {  net->target[1] = TARGET\_HIGH; /\* it's me, set target to HIGH \*/  } else {  net->target[1] = TARGET\_LOW; /\* not me, set it to LOW \*/  }  } |

|  |  |  |
| --- | --- | --- |
|  | **Max Accuracy** | **Epoch** |
| **Training Set** | 100 | 30 |
| **Validation Set** | 97.0588 | 27 |
| **Test Set** | 94.2308 | 45 |

**Pose:**

To modify the neural network to recognize each of the four poses, additional targets were added, for a total of four. The number of output units was also changed to four. This meant that the performance evaluation function needed to sum the total differences for delta and determine which target was on and whether it recognized poses successfully.

**imagenet.c**

|  |  |
| --- | --- |
| **Glickman** | **Poses** |
| load\_target(img, net)  IMAGE \*img;  BPNN \*net;  {  int scale;  char userid[40], head[40], expression[40], eyes[40], photo[40];  userid[0] = head[0] = expression[0] = eyes[0] = photo[0] = '\0';  /\*\*\* scan in the image features \*\*\*/  sscanf(NAME(img), "%[^\_]\_%[^\_]\_%[^\_]\_%[^\_]\_%d.%[^\_]",  userid, head, expression, eyes, &scale, photo);  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 \*/  }  } | load\_target(img, net)  IMAGE \*img;  BPNN \*net;  {  int scale;  char userid[40], head[40], expression[40], eyes[40], photo[40];  userid[0] = head[0] = expression[0] = eyes[0] = photo[0] = '\0';  /\*\*\* scan in the image features \*\*\*/  sscanf(NAME(img), "%[^\_]\_%[^\_]\_%[^\_]\_%[^\_]\_%d.%[^\_]",  userid, head, expression, eyes, &scale, photo);  if (!strcmp(head, "straight")) {  net->target[1] = TARGET\_HIGH; /\* it's straight, set target to HIGH \*/  net->target[2] = TARGET\_LOW; /\* not left, set it to LOW \*/  net->target[3] = TARGET\_LOW; /\* not right, set it to LOW \*/  net->target[4] = TARGET\_LOW; /\* not up, set it to LOW \*/  } else if (!strcmp(head, "left")) {  net->target[1] = TARGET\_LOW; /\* not straight, set it to LOW \*/  net->target[2] = TARGET\_HIGH; /\* it's left, set target to HIGH \*/  net->target[3] = TARGET\_LOW; /\* not right, set it to LOW \*/  net->target[4] = TARGET\_LOW; /\* not up, set it to LOW \*/  } else if (!strcmp(head, "right")) {  net->target[1] = TARGET\_LOW; /\* not straight, set it to LOW \*/  net->target[2] = TARGET\_LOW; /\* not left, set it to LOW \*/  net->target[3] = TARGET\_HIGH; /\* it's right, set target to HIGH \*/  net->target[4] = TARGET\_LOW; /\* not up, set it to LOW \*/  } else {  net->target[1] = TARGET\_LOW; /\* not straight, set it to LOW \*/  net->target[2] = TARGET\_LOW; /\* not left, set it to LOW \*/  net->target[3] = TARGET\_LOW; /\* not right, set it to LOW \*/  net->target[4] = TARGET\_HIGH; /\* it's up, set target to HIGH \*/  }  } |

**facetrain.c**

|  |  |
| --- | --- |
| **Glickman** | **Poses** |
| backprop\_face(trainlist, test1list, test2list, epochs, savedelta, netname, list\_errors)  IMAGELIST \*trainlist, \*test1list, \*test2list;  int epochs, savedelta, list\_errors;  char \*netname;  {  IMAGE \*iimg;  BPNN \*net;  int train\_n, epoch, i, imgsize;  double out\_err, hid\_err, sumerr;  train\_n = trainlist->n;  /\*\*\* Read network in if it exists, otherwise make one from scratch \*\*\*/  if ((net = bpnn\_read(netname)) == NULL) {  if (train\_n > 0) {  printf("Creating new network '%s'\n", netname);  iimg = trainlist->list[0];  imgsize = ROWS(iimg) \* COLS(iimg);  /\* bthom ===========================  make a net with:  imgsize inputs, 4 hiden units, and 1 output unit  \*/  net = bpnn\_create(imgsize, 4, 1); | backprop\_face(trainlist, test1list, test2list, epochs, savedelta, netname, list\_errors)  IMAGELIST \*trainlist, \*test1list, \*test2list;  int epochs, savedelta, list\_errors;  char \*netname;  {  IMAGE \*iimg;  BPNN \*net;  int train\_n, epoch, i, imgsize;  double out\_err, hid\_err, sumerr;  train\_n = trainlist->n;  /\*\*\* Read network in if it exists, otherwise make one from scratch \*\*\*/  if ((net = bpnn\_read(netname)) == NULL) {  if (train\_n > 0) {  printf("Creating new network '%s'\n", netname);  iimg = trainlist->list[0];  imgsize = ROWS(iimg) \* COLS(iimg);  /\* bthom ===========================  make a net with:  imgsize inputs, 6 hidden units, and 4 output units \*/  net = bpnn\_create(imgsize, 6, 4); |
| evaluate\_performance(net, err)  BPNN \*net;  double \*err;  {  double delta;  delta = net->target[1] - net->output\_units[1];  \*err = (0.5 \* delta \* delta);  /\*\*\* If the target unit is on... \*\*\*/  if (net->target[1] > 0.5) {  /\*\*\* If the output unit is on, then we correctly recognized me! \*\*\*/  if (net->output\_units[1] > 0.5) {  return (1);  /\*\*\* otherwise, we didn't think it was me... \*\*\*/  } else {  return (0);  }  /\*\*\* Else, the target unit is off... \*\*\*/  } else {  /\*\*\* If the output unit is on, then we mistakenly thought it was me \*\*\*/  if (net->output\_units[1] > 0.5) {  return (0);  /\*\*\* else, we correctly realized that it wasn't me \*\*\*/  } else {  return (1);  }  }  } | evaluate\_performance(net, err)  BPNN \*net;  double \*err;  {  double delta;  delta = net->target[1] - net->output\_units[1]  +net->target[2] - net->output\_units[2]  +net->target[3] - net->output\_units[3]  +net->target[4] - net->output\_units[4];  \*err = (0.5 \* delta \* delta);  /\*\*\* If the target unit is on... \*\*\*/  if (net->target[1] > 0.5) {  /\*\*\* If the output unit is on, then we correctly recognized straight! \*\*\*/  if (net->output\_units[1] > 0.5) {  return (1);  /\*\*\* otherwise, we didn't think it was straight... \*\*\*/  } else {  return (0);  }  } else if (net->target[2] > 0.5) {  /\*\*\* If the output unit is on, then we correctly recognized left! \*\*\*/  if (net->output\_units[2] > 0.5) {  return (1);  /\*\*\* else, we correctly realized that it wasn't left \*\*\*/  } else {  return (0);  }  } else if (net->target[3] > 0.5) {  /\*\*\* If the output unit is on, then we mistakenly thought it was right \*\*\*/  if (net->output\_units[3] > 0.5) {  return (1);  /\*\*\* else, we correctly realized that it wasn't right \*\*\*/  } else {  return (0);  }  } else {  /\*\*\* If the output unit is on, then we mistakenly thought it was up \*\*\*/  if (net->output\_units[4] > 0.5) {  return (1);  /\*\*\* else, we correctly realized that it wasn't up \*\*\*/  } else {  return (0);  }  }  } |

|  |  |  |
| --- | --- | --- |
|  | **Max Accuracy** | **Epoch** |
| **Training Set** | 100 | 77 |
| **Validation Set** | 92.8058 | 81 |
| **Test Set** | 96.1538 | 64 |

**Weights:**

A tool was provided to help visualize the distribution of weights with respect to each pixel. Since there were 6 layers for the pose net, 6 images were provided. You can see that in all but one, unit 5, there are definite boundaries and clustering of the weights.

     

Unit 1 Unit 2 Unit 3 Unit 4 Unit 5 Unit 6