EE 113DA: Digital Signal Processing Design

Mini-Project #1

Classify Handwritten Digits with Convolutional Neural Network (CNN)

Name: Khyle Calpe and Phillip Kwan

Lab Section: 1B

Objective:

The objective of this mini-project is to train a neural network in Python to recognize handwritten digits in the range of 0 to 9 and to implement the forward propagation portion of the CNN in CubeIDE for image recognition on the H7 board.

CNN Structure:

The CNN is composed of 5 main layers: 4 2D-convolutional layers and 1 dense (fully connected) layer. Max pooling layers are applied after every convolutional layer to reduce the number of parameters and complexity of calculations in subsequent layers. Our input layer is an 8 filter convolutional layer to pick out low level features such as straight lines. On every subsequent layer, we double the amount of filters to pick out increasingly complex features and combine the results in the fully connected layer. In total, we use 25,034 parameters with the largest number of parameters loaded concurrently into the H7 board being 18,496 parameters in the fourth convolutional layer.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 8)         80
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 8)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 16)        1168
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 16)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 32)          4640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 32)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 1, 1, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 10)                650
=================================================================
Total params: 25,034
Trainable params: 25,034
Non-trainable params: 0
```

Figure : CNN Structure

Design:

The testing phase of the CNN comprises three primary functions: convolution, pooling, and dense. Also, the code includes a flatten function to reshape a three-dimensional input to a one-dimensional output. The convolution function takes in a three-dimensional input and outputs to a pre-allocated three-dimensional output. Additionally, the parameters of the function include a one-dimensional kernel, one-dimensional bias, the shape of the input, and the depth of the kernel. For the purposes of simplifying the code, the function assumes a three-by-three kernel array. As for functionality, the function begins by zeroing out the output array. Then, based on same padding, the kernel sifts through the input array, dot products with the bounded values, adds the appropriate bias, and sets the result on the output array. The pooling function uses max pooling and takes in a three- dimensional input and outputs to a pre-allocated three-dimensional output. Additionally, the parameters of the function include the shape of the input and one of the dimensions of the pool. The function operates by determining the shape of the output based on the floored quotient of the input shape and the pool shape. Then, a pooling layer sifts through the input array based on the calculated output shape and outputs the maximum value in the bounded input layer onto the output array. the dense function takes in a one-dimensional input, one-dimensional kernel, one-dimensional bias, one-dimensional output, the input size, and the output size. The function operates by performing a matrix multiplication between the input and kernel, adding a bias, exponentiating each result, normalizing the values, then setting the results onto the output array as the predicted probabilities.

Code:

```
void conv2D(float*** input, float* kernel, float* bias, float*** output, int height, int width,
    int depth, int kernel_depth)
{
  // ONLY works for 3x3 filter
  // for each element in array, take surrounding elements in array equal to filter size
  // dot product with filter weights, assume same padding
  // apply relu on result

  int kernel_height = 3;
  int kernel_width = 3;
  int out_height = height;
  int out_width  = width;
  int out_depth  = kernel_depth;
  float input_val  = 0;
  float kernel_val = 0;
  float output_val = 0;

  for (int row = 0; row < out_height; row++)
  {
    for (int col = 0; col < out_width; col++)
    {
      for (int channel = 0; channel < out_depth; channel++)
      {
        output[row][col][channel] = 0;
      }
    }
  }
  for (int kernel_channel = 0; kernel_channel < kernel_depth; kernel_channel++)
  {
    for (int row = 0; row < height; row++)
    {

      for (int col = 0; col < width; col++)
      {
        for (int channel = 0; channel < depth; channel++)
        {
          for (int kernel_row = 0; kernel_row < kernel_height; kernel_row++)
          {
            // if far top or bottom, skip
            if (kernel_row+row-1 < 0)
              continue;
            if (kernel_row+row-1 >= height)
              continue;

            for (int kernel_col = 0; kernel_col < kernel_width; kernel_col++)
            {
              // if far left or right, skip
              if (kernel_col+col-1 < 0)
                continue;
              if (kernel_col+col-1 >= width)
                continue;

              input_val   = input[kernel_row+row-1][kernel_col+col-1][channel];
              kernel_val  = kernel[kernel_channel + kernel_depth*(channel + depth*(kernel_col + kernel_width*kernel_row))];
              output_val += (input_val * kernel_val);
            }
          }
        }
        output_val += bias[kernel_channel];
        // relu
        if (output_val < 0)
          output[row][col][kernel_channel] = 0;

        else
          output[row][col][kernel_channel] = output_val;
        // Reset output value
        output_val = 0;
      }
    }
  }
}
```

```
void pooling(float*** input, float*** output, int height, int width, int depth, int pool_size)
{
  // Round up max pool count based on input dimensions
  int max_count_pool_row = height/pool_size;
  int max_count_pool_col = width/pool_size;

  // Store max value element
  float input_value = 0;
  float max_value = 0;

  // Propagate through the arrays
  for(int channel = 0; channel < depth; channel++)
  {
    // Cycle through the input layer row by pool size
    for (int row_count = 0; row_count < max_count_pool_row; row_count++)
    {
      // Cycle through the input layer col by pool size
      for (int col_count = 0; col_count < max_count_pool_col; col_count++)
      {
        // Cycle through pool row
        for (int pool_row = row_count*pool_size;
            pool_row < height && pool_row < (row_count*pool_size+pool_size);
            pool_row++)
        {
          // Cycle through pool col
          for (int pool_col = col_count*pool_size;

              pool_col < width && pool_col < (col_count*pool_size+pool_size);
              pool_col++)
          {
            // Check for max value in a pool
            input_value = input[pool_row][pool_col][channel];

            if (input_value > max_value)
            {
              max_value = input_value;
            }
            input_value = 0;
          }
        }
        // Set output element to max value
        output[row_count][col_count][channel] = max_value;
        // Reset max value
        max_value = 0;
      }
    }
  }
}
```

```
146 void dense_S(float* input, float* kernel, float* bias, double output[], int input_size, int output_size)
147 {
148   // matrix multiplication between input and kernel
149   // add bias to the result
150   // calculate exp of the result
151   // divide each output exp by sum of exps to get probability
152   // should have 1 output for every image
153   float output_value = 0;
154   double output_sum = 0;
155
156   for (int row=0; row<output_size; row++)
157   {
158     for (int col=0; col<input_size; col++)
159     {
160       // dot product
161       // kernel transposed before flattened
162       output[row] += (kernel[col + row*input_size] * input[col]);
163     }
164     // add bias
165     output_value = output[row] + bias[row];
166     output[row] = (double) output_value;
167     output[row] = exp(output[row]);
168     output_sum += output[row];
169   }
170
171   // normalizing for probability
172   for (int i=0; i<output_size; i++)
173   {
174     output[i] /= output_sum;
175   }
176 }
```

Results:

The CNN functioned as expected and predicted every test case with a confidence of over 90% except for two cases. For those image files, the probability did not reach above 90%, but the CNN still predicted the appropriate digit. To improve the network even further, a better combination of weights and biases may be obtained from the training phase of the CNN.

#0

Probability of 0: 0.9999940351

Probability of 1: 0.0000000037

Probability of 2: 0.0000012573

Probability of 3: 0.0000000028

Probability of 4: 0.0000000030

Probability of 5: 0.0000000070

Probability of 6: 0.0000017245

Probability of 7: 0.0000000092

Probability of 8: 0.0000021776

Probability of 9: 0.0000007797

#1

Probability of 0: 0.0000000167

Probability of 1: 0.9999895212

Probability of 2: 0.0000007382

Probability of 3: 0.0000018433

Probability of 4: 0.0000000175

Probability of 5: 0.0000000353

Probability of 6: 0.0000000290

Probability of 7: 0.0000077520

Probability of 8: 0.0000000025

Probability of 9: 0.0000000443

#2

Probability of 0: 0.0000000000

Probability of 1: 0.0000000000

Probability of 2: 1.0000000000

Probability of 3: 0.0000000000

Probability of 4: 0.0000000000

Probability of 5: 0.0000000000

Probability of 6: 0.0000000000

Probability of 7: 0.0000000000

Probability of 8: 0.0000000000

Probability of 9: 0.0000000000

#3

Probability of 0: 0.0000000000

Probability of 1: 0.0000000000

Probability of 2: 0.0000000001

Probability of 3: 0.9999999550

Probability of 4: 0.0000000000

Probability of 5: 0.0000000003

Probability of 6: 0.0000000000

Probability of 7: 0.0000000005

Probability of 8: 0.0000000441

Probability of 9: 0.0000000000

Probability #4

Probability of 0: 0.0000000000

Probability of 1: 0.0000638106

Probability of 2: 0.0000001172

Probability of 3: 0.0000000000

Probability of 4: 0.9999339208

Probability of 5: 0.0000000002

Probability of 6: 0.0000000034

Probability of 7: 0.0000004686

Probability of 8: 0.0000000088

Probability of 9: 0.0000016704

#5

Probability of 0: 0.0000000002

Probability of 1: 0.0000000001

Probability of 2: 0.0000000005

Probability of 3: 0.0000060664

Probability of 4: 0.0000000033

Probability of 5: 0.9999925492

Probability of 6: 0.0000000000

Probability of 7: 0.0000000290

Probability of 8: 0.0000002311

Probability of 9: 0.0000011202

#6

Probability of 0: 0.0000002140

Probability of 1: 0.0000000000

Probability of 2: 0.0000000050

Probability of 3: 0.0000000000

Probability of 4: 0.0000000009

Probability of 5: 0.0000001221

Probability of 6: 0.9999772045

Probability of 7: 0.0000000000

Probability of 8: 0.0000224535

Probability of 9: 0.0000000000

#7

Probability of 0: 0.0000000000

Probability of 1: 0.0000122784

Probability of 2: 0.0000000945

Probability of 3: 0.0000014399

Probability of 4: 0.0000000846

Probability of 5: 0.0000000001

Probability of 6: 0.0000000000

Probability of 7: 0.9999858042

Probability of 8: 0.0000001501

Probability of 9: 0.0000001483

#8

Probability of 0: 0.0755620653

Probability of 1: 0.0018736267

Probability of 2: 0.0885219499

Probability of 3: 0.0014408828

Probability of 4: 0.0020278117

Probability of 5: 0.0111350573

Probability of 6: 0.0782000362

Probability of 7: 0.0003544639

Probability of 8: 0.7360262637

Probability of 9: 0.0048578426

#9

Probability of 0: 0.0000000004

Probability of 1: 0.0000006670

Probability of 2: 0.0000001174

Probability of 3: 0.0000000004

Probability of 4: 0.0229408738

Probability of 5: 0.0000000005

Probability of 6: 0.0000000022

Probability of 7: 0.0000133915

Probability of 8: 0.0000016600

Probability of 9: 0.9770432868

#0

Probability of 0: 0.9999940351

Probability of 1: 0.0000000037

Probability of 2: 0.0000012573

Probability of 3: 0.0000000028

Probability of 4: 0.0000000030

Probability of 5: 0.0000000070

Probability of 6: 0.0000017245

Probability of 7: 0.0000000092

Probability of 8: 0.0000021776

Probability of 9: 0.0000007797

#1

Probability of 0: 0.0000000167

Probability of 1: 0.9999895212

Probability of 2: 0.0000007382

Probability of 3: 0.0000018433

Probability of 4: 0.0000000175

Probability of 5: 0.0000000353

Probability of 6: 0.0000000290

Probability of 7: 0.0000077520

Probability of 8: 0.0000000025

Probability of 9: 0.0000000443

#2

Probability of 0: 0.0000000000

Probability of 1: 0.0000000000

Probability of 2: 1.0000000000

Probability of 3: 0.0000000000

Probability of 4: 0.0000000000

Probability of 5: 0.0000000000

Probability of 6: 0.0000000000

Probability of 7: 0.0000000000

Probability of 8: 0.0000000000

Probability of 9: 0.0000000000

#3

Probability of 0: 0.0000000000

Probability of 1: 0.0000000000

Probability of 2: 0.0000000001

Probability of 3: 0.9999999550

Probability of 4: 0.0000000000

Probability of 5: 0.0000000003

Probability of 6: 0.0000000000

Probability of 7: 0.0000000005

Probability of 8: 0.0000000441

Probability of 9: 0.0000000000

#4

Probability of 0: 0.0000000000

Probability of 1: 0.0000001190

Probability of 2: 0.0000000602

Probability of 3: 0.0000000000

Probability of 4: 0.9999916811

Probability of 5: 0.0000000003

Probability of 6: 0.0000000094

Probability of 7: 0.0000080037

Probability of 8: 0.0000000004

Probability of 9: 0.0000001259

#5

Probability of 0: 0.0000000183

Probability of 1: 0.0000000002

Probability of 2: 0.0000000162

Probability of 3: 0.0000066236

Probability of 4: 0.0000000008

Probability of 5: 0.9999876591

Probability of 6: 0.0000000297

Probability of 7: 0.0000000082

Probability of 8: 0.0000055763

Probability of 9: 0.0000000675

#6

Probability of 0: 0.0000151316

Probability of 1: 0.0000000015

Probability of 2: 0.0000001070

Probability of 3: 0.0000000145

Probability of 4: 0.0000011974

Probability of 5: 0.0000391253

Probability of 6: 0.9998591746

Probability of 7: 0.0000000000

Probability of 8: 0.0000852433

Probability of 9: 0.0000000048

#7

Probability of 0: 0.0000000000

Probability of 1: 0.0065864218

Probability of 2: 0.0011364910

Probability of 3: 0.0037865857

Probability of 4: 0.0000001004

Probability of 5: 0.0000000110

Probability of 6: 0.0000000000

Probability of 7: 0.9884878170

Probability of 8: 0.0000014590

Probability of 9: 0.0000011141

#8

Probability of 0: 0.0000029103

Probability of 1: 0.0000433023

Probability of 2: 0.0032396703

Probability of 3: 0.0004648336

Probability of 4: 0.0000000054

Probability of 5: 0.0000081442

Probability of 6: 0.0000001665

Probability of 7: 0.0000007293

Probability of 8: 0.9962261763

Probability of 9: 0.0000140618

#9

Probability of 0: 0.0000001110

Probability of 1: 0.0034772899

Probability of 2: 0.0025300084

Probability of 3: 0.0000637180

Probability of 4: 0.0032065302

Probability of 5: 0.0000558259

Probability of 6: 0.0000000906

Probability of 7: 0.0744447608

Probability of 8: 0.3153932156

Probability of 9: 0.6008284495