University at Buffalo
The State University of New York

CSE574 Introduction to Machine Learning

Project 4

Amir Baghdadi
50135018

Introduction

Artificial Neural Network (ANN), is inspired by biological neural networks in brain that learns and progressively improves performance in performing tasks by training through non task-specific programming. An ANN is a collection of connected units called artificial neurons. Each connection transmits a signal to another neuron and the receiving neuron can process all the signals from downstream neurons connected to it. The signal in output layer will determine the desired state of the variable to be classified.

Convolutional Neural Network (CNN) is a class of deep ANN that is recognized as state-of-the-art in analyzing visual imagery. CNNs use a variation of multilayer perceptron that minimizes the preprocessing requirements. Convolutional networks were designed based on biological processes in the organization of the animal visual cortex that inspires the connectivity patterns between neurons. In this setup, individual cortical neurons respond to stimuli only in a restricted region of the visual receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. A CNN consists of an input, multiple hidden, and an output layer. The hidden layer contains convolutional, pooling, dense, and normalization layers that builds up the flow of data and performs the computer based perception process[1].

The purpose of this project is to implement convolutional convolution neural network to determine whether the person in a portrait image is wearing glasses or not. The Celeb dataset was used which has more than 200k celebrity images in total. A publicly available convolutional neural network package from Tensorflow was used and trained on the Celeb images, hyperparameters were tuned and regularization was applied to improve the performance. The CelebFaces Attributes Dataset (CelebA) is a large-scale face attributes dataset with 202,599 celebrity images, each in color with an original definition of 178 x 218. Each image in the CelebA dataset is in ".jpg" format was imported and converted into 1D vector format. The evaluation will be performed by calculating classification error rate ($E = \frac{N_{wrong}}{N_V}$) in under the one-hot coding scheme for "Eyeglasses" attribute from list_attr_celeba.txt file indicating whether the person in the picture is wearing glasses or not.

Tasks and Results

Extract feature values and labels from the data:

The CelebA dataset was downloaded from the Internet and the original data file was processed into 1D Numpy arrays that containing the feature vectors and a Numpy array that contains the labels.

Reduce the size of training set:

Due to computational limitations on personal computer, a total number of 1000 samples were considered initially. In addition, the number of training samples in each batch was considered to be 10 samples.

[1] https://en.wikipedia.org/wiki/Convolutional_neural_network

Data Partition:

The imported data were partitioned into a training set, validation set, and testing set after shuffling to avoid any bias in the results. The training set takes the first 80% of the data, the validation set is the first 50% of the remaining data (10% of the whole), and the rest is regarded as the testing data. The three sets did not have overlap.

Reduce the resolution of the original images:

The resolution of the original images were reduced to 40(=10x4) x 32(=8x4). In this way the aspect ratio of the original image does not change and also the image can fit into our two layer CNN in terms of the pixel numbers in width and height. The results of trained model on testing set was E = 0.14 (86% accuracy) without dropout and E = 0.2 (80% accuracy) with dropout regularization.

Apply dropout or other regularization methods:

The dropout technique was used for regularization to avoid over fitting with the initial keep probability of 0.5.

Train model parameter:

For a given group of hyper-parameters such as dropout rates, the number of layers and the number of nodes in each layer, train the model parameters on the training set.

Tune hyper-parameters:

dropout rate:

```
keep probability = 0.1
testing error rate 0.27

keep probability = 0.5
testing error rate 0.13

keep probability = 1
testing error rate 0.11
```

number of convolutional layers:

```
number of convolutional layer = 1
testing error rate 0.09

number of convolutional layer = 2
testing error rate 0.11

number of convolutional layer = 3
testing error rate 0.05
```

The hyper-parameters were changed and the results were assessed. The above values of hyper-parameters resulted in a better performance on the testing set.

Retrain the model using higher resolutions:

The model was retrained using a high resolution of 160(=40x4) x 196(=49x4) and each side divisible to 4 so that can fit into our two layer CNN. The results of trained model on testing set was E = 0.03 (97% accuracy) without dropout and E = 0.06 (94% accuracy) with dropout regularization using two convolutional layers and 1000 input samples.

Use bigger sizes of the training set:

The number of total samples was increased to 2000 with 5% validation and 5% testing data, so the actual number of validation and testing data is not changed, however, the number of training data is increased to 1800. The results of trained model on testing set was E = 0.1 (90% accuracy) with dropout regularization and low resolution images.

Use even bigger sizes of the training set by data augmentation:

The number of newly increased train-data is increased even more to 5 times by means of

- Random rotation: each image is rotated by random degree in ranging [-15°, +15°].
- Random shift: each image is randomly shifted by a value ranging [-2 pix, +2 pix] at both axes.
- Zero-centered normalization: a pixel value is subtracted by (PIXEL_DEPTH/2) and divided by PIXEL_DEPTH.

The results of trained model on testing set was E = 0.09 (91% accuracy) with dropout regularization and low resolution images using two convolutional layers and 2000 input samples.

The following results are for the model with dropout regularization using 2000 augmented samples of low resolution images from which 90% is used for training after tuning the hyper-parameters (3 convolutional layers and keep probability = 1): E = 0.05 (95% accuracy)

```
UBitName = amirbagh
personNumber = 50135018
expanding data : 100 / 1800
expanding data : 200 / 1800
expanding data : 300 / 1800
expanding data : 400 / 1800
expanding data : 500 / 1800
expanding data : 600 / 1800
expanding data : 700 / 1800
expanding data : 800 / 1800
```

```
expanding data : 900 / 1800
expanding data : 1000 / 1800
expanding data : 1100 / 1800
expanding data : 1200 / 1800
expanding data : 1300 / 1800
expanding data : 1400 / 1800
expanding data : 1500 / 1800
expanding data : 1600 / 1800
expanding data : 1700 / 1800
expanding data : 1800 / 1800
step 0, training accuracy 0.2
step 10, training accuracy 0.9
step 20, training accuracy 1
step 30, training accuracy 0.9
step 40, training accuracy 0.7
step 50, training accuracy 0.3
step 60, training accuracy 0.9
step 70, training accuracy 1
step 80, training accuracy 1
step 90, training accuracy 1
step 100, training accuracy 0.9
step 110, training accuracy 0.9
step 120, training accuracy 0.9
step 130, training accuracy 0.9
step 140, training accuracy 1
step 150, training accuracy 0.9
step 160, training accuracy 0.9
step 170, training accuracy 0.9
step 180, training accuracy 1
step 190, training accuracy 0.9

testing error rate 0.05
```

main.py

```
# coding: utf-8

# In[1]:


# Project 4

print('UBitName = amirbagh')
print('personNumber = 50135018')

import numpy as np
from PIL import Image
import glob
from scipy import ndimage
import scipy

# setting the parameters
celebaImages_1d = []
new_width  = 32
new_height = 40
sample_size = 2000
itr = 200
tr_itr = 10
batch_size = 10
n_classes = 2
train_percent = 0.9
val_percent = 0.05
test_percent = 0.05

# read the CelebA image data
for fname in glob.glob('data/*.jpg'):
    img = scipy.ndimage.imread(fname, flatten=True, mode=None)
    img_resize = scipy.misc.imresize(img.astype('float32'), [new_width,
new_height])
    celebaImages_1d.append(img_resize.ravel())

celebaImages = np.asarray(celebaImages_1d)
```

main.py

```
# reading the labels from "list_attr_celeba.txt" file
f = open("data/list_attr_celeba.txt", "r")
list_attr_celeba = []
for line in f:
    list_attr_celeba.append(line)
eyeglass_label = []
list_attr_celeba = list_attr_celeba[0:sample_size+2]
for i in range(2,np.size(list_attr_celeba)):
    attr = list_attr_celeba[i].split()
    eyeglass_label.append(int((int(attr[16])+1)/2))

celebaLabels = eyeglass_label
celebaLabels_oneHot = np.zeros((np.size(list_attr_celeba)-2, 2))
celebaLabels_oneHot[np.arange(np.size(list_attr_celeba)-2), celebaLabels] =
1

def dataset_partition(train_percent, val_percent, test_percent, data,
indexes):
    data_lngth = np.shape(data)[0]
    train_data = data[indexes[0:round(train_percent * data_lngth)],:]
    val_data = data[indexes[round(train_percent *
data_lngth):round((train_percent + val_percent) * data_lngth)],:]
    test_data = data[indexes[round((train_percent + val_percent) *
data_lngth):data_lngth],:]
    return (train_data, val_data, test_data)

# Augment training data
def expend_training_data(images, labels, new_width, new_height):

    expanded_images = []
    expanded_labels = []

    j = 0 # counter
    for x, y in zip(images, labels):
        j = j+1
        if j%100==0:
            print ('expanding data : %03d / %03d' % (j,np.size(images,0)))

        # register original data
```

```
        expanded_images.append(x)
        expanded_labels.append(y)

        # get a value for the background
        # zero is the expected value, but median() is used to estimate
background's value
        bg_value = np.median(x) # this is regarded as background's
value
        image = np.reshape(x, (-1, new_width))

        for i in range(4):
            # rotate the image with random degree
            angle = np.random.randint(-15,15,1)
            new_img = ndimage.rotate(image,angle,reshape=False, cval=bg_value)

            # shift the image with random distance
            shift = np.random.randint(-2, 2, 2)
            new_img_ = ndimage.shift(new_img,shift, cval=bg_value)

            # register new training data
            expanded_images.append(np.reshape(new_img_, new_width*new_height))
            expanded_labels.append(y)

    # images and labels are concatenated for random-shuffle at each epoch
    # notice that pair of image and label should not be broken
    expanded_train_total_data = np.concatenate((expanded_images,
expanded_labels), axis=1)
    np.random.shuffle(expanded_train_total_data)

    trainData_aug = expanded_train_total_data[:,0:new_width*new_height]
    trainLabel_aug =
expanded_train_total_data[:,new_width*new_height:expanded_train_total_data.sha
pe[1]]
    expanded_train_total_data_seg = [trainData_aug, trainLabel_aug]

    return expanded_train_total_data_seg

# selecting the data
data_choose_in = celebaImages
```

```
data_choose_out = celebaLabels_oneHot

# shuffling the data
data_index = list(range(0, np.shape(data_choose_in)[0]))
np.random.shuffle(data_index)

# data partition
data_partition_in = dataset_partition(train_percent=train_percent,
val_percent=val_percent, test_percent=test_percent, data=data_choose_in,
indexes=data_index)
data_partition_out = dataset_partition(train_percent=train_percent,
val_percent=val_percent, test_percent=test_percent, data=data_choose_out,
indexes=data_index)

# test data assignment
testData = data_partition_in[1]
testLabel = data_partition_out[1]

# train data assignment/augmentation
celeba_train = [data_partition_in[0], data_partition_out[0]]
celeba_train_aug = expend_training_data(celeba_train[0], celeba_train[1],
new_width, new_height)

# comment/uncomment for training data augmentation
#trainData = celeba_train[0]
#trainLabel = celeba_train[1]
trainData = celeba_train_aug[0]
trainLabel = celeba_train_aug[1]


# Convolutional Neural Network

import tensorflow as tf

x = tf.placeholder(tf.float32, [None, new_width*new_height])  # input
W = tf.Variable(tf.zeros([new_width*new_height, 2]))          # Parameter
Variable
b = tf.Variable(tf.zeros([n_classes]))
y = tf.nn.softmax(tf.matmul(x, W) + b)       # Softmax regression model
```

```
y_ = tf.placeholder(tf.float32, [None, n_classes])   # add a new placeholder

# Weight Initialization
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)
def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

# Convolution and Pooling
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
    strides=[1, 2, 2, 1], padding='SAME')

# First Convolutional Layer
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
x_image = tf.reshape(x, [-1, new_width, new_height, 1])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

# Second Convolutional Layer
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) +
b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

# Third Convolutional Layer
W_conv3 = weight_variable([5, 5, 64, 128])
b_conv3 = bias_variable([128])
h_conv3 = tf.nn.relu(conv2d(h_pool2, W_conv3) +
b_conv3)
h_pool3 = max_pool_2x2(h_conv3)
```

```
# Densely Connected Layer
W_fc1 = weight_variable([(new_width//8) * (new_height//8) * 128, 2048])
b_fc1 = bias_variable([2048])
h_pool3_flat = tf.reshape(h_pool3, [-1, (new_width//8)*(new_height//8)*128])
h_fc1 = tf.nn.relu(tf.matmul(h_pool3_flat, W_fc1) + b_fc1)

# Dropout
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
# Readout Layer
W_fc2 = weight_variable([2048, n_classes])
b_fc2 = bias_variable([n_classes])
# comment/uncomment for dropout
#y_conv = tf.matmul(h_fc1, W_fc2) + b_fc2
y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2


# Train and Evaluate the Model
cross_entropy = tf.reduce_mean(
tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    test_err = []
    for i in range(itr-1):
        batch = [trainData[batch_size*(i//tr_itr):batch_size*(i//tr_itr+1)],
trainLabel[batch_size*(i//tr_itr):batch_size*(i//tr_itr+1)]]
        if i % tr_itr == 0:
            train_accuracy = accuracy.eval(feed_dict={
                x: batch[0], y_: batch[1], keep_prob: 1.0})
            print('step %d, training accuracy %g' % (i, train_accuracy))
        train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})
        test_err.append(1-accuracy.eval(feed_dict={
            x: testData, y_: testLabel, keep_prob: 1}))

    print('testing error rate %g' % np.round(np.mean(test_err),2))
```