Satvinder Singh Panesar

UBITName: satvinde Person No: 50248888

Project 4

Introduction to Deep Learning

1. **Extracting feature values and labels from the data:**

   Corresponding Python Functions:
   Misc.imread(): read an image
   Numpy.ravel(): return a flattened array
   Numpy.genfromtxt(): load data from a text file
   Numpy.reshpe(): reshape a numpy array

```
def read_resized_images(width, height, input_size):
    temp=[]
    for i in range(1,input_size+1):
        img=misc.imread("resized_"+str(width)+"_"+str(height)+"/"+str(i).rjust(6,'0')+".jpg",flatten=1)
        img=np.ravel(img)
        temp.append(img)
    print("Reading resized celeb images completed")
    return np.array(temp)
```

Fig: Reading features of celeb image dataset (python code)

```
def read_labels(input_size):
    temp=np.genfromtxt("list_attr_celeba.txt", delimiter=" ", skip_header=2, max_rows=input_size)
    temp=temp[0:,16]
    temp=np.reshape(temp,(-1,1))
    labels=[]
    for ele in temp:
        if ele == 1:
            labels.append([1,0])
        else:
            labels.append([0,1])
    print("Reading labels completed")
    return np.array(labels)
```

Fig: Reading labels of celeb image dataset (python code)

2. **Reducing the resolution of the original images**

   Corresponding Python Functions:
   Image.open(): open an image
   Image.resize(): resize an image
   Image.save(): save an image

```
def resize_images(width, height, input_size):
   dir_name = "resized_"+str(width)+"_"+str(height)
   if not os.path.exists(dir_name):
      os.makedirs(dir_name)
   for filename in g.glob(os.path.join("img_align_celeba/", '*.jpg'))[0:input_size]:
      img=Image.open(filename.replace("\\","/"))
      img=img.resize((width,height), Image.NEAREST)
      img.save(filename.replace("img_align_celeba","resized_"+str(width)+"_"+str(height)))
   print("Resizing celeb images completed”)
```

Fig: Reducing image resolution (python code)

3. Reducing the size of training set

Restricting size of training set using variables **input_size** in code.

4. Data Partitioning

```
#creating training and testing sets
celeb_image_train_data = celeb_image_data[0:int(len(celeb_image_data)*0.8)]
celeb_image_train_labels = celeb_image_labels[0:int(len(celeb_image_labels)*0.8)]

celeb_image_test_data = celeb_image_data[int(len(celeb_image_data)*0.8):int(len(celeb_image_data))]
celeb_image_test_labels =
celeb_image_labels[int(len(celeb_image_labels)*0.8):int(len(celeb_image_labels))]
```

Fig: Data partitioning into training and testing sets (python code)

5. Applying dropout for regularization

To reduce overfitting, we will apply dropout before the readout layer. We create a placeholder for the probability that a neuron's output is kept during dropout. This allows us to turn dropout on during training, and turn it off during testing. Tensor Flow’s tf.nn.dropout op automatically handles scaling neuron outputs in addition to masking them, so dropout just works without any additional scaling

```
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

Fig: Applying dropout to avoid over fitting (python code)

6. Training model parameters

```
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)

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')

def train_and_get_accuracy(width, height, step_size, batch_size, no_of_neurons, prob, train_data,
train_labels, test_data, test_labels):
    x = tf.placeholder(tf.float32, [None, width*height])
    y_ = tf.placeholder(tf.float32, [None, 2])

    W = tf.Variable(tf.zeros([width*height, 2]))
    b = tf.Variable(tf.zeros([1]))

    y = tf.matmul(x,W) + b

    W_conv1 = weight_variable([5, 5, 1, 32])
    b_conv1 = bias_variable([32])
    x_image = tf.reshape(x, [-1, height, width, 1])
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
    h_pool1 = max_pool_2x2(h_conv1)

    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)

    W_fc1 = weight_variable([int(height/4) * int(width/4) * 64, no_of_neurons])
    b_fc1 = bias_variable([no_of_neurons])
    h_pool2_flat = tf.reshape(h_pool2, [-1, int(height/4) * int(width/4) * 64])
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

    keep_prob = tf.placeholder(tf.float32)
    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

    W_fc2 = weight_variable([no_of_neurons, 2])
    b_fc2 = bias_variable([2])
    y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2

    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())
        for i in range(step_size):
            arr = random.sample(range(0,len(train_data)-1),batch_size)
            x_batch = []
```

```
        y_batch = []
        for ele in arr:
          x_batch.append(train_data[ele])
          y_batch.append(train_labels[ele])
        if i % 100 == 0:
          train_accuracy = accuracy.eval(feed_dict={x: x_batch, y_: y_batch, keep_prob: 1.0})
          print('step %d, training accuracy %g' % (i, train_accuracy))
        train_step.run(feed_dict={x: x_batch, y_: y_batch, keep_prob: prob})
      print('Accuracy with Dataset: %g' % accuracy.eval(feed_dict={x: test_data, y_: test_labels, keep_prob:
1.0}))
```

Fig: Training model (python code)

7. **Tuning hyper-parameters:**

   For multi-layer convolutional network, we can tune **number of neurons and dropout** to enhance the performance.

   Detailed explanation is in following sections**.**

8. **Retraining the model using higher resolutions:**

   For this we will again resize images using:
   resize_images(width, height, input_size), but pass higher values for width and height.:

   And then retrain model using:
   train_and_get_accuracy(width, height, train_data, train_labels, test_data, test_labels)

9. **Using bigger sizes of the training set**

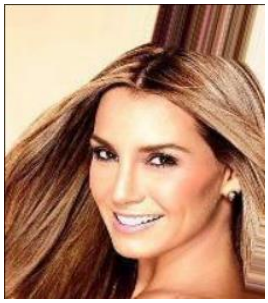
   By changing variable **input_size**, we can adjust the size of training set and then complete training using below functions:

   train_and_get_accuracy(width, height, train_data, train_labels, test_data, test_labels)

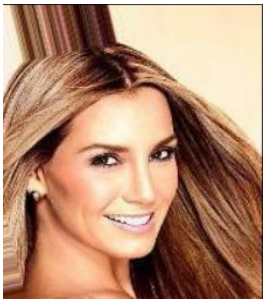
10. **Using bigger sizes of the training set with augmented data**

    Augmented data is obtained by **flipping or rotating images, adding noise, adjusting image shadows etc.** This changes image perspective but doesn't change the label associated with the image**.**

    Corresponding Python Functions:
    Image.transpose(): flips an image



Original Image



Flipped Image

```
def flip_images(width, height, input_size):
    dir_name = "flipped_"+str(width)+"_"+str(height)
    if not os.path.exists(dir_name):
        os.makedirs(dir_name)
    for filename in g.glob(os.path.join("resized_"+str(width)+"_"+str(height)+"/", '*.jpg'))[0:input_size]:
        img=Image.open(filename.replace("\\","/"))
        img=img.transpose(Image.FLIP_LEFT_RIGHT)
        img.save(filename.replace("resized","flipped"))
    print("Flipping images completed")

def read_flipped_images(width, height, input_size):
    temp=[]
    for i in range(1,input_size+1):
        img=misc.imread("flipped_"+str(width)+"_"+str(height)+"/"+str(i).rjust(6,'0')+".jpg",flatten=1)
        img=np.ravel(img)
        temp.append(img)
    print("Reading flipped celeb images completed")
    return np.array(temp)

def get_augmented_data(width, height, input_size):
    resize_images(width, height, input_size)
    temp_img = read_resized_images(width, height, input_size)
    flip_images(width, height, input_size)
    temp_rotated_img = read_flipped_images(width, height, input_size)
    data=np.concatenate((temp_img,temp_rotated_img),axis=0)
    return data
```

Fig: Getting augmented data (python code)

CELEB IMAGE DATASET

Training set size = 80%
Testing set size = 20%

1. Training with lower resolution images

Training phase (# of neurons = 1024, # of layers = 2, keep_prob = 0.5)

```
step 0, training accuracy 0.41
step 100, training accuracy 0.93
step 200, training accuracy 0.97
step 300, training accuracy 0.94
step 400, training accuracy 0.95
step 500, training accuracy 0.98
step 600, training accuracy 0.98
step 700, training accuracy 0.99
step 800, training accuracy 0.99
step 900, training accuracy 0.97
step 1000, training accuracy 1
step 1100, training accuracy 0.98
step 1200, training accuracy 0.99
step 1300, training accuracy 1
step 1400, training accuracy 0.99
```

Validation phase (Hyper parameter tuning)

Tuning values of # of neurons and keep_prob (dropout controller)

```
Results with # of neurons = 100 and keep_prob = 0.1
-----------------------------------------------------------------------
Accuracy with Dataset:  0.932 (Accuracy is lower with smaller values)

Results with # of neurons = 1024 and keep_prob = 0.5
-----------------------------------------------------------------------
Accuracy with Dataset:  0.945 (Accuracy is higher with higher values)
```

Testing phase with tuned parameters

```
Results with tuned parameters:
------------------------
Accuracy with Dataset: 0.945
```

2. Training with higher resolution images

Training phase

```
step 0, training accuracy 0.66
step 100, training accuracy 0.94
step 200, training accuracy 0.97
step 300, training accuracy 0.94
step 400, training accuracy 0.96
step 500, training accuracy 0.99
step 600, training accuracy 1
step 700, training accuracy 1
step 800, training accuracy 0.99
step 900, training accuracy 1
step 1000, training accuracy 1
step 1100, training accuracy 1
step 1200, training accuracy 1
step 1300, training accuracy 1
step 1400, training accuracy 1
```

Testing phase using tuned parameters

```
Results with tuned parameters:
-------------------------
Accuracy with Dataset: 0.956 (Improvement over lower resolution images)
```

3. Training with higher resolution images and larger training set

Training phase

```
step 0, training accuracy 0.92
step 100, training accuracy 0.96
step 200, training accuracy 0.96
step 300, training accuracy 0.92
step 400, training accuracy 0.93
step 500, training accuracy 0.94
step 600, training accuracy 0.91
step 700, training accuracy 0.99
step 800, training accuracy 0.97
step 900, training accuracy 0.96
step 1000, training accuracy 0.96
step 1100, training accuracy 0.97
step 1200, training accuracy 0.94
step 1300, training accuracy 0.98
step 1400, training accuracy 0.98
```

Testing phase using tuned parameters

```
Results with tuned parameters:
-------------------------
Accuracy with Dataset: 0.963 (Improvement over lower resolution images and
higher resolution images with smaller training set)
```

4. Training with higher resolution images and larger training set using Data Augmentation

Training phase

```
step 0, training accuracy 0.05
step 100, training accuracy 0.96
step 200, training accuracy 0.96
step 300, training accuracy 0.94
step 400, training accuracy 0.97
step 500, training accuracy 0.95
step 600, training accuracy 0.93
step 700, training accuracy 0.99
step 800, training accuracy 0.98
step 900, training accuracy 0.97
step 1000, training accuracy 0.95
step 1100, training accuracy 0.97
step 1200, training accuracy 0.98
step 1300, training accuracy 0.99
step 1400, training accuracy 0.99
```

Testing phase using tuned parameters

```
Results with tuned parameters:
-------------------------
Accuracy with Dataset: 0.9632 (Improvement)
```

Summary

Training data	Testing Accuracy
Lower resolution images	**0.945**
Higher resolution images	**0.956**
Higher resolution images (larger training set)	**0.963**
Higher resolution images (data augmentation)	**0.9632**

From above table, we can draw below conclusions:

1. **Higher resolution images improve performance.**
2. **Increasing the size of training set improves performance.**
3. **Data augmentation improves performance.**