Image Classification

컴퓨터 소프트웨어 학부

2018008059 김은수

- 환경
- 코드설명
- 결과설명

1. 환경

   - Tensorflow 1.13.1
   - Python 3.7.10

```
import tensorflow as tf
import sys

print(tf.__version__)
print(sys.version)
```

```
/usr/local/lib/python3.7/dist-packages/ten
  _np_qint8 = np.dtype([("qint8", np.int8
/usr/local/lib/python3.7/dist-packages/ten
  _np_quint8 = np.dtype([("quint8", np.uin
/usr/local/lib/python3.7/dist-packages/ten
  _np_qint16 = np.dtype([("qint16", np.in
/usr/local/lib/python3.7/dist-packages/ten
  _np_quint16 = np.dtype([("quint16", np.u
/usr/local/lib/python3.7/dist-packages/ten
  _np_qint32 = np.dtype([("qint32", np.in
/usr/local/lib/python3.7/dist-packages/ten
  np_resource = np.dtype([("resource", np
1.13.1
3.7.10 (default, May  3 2021, 02:48:31)
[GCC 7.5.0]
```

2. 코드 설명

- 학습 방법: 첫번째로 학습을 돌린 뒤 학습된 파라매터를 check point를 통해 저장하고 learning rate와 batch size를 조정하여 한 번 더 학습하였다.

- 하이퍼파라매터

  첫번째

```
learning_rate = 0.001
training_epochs = 50
batch_size = 128
```

  두번째

```
learning_rate = 0.0001
training_epochs = 50
batch_size = 64
```

  Dropout

```
keep_prob = tf.placeholder(tf.float32)
KEEP_PROB = 0.7
```

  KEEP_PROB는 0.7로 70%의 노드를 활성화한다.

- 딥러닝 layer

```
def build_CNN_classifier(x):
    x_image = x

    W1 = tf.get_variable(name="W1", shape=[5, 5, 3, 64], initializer=tf.contrib.layers.xavier_initializer())
    b1 = tf.get_variable(name="b1", shape=[64], initializer=tf.contrib.layers.xavier_initializer())
    c1 = tf.nn.conv2d(x_image, W1, strides=[1, 1, 1, 1], padding='SAME')
    l1 = tf.nn.bias_add(c1, b1)
    l1 = tf.layers.batch_normalization(l1)
    l1 = tf.nn.relu(l1)
    l1_pool = tf.nn.max_pool(l1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding='SAME')

    W2 = tf.get_variable(name="W2", shape = [5,5,64,128], initializer= tf.contrib.layers.xavier_initializer())
    b2 = tf.get_variable(name="b2", shape = [128], initializer = tf.contrib.layers.xavier_initializer())
    c2 = tf.nn.conv2d(l1_pool, W2, strides=[1, 1, 1, 1], padding='SAME')
    l2 = tf.nn.bias_add(c2, b2)
    l2 = tf.layers.batch_normalization(l2)
    l2 = tf.nn.relu(l2)
    l2 = tf.nn.dropout(l2,keep_prob = KEEP_PROB)
    l2_pool = tf.nn.max_pool(l2, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding='SAME')
```

  Convolutional layer, pooling layer가 각 4개씩 있는 neural network이다. Input 이미지

의 채널은 처음에 RGB 3개이다. 첫번째 layer에서 여기에 5X5 필터 64개를 사용한다. Convolutional layer를 거치고 나오면 batch normalization을 해주고 그 다음에 activation함수로 ReLU함수를 적용시켜준다. 다음으로 dropout으로 overfitting을 방지해준 뒤 pooling layer를 통해 down sampling해준다. Pooling은 3X3의 커널을 만들어주고 stride는 2로 2칸씩 이동한다.

필터 사이즈는 각 convolutional layer마다 5X5로 같고 filter의 개수는 3->64->128->256->256이다. 4개의 layer를 거치고 나면 2*2이 256개 있다. 따라서 flat하려면 이를 다 곱한 2X2X256이다.

```
shape=(?, 2, 2, 256)
```

```
l4_flat = tf.reshape(l4_pool, [-1, 256*2*2])
```

```
W1_fc = tf.get_variable(name="W1_fc", shape=[256*2*2, 256*4], initializer=tf.contrib.layers.xavier_initializer())
b1_fc = tf.get_variable(name="b1_fc", shape=[256*4], initializer=tf.contrib.layers.xavier_initializer())
l1_fc = tf.nn.relu(tf.nn.bias_add(tf.matmul(l4_flat, W1_fc), b1_fc))
l1_fc = tf.nn.dropout(l1_fc, keep_prob=KEEP_PROB)

W2_fc = tf.get_variable(name="W2_fc", shape=[256*4, 256], initializer=tf.contrib.layers.xavier_initializer())
b2_fc = tf.get_variable(name="b2_fc", shape=[256], initializer=tf.contrib.layers.xavier_initializer())
l2_fc = tf.nn.relu(tf.nn.bias_add(tf.matmul(l1_fc, W2_fc), b2_fc))
l2_fc = tf.nn.dropout(l2_fc, keep_prob=KEEP_PROB)

W3_fc = tf.get_variable(name="W3_fc", shape=[256, 10], initializer=tf.contrib.layers.xavier_initializer())
b3_fc = tf.get_variable(name="b3_fc", shape=[10], initializer=tf.contrib.layers.xavier_initializer())
logits = tf.nn.bias_add(tf.matmul(l2_fc, W3_fc), b3_fc)

hypothesis = tf.nn.softmax(logits)

return hypothesis, logits
```

Fully connected layer부분은 다음과 같다. 3개의 layer를 거쳐서 256*2*2에서 10개의 뉴런으로 나타난다. 1,2 layer에서는 활성화 함수로 Relu함수를 사용하고, 마지막에는 softmax함수를 사용한다.

- Check point.

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    #saver = tf.train.Saver()
    #saver.restore(sess, ckpt_path)

    print("학습시작")

    for epoch in range(training_epochs):
        print("Epoch", epoch+1)
        start = 0
        average_cost = 0
        shuffled_idx = np.arange(0, len(x_train))
        np.random.shuffle(shuffled_idx)

        for i in range(total_batch):
            batch = batch_data(shuffled_idx, batch_size, x_train, y_train_one_hot.eval(), i*batch_size)
            c,_ = sess.run([cost,train_step], feed_dict={x: batch[0], y: batch[1]})
            average_cost += c /total_batch
        print("Epoch:", '%d'%(epoch+1), 'cost = ', '{:.9f}'.format(average_cost))
    saver = tf.train.Saver()
    saver.save(sess, ckpt_path)
    saver.restore(sess, ckpt_path)
```

총 2번 학습을 진행하기 때문에 처음에는 다음과 같이 진행하고 2번째는 주석을 풀어서 학습을 시켜주면 된다.

- Mini batch를 위한 함수

```
def batch_data(shuffled_idx, batch_size, data, labels, start_idx):
    idx = shuffled_idx[start_idx:start_idx+batch_size]
    data_shuffle = [data[i] for i in idx]
    labels_shuffle = [labels[i] for i in idx]

    return np.asarray(data_shuffle), np.asarray(labels_shuffle)
```

- cost함수와 Optimizer

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y, logits=logits))
train_step = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

- epoch당 cost 함수 출력.

```
for i in range(total_batch):
    batch = batch_data(shuffled_idx, batch_size, x_train, y_train_one_hot.eval(), i*batch_size)
    c,_ = sess.run([cost,train_step], feed_dict={x: batch[0], y: batch[1]})
    average_cost += c /total_batch
print("Epoch:", '%d'%(epoch+1), 'cost = ', '{:.9f}'.format(average_cost))
```

3. 결과 설명

처음	Epoch 47 Epoch: 47 cost = 0.299286700 Epoch 48 Epoch: 48 cost = 0.260747775 Epoch 49 Epoch: 49 cost = 0.260662157 Epoch 50 Epoch: 50 cost = 0.254786967 WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packag Instructions for updating: Use standard file APIs to check for files with this prefix. dev 데이터 f1 score: 0.687575
두번째	Epoch 47 Epoch: 47 cost = 0.014199762 Epoch 48 Epoch: 48 cost = 0.013574072 Epoch 49 Epoch: 49 cost = 0.012171109 Epoch 50 Epoch: 50 cost = 0.011437897 dev 데이터 f1 score: 0.744835

- 한번 학습 당 50번의 epoch을 돌렸다.
- Train 데이터에 대한 cost는 0.2 -> 0.01 로 많이 줄어들고, dev 데이터에 대한 f1 score 또한 0.68에서 0.74로 많이 증가한다.
- **F1 score 0.744835**