Instructions on how to use the code:

Part 1: CODE OF CNN

This code is divided into five files.

1. data_preprocessing.py
2. forward_propagation.py
3. back_propagation.py
4. data_testing.py
5. extract_kernels.py

Explaination:

1. data_preprocessing.py

This is used to generate our dataset that can be input into CNN. You should firstly use this code to generate the array, and then save them to your folder.

Detail annotations can be found in the code.

2. forward_propagation.py

This code is used to define the architecture of the neural network. Detail annotations can be found in the code.

Don't run this code directly, you should run the back_propagation.py, and it will import the forward_propagation.py.

3. back_propagation.py

This code is used to define the process of back propagation of neural network. When you already generate the array that can be input into the CNN (by using data_preprocessing.py). You can run this code and start the training process of your model.

```
16
17 BATCH_SIZE = 100
18 LEARNING_RATE_BASE = 0.0005
19 LEARNING_RATE_DECAY = 0.99
20 REGULARIZER = 0.0001
21 STEPS = 10000
22 MOVING_AVERAGE_DECAY = 0.99
23 MODEL_SAVE_PATH="./training_model_exp/"
24 MODEL_NAME="model"
25
```

The trained model will be saved in this path. What's more, don't be afraid of the power cut when you run the code. The information of the trained model will be saved and this code can continue to train when your power is return.

!! NOTE: see the following figure. The number in the red bracket should change with the size of training dataset.

```
for i in range(STEPS):
    start = (i*BATCH_SIZE) % 6000  #note!!: This number should change with the size of training dataset
    end = start + BATCH_SIZE
    _, loss_value, step, accuracy_score = sess.run([train_op, loss, global_step, accuracy], feed_dict={x: fluid_data[start:end,:,:,:], y_: fluid_label[start:end,:]})
    if i % 100 == 0:
        print("After %d training step(s), loss on training batch is %g." % (step, loss_value))
        print("accuracy for training dataset is %d" % (accuracy_score))
        saver.save(sess, os.path.join(MODEL_SAVE_PATH, MODEL_NAME), global_step=global_step)
```

4. data_testing.py

This code is testing the accuracy of data. You can run this code and back_propagation.py at the same time. And you can also run this code after you train you model.

```
def main():
    fluid_data = np.load('G:/fluid_data/interpolated_data_near_the_airfoil/1M_9000_10000_data.npy')
    fluid_data = fluid_data[:,:,:,0:1]
    fluid_label = np.load('G:/fluid_data/test_label_3.npy')
    #fluid_data1 = fluid_data[:,:,:,0]
```

Here, you should prepare your test data and label.

5. extract_kernels.py

This code is used to extract the kernels and feature maps in the trained model. Also, you can also extract other weights in the architecture.

It should be noted that this code will import forward_propagation.py and back_propagation.py. **You should change the output of 'forward' function in the forward_propagation.py. Just like the following figure.**

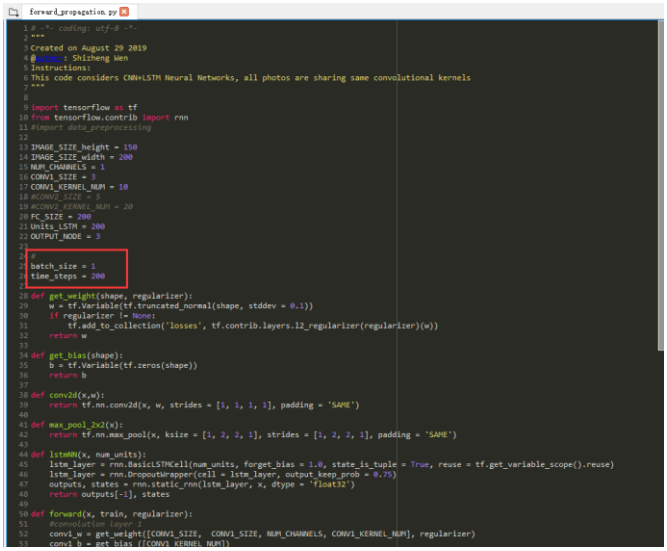
forward_propagation.py* | back_propagation.py | data_testing.py | extract_kernels.py

```
17 CONV1_SIZE = 3
18 CONV1_KERNEL_NUM = 10
19 #CONV2_SIZE = 5
20 #CONV2_KERNEL_NUM = 20
21 FC_SIZE = 200
22 OUTPUT_NODE = 3
23
24 def get_weight(shape, regularizer):
25     w = tf.Variable(tf.truncated_normal(shape, stddev = 0.1))
26     if regularizer != None:
27         tf.add_to_collection('losses', tf.contrib.layers.l2_regularizer(regularizer)(w))
28     return w
29
30 def get_bias(shape):
31     b = tf.Variable(tf.zeros(shape))
32     return b
33
34 def conv2d(x,w):
35     return tf.nn.conv2d(x, w, strides = [1, 1, 1, 1], padding = 'SAME')
36
37 def max_pool_2x2(x):
38     return tf.nn.max_pool(x, ksize = [1, 2, 2, 1], strides = [1, 2, 2, 1], padding = 'SAME')
39
40 def forward(x, train, regularizer):
41     #convolution layer 1
42     conv1_w = get_weight([CONV1_SIZE,  CONV1_SIZE, NUM_CHANNELS, CONV1_KERNEL_NUM], regularizer)
43     conv1_b = get_bias ([CONV1_KERNEL_NUM])
44     conv1 = conv2d(x, conv1_w)
45     relu1 = tf.nn.relu(tf.nn.bias_add(conv1, conv1_b))
46     pool1 = max_pool_2x2(relu1)
47
48     #convolution layer 2
49     #conv2_w = get_weight([CONV2_SIZE, CONV2_SIZE, CONV1_KERNEL_NUM, CONV2_KERNEL_NUM], regularizer)
50     #conv2_b = get_bias([CONV2_KERNEL_NUM])
51     #conv2 = conv2d(pool1, conv2_w)
52     #relu2 = tf.nn.relu(tf.nn.bias_add(conv2, conv2_b))
53     #pool2 = max_pool_2x2(relu2)
54
55     #reshape
56     pool_shape = pool1.get_shape().as_list()
57     nodes = pool_shape[1] * pool_shape[2] * pool_shape[3]
58     reshaped = tf.reshape(pool1, [pool_shape[0], nodes]) #batch 行，所有特征点作为列的二维张量
59
60     #fully connected layer(one hidden layer)
61     fc1_w = get_weight([nodes, FC_SIZE], regularizer)
62     fc1_b = get_bias([FC_SIZE])
63     fc1 = tf.nn.relu(tf.matmul(reshaped, fc1_w) + fc1_b)
64     # when training, dropout some neurons at hidden layer randomly.
65     if train:
66         fc1 = tf.nn.dropout(fc1, 0.5)
67
68     fc2_w = get_weight([FC_SIZE, OUTPUT_NODE], regularizer)
69     fc2_b = get_bias([OUTPUT_NODE])
70     y = tf.matmul(fc1, fc2_w) + fc2_b
71     return conv1_w, conv1, relu1, pool1, fc2_w
72
```

Part 2: CODE OF CNN-LSTM

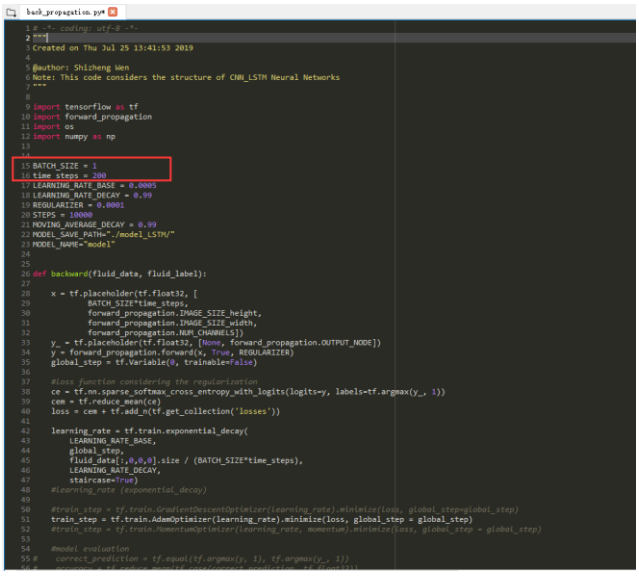
The code for CNN-LSTM is similar with the CNN. The difference is as follows:
1. You should reconstruct your training dataset. What I mean is that snapshots in the training dataset should have the sequence.

2.In the forward_propagation.py and back_propagation.py, you should set your batch_size and time_steps:



```
# -*- coding: utf-8 -*-
"""
Created on August 29 2019
@author: Shizheng Wen
Instructions:
This code considers CNN+LSTM Neural Networks, all photos are sharing same convolutional kernels
"""

import tensorflow as tf
from tensorflow.contrib import rnn
#import data_preprocessing

IMAGE_SIZE_height = 150
IMAGE_SIZE_width = 200
NUM_CHANNELS = 1
CONV1_SIZE = 3
CONV1_KERNEL_NUM = 10
#CONV2_SIZE = 3
#CONV2_KERNEL_NUM = 20
FC_SIZE = 200
Units_LSTM = 200
OUTPUT_NODE = 3

#
batch_size = 1
time_steps = 200

def get_weight(shape, regularizer):
    w = tf.Variable(tf.truncated_normal(shape, stddev = 0.1))
    if regularizer != None:
        tf.add_to_collection('losses', tf.contrib.layers.l2_regularizer(regularizer)(w))
    return w

def get_bias(shape):
    b = tf.Variable(tf.zeros(shape))
    return b

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 lstmNN(x, num_units):
    lstm_layer = rnn.BasicLSTMCell(num_units, forget_bias = 1.0, state_is_tuple = True, reuse = tf.get_variable_scope().reuse)
    lstm_layer = rnn.DropoutWrapper(cell = lstm_layer, output_keep_prob = 0.75)
    outputs, states = rnn.static_rnn(lstm_layer, x, dtype = 'float32')
    return outputs[-1], states

def forward(x, train, regularizer):
    #convolution layer 1
    conv1_w = get_weight([CONV1_SIZE,  CONV1_SIZE, NUM_CHANNELS, CONV1_KERNEL_NUM], regularizer)
    conv1_b = get_bias ([CONV1_KERNEL_NUM])
```

forward_propagation.py



```
# -*- coding: utf-8 -*-
"""
Created on Thu Jul 25 13:41:53 2019

@author: Shizheng Wen
Note: This code considers the structure of CNN_LSTM Neural Networks
"""

import tensorflow as tf
import forward_propagation
import os
import numpy as np

BATCH_SIZE = 1
time_steps = 200
LEARNING_RATE_BASE = 0.0005
LEARNING_RATE_DECAY = 0.99
REGULARIZER = 0.0001
STEPS = 10000
MOVING_AVERAGE_DECAY = 0.99
MODEL_SAVE_PATH="./model_LSTM/"
MODEL_NAME="model"


def backward(fluid_data, fluid_label):

    x = tf.placeholder(tf.float32, [
            BATCH_SIZE*time_steps,
            forward_propagation.IMAGE_SIZE_height,
            forward_propagation.IMAGE_SIZE_width,
            forward_propagation.NUM_CHANNELS])
    y_ = tf.placeholder(tf.float32, [None, forward_propagation.OUTPUT_NODE])
    y = forward_propagation.forward(x, True, REGULARIZER)
    global_step = tf.Variable(0, trainable=False)

    #loss function considering the regularization
    ce = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=y, labels=tf.argmax(y_, 1))
    cem = tf.reduce_mean(ce)
    loss = cem + tf.add_n(tf.get_collection('losses'))

    learning_rate = tf.train.exponential_decay(
        LEARNING_RATE_BASE,
        global_step,
        fluid_data[:,0,0,0].size / (BATCH_SIZE*time_steps),
        LEARNING_RATE_DECAY,
        staircase=True)
    #learning_rate (exponential_decay)

    #train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss, global_step=global_step)
    train_step = tf.train.AdamOptimizer(learning_rate).minimize(loss, global_step = global_step)
    #train_step = tf.train.MomentumOptimizer(learning_rate, momentum).minimize(loss, global_step = global_step)

    #model evaluation
    correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
```

back_propagation.py

In summary, the main difference is the change of architecture in forward_propagation.py.

If you meet other problems when using this code, please be free to contact me:
szwen@nuaa.edu.cn