Supplementary S1 | BPNN model code for muskmelon

```
%This is BPNN model
clc
clear
tic
%load data from Excel
Data = xlsread('pcainputdata');
k = rand(1,156);
[m,n] = sort(k);
input_train = Data(n(1:124),1:3);
output_train = Data(n(1:124),4);
input_test = Data(n(125:156),1:3);
output_test = Data(n(125:156),4);
%normalize data
[inputn,inputps] = mapminmax(input_train(:,1:3)');
[outputn,outputps] = mapminmax(output_train');
inputn_test = mapminmax('apply',input_test(:,1:3)',inputps);
outputn_test = mapminmax('apply',output_test',outputps);
input_train(:,1:3) = inputn';
output_train = outputn';
input_test(:,1:3) = inputn_test';
output_test = outputn_test';
%network
innum = 3;
hidenum = 12;
outnum = 1;
%initialize weight values and threshold values
w1 = rands(hidenum,innum);
b1 = rands(hidenum,1);
w2 = rands(hidenum,outnum);
b2 = rands(outnum,1);
w1_1 = w1;
w1_2 = w1_1;
w2_1 = w2;
w2_2 = w2_1;
b1_1 = b1;
b1_2 = b1_1;
b2_1 = b2;
b2_2 = b2_1;
xite_max = 0.2;
xite_min = 0.02;
```

```
alfa = 0.02;
loopNumber = 200;
I = zeros(1,hidenum);
Iout = zeros(1,hidenum);
FI = zeros(1,hidenum);
dw1 = zeros(innum,hidenum);
db1 = zeros(1,hidenum);
%training
E = zeros(1,loopNumber);
for ii = 1:loopNumber
   E(ii) = 0;
   xite(ii) = xite_max-ii*(xite_max-xite_min)/loopNumber;
   for i = 1:124
      x = input_train(i,:)';
      for j = 1:hidenum
         I(j) = input_train(i,:)*w1(j,:)'+b1(j);
         Iout(j) = 1/(1+exp(-I(j)));
      end
      yn = w2'*Iout'+b2;
      e = output_train(i,:)-yn';
      E(ii) = E(ii)+sum(abs(e));
      dw2 = e'*Iout;
      db2 = e';
      for j = 1:hidenum
         S = 1/(1+exp(-I(j)));
         FI(j) = S*(1-S);
      end
      for k = 1:innum
         for j = 1:hidenum
            dw1(k,j) = FI(j)*x(k)*e(1)*w2(j,1);
            db1(j) = FI(j)*e(1)*w2(j,1);
         end
      end
      w1 = w1_1+xite(ii)*dw1'+alfa*(w1_1-w1_2);
      b1 = b1_1+xite(ii)*db1'+alfa*(b1_1-b1_2);
      w2 = w2_1+xite(ii)*dw2'+alfa*(w2_1-w2_2);
      b2 = b2_1+xite(ii)*db2+alfa*(b2_1-b2_2);
      w1_2 = w1_1;
      w1_1 = w1;
      w2_2 = w2_1;
      w2_1 = w2;
      b1_2 = b1_1;
      b1_1 = b1;
```

```
        b2_2 = b2_1;
        b2_1 = b2;
    end
end
for i = 1:32
    for j = 1:hidenum
        I(j) = input_test(i,:)*w1(j,:)'+b1(j);
        Iout(j) = 1/(1+exp(-I(j)));
    end
    %predictive output
    an(i,:) = w2'*Iout'+b2;
end
%renormalization
output1 = mapminmax('reverse',an,outputps);
BPoutput = output1';
output2 = mapminmax('reverse',output_test,outputps);
Realoutput = output2';
figure(1)
plot(BPoutput(:,1),'-*')
hold on
plot(Realoutput(:,1),'-o');
ylabel('Value','fontsize',12)
xlabel('Num','fontsize',12)
error1=BPoutput(:,1)-Realoutput(:,1);
%results of R2 and RMSE
r1 = 1 - sum((Realoutput(:,1) - BPoutput(:,1)).^2) / sum((Realoutput(:,1) - (sum(BPoutput(:,1)) /
32)).^2)
se1 = sqrt(sum((Realoutput(:,1) - BPoutput(:,1)).^2)/32)
BPoutput
Realoutput
toc
```

Supplementary S2 | CNN model code for muskmelon

```
from skimage import io, transform
from xlutils.copy import copy
import matplotlib.pyplot as plt
import xlwt
import xlrd
import math
import numpy as np
import pandas as pd
import os
from keras import backend as K
from keras.models import Sequential
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.utils import np_utils
from keras.optimizers import SGD, RMSprop, Adam
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
def suffer(data, label):
    # createst random data of training set and test set
    num_example = data.shape[0]
    arr = np.arange(num_example)
    np.random.shuffle(arr)
    data = data[arr]
    label = label[arr]
    ratio = 0.9
    s = np.int(num_example * ratio)
    x_t = data[:s]
    y_t = label[:s]
    x_v = data[s:]
    y_v = label[s:]
    return x_t, y_t, x_v, y_v
image_datas = np.load(".../imageset_128.npy")
label_datas = np.load(".../labelset.npy")
x_train, y_train, x_val, y_val = suffer(image_datas, label_datas)
class LeNet:
    @staticmethod
    def build(input_shape):
        model = Sequential()
        # CONV + CONV + MAXPOOL + DROPOUT + CONV + CONV + MAXPOOL
        model.add(Conv2D(32, kernel_size=5, padding="same", input_shape=input_shape))
        model.add(Activation("relu"))
```

```
        model.add(Conv2D(32, kernel_size=5, padding="same", input_shape=input_shape))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(Conv2D(64, kernel_size=5, padding="same"))
        model.add(Activation("relu"))
        model.add(Conv2D(64, kernel_size=5))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(Flatten())
        model.add(Dense(512))
        model.add(Activation("relu"))
        model.add(Dense(1))
        model.summary()
        return model
# network and training
NB_EPOCH = 20
BATCH_SIZE = 12
VERBOSE = 1
OPTIMIZER = Adam()
VALIDATION_SPLIT = 0.1
IMG_ROWS, IMG_COLS = 256, 256  # input image dimensions
INPUT_SHAPE = (IMG_ROWS, IMG_COLS, 3)
# data: shuffled and split between train and test sets, input_shape must be (height, width,
channels)
K.set_image_dim_ordering("tf")
# consider them as float and normalize
X_train = x_train.astype('float32')
X_val = x_val.astype('float32')
X_train /= 255
X_val /= 255
X_train = X_train.reshape(X_train.shape[0], IMG_ROWS, IMG_COLS, 3)
X_val = X_val.reshape(X_val.shape[0], IMG_ROWS, IMG_COLS, 3)
# we need  [4 x 256 x 256 x 3] shape as input to the CONVNET
X_train = X_train[:, :, :, :]
X_val = X_val[:, :, :, :]
print(X_train.shape[0], 'train samples')
print(X_val.shape[0], 'test samples')
# initialize the optimizer and model
model = LeNet.build(input_shape=INPUT_SHAPE)
model.compile(loss="MSE", optimizer=OPTIMIZER,
            metrics=["mean_squared_error"])
history = model.fit(X_train, y_train,
                  batch_size=BATCH_SIZE, epochs=NB_EPOCH,
```

```
                    verbose=VERBOSE, validation_split=VALIDATION_SPLIT)
y_pred = model.predict(X_val)
yval = y_val.flat[:]
yp = y_pred.flat[:]
filename = '.../pred_data.xlsx'
data = pd.DataFrame(yp)
writer = pd.ExcelWriter(filename)
data.to_excel(writer, 'pred_data',header=["PRED_DATA"], float_format='%.5f')
writer.save()
writer.close()
open_book = xlrd.open_workbook(filename)
work_book = copy(open_book)
sheet = work_book.get_sheet(0)
sheet.write(0, 2, "REAL_DATA")
for i in range(1, 95):
    sheet.write(i, 2, yval[i-1])
os.remove(filename)
work_book.save('.../pred_data.xls')
score, _ = model.evaluate(X_val, y_val, verbose=VERBOSE)
rmse = math.sqrt(score)
print("MSE: {:.3F}, RMSE:{:.3f}".format(score, rmse))
# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('model mse')
plt.ylabel('mse')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Supplementary S3 | DCNN model code for muskmelon

```
from skimage import io, transform
from xlutils.copy import copy
import matplotlib.pyplot as plt
import xlwt
import xlrd
import math
import numpy as np
import pandas as pd
import pydot
import os
from keras import backend as K
from keras.models import Sequential
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Dense, Activation, Flatten
from keras.utils import np_utils
from keras.optimizers import SGD, RMSprop, Adam
from keras.utils import plot_model
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
np.random.seed(0)
def suffer(data, label):
    # createst random data of training set and test set
    num_example = data.shape[0]
    arr = np.arange(num_example)
    np.random.shuffle(arr)
    data = data[arr]
    label = label[arr]
    ratio = 0.9
    s = np.int(num_example * ratio)
    x_t = data[:s]
    y_t = label[:s]
    x_v = data[s:]
    y_v = label[s:]
    return x_t, y_t, x_v, y_v
image_datas = np.load(".../imageset_128.npy")
label_datas = np.load(".../labelset.npy")
x_train, y_train, x_val, y_val = suffer(image_datas, label_datas)
class LeNet:
    @staticmethod
    def build(input_shape):
        model = Sequential()
```

```
        model.add(Conv2D(16, kernel_size=5, padding="same", input_shape=input_shape))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(Conv2D(32, kernel_size=5, padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(Conv2D(64, kernel_size=5, padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(Conv2D(128, kernel_size=5, padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(Conv2D(256, kernel_size=5, padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(Flatten())
        model.add(Dense(1024))
        model.add(Activation("relu"))
        model.add(Dense(1))
        model.summary()
        return model
# network and training
NB_EPOCH = 40
BATCH_SIZE = 12
VERBOSE = 1
OPTIMIZER = Adam()
VALIDATION_SPLIT = 0.1
IMG_ROWS, IMG_COLS = 256, 256  # input image dimensions
INPUT_SHAPE = (IMG_ROWS, IMG_COLS, 3)
# data: shuffled and split between train and test sets, input_shape must be (height, width,
channels)
K.set_image_dim_ordering("tf")
# consider them as float and normalize
X_train = x_train.astype('float32')
X_val = x_val.astype('float32')
X_train /= 255
X_val /= 255
X_train = X_train.reshape(X_train.shape[0], IMG_ROWS, IMG_COLS, 3)
X_val = X_val.reshape(X_val.shape[0], IMG_ROWS, IMG_COLS, 3)
# we need  [4 x 256 x 256 x 3] shape as input to the CONVNET
X_train = X_train[:, :, :, :]
X_val = X_val[:, :, :, :]
print(X_train.shape[0], 'train samples')
```

```
print(X_val.shape[0], 'test samples')
# initialize the optimizer and model
model = LeNet.build(input_shape=INPUT_SHAPE)
model.compile(loss="MSE", optimizer=OPTIMIZER,
          metrics=["mean_squared_error"])
history = model.fit(X_train, y_train,
                batch_size=BATCH_SIZE, epochs=NB_EPOCH, validation_data=(X_val, y_val),
                verbose=VERBOSE, validation_split=VALIDATION_SPLIT)
y_pred = model.predict(X_val)
yval = y_val.flat[:]
yp = y_pred.flat[:]
filename = '.../pred_data.xls'
data = pd.DataFrame(yp)
writer = pd.ExcelWriter(filename)
data.to_excel(writer, 'pred_data',header=["PRED_DATA"], float_format='%.5f')
writer.save()
writer.close()
open_book = xlrd.open_workbook(filename)
work_book = copy(open_book)
sheet = work_book.get_sheet(0)
sheet.write(0, 2, "REAL_DATA")
num_row = X_val.shape[0] + 1
for i in range(1, num_row):
   sheet.write(i, 2, yval[i-1])
os.remove(filename)
work_book.save('.../pred_data.xls')
score, _ = model.evaluate(X_val, y_val, verbose=VERBOSE)
rmse = math.sqrt(score)

print("MSE: {:.3F}, RMSE:{:.3f}".format(score, rmse))
# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('model mse')
plt.ylabel('mse')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right', frameon=False)
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right', frameon=False)
plt.show()
# plot_model(model, to_file='.../model.png', show_shapes=True)
```

Supplementary S4 | DCNN-LSTM model code for muskmelon

```
from skimage import io, transform
from xlutils.copy import copy
import matplotlib.pyplot as plt
import xlwt
import xlrd
import math
import numpy as np
import pandas as pd
import os
from keras import backend as K
from keras.models import Model
from keras.models import Sequential
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Dense, Activation, Flatten, Dropout
from keras.layers.recurrent import LSTM
from keras.utils import np_utils
from keras.optimizers import SGD, RMSprop, Adam
from keras.layers import concatenate
from keras.utils import plot_model
from collections import Iterator
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
np.random.seed(0)
def suffer(data1, data2, label):
    # createst random data of training set and test set
    num_example = data1.shape[0]
    arr = np.arange(num_example)
    np.random.shuffle(arr)
    data1 = data1[arr]
    data2 = data2[arr]
    label = label[arr]
    ratio = 0.9
    s = np.int(num_example * ratio)
    x_t1 = data1[:s]
    x_t2 = data2[:s]
    y_t = label[:s]
    x_v1 = data1[s:]
    x_v2 = data2[s:]
    y_v = label[s:]
    return x_t1, x_t2, y_t, x_v1, x_v2, y_v
```

```
image_datas = np.load(".../imageset_128.npy")
label_datas = np.load(".../labelset.npy")
tep_datas = np.load(".../tepset.npy")
tep_datas = np.expand_dims(tep_datas, axis=1)
x_train1, x_train2, y_train, x_val1, x_val2, y_val = suffer(image_datas, tep_datas, label_datas)
def create_cnn(input_shape):
    model = Sequential()
    model.add(Conv2D(16, kernel_size=5, padding="same", input_shape=input_shape))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Conv2D(32, kernel_size=5, padding="same"))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Conv2D(64, kernel_size=5, padding="same"))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Conv2D(128, kernel_size=5, padding="same"))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Conv2D(256, kernel_size=5, padding="same"))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Flatten())
    model.add(Dense(1024))
    model.add(Activation("relu"))
    model.add(Dropout(0.1))
    model.add(Dense(4))
    model.summary()
    return model
def create_lstm(hidden_size1, hidden_size2, hidden_size3):
    model = Sequential()
    model.add(LSTM(hidden_size1, batch_input_shape=(None, 1, 1), stateful=False,
return_sequences=True))
    model.add(LSTM(hidden_size2, return_sequences=True))
    model.add(LSTM(hidden_size3, return_sequences=False))
    model.add(Dense(4))
    model.summary()
    return model
# network and training
BATCH_SIZE = 12
NB_EPOCH = 40
VERBOSE = 1
OPTIMIZER = Adam()
```

```
VALIDATION_SPLIT = 0.1
HIDDEN_SIZE1 = 50
HIDDEN_SIZE2 = 30
HIDDEN_SIZE3 = 10
IMG_ROWS, IMG_COLS = 256, 256  # input image dimensions
INPUT_SHAPE = (IMG_ROWS, IMG_COLS, 3)
# data: shuffled and split between train and test sets, input_shape must be (height, width, channels)
K.set_image_dim_ordering("tf")
# consider them as float and normalize
X_train1 = x_train1.astype('float32')
X_val1 = x_val1.astype('float32')
X_train1 /= 255
X_val1 /= 255
X_train1 = X_train1.reshape(X_train1.shape[0], IMG_ROWS, IMG_COLS, 3)
X_val1 = X_val1.reshape(X_val1.shape[0], IMG_ROWS, IMG_COLS, 3)
X_train1 = X_train1[:, :, :, :]
X_val1 = X_val1[:, :, :, :]
train_size = (x_train2.shape[0] // BATCH_SIZE) * BATCH_SIZE
test_size = (x_val2.shape[0] // BATCH_SIZE) * BATCH_SIZE
X_train1, X_train2, Y_train = X_train1[0:train_size], x_train2[0:train_size], y_train[0:train_size]
X_val1, X_val2, Y_val = X_val1[0:test_size], x_val2[0:test_size], y_val[0:test_size]
print(X_train1.shape[0], 'train samples')
print(X_val1.shape[0], 'test samples')
cnn = create_cnn(INPUT_SHAPE)
lstm = create_lstm(HIDDEN_SIZE1, HIDDEN_SIZE2, HIDDEN_SIZE3)
combinedInput = concatenate([cnn.output, lstm.output])
x = Dense(4, activation="relu")(combinedInput)
x = Dense(1, activation="linear")(x)
model = Model(inputs=[cnn.input, lstm.input], outputs=x)
model.compile(loss="MSE", optimizer=OPTIMIZER, metrics=["mean_squared_error"])
history = model.fit([X_train1, X_train2], Y_train, validation_data=([X_val1, X_val2], Y_val),
                    batch_size=BATCH_SIZE, epochs=NB_EPOCH,
                    verbose=VERBOSE, validation_split=VALIDATION_SPLIT)
Y_pred = model.predict([X_val1, X_val2])
# export predicted value
yval = Y_val.flat[:]
yp = Y_pred.flat[:]
filename = '.../pred_data.xls'
data = pd.DataFrame(yp)
writer = pd.ExcelWriter(filename)
data.to_excel(writer, 'pred_data',header=["PRED_DATA"], float_format='%.5f')
writer.save()
writer.close()
```

```
open_book = xlrd.open_workbook(filename)
work_book = copy(open_book)
sheet = work_book.get_sheet(0)
sheet.write(0, 2, "REAL_DATA")
num_row = test_size + 1
for i in range(1, num_row):
    sheet.write(i, 2, yval[i-1])
os.remove(filename)
work_book.save('.../pred_data.xls')
score, _ = model.evaluate([X_val1, X_val2], Y_val, verbose=VERBOSE)
rmse = math.sqrt(score)
print("MSE: {:.3F}, RMSE:{:.3f}".format(score, rmse))
# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['mean_squared_error'])
plt.plot(history.history['val_mean_squared_error'])
plt.title('model mse')
plt.ylabel('mse')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right', frameon=False)
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right', frameon=False)
plt.show()
# plot_model(model, to_file='.../cnnlstmmodel.png', show_shapes=True)
# check input image
fig1, ax1 = plt.subplots(figsize=(4, 4))
ax1.imshow(X_val1[1])
plt.show()
image_arr = np.reshape(X_val1[1], (-1, 256, 256, 3))
# Visualization result of first MaxPooling2D
layer_1 = K.function([model.layers[0].input], [model.layers[1].output])
# 只修改input_image
f1 = layer_1([image_arr])[0]
re1 = np.transpose(f1, (0, 3, 1, 2))
for i in range(16):
    plt.subplot(4, 4, i+1)
```

```
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(re1[0][i])
plt.show()
# Visualization result of second MaxPooling2D
layer_2 = K.function([model.layers[0].input], [model.layers[4].output])
f2 = layer_2([image_arr])[0]
re2 = np.transpose(f2, (0, 3, 1, 2))
for i in range(32):
    plt.subplot(4, 8, i+1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(re2[0][i])
plt.show()
# Visualization result of third MaxPooling2D
layer_3 = K.function([model.layers[0].input], [model.layers[7].output])
f3 = layer_3([image_arr])[0]
re3 = np.transpose(f3, (0, 3, 1, 2))
for i in range(64):
    plt.subplot(8, 8, i+1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(re3[0][i])
plt.show()
# Visualization result of forth MaxPooling2D
layer_4 = K.function([model.layers[0].input], [model.layers[10].output])
f4 = layer_4([image_arr])[0]
re4 = np.transpose(f4, (0, 3, 1, 2))
for i in range(128):
    plt.subplot(8, 16, i+1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(re4[0][i])
plt.show()
# Visualization result of fifth MaxPooling2D
layer_5 = K.function([model.layers[0].input], [model.layers[13].output])
f5 = layer_5([image_arr])[0]
re5 = np.transpose(f5, (0, 3, 1, 2))
for i in range(256):
```

```
    plt.subplot(16, 16, i+1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(re5[0][i])
plt.show()
```