Machine Learning 2017 HW3 Report

Image Sentiment Classification

B03901156 Yu Xuan Huang

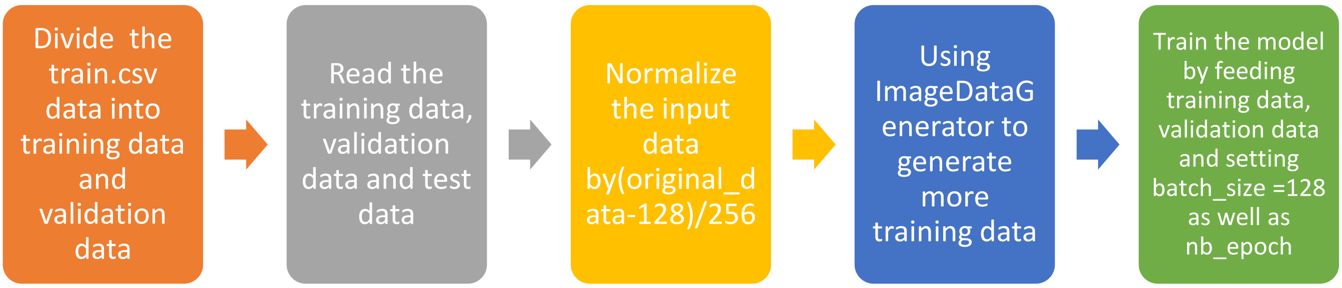
1. 請說明你實作的 CNN model，其模型架構、訓練過程和準確率為何？

Structure:

Layer (type)	Output Shape	Param #
Convolution Part		
# First Layer		
conv2d_1 (Conv2D)	(None, 48, 48, 32)	320
activation_1 (Activation)	(None, 48, 48, 32)	0
conv2d_2 (Conv2D)	(None, 46, 46, 32)	9248
activation_2 (Activation)	(None, 46, 46, 32)	0
max_pooling2d_1 (MaxPooling2)	(None, 23, 23, 32)	0
dropout_1 (Dropout)	(None, 23, 23, 32)	0
# Second Layer		
conv2d_3 (Conv2D)	(None, 23, 23, 64)	18496
activation_3 (Activation)	(None, 23, 23, 64)	0
conv2d_4 (Conv2D)	(None, 21, 21, 64)	36928
activation_4 (Activation)	(None, 21, 21, 64)	0
max_pooling2d_2 (MaxPooling2)	(None, 10, 10, 64)	0
dropout_2 (Dropout)	(None, 10, 10, 64)	0
# Third Layer		
conv2d_5 (Conv2D)	(None, 10, 10, 128)	73856
activation_5 (Activation)	(None, 10, 10, 128)	0
conv2d_6 (Conv2D)	(None, 8, 8, 128)	147584
activation_6 (Activation)	(None, 8, 8, 128)	0
max_pooling2d_3 (MaxPooling2)	(None, 4, 4, 128)	0
dropout_3 (Dropout)	(None, 4, 4, 128)	0
Fully-connected Part		
# Feed Forward		
flatten_1 (Flatten)	(None, 2048)	0
dense_1 (Dense)	(None, 1024)	2098176
activation_7 (Activation)	(None, 1024)	0
dense_2 (Dense)	(None, 512)	524800
activation_8 (Activation)	(None, 512)	0
dense_3 (Dense)	(None, 256)	131328
activation_9 (Activation)	(None, 256)	0

dense_4 (Dense)	(None, 128)	32896
activation_10 (Activation)	(None, 128)	0
dropout_4 (Dropout)	(None, 128)	0
# Output		
dense_5 (Dense)	(None, 7)	903
activation_11 (Activation)	(None, 7)	0
# Compile the model		
model.compile(loss='categorical_crossentropy',optimizer='adadelta',metrics=['accuracy'])		
Total params: 3,074,535 Trainable params: 3,074,535 Non-trainable params: 0		

Training Process:



Accurate rate:

Model	Accurate Rate (Kaggle)
Without ImageDataGenerator, nb_epoch = 30	0.57788
Without ImageDataGenerator, nb_epoch = 40	0.62469
Without ImageDataGenerator, nb_epoch = 70	0.61856
With ImageDataGenerator, nb_epoch = 30	0.62469
With ImageDataGenerator, nb_epoch = 40	0.64140
With ImageDataGenerator, nb_epoch = 70	0.65589
With ImageDataGenerator, nb_epoch = 100	0.66871

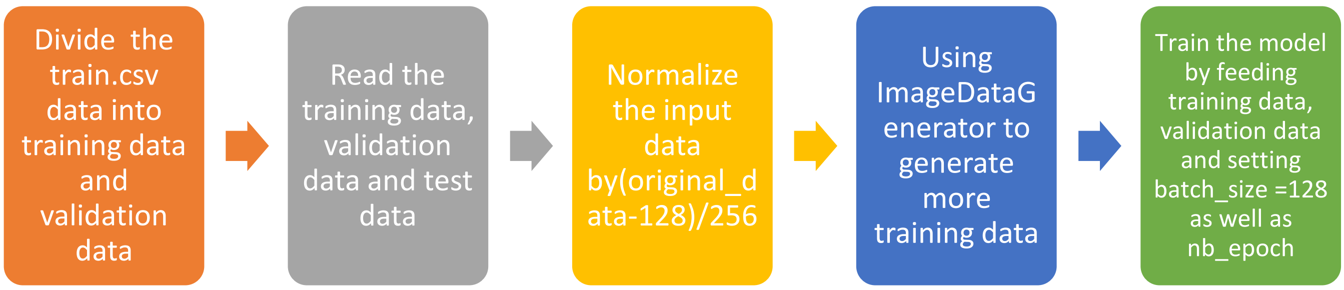
2. 承上題，請用與上述 CNN 接近的參數量，實做簡單的 DNN model，其模型架構、訓練過程和準確率為何？試與上題結果做比較，並說明你觀察到了什麼？

Structure:

Layer (type)	Output Shape	Param #
max_pooling2d_1 (MaxPooling2)	(None, 24, 24, 1)	0
dropout_1 (Dropout)	(None, 24, 24, 1)	0
Fully-connected Part		
# Feed Forward		
flatten_1 (Flatten)	(None, 576)	0

dense_1 (Dense)	(None, 1024)	590848
activation_1 (Activation)	(None, 1024)	0
dense_2 (Dense)	(None, 1024)	1049600
activation_2 (Activation)	(None, 1024)	0
dense_3 (Dense)	(None, 512)	524800
activation_3 (Activation)	(None, 512)	0
dense_4 (Dense)	(None, 512)	262656
activation_4 (Activation)	(None, 512)	0
dense_5 (Dense)	(None, 512)	262656
activation_5 (Activation)	(None, 512)	0
dense_6 (Dense)	(None, 256)	131328
activation_6 (Activation)	(None, 256)	0
dense_7 (Dense)	(None, 128)	32896
activation_7 (Activation)	(None, 128)	0
dropout_2 (Dropout)	(None, 128)	0
# Output		
dense_8 (Dense)	(None, 7)	903
activation_8 (Activation)	(None, 7)	0
# Compile the model		
model.compile(loss='categorical_crossentropy',optimizer='adadelta',metrics=['accuracy'])		
Total params: 2,855,687 Trainable params: 2,855,687 Non-trainable params: 0		

Training Process:



Accurate rate:

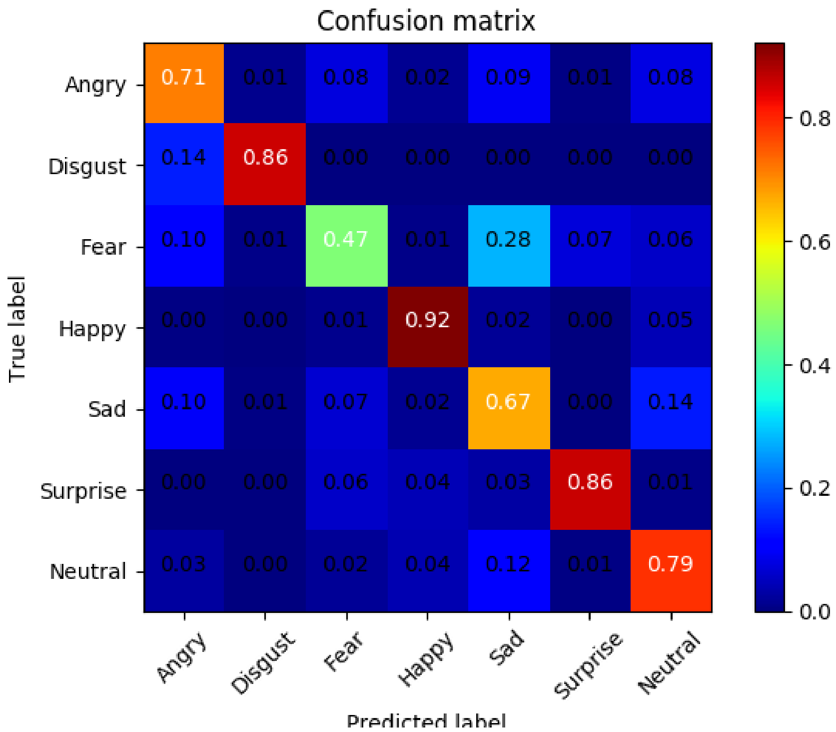
Model	Accurate Rate (Kaggle)
Without ImageDataGenerator, nb_epoch = 30	0.24464
Without ImageDataGenerator, nb_epoch = 40	0.30375
Without ImageDataGenerator, nb_epoch = 70	0.34789
With ImageDataGenerator, nb_epoch = 30	0.30797

With ImageDataGenerator, nb_epoch = 40	0.38878
With ImageDataGenerator, nb_epoch = 70	0.39008

Observation:

From the experiment of CNN model and DNN model, it can be observed that although the training process of DNN model is shorter than CNN when given similar parameters, CNN usually performs better than DNN in image classification under the same condition, which indicates that CNN is more adequate than DNN to do the image processing.

3. (1%) 觀察答錯的圖片中，哪些 class 彼此間容易用混？[繪出 confusion matrix 分析]



Ans. According to the above confusion matrix, the following pairs (True label→Predicted label) are possible to be confused with each other:

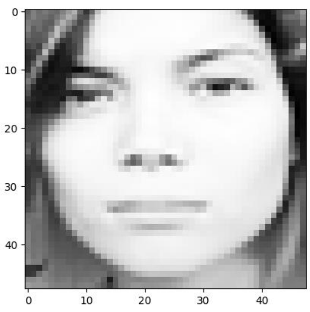
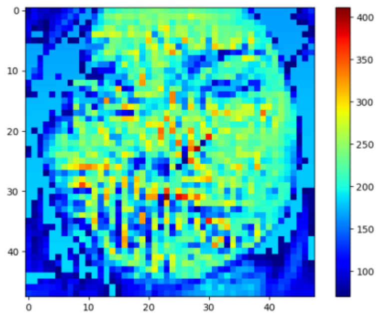
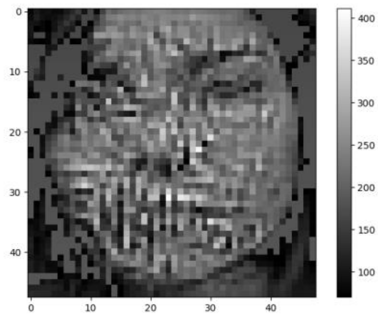
Fear → Sad ---Possibility: 0.28

Sad → Neutral ---Possibility: 0.14

Disgust → Angry ---Possibility: 0.14

Neutral → Sad ---Possibility: 0.12

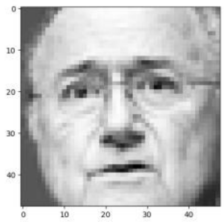
4. (1%) 從(1)(2)可以發現，使用 CNN 的確有些好處，試繪出其 saliency maps 觀察模型在做 classification 時，是 focus 在圖片的哪些部份？

Original Image (Class : Neutral)	Saliency Map	Mask 掉 heat 小的部份
		

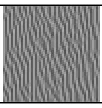
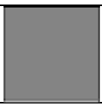
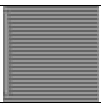
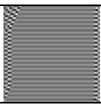
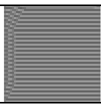
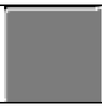
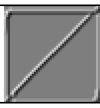
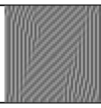
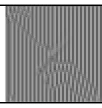
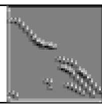
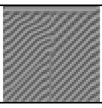
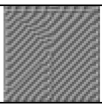
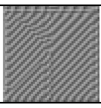
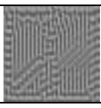
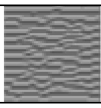
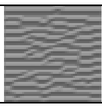
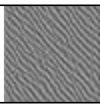
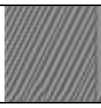
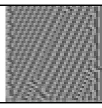
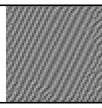
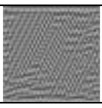
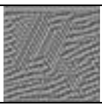
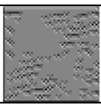
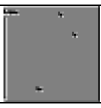
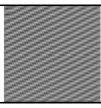
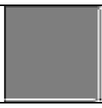
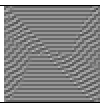
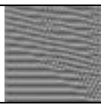
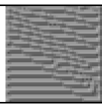
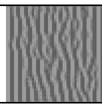
Ans. According to the above result, it can be observed that when the model does classification, it always focuses on the facial features.

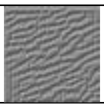
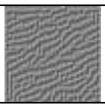
5. (1%) 承(1)(2)，利用上課所提到的 gradient ascent 方法，觀察特定層的 filter 最容易被哪種圖片 activate。

Ans. According to the result of many experiments I did on the layer activation_1, I find that it is more possible to be activated by class Neutral images. The sample results are on the below:

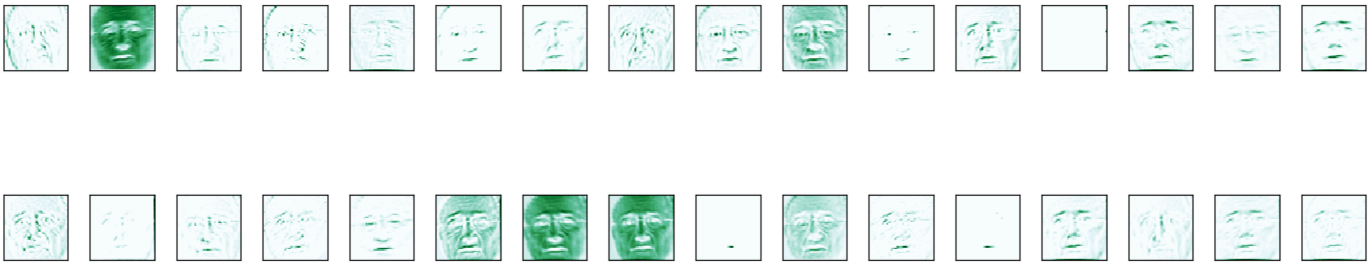


Class : Neutral

Filters of layer activation_1 (#Ascent Epoch 160)									
									
8904.07	4940.14	6944.14	7469.61	5110.9	6868.79	5530.72	11370.2	7155.79	4913.89
									
7073.17	8197.83	6866.65	6898.25	10865.2	8831.69	8326.24	6810.98	6952.76	7101.83
									
7395.73	5122.91	5903.94	6843.09	8585.94	5356.57	6339.77	6177.95	6097.38	9214.44

		
6161.09	6973.58	

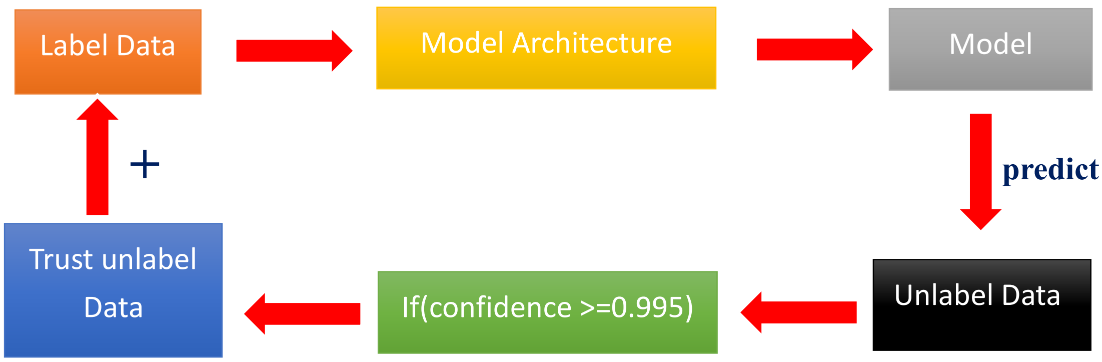
Output of layer activation_1



6. [Bonus](1%) 從 training data 中移除部份 label，實做 semi-supervised learning

Ans.

Structure:



First design a not bad model, and then utilize this model to predict unlabeled data, adding the unlabeled data with high confidence (>=0.995) to the labeled data (self-training). Repeat this procedure for several time and we can get better performance (accurate rate= 0.64140)

Script usage:

bash semi_train.sh <training data> <validation data> <testing data> < prediction file >

bash semi_test.sh <testing data> <prediction file>

Implementation:

```python
import os
import numpy as np
import sys
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.layers import Input,Dense,Dropout,Flatten,Activation,BatchNormalization
from keras.layers import Convolution2D,MaxPooling2D
from keras.optimizers import SGD
from keras.models import load_model
from keras.utils import np_utils
from keras.callbacks import EarlyStopping
from keras.preprocessing.image import ImageDataGenerator
from keras.layers.normalization import BatchNormalization
nb_classes = 7
# read train data
def read_dataset(train_file,isFeat=True):
    datas = []
    with open(train_file) as file:
        next(file)
        for line_id,line in enumerate(file):
            label, feat=line.split(',')
            feat = np.fromstring(feat,dtype='float32',sep=' ')
            # normalization
            feat = (feat-128)/255
            #print(feat)
            feat = np.reshape(feat,(48,48,1))

            datas.append((feat,int(label),line_id))

    #random.shuffle(datas)  # shuffle outside
    feats,labels,line_ids = zip(*datas)
    feats = np.asarray(feats)
    labels = to_categorical(np.asarray(labels,dtype=np.int32))
    return feats,labels,line_ids

```

```python
# read test data
def read_test_dataset(test_file):
    datas = []

    with open(test_file) as file:
        next(file)
        for line_id,line in enumerate(file):
            _,feat = line.split(',')
            feat = np.fromstring(feat,dtype='float32',sep=' ')
            # normalization
            feat = (feat-128)/255
            #print(feat)
            feat = np.reshape(feat,(48,48,1))
            datas.append(feat)

    #random.shuffle(datas)  # shuffle outside

    feats = datas
    feats = np.asarray(feats)
    return feats

# add unlabel feature to training data
def add_unlabel_featrue_to_training(training_feature, training_yhat, unlabel_feature, model, confidence_th):
    unlabel_prediction = model.predict(unlabel_feature)
    # print("nb_unlabel_feature:" + str(len(unlabel_feature) ))
    # print("nb_unlabel_prediction:" + str(len(unlabel_prediction)))
    nb_unlabel_data = len(unlabel_prediction)

    delete_array = np.array([0]*nb_unlabel_data).astype(dtype='int')
    delete_index = np.array([]).astype(dtype='int')
    count = 0

    print("[ original data ] : training->" + str(len(training_feature)) + "  unlabel->"+str(len(unlabel_feature)) )


    for i in range(nb_unlabel_data):
```

```python
    for i in range(nb_unlabel_data):
        if(  np.amax( unlabel_prediction[i] ) >= confidence_th   ):

            delete_array[i] = 1
            max_index = np.argmax(unlabel_prediction[i])
            tmp_prediction = unlabel_prediction[i]
            tmp_prediction.fill(0)
            tmp_prediction[max_index] = 1.0

            tmp_feature =  unlabel_feature[i]
            tmp_feature = np.array( tmp_feature ).astype(dtype='float32').reshape(1,48,48,1)
            #print("!!!!!!!!!!!!!!!!!!",training_feature.shape )
            #print("!!!!!!!!!!!!!!!!!!",tmp_feature.shape )
            training_feature = np.vstack((training_feature, tmp_feature ))


            tmp_prediction = np.array( tmp_prediction ).astype(dtype='float32').reshape(1,7)
            training_yhat= np.vstack((training_yhat, tmp_prediction ))

            count += 1
        if(i%1000 == 0 ):
            print("check unlabel confidence :" + str(i) +"  data")


    for i in range( nb_unlabel_data):
        if(delete_array[i] == 1):
            delete_index = np.append(delete_index, i )
        if(i%1000 == 0 ):
            print("delete unlabel feature :" + str(i) +"  data")

    unlabel_feature = np.delete(unlabel_feature, delete_index, axis = 0 )


    print("after confidence similar trimming:")
    print("total move :" + str(count) + "  data from unlabel to training ")
    print("[after data] : training->" + str(len(training_feature)) + "  unlabel->" + str(len(unlabel_feature)) )
```

```python
    return training_feature, training_yhat, unlabel_feature


# split the data into training and validation data
def split_data(training_feature, training_yhat, validation_percent):

    training_percent = 1 - validation_percent
    num_training = int(training_percent * training_feature.shape[0])
    indices = np.random.permutation(training_feature.shape[0])
    training_idx,validation_idx = indices[:num_training], indices[num_training:]
    #print("training_feature.shape[0]",training_feature.shape[0])
    #print("training_yhat.shape[0]",training_yhat.shape[0])
    #print()

    training_feature ,validation_feature = training_feature[training_idx,:], training_feature[validation_idx,:]
    training_yhat ,validation_yhat = training_yhat[training_idx,:], training_yhat[validation_idx,:]

    return training_feature , training_yhat , validation_feature , validation_yhat

def build_model(mode):
    model = Sequential()
    if mode == 'easy':
    # CNN part (you can repeat this part several times)
        model.add(Convolution2D(32,3,3,border_mode='same',input_shape=(48,48,1)))
        model.add(Activation('relu'))
        model.add(Convolution2D(32, 3, 3))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        model.add(Convolution2D(64, 3, 3, border_mode='same'))
        model.add(Activation('relu'))
        model.add(Convolution2D(64, 3, 3))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))
```

```
        model.add(Convolution2D(128, 3, 3, border_mode='same'))
        model.add(Activation('relu'))
        model.add(Convolution2D(128, 3, 3))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        # fully-connected part
        model.add(Flatten())
        model.add(Dense(1024))
        model.add(Activation('relu'))
        model.add(Dense(512))
        model.add(Activation('relu'))
        model.add(Dense(256))
        model.add(Activation('relu'))
        model.add(Dense(128))
        model.add(Activation('relu'))
        model.add(Dropout(0.25))
        model.add(Dense(nb_classes))
        model.add(Activation('softmax'))

        opt = SGD(lr=0.01,decay=1e-6,momentum=0.9, nesterov=True)
    model.compile(loss='categorical_crossentropy',
                  optimizer='adadelta',
                  metrics=['accuracy'])
    model.summary() # show the whole model in terminal
    return model


# read training raw data
tr_feats, tr_labels, _ = read_dataset(sys.argv[1])
# read validation data
dev_feats, dev_labels,_ = read_dataset(sys.argv[2])
# read testing raw data
test_feats= read_test_dataset(sys.argv[3])
```

```
# get training_feature , training_yhat , validation_feature , validation_yhat
(training_feature , training_yhat , validation_feature , validation_yhat) = split_data(tr_feats , tr_labels , 0.1 )
# confidence threshold
confidence_th = 0.995
#unlabel data
unlabel_feature = test_feats
#early stop
early_stop = EarlyStopping(monitor='val_loss' , patience=20, verbose=1)

nb_add_unlabel = 5

for i in range(nb_add_unlabel):
    if i == nb_add_unlabel -1:
        emotion_classifier = build_model('easy')
        emotion_classifier.fit(training_feature, training_yhat, batch_size=128, nb_epoch=40,validation_data=(validation_feature, validation_yhat), callbacks=[ear
    else:
        emotion_classifier = build_model('easy')
        emotion_classifier.fit(training_feature, training_yhat, batch_size=128, nb_epoch =40, validation_data=(validation_feature, validation_yhat), callbacks=[e
        (training_feature, training_yhat, unlabel_feature) = add_unlabel_featrue_to_training(training_feature, training_yhat, unlabel_feature, emotion_classifier
emotion_classifier = build_model('easy')

# save model
emotion_classifier.save('semi_model')
```