**SGN-26006 Advanced Signal Processing Image Recognition**

1. Task

In this task, we are required to use deep learning techniques to build networks processing the Image and categorize them as smile and non-smile two categorizations.

2. Dataset

The datasets are the GENKI-4k face, Expression and Pose Dataset, In this binary classification task, the whole 4000 images with the label as 2162 smiles and 1838 nonsmile are supposed to be categorized into two sets as accurate as possible with the input of size 64\*64\*3. We choose the input size as a power of two.

3. requirements

The topology of the small network is from the input of 64\*64\*3 to 64\*64\*32, 32\*32\*32 and then from 32\*32\*32 to 16\*16\*32 and from 16\*16\*32 to 8\*8\*32, and then flatten it into 1\*2048, dense layer with 128 and the tools included will be tensorflow, keras and OpenCV.

4. Solutions

4.1 Preprocess the dataset to get the training data and test data

In the load\_GE function, each picture of in the .jpg datasets are read through OpenCV and converted into the img matrix which has the size 64\*64\*3 with value between [0, 1]. Then, they are appended into the list imlist. As for the labels, they are read through the labels.txt and also appended into the list labels. Finally, the array imlist and labels are returned.

Notice that, because of the classification is binary, the labels are also binarized using np\_utils function. After shuffle, these two are made into arrays and prepared for fitting and compiling.

One last step is to change augment the pictures, the rotation with 45 degree is implemented along with the width and height shifting 0.15. The brightness is also adjusted into (0.6, 1.0) filling with the ’nearest’ mode.

4.2 Network construction

The network build for the fitting and compiling uses the convolution network with sequential model. The input is of the size 64\*64\*3, kernel size 5\*5. After the max-pooling into half 3 times, the output is flatten and densed by two layers as 128 and 1 separately.

In every convolution layer, the activation function is relu and with the same padding; In every pooing layer, the maxpooling with stride 2 of 2D is implemented; In the dense layer, the output are chosen as 1\*128 processed by relu and 1\*2(because of the binary type) processed by softmax.

Layer (type) Output Shape Param #

=================================================================

conv2d\_101 (Conv2D) (None, 62, 62, 64) 1792

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_49 (MaxPooling (None, 31, 31, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_102 (Conv2D) (None, 31, 31, 32) 51232

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_103 (Conv2D) (None, 31, 31, 32) 25632

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_50 (MaxPooling (None, 15, 15, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_104 (Conv2D) (None, 15, 15, 16) 12816

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_105 (Conv2D) (None, 15, 15, 16) 6416

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_51 (MaxPooling (None, 7, 7, 16) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_106 (Conv2D) (None, 7, 7, 8) 3208

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_15 (Flatten) (None, 392) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_29 (Dense) (None, 128) 50304

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_30 (Dense) (None, 2) 258

=================================================================

Total params: 151,658

Trainable params: 151,658

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

None

4.3 Learning

The whole dataset is split using train\_test\_split with the default ratio at 0.75 as the size of the training data for the first stage. The batch size is chosen to be 16. The model is sequential model connecting the built network the stop iteration condition is the accuracy reaching 0.9 or epochs reaching 50.

For final evaluation, 0.8 of the dataset are set as the training data, and complied by sgd optimizer, loss function as binary\_crossentropy and measured with meatrics accuracy.

5. Result

Finally, with the training dataset being 3200 and test dataset being 800, we have the accuracy as 0.58 with loss as 0.64. The confusion matrix(see Table 1) shows my model’s largest problem in predicting smile expression. The process of label might be wrong, since all the smile pictures are all predicted as non-smile. On contrary, the one non-smile pictures are predicted as all correct.(See figure1)

|  |  |  |
| --- | --- | --- |
| Prediction/GT | Smile | Non-smile |
| Smile | 0 | 345 |
| Non-smile | 0 | 455 |

Table 1

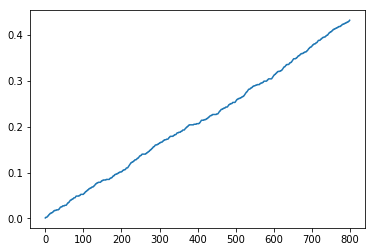
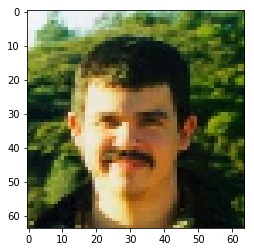
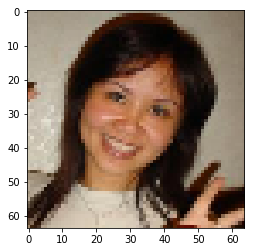
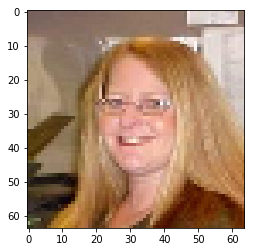
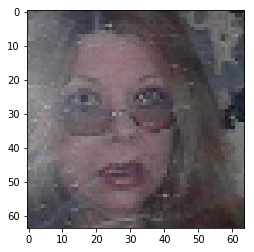
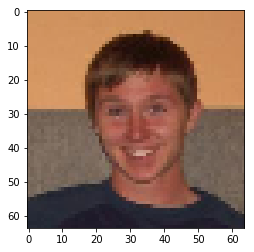
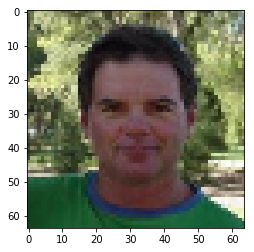


Figure 1

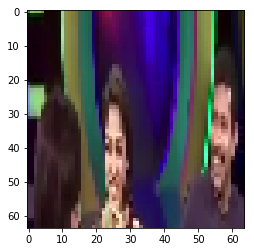
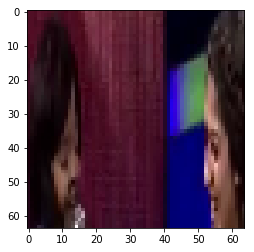
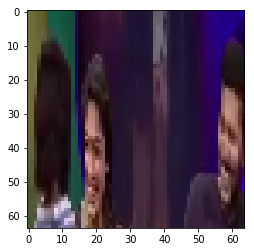
Examples of correct predictions:

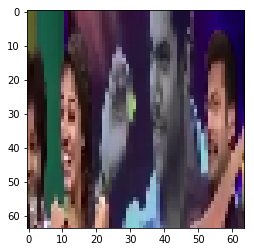
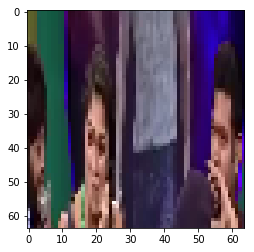
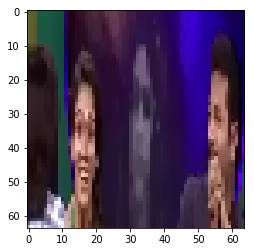


Examples of wrong predictions:



It is basically a failure to use this model to detect smile as the accuracy is really low although given 50 epochs and with good preprocessing. The improvement can be made with using other models instead. For example, the vgg16, adadelta. More layers as drop out can also be added as against overfitting.





Later, the model was changed to adadelta and added the dropout layer(which ratio is 0.8712) with validation data set added.

According to the keras document, the adadelta is the more robust extension of ada grad which adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients. This way, Adadelta continues learning even when many updates have been done. And It is recommended to leave the parameters as the default values.

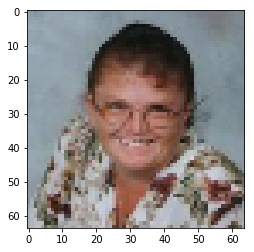
After spliting the trainining data with 0.15 as validation\_data, within 50 epochsthe validation dataset reaches the accuracy as 0.8712(thus it does not stop iteration) and the loss function value as 0.4043. Meanwhile the one on the test dataset reaches the 0.9209.

The confusion matrix is then:

|  |  |  |
| --- | --- | --- |
| Prediction/GT | Smile | Non-smile |
| Smile | 311 | 53 |
| Non-smile | 54 | 382 |

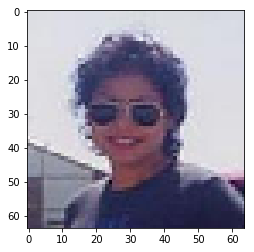
The smile and non-smile correctly detected are 311 and 382 in amount number with the false alarm detecting as 54/364 and 53/435 separately.

Example of the smile dataset:

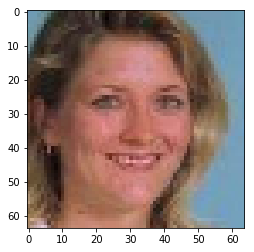


prediction: [[1.6651839e-07 9.9999988e-01]]

prediction: ground true: [0. 1.]



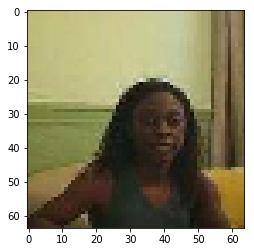
prediction: [[0.01000388 0.9899961 ]]prediction: ground true: [0. 1.]



prediction: [[1.5495857e-04 9.9984503e-01]]

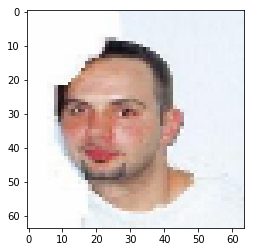
ground true: [0. 1.]

Example of the non-smile dataset:



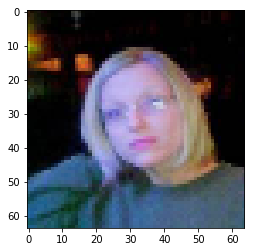
[[0.9885416 0.01145836]]

ground true: [1. 0.]



[[0.9885416 0.01145836]]

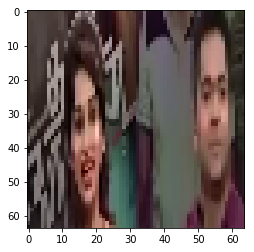
ground true: [1. 0.]



prediction: [[0.97023803 0.02976197]]

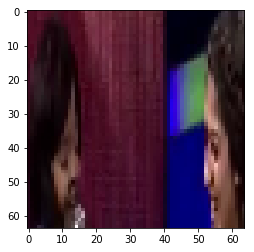
ground true: [1. 0.]

Using the OpenCV to capture the frame with fps as 25 croping the image with the ROI as [25, 25, 120, 400], we can see some of the results:



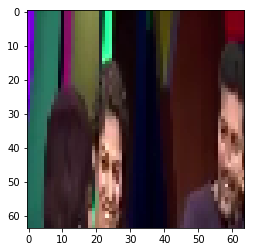
[[9.9997258e-01 2.7361357e-05]]

SMILE



[[1. 0.]]

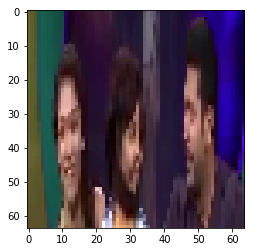
SMILE



[[1. 0.]]

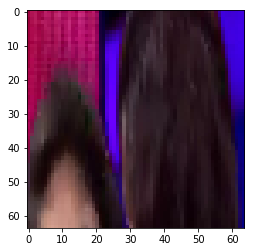
SMILE

Example of the non-smile dataset:



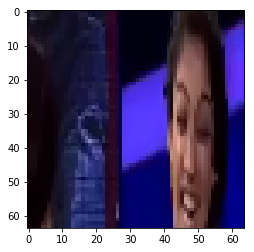
[[3.845949e-09 1.000000e+00]]

NON-SMILE



[[0. 1.]]

NON-SMILE



[[0. 1.]]

NON-SMILE

It can be seen that

Smile part of the picture are detected well however, non-smile part is still with error. This may also be related to my resizing the picture into 64 \*64\*3 for the model we trained before.