SmileRecognition

Supporting Real-time recognition using deeplearning

Project Definition:

We wanted to implement a program which detects the smiles in the pictures So
We will use deep learning tools in order to detect smiling or non-smiling gestures in the pictures

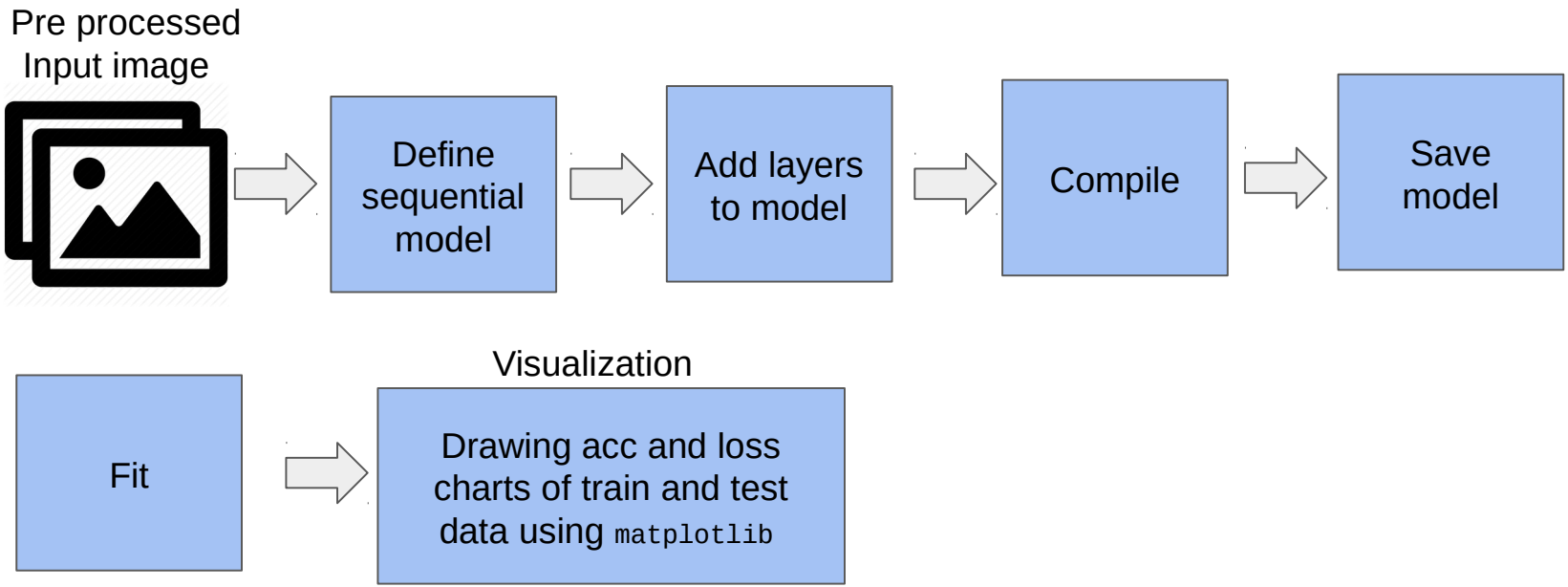
Our project has two main parts :

TRAIN

TEST


TRAINING

Training Steps taken



Train Details

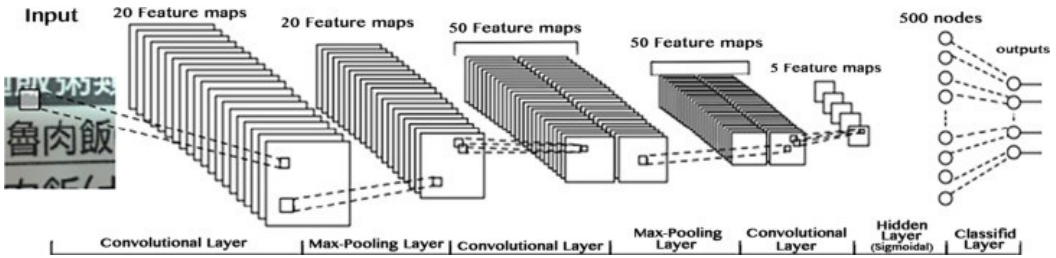
Tools:

- We use “Keras” Deep Learning python tool
- Our backend is “Tensorflow”
- Our class of deep neural networks was “CNN”
- For visualization and showing charts we use “maplotlib”

Train Details

Neural network details

A typical architecture of a convolutional neural network will contain an input layer, some convolutional layers, some dense(fully-connected) layers



Train Details

Neural network details

Compilation:

Configures the model for training.

Compiling is done using “Adadelta” **optimizer**

And the **metric** is “accuracy”

Train Details

Neural network details

Fitting :

Trains the model for a given number of epochs (iterations on a dataset)

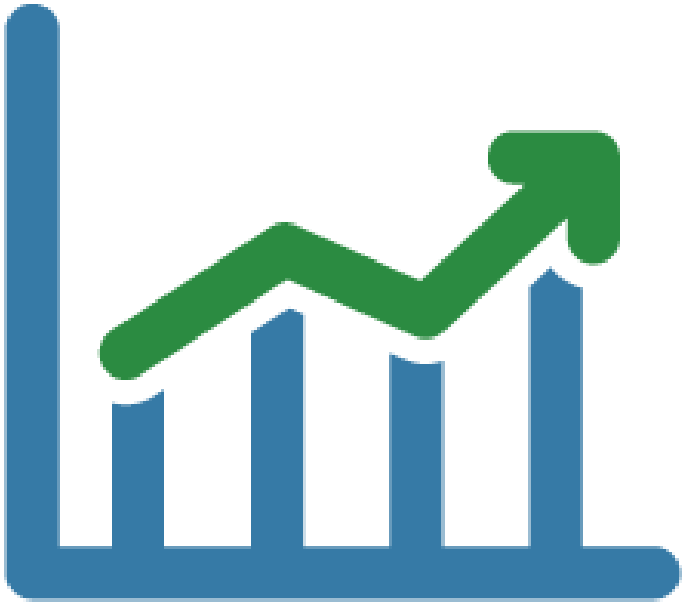
epochs = 50

batch_size = 128

* call_back = [checkpointer]

* validation_data = x_dev, y_dev

MODEL VALIDATION



Train Details

Neural network details

We have 3 conv2D layers and 2 dense layers (one fully connected layer with an output) using dropout in order to prevent overfitting

We use maxpooling2D with size of 2*2 windows for all layers

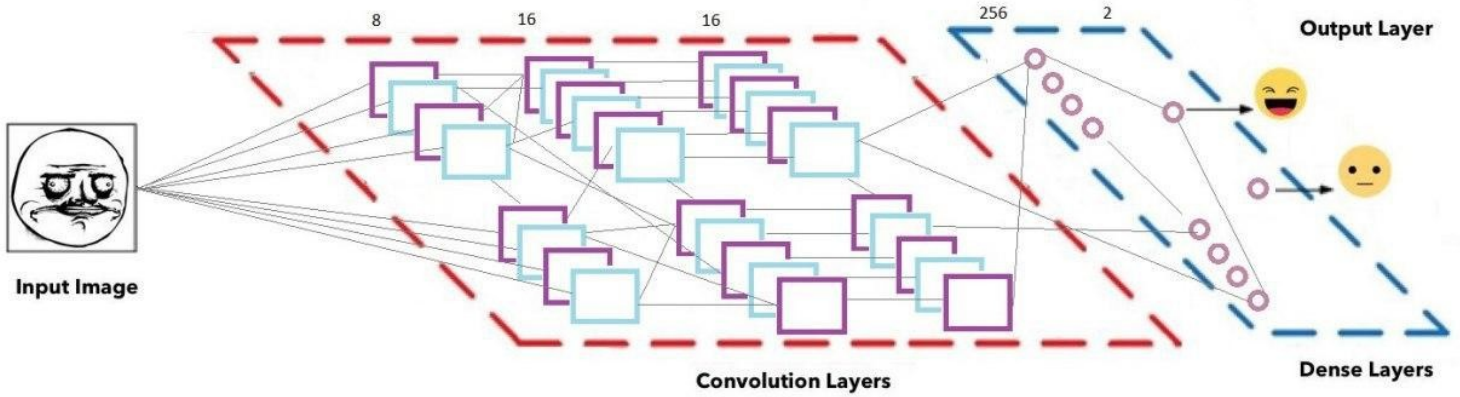
Filters and Sizes:

Using 8, 16, 16 filters in order(the numbers were given by experiment)

And also our activation function is “relu”

Train Details

Neural network structure



Saving Model

By keras callback library using ModelCheckpoint we save our model after every epoch

We save the best model in these 50 epochs according to high accuracy and low loss

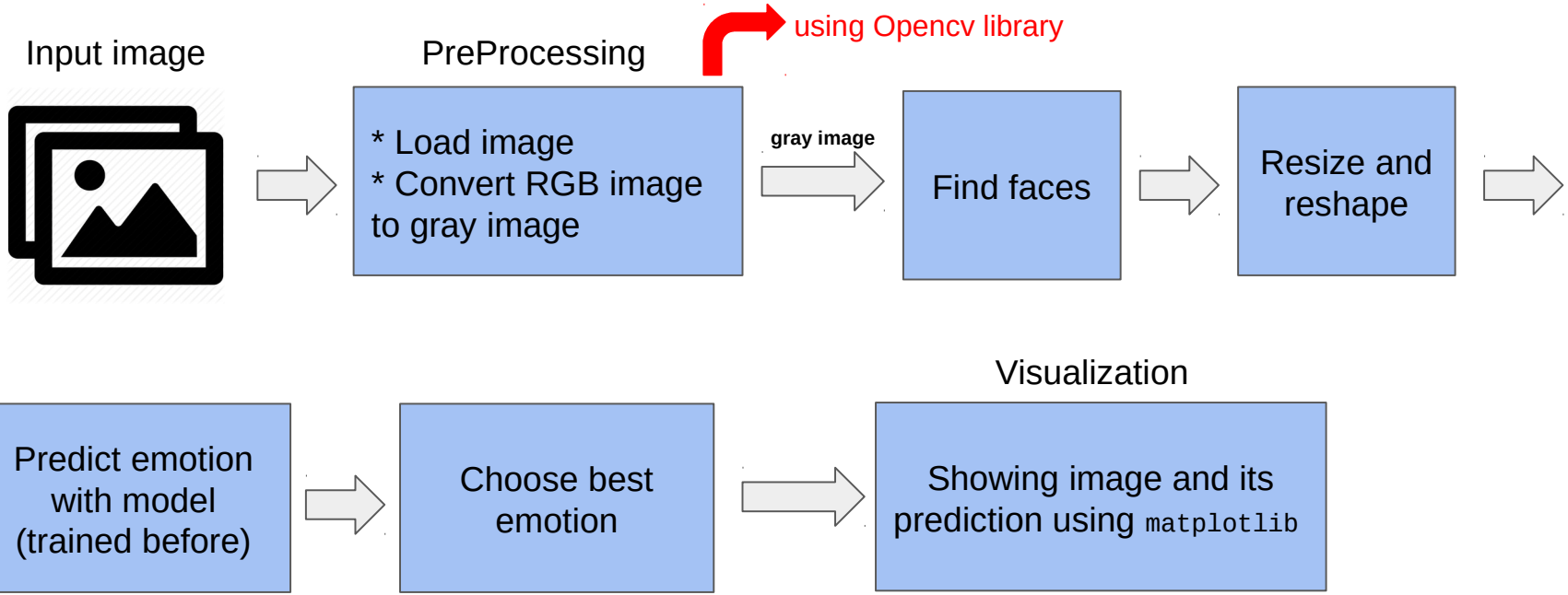
the function looks like this :

```
checkpointer = keras.callbacks.ModelCheckpoint(
   'models/model.h5',
   save_best_only = True,
   monitor = 'val_acc',
   mode = 'auto',
   verbose = 1
)
```

Actually we have two classes smile and not-smiling.

Prediction level

Prediction Steps taken




CHALLENGE

We were not sure about the numeric metrics of our code
Of course Nobody is !

for example: number of filters applied to conv2D layers

So we had to

CHANGE and TEST

and

CHANGE and TEST

and

….

Checking out an example:

at first we used (16, 32, 32) filters in order for each layer and the result after 50 epochs was this:

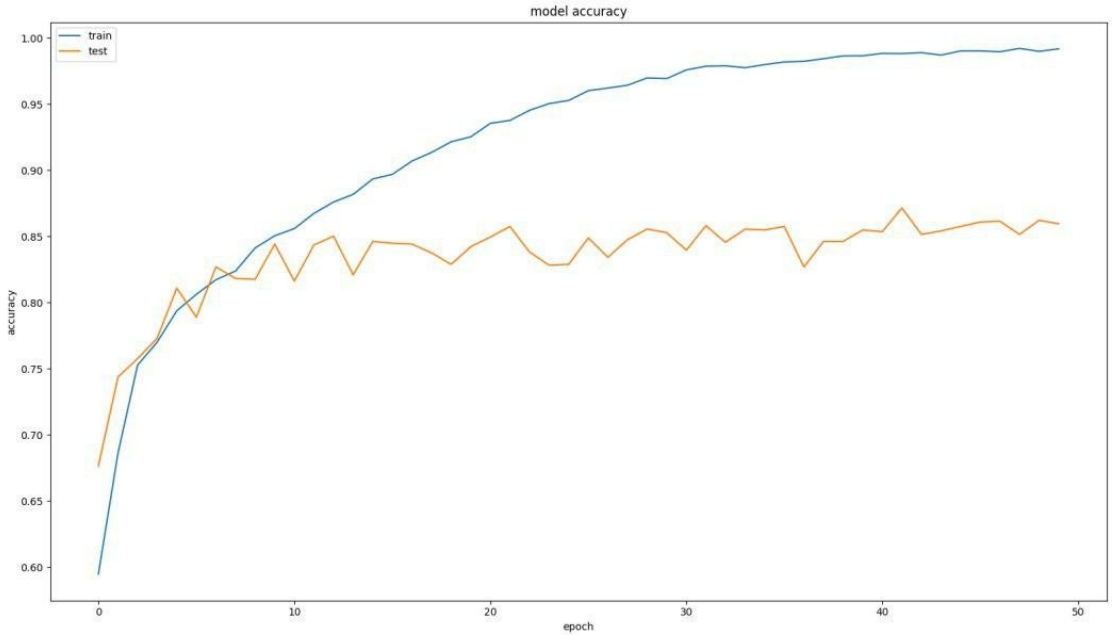
```
y_test = keras.utils.to_categorical(y_test, num_classes)

# initializing model
model = Sequential()
model.add(Conv2D(16, kernel_size=(5, 5), activation='relu', input_shape=(48, 48, 1)))
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

The previous code

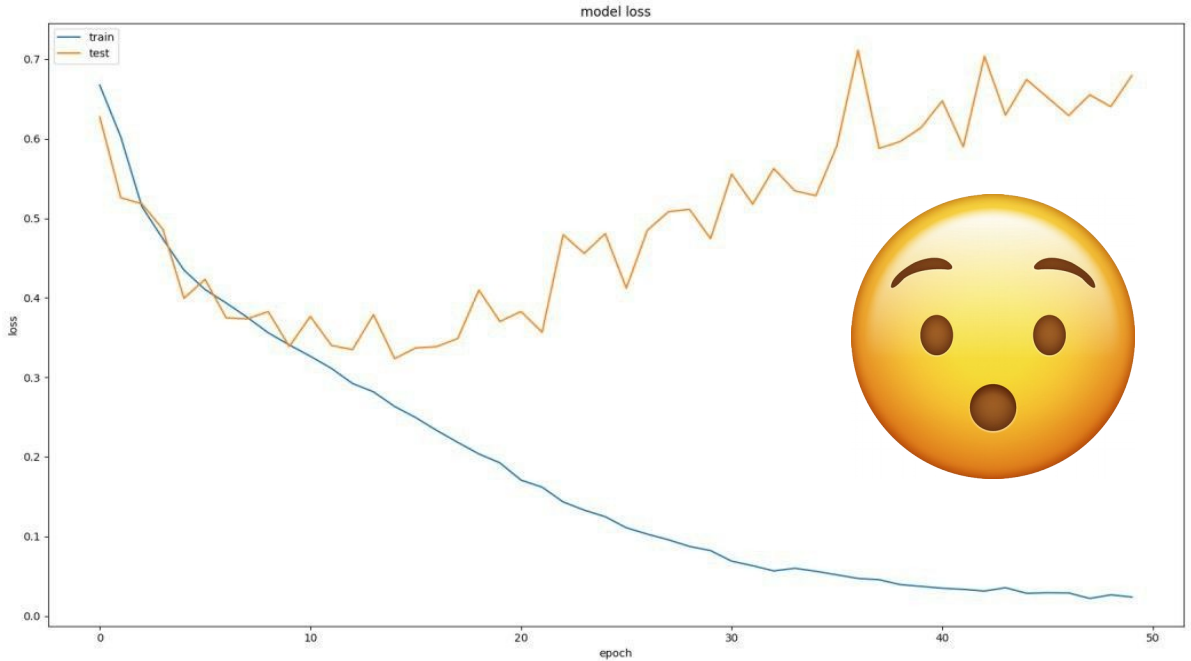
Charts of accuracy (in case of (16,32,32))



num_of_epochs = 50

As you can see it’s not good enough
the test_data has about 15% difference with train_data

Charts of loss (in case of (16,32,32))



num_of_epochs = 50

This state is almost terrible
our test_data loss is BAD and more than train_data

The previous code details are here

12180/12180 [==============================] - 14s 1ms/step - loss: 0.0633 - acc: 0.9783 - val_loss: 0.4627 - val_acc: 0.8609

Test loss: 0.6608016912129234
Test accuracy: 0.8644518272425249

Properties of this network against our desired one:

- **more loss**
- **less accuracy**
- **more processes**
- **bigger network**
- **worse test accuracy**


CHALLENGE
SOLUTION

The solution

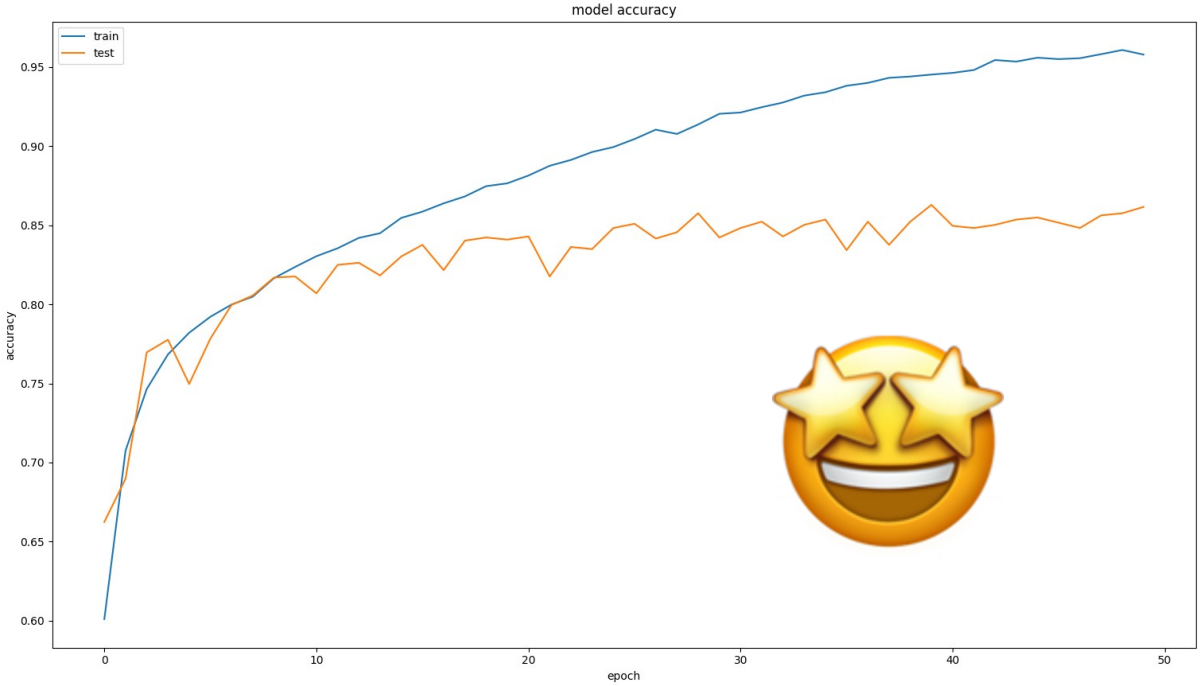
then after testing a lot of numbers we used (8,16,16) filters in order for each layer and the result after 50 epochs was this:

```
# initializing model
model = Sequential()
model.add(Conv2D(8, kernel_size=(5, 5), activation='relu', input_shape=(48, 48, 1)))
model.add(Conv2D(16, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(16, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

The current code

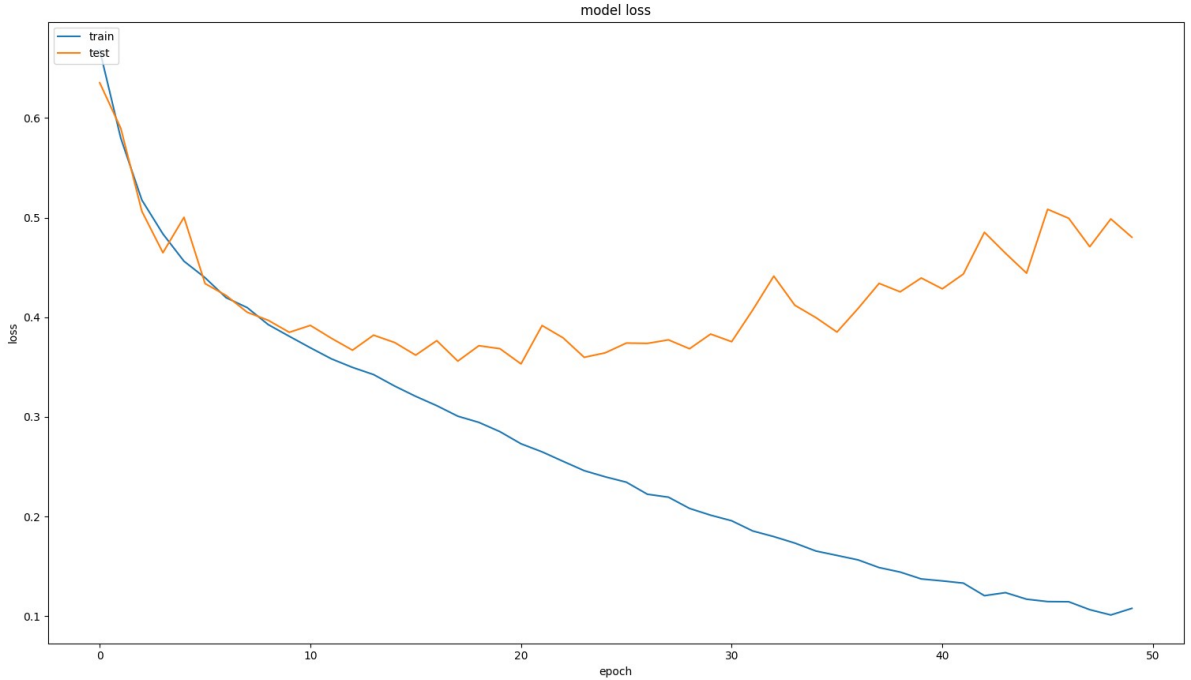
Charts of accuracy (in case of (8,16,16))



num_of_epochs = 50

As you can see this is arelly steady state
And it's optimal

Charts of loss (in case of (8,16,16))



num_of_epochs = 50

Actually not bad and best in our cases

Dataset

images size: 48*48

Train set: 80% (12180)

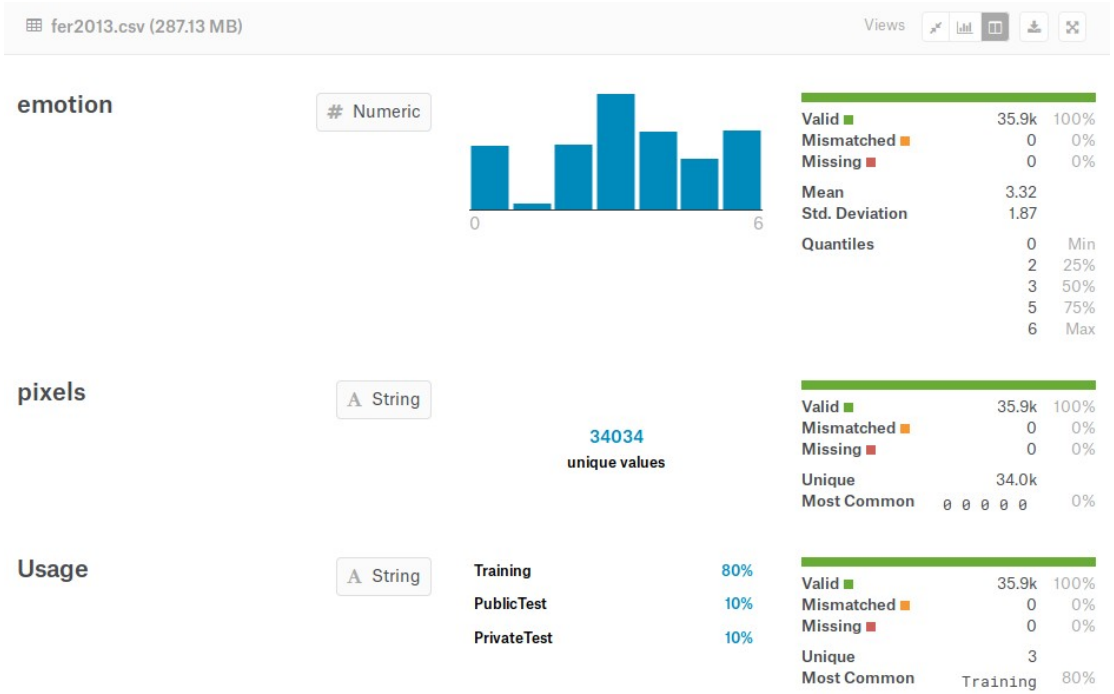
PublicTest set: 10%(1502)

PrivateTest set: 10%(1505)

fer2013 kaggle

https://www.kaggle.com/deadskull7/fer2013

fer2013 dataset



	# emotion	A pixels	A Usage
	 0 6	34034 unique values	Training 80% PublicTest 10% Other (1) 10%
1	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121 119 115 110 98 91 84 84 90 99 110 126 143 153 158 171 169 172 169 165 129 110 113 107 95 79 66 62 56 57 61 52 43 41 65 61 58 57 56 69 75 70 65 56 54 105 ...	Training
2	0	151 150 147 155 148 133 111 140 170 174 182 154 153 164 173 178 185 185 189 187 186 193 194 185 183 186 180 173 166 161 147 133 172 151 114 161 161 146 131 104 95 132 163 123 119 129 140 120 151 149 1...	Training

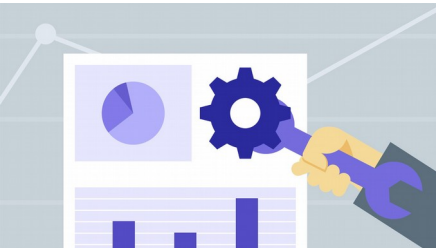
Data set entries

Analysis

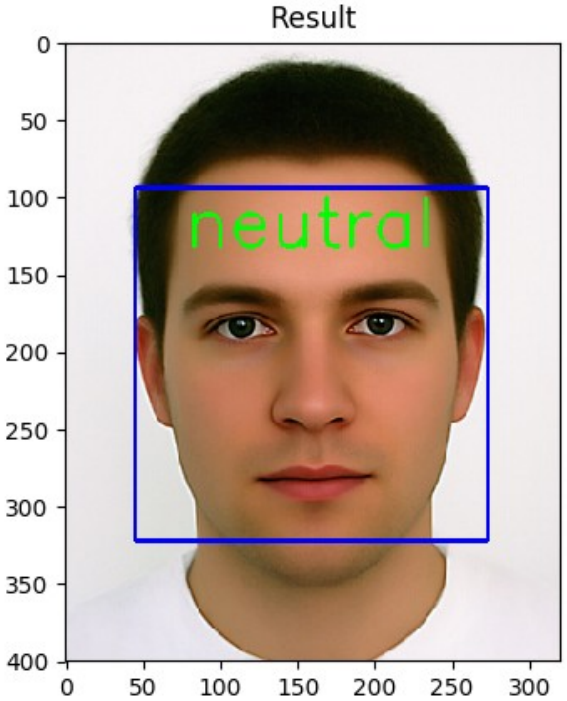
Analysis

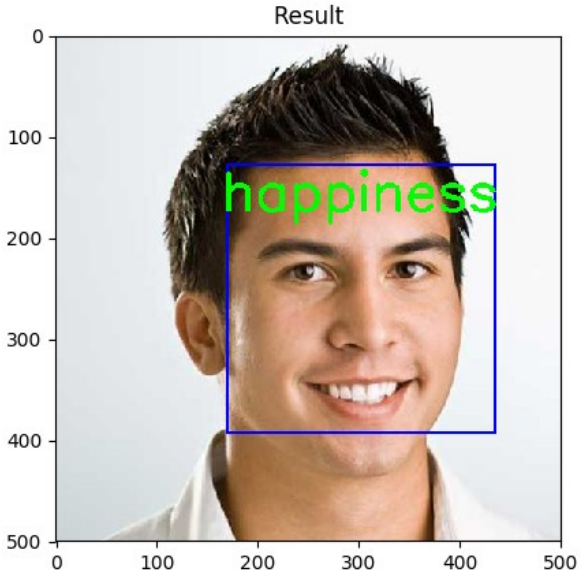
Our final results are given here :

	accuracy	loss
Train	95.78	10.78
Dev	86	48.04
Test	87.97	42.17

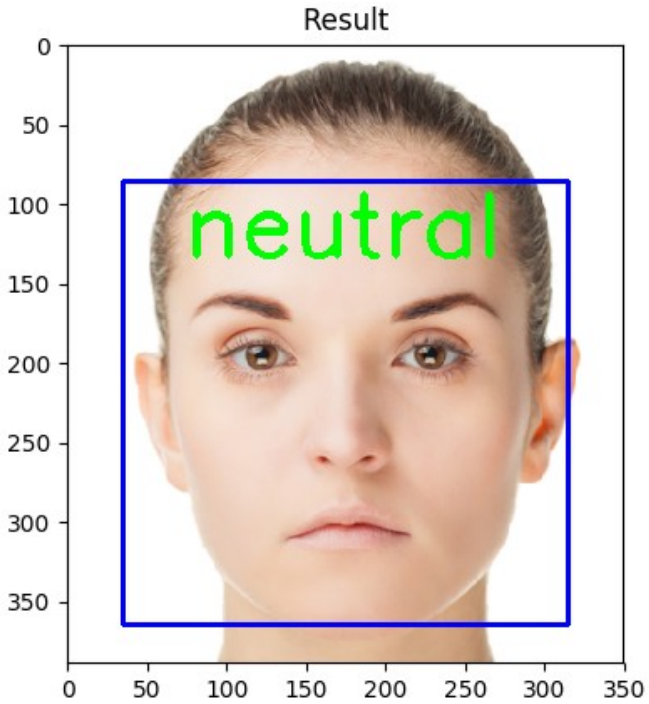


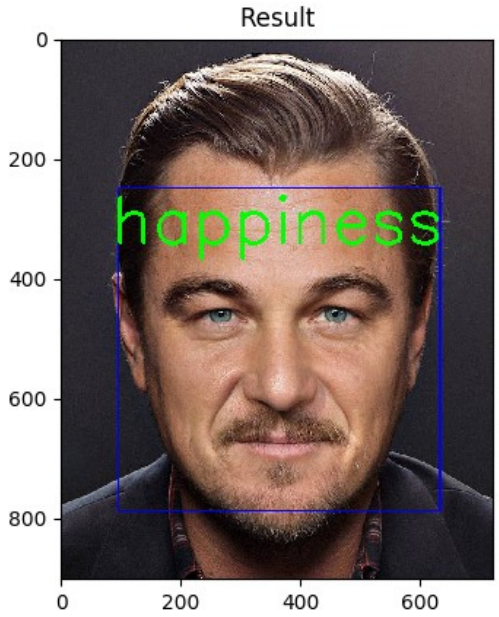
Testing (valid datas)





Testing (valid datas)





Invalid datas !!

This image is not valid because the face is not complete in the picture

Opencv can’t recognize the face !!

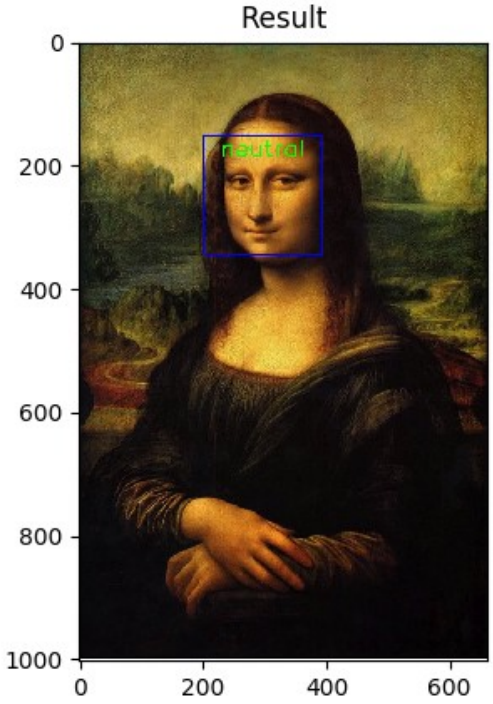
Valid datas which get the wrong answer

Mona Lisa !!!

Valid data which get the wrong answer

But she is almost happy and smiling !!!

Reason : we have 5 percent error according to our accuracy



Mistakes 😐

Using a similar code to previous parts we have a mistake.py file which recognizes which images of data-set give us the wrong result

It makes us sure about calculations.

Mistakes

We have categorized wrong recognized pictures in three directories(train-dev-test) and each of them contain folders named **“wh”** and **“wn”** (in our project directory)

wh ---> pictures that were happy but said to be netura

wn ---> pictures that were netural but said to be happ

You can see the examples on the next page

Train Mistakes (smiling)

total number of wh images = 71

Train Mistakes (not smiling)

total number of wn images = 102

Calculations (Train)

total train dataset = 12180

total mistakes = wh + wn = 71 + 102

error percentage = (173/12180)*100 = 1.42

correctness = 100 - 1.42 = 98.58

train accuracy = 95.78

Difference Reason : while training the acc of each batch is “average acc” and this is why we have 2-3 percent difference here

Test Mistakes (smiling)

total number of wh images = 80

Train Mistakes (not smiling)

total number of wn images = 113

Calculations (Test)

total train dataset = 1505

total mistakes = wh + wn = 80+113

error percentage = (193/1505)*100 = 12.82

Correctness = 100 – 12.82 = 87.18

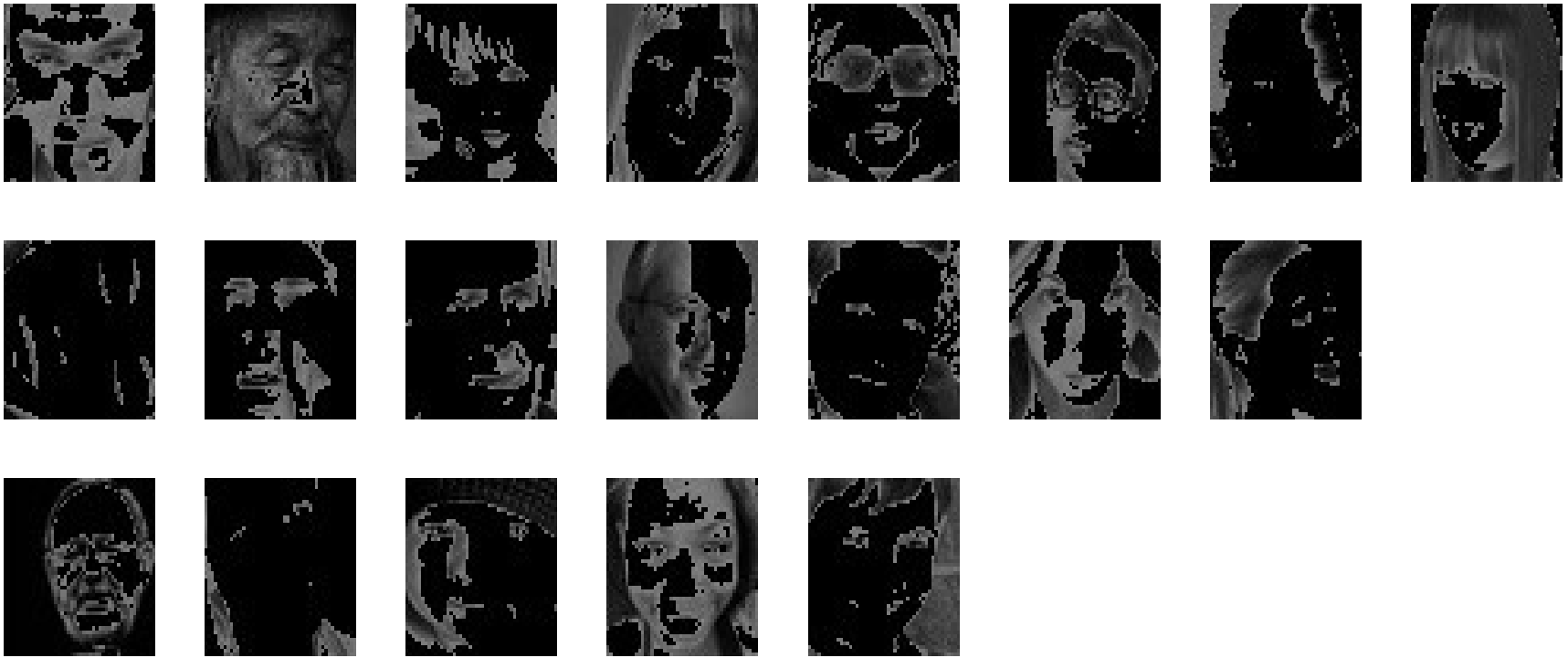
train accuracy = 87.97

WOW ...Almost exactly the same

Dev Mistakes (smiling)

total number of wh images = 94

Dev Mistakes (not smiling)



total number of wn images = 112

Calculations (Dev)

total train dataset = 1502

total mistakes = wh + wn = 94 + 112

error percentage = (206/1502)*100 = 13.71

correctness = 100 - 13.71 = 86.29

train accuracy = 86 🤩

Filters

We visualized the 8 filters of first layers as you can see below



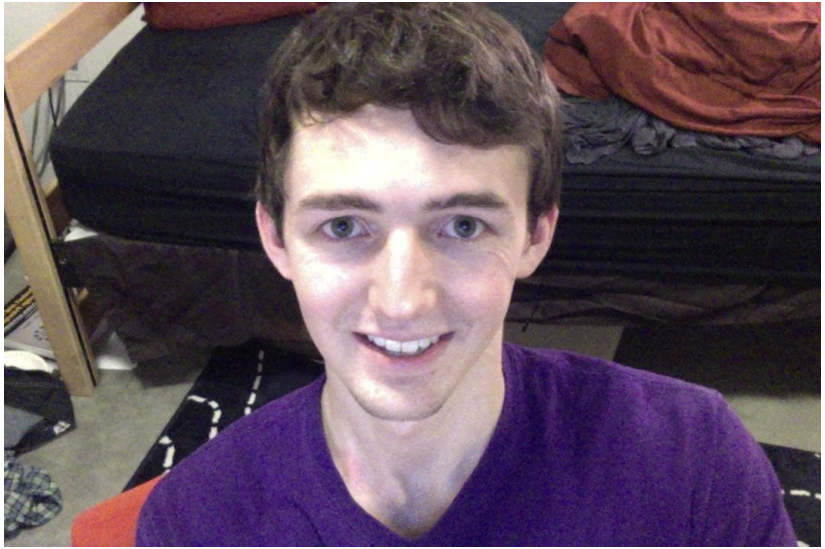





Filters

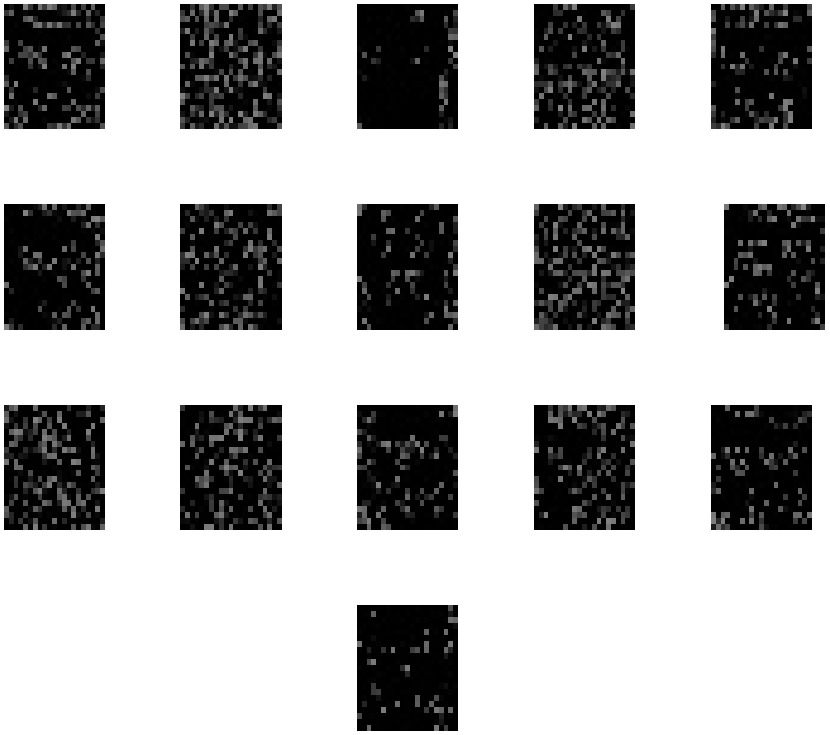
We will see the filters applied to the following image on each layer of network in the next slides

The idea of code is using model.get_weights() and visualizing it



Filters

8 filters of first layer



16 filters of second layer

Filters

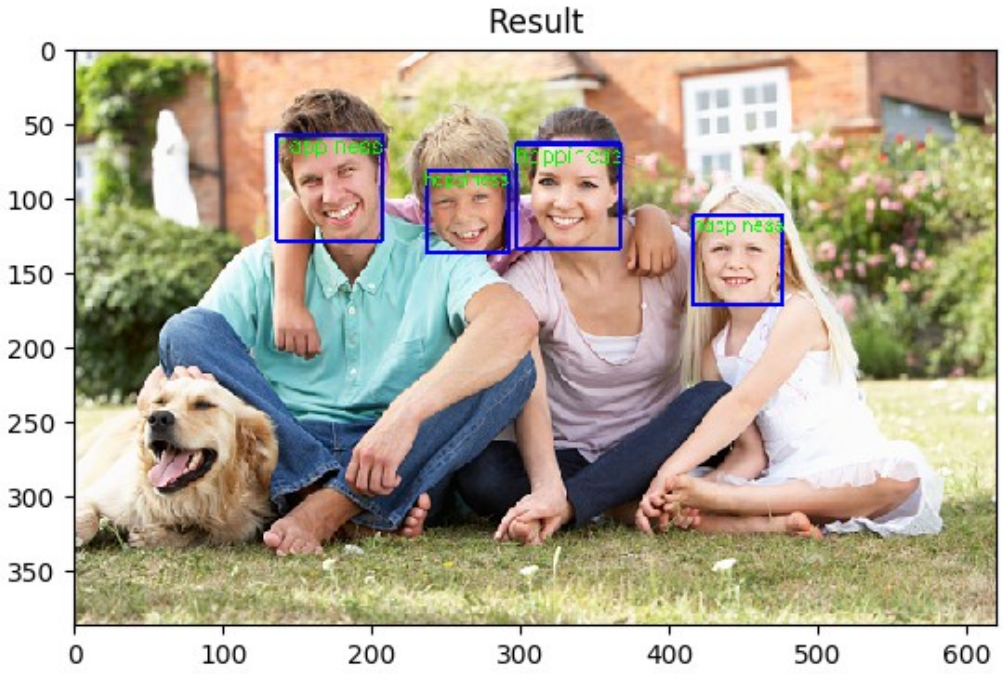
16 filters of third layer

Performance and Speed

image size : 640 * 480

number of faces = 4

max response_time = 0.18397s



Performance and Speed

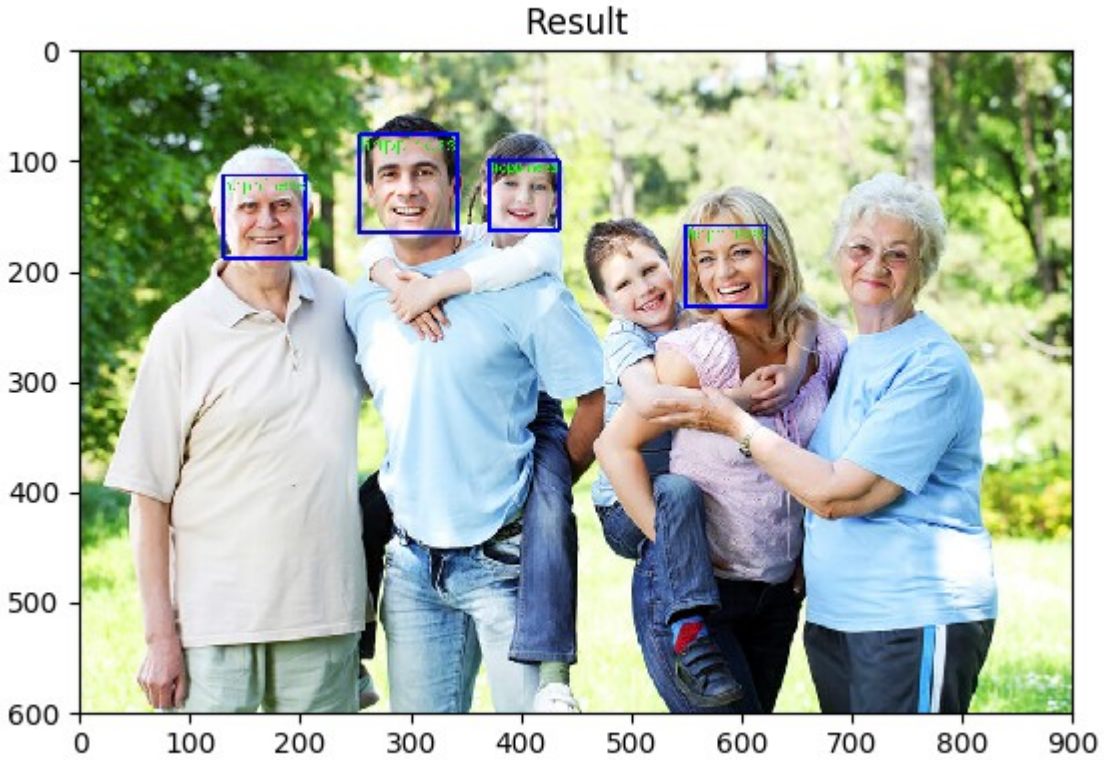
image size : 640 * 480

number of faces = 4

max response_time = 0.3157s

Reason of not detection :

face angles

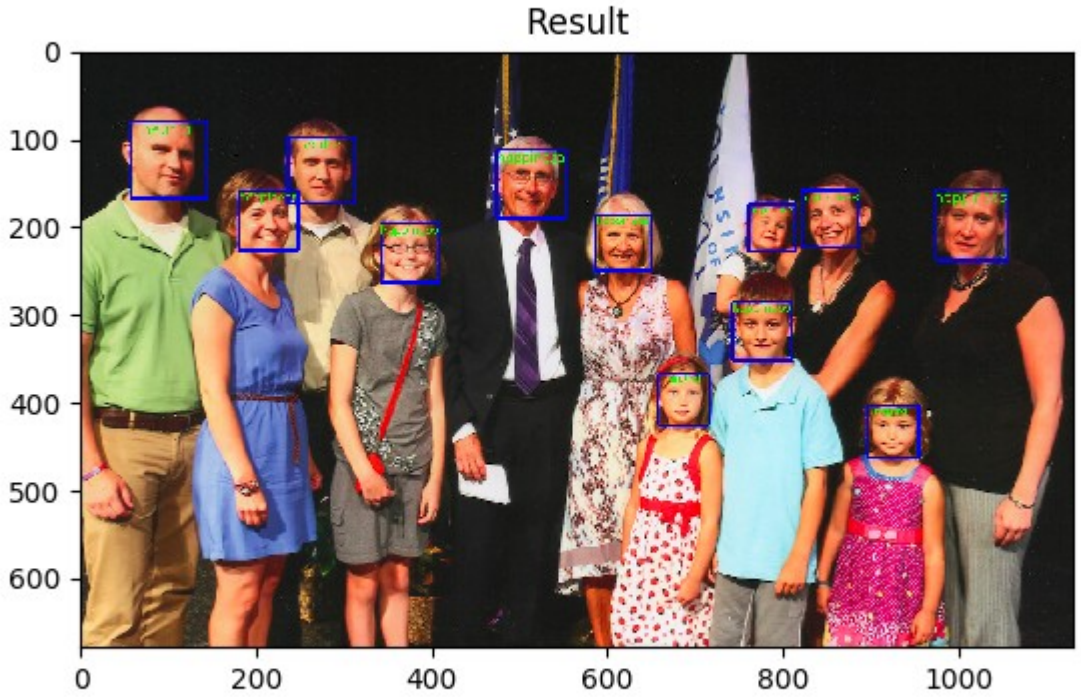


Performance and Speed

image size : 1131* 679

number of faces = 12

max response_time = 0.4209s



Good news ^__^

Our program also supports using Webcam in real-time recognition

By running webcam.py

Response Time of RealTime system

with an average time of 0.11053 s in 30 seconds testing

You can access to our project from link below:

https://github.com/zahrabashir98/SmileDetection/

ANY QUESTIONS? :))

Thanks for your attention and time

^_^