Facial Expression Recognition System

Progress Report

In fulfillment of the requirements for the

NU 302 R&D Project

At NIIT University

Submitted by

Mrinal Sharma

Akshay Kumar Raut

Dhruv Parashar

Nishi Chandra

Kumar Aniket

Area

NIIT University

Neemrana

Rajasthan

CERTIFICATE

*This is to certify that the present research work entitled "**Facial Expression Recognition System**" being submitted to NIIT University, Neemrana, Rajasthan, in the fulfillment of the requirements for the course at NIIT University, Neemrana, embodies authentic and faithful record of original research carried out by **Mrinal Sharma**/s, **Dhruv Parashar**/s, **Nishi Chandra**/s, **Akshay Kumar Raut**/s, **Kumar Aniket**/s, student/s of B Tech (**CSE/ECE**),at NIIT University, Neemrana,. She /He has worked under our supervision and that the matter embodied in this project work has not been submitted, in part or full, as a project report for any course of NIIT University, Neemrana or any other university.*

Name and Title of the Mentor: Prof. Abdul Mazid

LIST OF FIGURES

LIST OF TABLES

CONTENTS

Rational

- To propose a system for better human-computer interaction.
- To provide robustness and high accuracy to the current systems.
- To cope up with variability in the environment and adapt to real time scenarios.

Introduction

The success of service robotics decisively depends on a smooth robot to user interaction. Thus, a robot should be able to extract information just from the face of its user, e.g. identify the emotional state or deduce gender. Interpreting correctly any of these elements using machine learning (ML) techniques has proven to be complicated due the high variability of the samples within each task. This leads to models with millions of parameters trained under thousands of samples. Furthermore, the human accuracy for classifying an image of a face in one of 7 different emotions is 65% _ 5%. One can observe the difficulty of this task by trying to manually classify the FER-2013 dataset images in within the following classes {“angry”, “disgust”, “fear”, “happy”, “sad”, “surprise”, “neutral”} .

Figure 1: FERC-2011 Dataset

In spite of these difficulties, robot platforms oriented to attend and solve household tasks require facial expressions systems that are robust and computationally efficient. Moreover, the state-of-the-art methods in image-related tasks such as image classification and object detection are all based on Convolutional Neural Networks (CNNs). These tasks require CNN architectures with millions of parameters; therefore, their deployment in robot platforms and real-time systems becomes unfeasible. In this paper we propose an implement a general CNN building framework for designing real-time

CNNs. The implementations have been validated in a real-time facial expression system that provides face-detection, gender classification and that achieves human-level performance when classifying emotions.

Furthermore, CNNs are used as black-boxes and often their learned features remain hidden, making it complicated to establish a balance between their classification accuracy and unnecessary parameters.

Figure 2: IMDB Dataset

Literature Review

1)

Title: Facial Expression Recognition using Weighted Mixture Deep Neural Network Based on two channel Facial Images

Author: Biao Yang, Jinmeng Cao, Rongrong Ni, Yuyu Zhang

Journal: JOURNAL OF LATEX CLASS FILES, VOL. 14, NO. 8, AUGUST 2015

Method: This study proposes a FER method based on WMDNN that can process facial grayscale and LBP facial images simultaneously.It takes about 1.3s to process a facial image, including 0.5s for pre-processing and 0.8s for recognizing different expressions.

2)

Title: Image based Static Facial Expression Recognition with Multiple Deep Network Learning

Author: Zhiding Yu,

Cha Zhang

Journal: ICMI'15 Proceedings of the 2015 ACM on International Conference on Multimodal Interaction

Method: This paper focusses on the task of image based static facial expression recognition on SFEW with deep CNNs.

3)

Title: A Study of Techniques for Facial Detection and Expression Classification

Author: G.Hemalatha , C.P. Sumathi

Journal: International Journal of Computer Science & Engineering Survey (IJCSES) Vol.5, No.2, April 2014

Method: This paper analysis the facial expression recognition with various methods of facial detection, facial feature extraction and classification.

4)

Title: A Study of Techniques for Facial Detection and Expression Classification

Author: G.Hemalatha , C.P. Sumathi

Journal: International Journal of Computer Science & Engineering Survey (IJCSES) Vol.5, No.2, April 2014

Method: This paper analysis the facial expression recognition with various methods of facial detection, facial feature extraction and classification.

4)

Title: Facial Expression Recognition using Convolutional Neural Networks: State of the Art

Author: Christopher Pramerdorfer, Martin Kampel

Journal: Computer Vision Lab, TU Wien Vienna, Austria

Method: This paper reviews the state of the art in image-based facial expression recognition using CNN. Using ensemble of modern deep CNNs, test accuracy of 75.2% is obtained on FERC=2013

5)

Title: DeXpression: Deep Convolutional Neural Network for Expression Recognition

Author: Peter Burkert, Felix Trier, Muhammad Zeshan Afzal, Andreas Dengel and Marcus Liwicki.

Journal: German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany ,University of Kaiserslautern, Gottlieb-Daimler-Str. Kaiserslautern Germany

Method: In this paper, DeXpression named neural network is presented which works fully automatically and has little computational effort compared to other state of the art CNN architectures.

Objectives

- Automatic facial expression recognition system for characterizing 6 basic emotions namely –

1)	Disgust
2)	Anger
3)	Happiness
4)	Sadness
5)	Surprise
6)	Fear

Table 1: Six Basic Emotions

- It is designed to be person independent and tailored only for static images.

Methodology

Database	Anger	Disgust	Fear	Happiness	Neutral	Sadness	Surprise
MultiPie	0	22696	0	47338	114305	0	19817
MMI	1959	1517	1313	2785	0	2169	1746
CK+	45	59	25	69	0	28	83
FERA	1681	0	1467	1882	0	2115	0
SFEW	104	81	90	112	98	92	86
FERC-2013	4953	547	5121	8989	6198	6077	4002

Table 2: different datasets

Step 1: Resizing the image and converting it to grayscale.

An RGB Image consists of 3 layers R,G,B as it is clearly see through its name. It's a 3 dimensional matrix, for example, 3 consecutive pages in your book. where grayscale image is of only 2 dimensions, and the values ranges between 0–255 (8-bit

unsigned integers). Therefore, some algorithms can only applied on 2-D image rather than 3-D, hence we convert an RGB image into a grayscale image, for instance, Black and White conversion of an image, convolution of an image, etc.

Step 2: Face detection using frontal face haar classifiers.

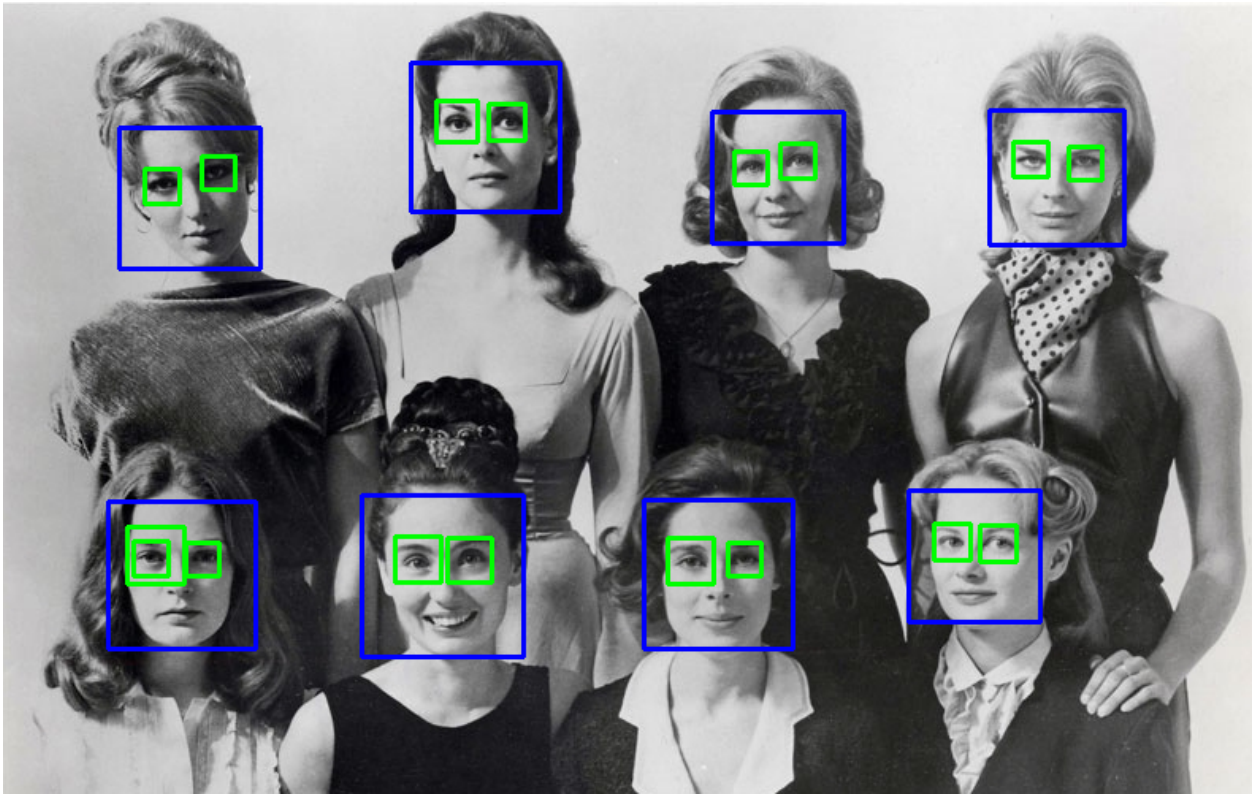
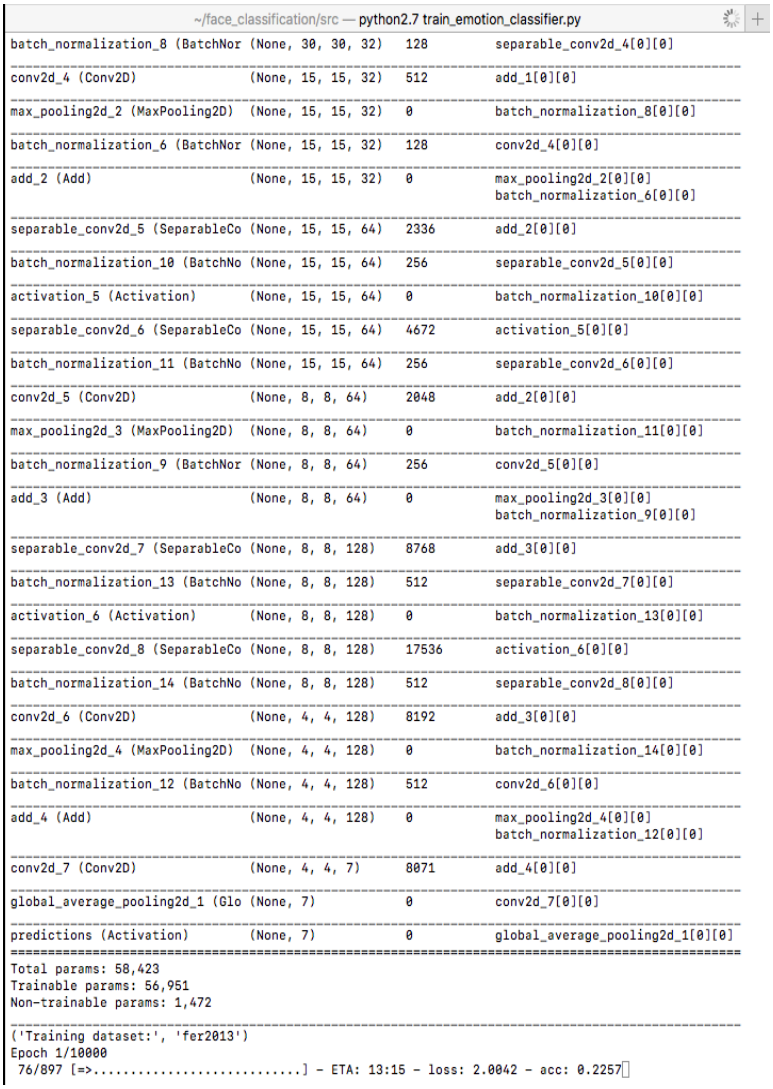


Image Source: http://blog.tibarazmi.com/wp-content/uploads/faceeyedection.png
Figure 3: Face Detection using HAAR Classifier

Step 3: Training the CNN classifier for FERC-2013 and IMDB dataset.



~/face_classification/src — python2.7 train_emotion_classifier.py

batch_normalization_8 (BatchNor	(None, 30, 30, 32)	128	separable_conv2d_4[0][0]
conv2d_4 (Conv2D)	(None, 15, 15, 32)	512	add_1[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 15, 15, 32)	0	batch_normalization_8[0][0]
batch_normalization_6 (BatchNor	(None, 15, 15, 32)	128	conv2d_4[0][0]
add_2 (Add)	(None, 15, 15, 32)	0	max_pooling2d_2[0][0] batch_normalization_6[0][0]
separable_conv2d_5 (SeparableCo	(None, 15, 15, 64)	2336	add_2[0][0]
batch_normalization_10 (BatchNo	(None, 15, 15, 64)	256	separable_conv2d_5[0][0]
activation_5 (Activation)	(None, 15, 15, 64)	0	batch_normalization_10[0][0]
separable_conv2d_6 (SeparableCo	(None, 15, 15, 64)	4672	activation_5[0][0]
batch_normalization_11 (BatchNo	(None, 15, 15, 64)	256	separable_conv2d_6[0][0]
conv2d_5 (Conv2D)	(None, 8, 8, 64)	2048	add_2[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 64)	0	batch_normalization_11[0][0]
batch_normalization_9 (BatchNor	(None, 8, 8, 64)	256	conv2d_5[0][0]
add_3 (Add)	(None, 8, 8, 64)	0	max_pooling2d_3[0][0] batch_normalization_9[0][0]
separable_conv2d_7 (SeparableCo	(None, 8, 8, 128)	8768	add_3[0][0]
batch_normalization_13 (BatchNo	(None, 8, 8, 128)	512	separable_conv2d_7[0][0]
activation_6 (Activation)	(None, 8, 8, 128)	0	batch_normalization_13[0][0]
separable_conv2d_8 (SeparableCo	(None, 8, 8, 128)	17536	activation_6[0][0]
batch_normalization_14 (BatchNo	(None, 8, 8, 128)	512	separable_conv2d_8[0][0]
conv2d_6 (Conv2D)	(None, 4, 4, 128)	8192	add_3[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 4, 4, 128)	0	batch_normalization_14[0][0]
batch_normalization_12 (BatchNo	(None, 4, 4, 128)	512	conv2d_6[0][0]
add_4 (Add)	(None, 4, 4, 128)	0	max_pooling2d_4[0][0] batch_normalization_12[0][0]
conv2d_7 (Conv2D)	(None, 4, 4, 7)	8071	add_4[0][0]
global_average_pooling2d_1 (Glo	(None, 7)	0	conv2d_7[0][0]
predictions (Activation)	(None, 7)	0	global_average_pooling2d_1[0][0]

Total params: 58,423
Trainable params: 56,951
Non-trainable params: 1,472

('Training dataset:', 'fer2013')
Epoch 1/10000
76/897 [=>.....] - ETA: 13:15 - loss: 2.0042 - acc: 0.2257

Figure 4: Training Datasets

Step 4: Feeding the image to the classifier to get the output.

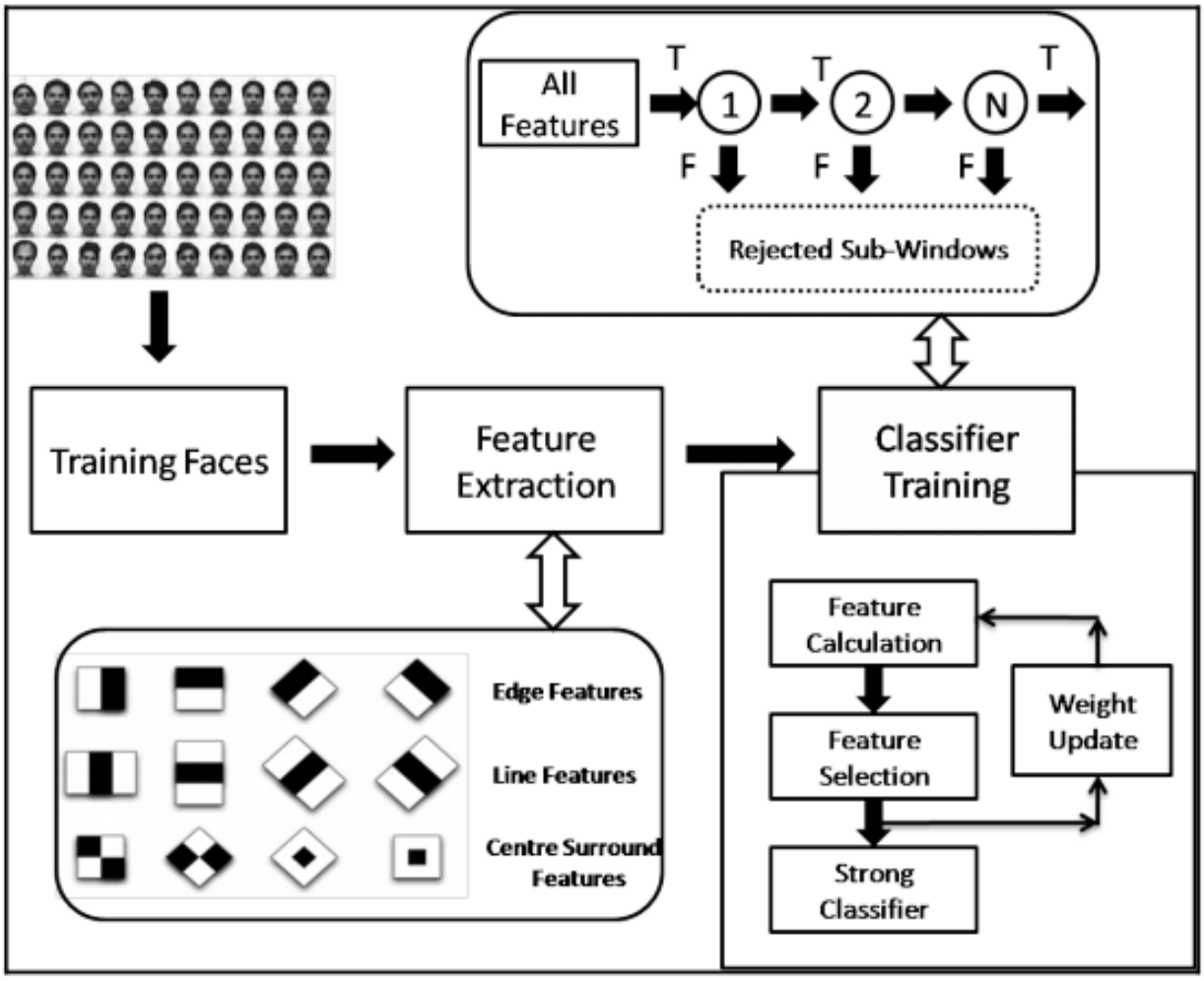


Image Source: https://datahub.packtpub.com/wp-content/uploads/2018/02/image7-1.png
Figure 5: Image Feeding Procedure

CNN Model

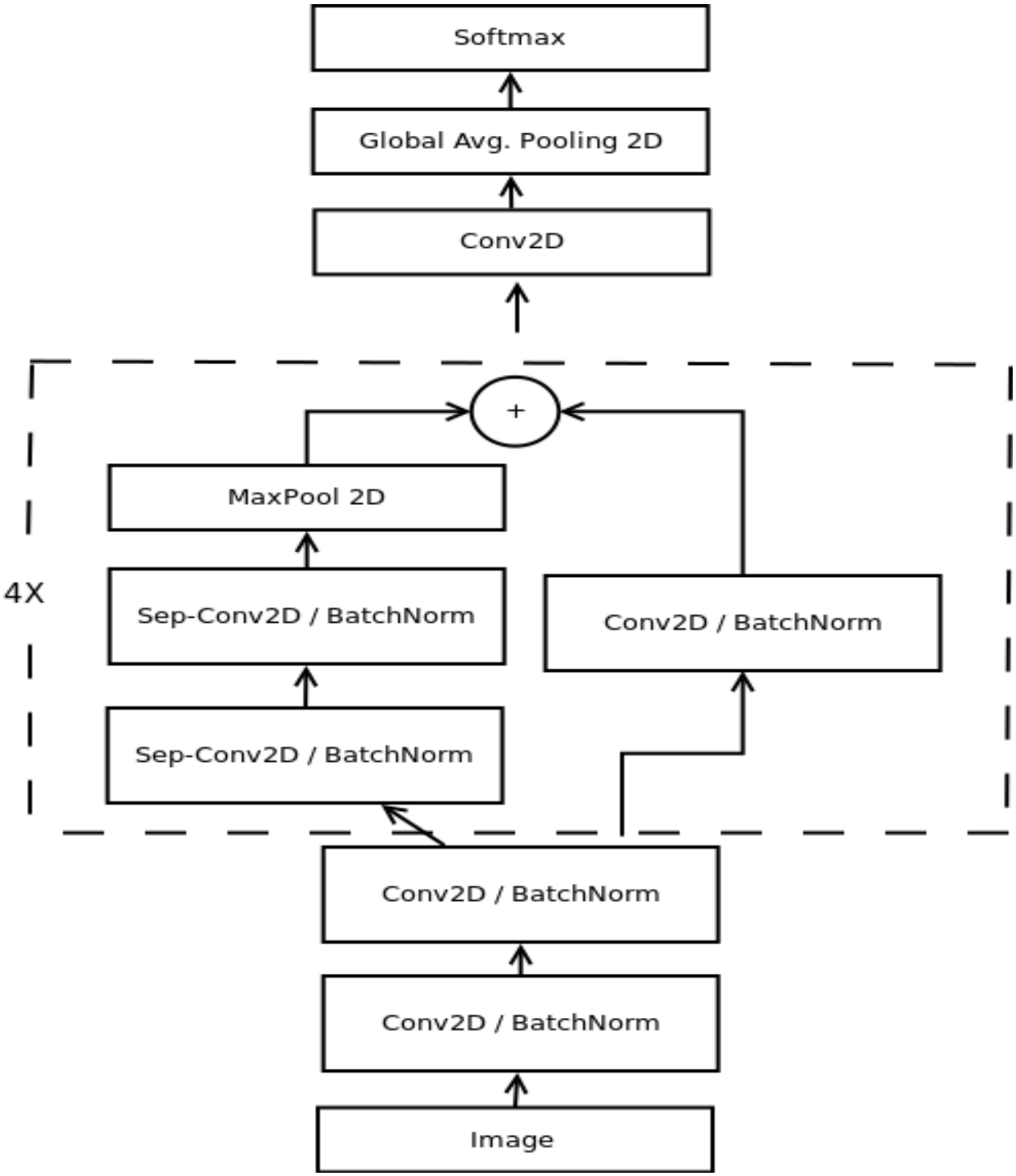


Figure 6: Proposed CNN Model

Results

Results of the real-time emotion classification task in unseen faces can be observed in Figure 5. Our complete real time pipeline including: face detection, emotion and gender classification have been fully integrated.

An example of our complete pipeline can be seen in Figure 8 in which we provide emotion and gender classification.

In Figure 19 we provide the confusion matrix results of our emotion classification mini-Xception model. We can observe several common misclassifications such as predicting “sad” instead of “fear” and predicting “angry” instead “disgust”.

Image Source: https://4.bp.blogspot.com/-PA2HL22Xylk/VIvk5dGW7KI/AAAAAAAAJ10/b-XzcYiZB54/s1600/The%2BSolvay%2BConference%2C%2Bprobably%2Bthe%2Bmost%2Bintelligent%2Bpicture%2Bever%2Btaken%2C%2B1927%2B(1).jpg

Figure 7: Solvay Conference Face emotion & Gender Classification

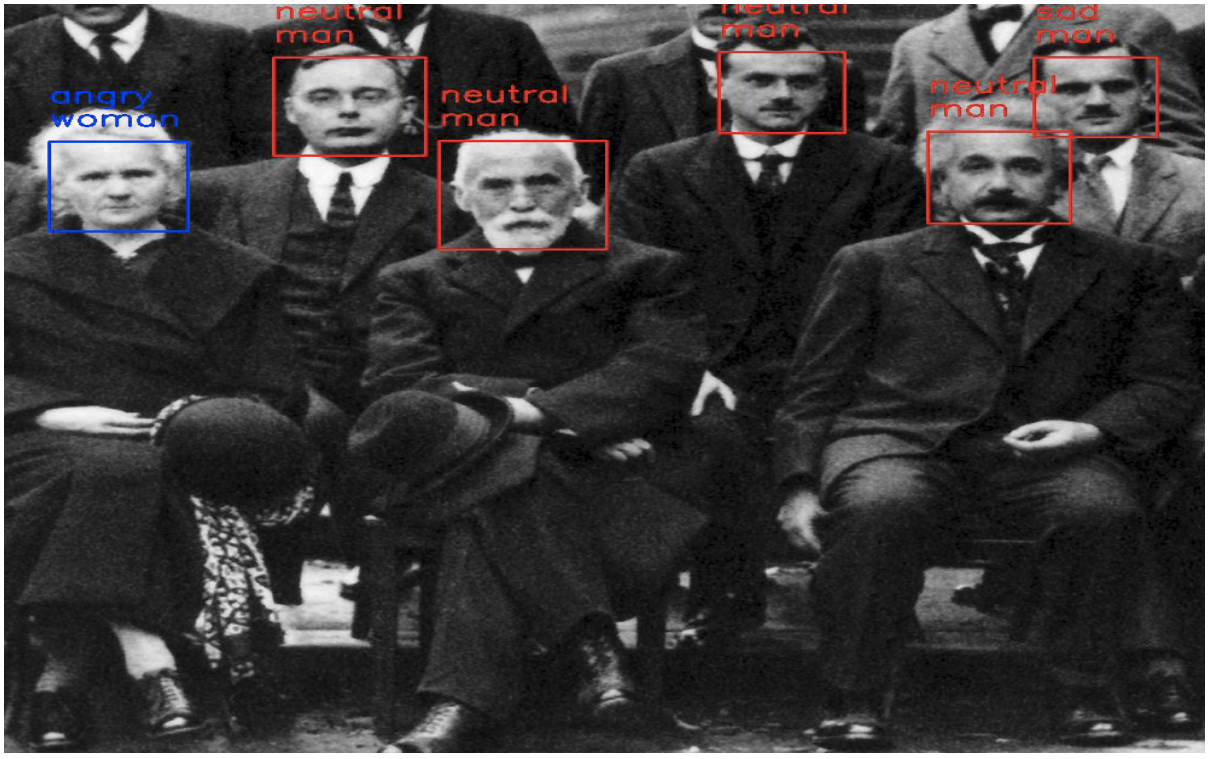


Figure 8: Face Detection with emotion & Gender Classification

Static Image Face expression & gender classification:

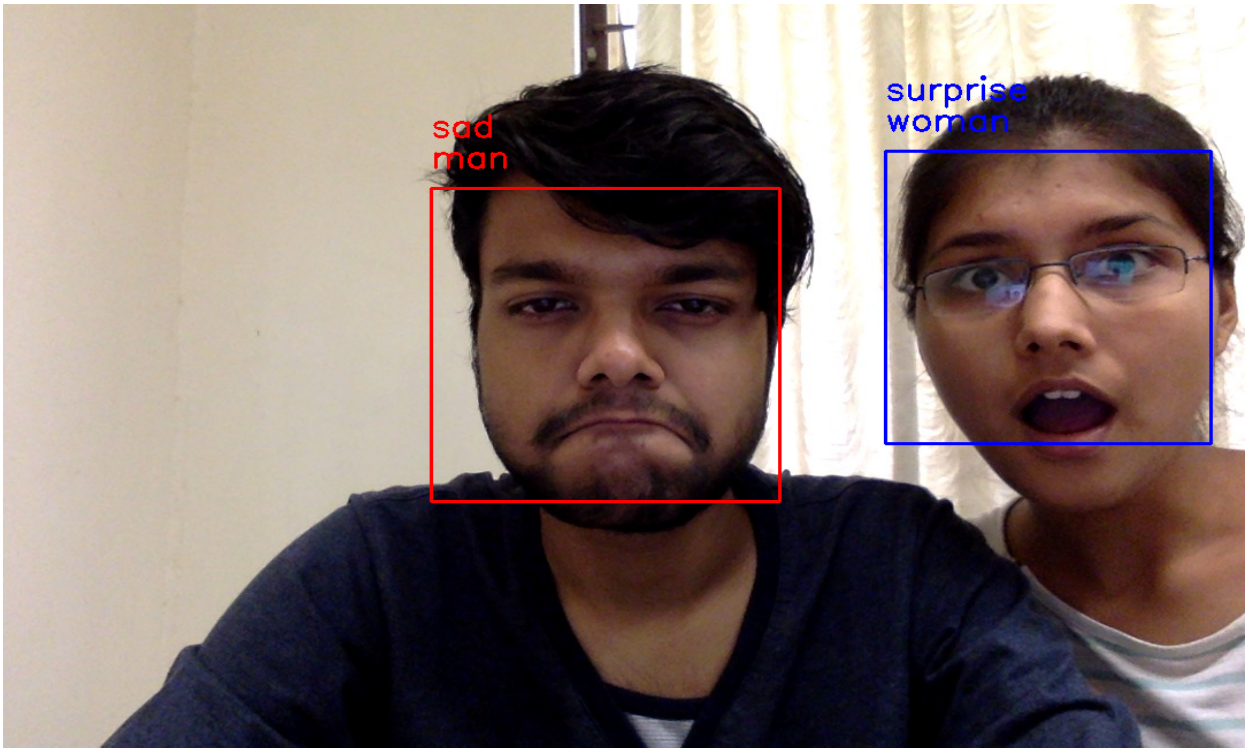


Figure 9: Face Detection with emotion & Gender Classification

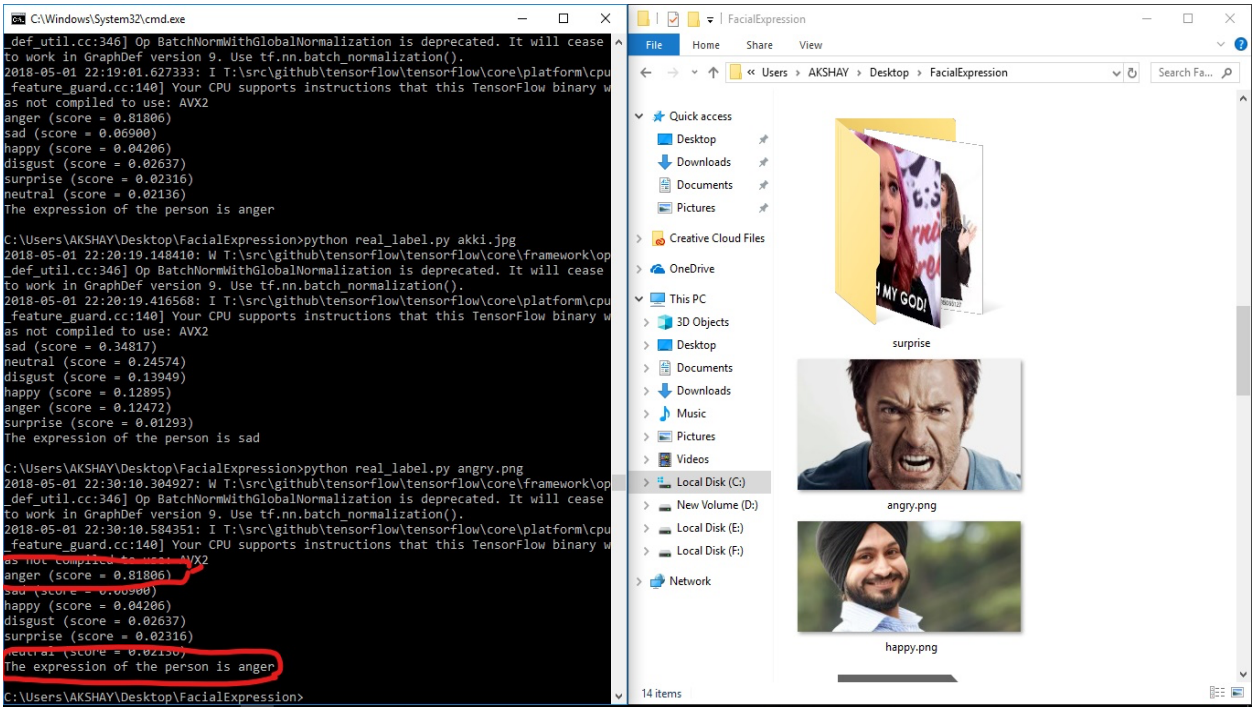


Figure 10: Face Detection with emotion Classification

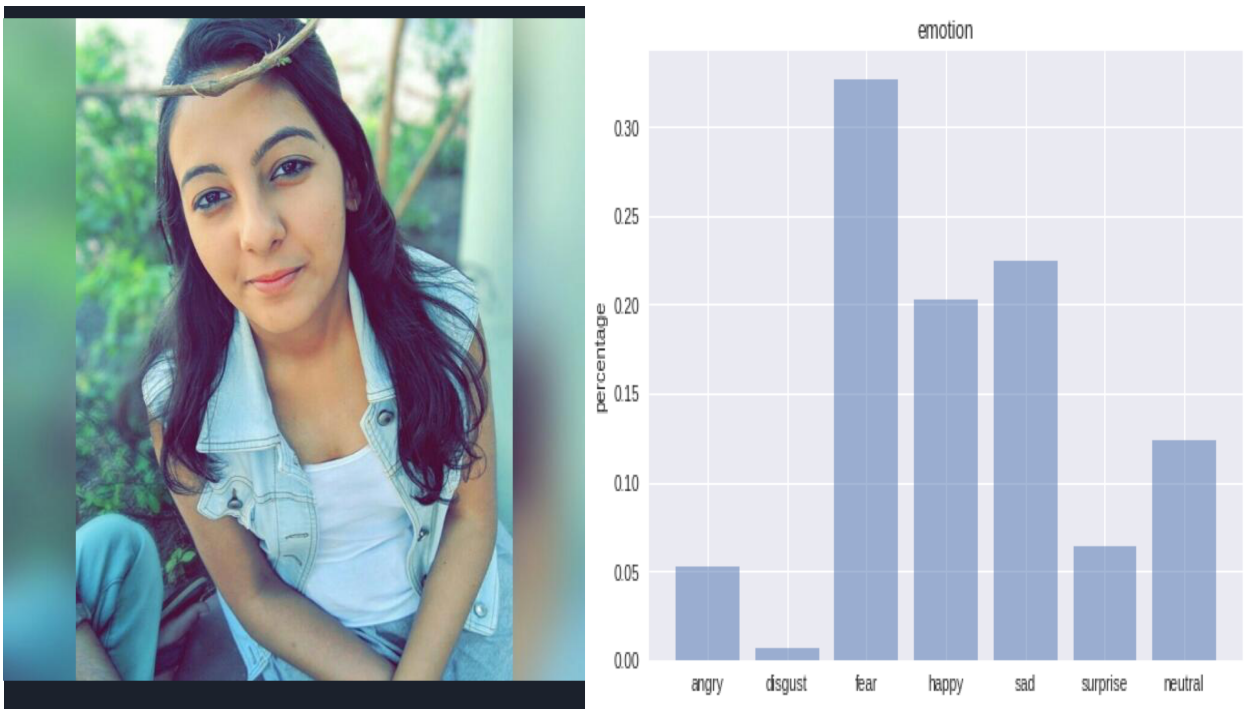


Figure 11: Face Detection with emotion Classification

Dynamic Image (Real Time) Face expression & gender classification:

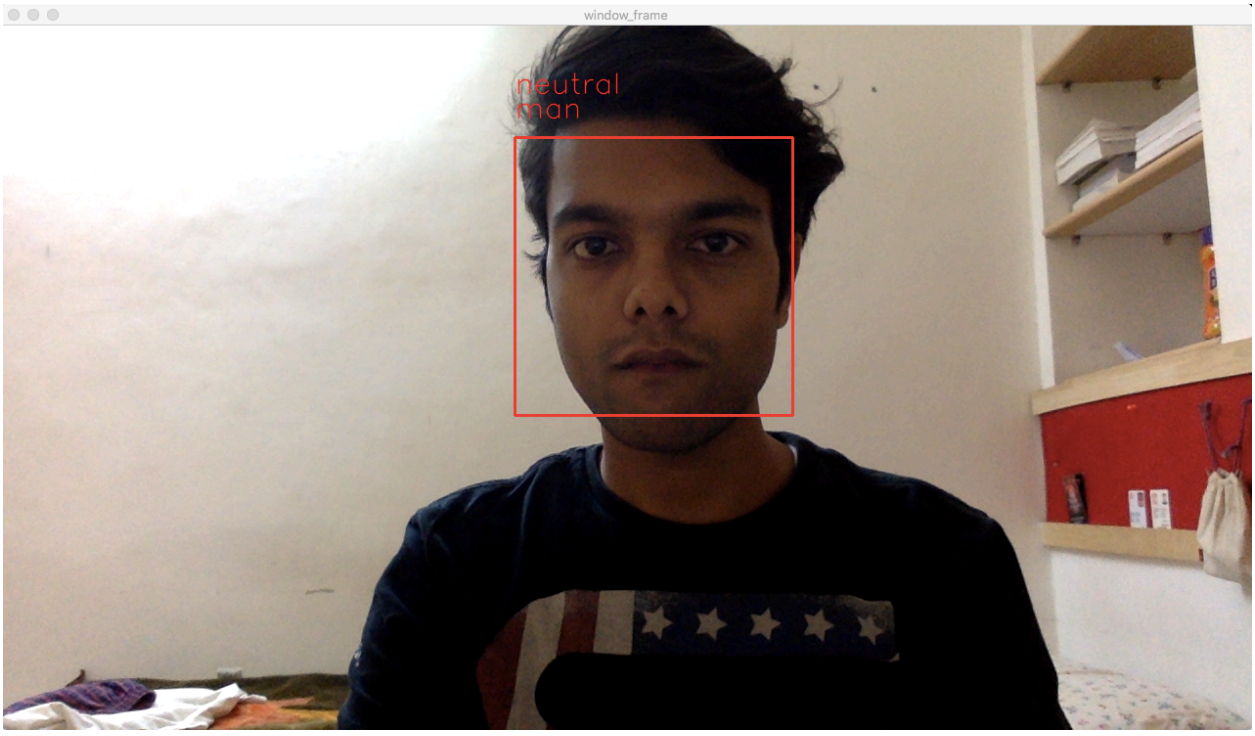


Figure 12: Face Detection with emotion & Gender Classification(Neutral)

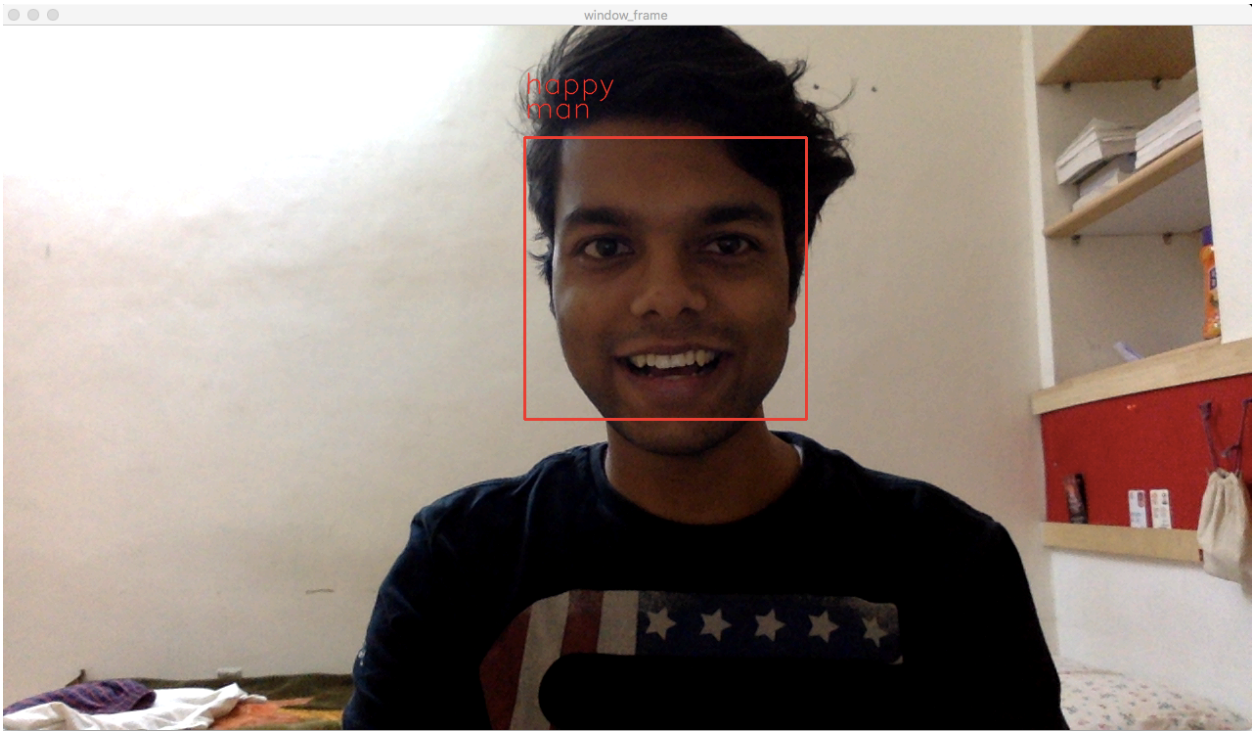


Figure 13: Face Detection with emotion & Gender Classification(Happy)

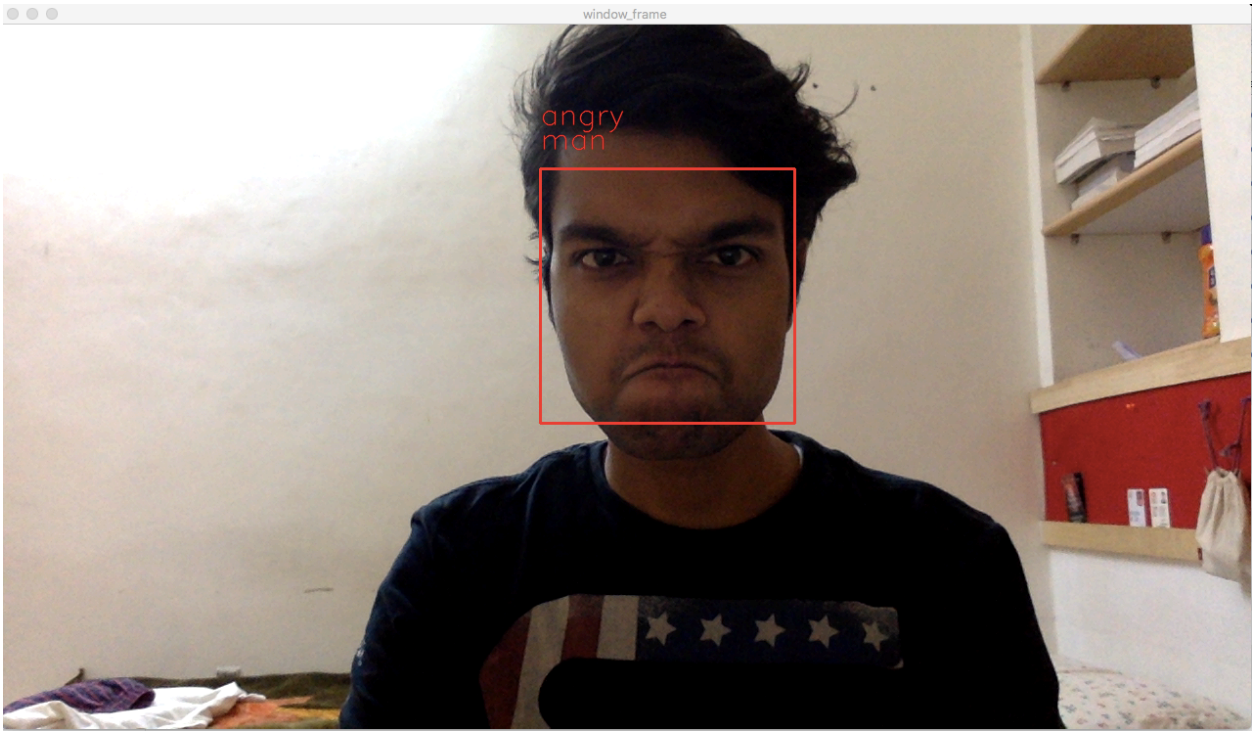


Figure 14: Face Detection with emotion & Gender Classification(Angry)

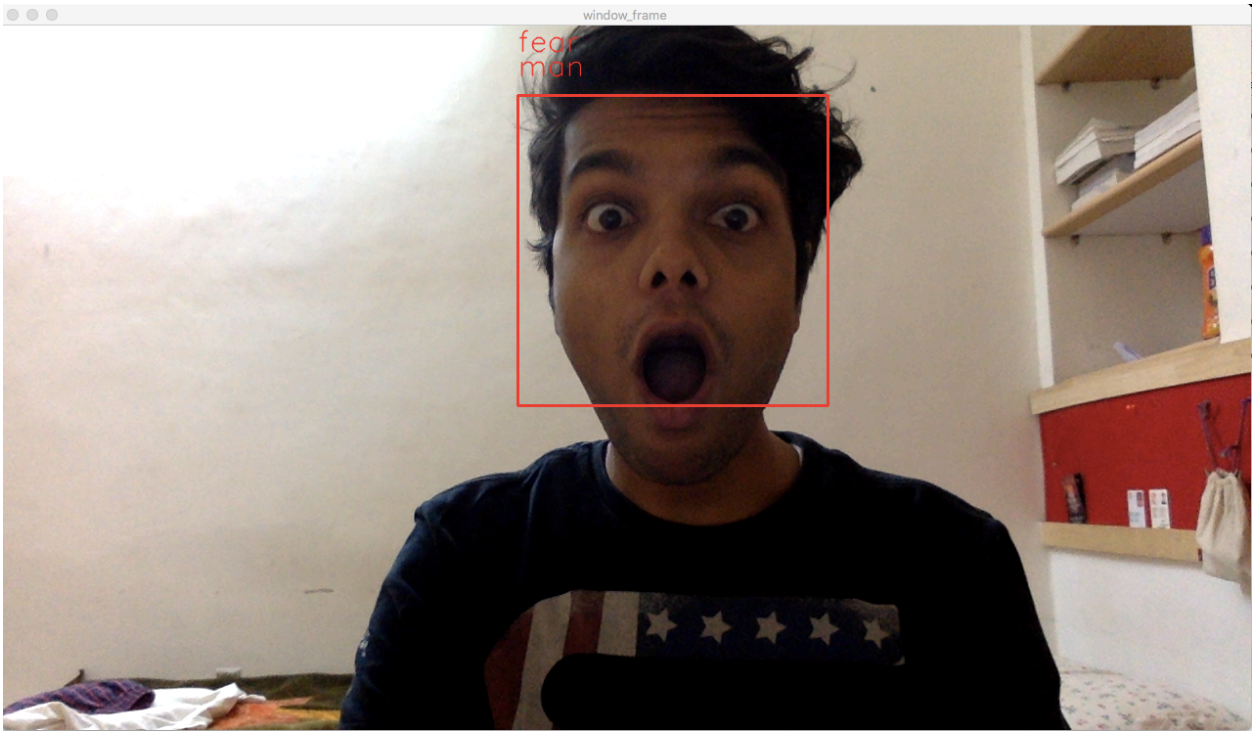


Figure 15: Face Detection with emotion & Gender Classification(Fear)

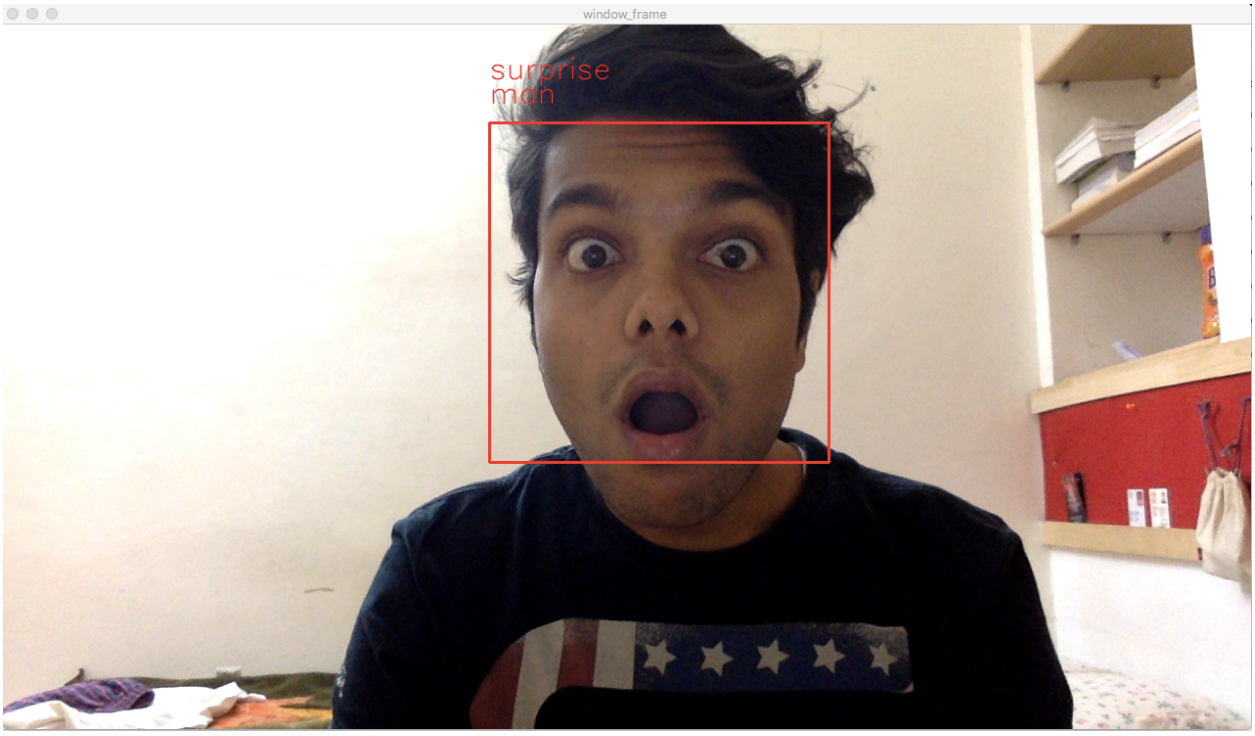


Figure 16: Face Detection with emotion & Gender Classification(Surprise)

Figure 17: Face Detection with emotion & Gender Classification(Sad)

Difficulties Faced:

- Multiple Face Detection.
- Gender Based Classification.
- Beard & Spectacles Occlusions.

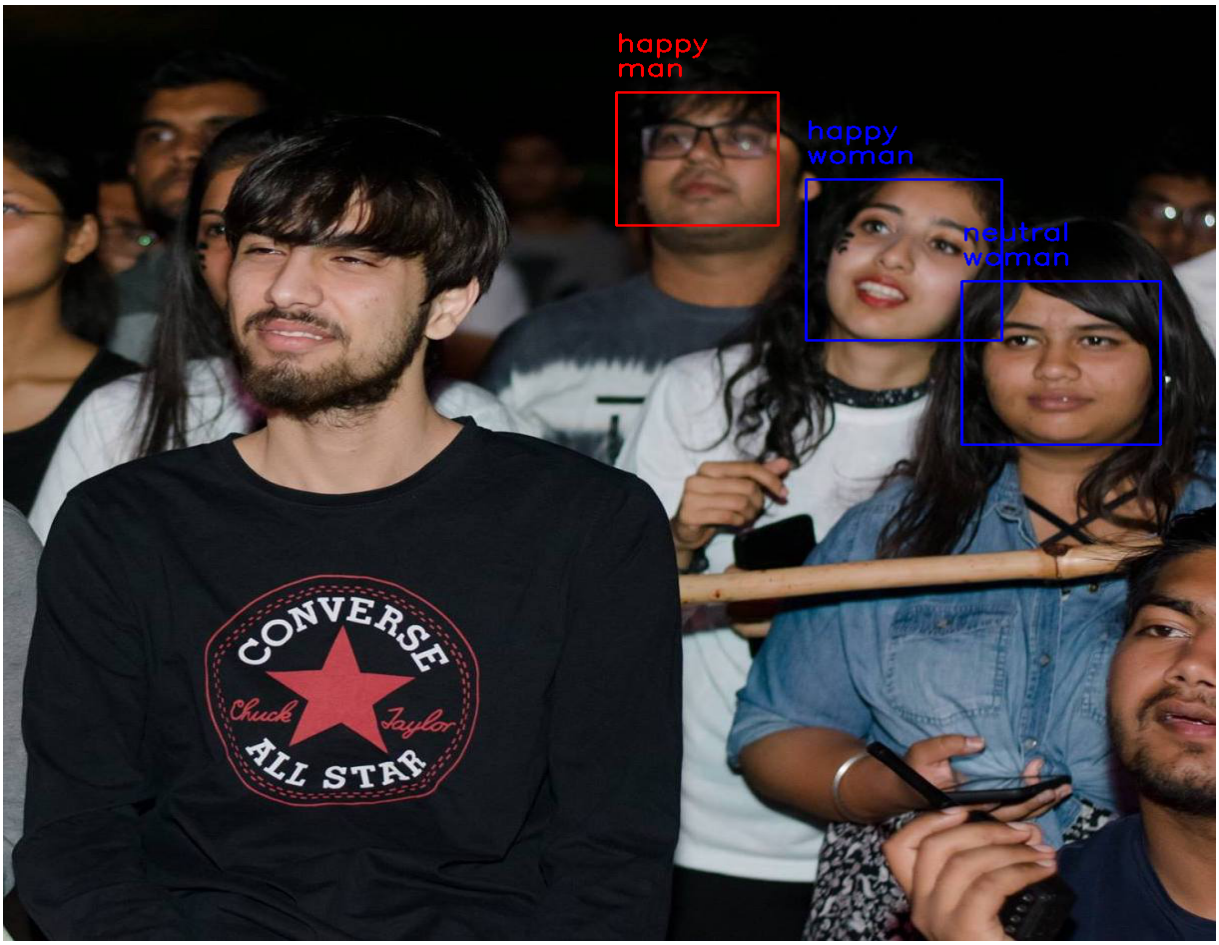


Figure 18: Multiple Face Detection with emotion & Gender Classification

CODES:

HAAR Cascade Frontal Face Detection (Code by Intel of almost 600 Pages)

```
<!--
    Stump-based 24x24 discrete(?) adaboost frontal face detector.
    Created by Rainer Lienhart.

////////////////////////////////////////////////////////////////////////
//////////////////

  IMPORTANT: READ BEFORE DOWNLOADING, COPYING, INSTALLING OR USING.

  By downloading, copying, installing or using the software you agree
to this license.
  If you do not agree to this license, do not download, install,
  copy or use the software.


                        Intel License Agreement
                For Open Source Computer Vision Library

 Copyright (C) 2000, Intel Corporation, all rights reserved.
 Third party copyrights are property of their respective owners.

 Redistribution and use in source and binary forms, with or without
modification,
 are permitted provided that the following conditions are met:

   * Redistribution's of source code must retain the above copyright
notice,
     this list of conditions and the following disclaimer.

   * Redistribution's in binary form must reproduce the above
copyright notice,
     this list of conditions and the following disclaimer in the
documentation
     and/or other materials provided with the distribution.

   * The name of Intel Corporation may not be used to endorse or
promote products
     derived from this software without specific prior written
permission.
```

```
 This software is provided by the copyright holders and contributors
"as is" and
 any express or implied warranties, including, but not limited to, the
implied
 warranties of merchantability and fitness for a particular purpose
are disclaimed.
 In no event shall the Intel Corporation or contributors be liable for
any direct,
 indirect, incidental, special, exemplary, or consequential damages
 (including, but not limited to, procurement of substitute goods or
services;
 loss of use, data, or profits; or business interruption) however
caused
 and on any theory of liability, whether in contract, strict
liability,
 or tort (including negligence or otherwise) arising in any way out of
 the use of this software, even if advised of the possibility of such
damage.
                                -->
```

image_emotion_gender_demo.py (For Static Images)

```
import sys

import cv2
from keras.models import load_model
import numpy as np

from utils.datasets import get_labels
from utils.inference import detect_faces
from utils.inference import draw_text
from utils.inference import draw_bounding_box
from utils.inference import apply_offsets
from utils.inference import load_detection_model
from utils.inference import load_image
from utils.preprocessor import preprocess_input

# parameters for loading data and images
image_path = sys.argv[1]
detection_model_path =
'../trained_models/detection_models/haarcascade_frontalface_default.xm
l'
emotion_model_path =
'../trained_models/emotion_models/fer2013_mini_XCEPTION.102-0.66.hdf5'
gender_model_path = '../trained_models/gender_models/simple_CNN.81-
0.96.hdf5'
emotion_labels = get_labels('fer2013')
gender_labels = get_labels('imdb')
font = cv2.FONT_HERSHEY_SIMPLEX
```

```
# hyper-parameters for bounding boxes shape
gender_offsets = (30, 60)
gender_offsets = (10, 10)
emotion_offsets = (20, 40)
emotion_offsets = (0, 0)

# loading models
face_detection = load_detection_model(detection_model_path)
emotion_classifier = load_model(emotion_model_path, compile=False)
gender_classifier = load_model(gender_model_path, compile=False)

# getting input model shapes for inference
emotion_target_size = emotion_classifier.input_shape[1:3]
gender_target_size = gender_classifier.input_shape[1:3]

# loading images
rgb_image = load_image(image_path, grayscale=False)
gray_image = load_image(image_path, grayscale=True)
gray_image = np.squeeze(gray_image)
gray_image = gray_image.astype('uint8')

faces = detect_faces(face_detection, gray_image)
for face_coordinates in faces:
    x1, x2, y1, y2 = apply_offsets(face_coordinates, gender_offsets)
    rgb_face = rgb_image[y1:y2, x1:x2]

    x1, x2, y1, y2 = apply_offsets(face_coordinates, emotion_offsets)
    gray_face = gray_image[y1:y2, x1:x2]

    try:
        rgb_face = cv2.resize(rgb_face, (gender_target_size))
        gray_face = cv2.resize(gray_face, (emotion_target_size))
    except:
        continue

    rgb_face = preprocess_input(rgb_face, False)
    rgb_face = np.expand_dims(rgb_face, 0)
    gender_prediction = gender_classifier.predict(rgb_face)
    gender_label_arg = np.argmax(gender_prediction)
    gender_text = gender_labels[gender_label_arg]

    gray_face = preprocess_input(gray_face, True)
    gray_face = np.expand_dims(gray_face, 0)
    gray_face = np.expand_dims(gray_face, -1)
    emotion_label_arg =
np.argmax(emotion_classifier.predict(gray_face))
    emotion_text = emotion_labels[emotion_label_arg]

    if gender_text == gender_labels[0]:
```

```
        color = (0, 0, 255)
    else:
        color = (255, 0, 0)

    draw_bounding_box(face_coordinates, rgb_image, color)
    draw_text(face_coordinates, rgb_image, gender_text, color, 0, -20,
1, 2)
    draw_text(face_coordinates, rgb_image, emotion_text, color, 0, -
50, 1, 2)

bgr_image = cv2.cvtColor(rgb_image, cv2.COLOR_RGB2BGR)
cv2.imwrite('../images/aniket_sample.png', bgr_image)
```

video_emotion_color_demo.py(For Real Time Emotion Detection)

```
from statistics import mode

import cv2
from keras.models import load_model
import numpy as np

from utils.datasets import get_labels
from utils.inference import detect_faces
from utils.inference import draw_text
from utils.inference import draw_bounding_box
from utils.inference import apply_offsets
from utils.inference import load_detection_model
from utils.preprocessor import preprocess_input

# parameters for loading data and images
detection_model_path =
'../trained_models/detection_models/haarcascade_frontalface_default.xm
l'
emotion_model_path =
'../trained_models/emotion_models/fer2013_mini_XCEPTION.102-0.66.hdf5'
emotion_labels = get_labels('fer2013')

# hyper-parameters for bounding boxes shape
frame_window = 10
emotion_offsets = (20, 40)

# loading models
face_detection = load_detection_model(detection_model_path)
emotion_classifier = load_model(emotion_model_path, compile=False)
```

```
# getting input model shapes for inference
emotion_target_size = emotion_classifier.input_shape[1:3]

# starting lists for calculating modes
emotion_window = []

# starting video streaming
cv2.namedWindow('window_frame')
video_capture = cv2.VideoCapture(0)
while True:
    bgr_image = video_capture.read()[1]
    gray_image = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2GRAY)
    rgb_image = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2RGB)
    faces = detect_faces(face_detection, gray_image)

    for face_coordinates in faces:

        x1, x2, y1, y2 = apply_offsets(face_coordinates,
emotion_offsets)
        gray_face = gray_image[y1:y2, x1:x2]
        try:
            gray_face = cv2.resize(gray_face, (emotion_target_size))
        except:
            continue

        gray_face = preprocess_input(gray_face, True)
        gray_face = np.expand_dims(gray_face, 0)
        gray_face = np.expand_dims(gray_face, -1)
        emotion_prediction = emotion_classifier.predict(gray_face)
        emotion_probability = np.max(emotion_prediction)
        emotion_label_arg = np.argmax(emotion_prediction)
        emotion_text = emotion_labels[emotion_label_arg]
        emotion_window.append(emotion_text)

        if len(emotion_window) > frame_window:
            emotion_window.pop(0)
        try:
            emotion_mode = mode(emotion_window)
        except:
            continue

        if emotion_text == 'angry':
            color = emotion_probability * np.asarray((255, 0, 0))
        elif emotion_text == 'sad':
            color = emotion_probability * np.asarray((0, 0, 255))
        elif emotion_text == 'happy':
            color = emotion_probability * np.asarray((255, 255, 0))
        elif emotion_text == 'surprise':
```

```
            color = emotion_probability * np.asarray((0, 255, 255))
        else:
            color = emotion_probability * np.asarray((0, 255, 0))

        color = color.astype(int)
        color = color.tolist()

        draw_bounding_box(face_coordinates, rgb_image, color)
        draw_text(face_coordinates, rgb_image, emotion_mode,
                  color, 0, -45, 1, 1)

    bgr_image = cv2.cvtColor(rgb_image, cv2.COLOR_RGB2BGR)
    cv2.imshow('window_frame', bgr_image)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

video_emotion_gender_demo.py

```
from statistics import mode

import cv2
from keras.models import load_model
import numpy as np

from utils.datasets import get_labels
from utils.inference import detect_faces
from utils.inference import draw_text
from utils.inference import draw_bounding_box
from utils.inference import apply_offsets
from utils.inference import load_detection_model
from utils.preprocessor import preprocess_input

# parameters for loading data and images
detection_model_path =
'../trained_models/detection_models/haarcascade_frontalface_default.xm
l'
emotion_model_path =
'../trained_models/emotion_models/fer2013_mini_XCEPTION.102-0.66.hdf5'
gender_model_path = '../trained_models/gender_models/simple_CNN.81-
0.96.hdf5'
emotion_labels = get_labels('fer2013')
gender_labels = get_labels('imdb')
font = cv2.FONT_HERSHEY_SIMPLEX

# hyper-parameters for bounding boxes shape
frame_window = 10
gender_offsets = (30, 60)
emotion_offsets = (20, 40)
```

```
# loading models
face_detection = load_detection_model(detection_model_path)
emotion_classifier = load_model(emotion_model_path, compile=False)
gender_classifier = load_model(gender_model_path, compile=False)

# getting input model shapes for inference
emotion_target_size = emotion_classifier.input_shape[1:3]
gender_target_size = gender_classifier.input_shape[1:3]

# starting lists for calculating modes
gender_window = []
emotion_window = []

# starting video streaming
cv2.namedWindow('window_frame')
video_capture = cv2.VideoCapture(0)
while True:

    bgr_image = video_capture.read()[1]
    gray_image = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2GRAY)
    rgb_image = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2RGB)
    faces = detect_faces(face_detection, gray_image)

    for face_coordinates in faces:

        x1, x2, y1, y2 = apply_offsets(face_coordinates,
gender_offsets)
        rgb_face = rgb_image[y1:y2, x1:x2]

        x1, x2, y1, y2 = apply_offsets(face_coordinates,
emotion_offsets)
        gray_face = gray_image[y1:y2, x1:x2]
        try:
            rgb_face = cv2.resize(rgb_face, (gender_target_size))
            gray_face = cv2.resize(gray_face, (emotion_target_size))
        except:
            continue
        gray_face = preprocess_input(gray_face, False)
        gray_face = np.expand_dims(gray_face, 0)
        gray_face = np.expand_dims(gray_face, -1)
        emotion_label_arg =
np.argmax(emotion_classifier.predict(gray_face))
        emotion_text = emotion_labels[emotion_label_arg]
        emotion_window.append(emotion_text)

        rgb_face = np.expand_dims(rgb_face, 0)
        rgb_face = preprocess_input(rgb_face, False)
        gender_prediction = gender_classifier.predict(rgb_face)
```

```
            gender_label_arg = np.argmax(gender_prediction)
            gender_text = gender_labels[gender_label_arg]
            gender_window.append(gender_text)

            if len(gender_window) > frame_window:
                emotion_window.pop(0)
                gender_window.pop(0)
            try:
                emotion_mode = mode(emotion_window)
                gender_mode = mode(gender_window)
            except:
                continue

            if gender_text == gender_labels[0]:
                color = (0, 0, 255)
            else:
                color = (255, 0, 0)

            draw_bounding_box(face_coordinates, rgb_image, color)
            draw_text(face_coordinates, rgb_image, gender_mode,
                      color, 0, -20, 1, 1)
            draw_text(face_coordinates, rgb_image, emotion_mode,
                      color, 0, -45, 1, 1)

        bgr_image = cv2.cvtColor(rgb_image, cv2.COLOR_RGB2BGR)
        cv2.imshow('window_frame', bgr_image)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
```

train_gender_classifier.py

```
from keras.callbacks import CSVLogger, ModelCheckpoint, EarlyStopping
from keras.callbacks import ReduceLROnPlateau
from utils.datasets import DataManager
from models.cnn import mini_XCEPTION
from utils.data_augmentation import ImageGenerator
from utils.datasets import split_imdb_data

# parameters
batch_size = 32
num_epochs = 1000
validation_split = .2
do_random_crop = False
patience = 100
num_classes = 2
dataset_name = 'imdb'
```

```
input_shape = (64, 64, 1)
if input_shape[2] == 1:
    grayscale = True
images_path = '../datasets/imdb_crop/'
log_file_path = '../trained_models/gender_models/gender_training.log'
trained_models_path =
'../trained_models/gender_models/gender_mini_XCEPTION'

# data loader
data_loader = DataManager(dataset_name)
ground_truth_data = data_loader.get_data()
train_keys, val_keys = split_imdb_data(ground_truth_data,
validation_split)
print('Number of training samples:', len(train_keys))
print('Number of validation samples:', len(val_keys))
image_generator = ImageGenerator(ground_truth_data, batch_size,
                                 input_shape[:2],
                                 train_keys, val_keys, None,
                                 path_prefix=images_path,
                                 vertical_flip_probability=0,
                                 grayscale=grayscale,
                                 do_random_crop=do_random_crop)

# model parameters/compilation
model = mini_XCEPTION(input_shape, num_classes)
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()

# model callbacks
early_stop = EarlyStopping('val_loss', patience=patience)
reduce_lr = ReduceLROnPlateau('val_loss', factor=0.1,
                              patience=int(patience/2), verbose=1)
csv_logger = CSVLogger(log_file_path, append=False)
model_names = trained_models_path + '.{epoch:02d}-{val_acc:.2f}.hdf5'
model_checkpoint = ModelCheckpoint(model_names,
                                   monitor='val_loss',
                                   verbose=1,
                                   save_best_only=True,
                                   save_weights_only=False)
callbacks = [model_checkpoint, csv_logger, early_stop, reduce_lr]

# training model
model.fit_generator(image_generator.flow(mode='train'),
                    steps_per_epoch=int(len(train_keys) / batch_size),
                    epochs=num_epochs, verbose=1,
                    callbacks=callbacks,
                    validation_data=image_generator.flow('val'),
                    validation_steps=int(len(val_keys) / batch_size)
```

train_emotion_classifier.py

```
from keras.callbacks import CSVLogger, ModelCheckpoint, EarlyStopping
from keras.callbacks import ReduceLROnPlateau
from keras.preprocessing.image import ImageDataGenerator

from models.cnn import mini_XCEPTION
from utils.datasets import DataManager
from utils.datasets import split_data
from utils.preprocessor import preprocess_input

# parameters
batch_size = 32
num_epochs = 10000
input_shape = (64, 64, 1)
validation_split = .2
verbose = 1
num_classes = 7
patience = 50
base_path = '../trained_models/emotion_models/'

# data generator
data_generator = ImageDataGenerator(
                        featurewise_center=False,
                        featurewise_std_normalization=False,
                        rotation_range=10,
                        width_shift_range=0.1,
                        height_shift_range=0.1,
                        zoom_range=.1,
                        horizontal_flip=True)

# model parameters/compilation
model = mini_XCEPTION(input_shape, num_classes)
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()


datasets = ['fer2013']
for dataset_name in datasets:
    print('Training dataset:', dataset_name)

    # callbacks
    log_file_path = base_path + dataset_name + '_emotion_training.log'
    csv_logger = CSVLogger(log_file_path, append=False)
    early_stop = EarlyStopping('val_loss', patience=patience)
    reduce_lr = ReduceLROnPlateau('val_loss', factor=0.1,
```

```
                                    patience=int(patience/4), verbose=1)
    trained_models_path = base_path + dataset_name + '_mini_XCEPTION'
    model_names = trained_models_path + '.{epoch:02d}-
{val_acc:.2f}.hdf5'
    model_checkpoint = ModelCheckpoint(model_names, 'val_loss',
verbose=1,

save_best_only=True)
    callbacks = [model_checkpoint, csv_logger, early_stop, reduce_lr]

    # loading dataset
    data_loader = DataManager(dataset_name,
image_size=input_shape[:2])
    faces, emotions = data_loader.get_data()
    faces = preprocess_input(faces)
    num_samples, num_classes = emotions.shape
    train_data, val_data = split_data(faces, emotions,
validation_split)
    train_faces, train_emotions = train_data
    model.fit_generator(data_generator.flow(train_faces,
train_emotions,
                                            batch_size),
                        steps_per_epoch=len(train_faces) / batch_size,
                        epochs=num_epochs, verbose=1,
callbacks=callbacks,
                        validation_data=val_data)
```

cnn.py

```
from keras.layers import Activation, Convolution2D, Dropout, Conv2D
from keras.layers import AveragePooling2D, BatchNormalization
from keras.layers import GlobalAveragePooling2D
from keras.models import Sequential
from keras.layers import Flatten
from keras.models import Model
from keras.layers import Input
from keras.layers import MaxPooling2D
from keras.layers import SeparableConv2D
from keras import layers
from keras.regularizers import l2

def simple_CNN(input_shape, num_classes):

    model = Sequential()
    model.add(Convolution2D(filters=16, kernel_size=(7, 7),
padding='same',
```

```
                                name='image_array',
input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=16, kernel_size=(7, 7),
padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.5))

    model.add(Convolution2D(filters=32, kernel_size=(5, 5),
padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=32, kernel_size=(5, 5),
padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.5))

    model.add(Convolution2D(filters=64, kernel_size=(3, 3),
padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=64, kernel_size=(3, 3),
padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.5))

    model.add(Convolution2D(filters=128, kernel_size=(3, 3),
padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=128, kernel_size=(3, 3),
padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.5))

    model.add(Convolution2D(filters=256, kernel_size=(3, 3),
padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=num_classes, kernel_size=(3, 3),
padding='same'))
    model.add(GlobalAveragePooling2D())
    model.add(Activation('softmax',name='predictions'))
    return model
```

```
def simpler_CNN(input_shape, num_classes):

    model = Sequential()
    model.add(Convolution2D(filters=16, kernel_size=(5, 5),
padding='same',
                            name='image_array',
input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=16, kernel_size=(5, 5),
                            strides=(2, 2), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(.25))

    model.add(Convolution2D(filters=32, kernel_size=(5, 5),
padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=32, kernel_size=(5, 5),
                            strides=(2, 2), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(.25))

    model.add(Convolution2D(filters=64, kernel_size=(3, 3),
padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=64, kernel_size=(3, 3),
                            strides=(2, 2), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(.25))

    model.add(Convolution2D(filters=64, kernel_size=(1, 1),
padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=128, kernel_size=(3, 3),
                            strides=(2, 2), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(.25))

    model.add(Convolution2D(filters=256, kernel_size=(1, 1),
padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=128, kernel_size=(3, 3),
                            strides=(2, 2), padding='same'))

    model.add(Convolution2D(filters=256, kernel_size=(1, 1),
padding='same'))
```

```
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=num_classes, kernel_size=(3, 3),
                            strides=(2, 2), padding='same'))

    model.add(Flatten())
    #model.add(GlobalAveragePooling2D())
    model.add(Activation('softmax',name='predictions'))
    return model

def tiny_XCEPTION(input_shape, num_classes, l2_regularization=0.01):
    regularization = l2(l2_regularization)

    # base
    img_input = Input(input_shape)
    x = Conv2D(5, (3, 3), strides=(1, 1),
kernel_regularizer=regularization,
                                            use_bias=False)(img_input)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(5, (3, 3), strides=(1, 1),
kernel_regularizer=regularization,
                                            use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    # module 1
    residual = Conv2D(8, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(8, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(8, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    # module 2
    residual = Conv2D(16, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(16, (3, 3), padding='same',
```

```
                    kernel_regularizer=regularization,
                    use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(16, (3, 3), padding='same',
                    kernel_regularizer=regularization,
                    use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    # module 3
    residual = Conv2D(32, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(32, (3, 3), padding='same',
                    kernel_regularizer=regularization,
                    use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(32, (3, 3), padding='same',
                    kernel_regularizer=regularization,
                    use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    # module 4
    residual = Conv2D(64, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(64, (3, 3), padding='same',
                    kernel_regularizer=regularization,
                    use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(64, (3, 3), padding='same',
                    kernel_regularizer=regularization,
                    use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    x = Conv2D(num_classes, (3, 3),
```

```
            #kernel_regularizer=regularization,
            padding='same')(x)
    x = GlobalAveragePooling2D()(x)
    output = Activation('softmax',name='predictions')(x)

    model = Model(img_input, output)
    return model


def mini_XCEPTION(input_shape, num_classes, l2_regularization=0.01):
    regularization = l2(l2_regularization)

    # base
    img_input = Input(input_shape)
    x = Conv2D(8, (3, 3), strides=(1, 1),
kernel_regularizer=regularization,
                                            use_bias=False)(img_input)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(8, (3, 3), strides=(1, 1),
kernel_regularizer=regularization,
                                            use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    # module 1
    residual = Conv2D(16, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(16, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(16, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    # module 2
    residual = Conv2D(32, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(32, (3, 3), padding='same',
```

```
                    kernel_regularizer=regularization,
                    use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(32, (3, 3), padding='same',
                    kernel_regularizer=regularization,
                    use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    # module 3
    residual = Conv2D(64, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(64, (3, 3), padding='same',
                    kernel_regularizer=regularization,
                    use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(64, (3, 3), padding='same',
                    kernel_regularizer=regularization,
                    use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    # module 4
    residual = Conv2D(128, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(128, (3, 3), padding='same',
                    kernel_regularizer=regularization,
                    use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(128, (3, 3), padding='same',
                    kernel_regularizer=regularization,
                    use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    x = Conv2D(num_classes, (3, 3),
```

```
            #kernel_regularizer=regularization,
            padding='same')(x)
    x = GlobalAveragePooling2D()(x)
    output = Activation('softmax',name='predictions')(x)

    model = Model(img_input, output)
    return model

def big_XCEPTION(input_shape, num_classes):
    img_input = Input(input_shape)
    x = Conv2D(32, (3, 3), strides=(2, 2), use_bias=False)(img_input)
    x = BatchNormalization(name='block1_conv1_bn')(x)
    x = Activation('relu', name='block1_conv1_act')(x)
    x = Conv2D(64, (3, 3), use_bias=False)(x)
    x = BatchNormalization(name='block1_conv2_bn')(x)
    x = Activation('relu', name='block1_conv2_act')(x)

    residual = Conv2D(128, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(128, (3, 3), padding='same',
use_bias=False)(x)
    x = BatchNormalization(name='block2_sepconv1_bn')(x)
    x = Activation('relu', name='block2_sepconv2_act')(x)
    x = SeparableConv2D(128, (3, 3), padding='same',
use_bias=False)(x)
    x = BatchNormalization(name='block2_sepconv2_bn')(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    residual = Conv2D(256, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = Activation('relu', name='block3_sepconv1_act')(x)
    x = SeparableConv2D(256, (3, 3), padding='same',
use_bias=False)(x)
    x = BatchNormalization(name='block3_sepconv1_bn')(x)
    x = Activation('relu', name='block3_sepconv2_act')(x)
    x = SeparableConv2D(256, (3, 3), padding='same',
use_bias=False)(x)
    x = BatchNormalization(name='block3_sepconv2_bn')(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])
    x = Conv2D(num_classes, (3, 3),
            #kernel_regularizer=regularization,
```

```
            padding='same')(x)
    x = GlobalAveragePooling2D()(x)
    output = Activation('softmax',name='predictions')(x)

    model = Model(img_input, output)
    return model


if __name__ == "__main__":
    input_shape = (64, 64, 1)
    num_classes = 7
    #model = tiny_XCEPTION(input_shape, num_classes)
    #model.summary()
    #model = mini_XCEPTION(input_shape, num_classes)
    #model.summary()
    #model = big_XCEPTION(input_shape, num_classes)
    #model.summary()
    model = simple_CNN((48, 48, 1), num_classes)
    model.summary()
```

faces.py

```
from flask import Flask, jsonify, make_response, request, abort,
redirect, send_file
import logging

import emotion_gender_processor as eg_processor

app = Flask(__name__)

@app.route('/')
def index():
    return redirect("https://ekholabs.ai", code=302)

@app.route('/classifyImage', methods=['POST'])
def upload():
    try:
        image = request.files['image'].read()
        eg_processor.process_image(image)
        return send_file('/ekholabs/face-
classifier/result/predicted_image.png', mimetype='image/png')
    except Exception as err:
        logging.error('An error has occurred whilst processing the
file: "{0}"'.format(err))
        abort(400)
```

```
@app.errorhandler(400)
def bad_request(erro):
    return make_response(jsonify({'error': 'We cannot process the file
sent in the request.'}), 400)

@app.errorhandler(404)
def not_found(error):
    return make_response(jsonify({'error': 'Resource no found.'}),
404)

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=8084)
```

emotion_gender_processor.py

```
import os
import sys
import logging

import cv2
from keras.models import load_model
import numpy as np

from utils.datasets import get_labels
from utils.inference import detect_faces
from utils.inference import draw_text
from utils.inference import draw_bounding_box
from utils.inference import apply_offsets
from utils.inference import load_detection_model
from utils.inference import load_image
from utils.preprocessor import preprocess_input

def process_image(image):

    try:
        # parameters for loading data and images
        detection_model_path =
'./trained_models/detection_models/haarcascade_frontalface_default.xml
'
        emotion_model_path =
'./trained_models/emotion_models/fer2013_mini_XCEPTION.102-0.66.hdf5'
        gender_model_path =
'./trained_models/gender_models/simple_CNN.81-0.96.hdf5'
        emotion_labels = get_labels('fer2013')
        gender_labels = get_labels('imdb')
        font = cv2.FONT_HERSHEY_SIMPLEX
```

```
        # hyper-parameters for bounding boxes shape
        gender_offsets = (30, 60)
        gender_offsets = (10, 10)
        emotion_offsets = (20, 40)
        emotion_offsets = (0, 0)

        # loading models
        face_detection = load_detection_model(detection_model_path)
        emotion_classifier = load_model(emotion_model_path,
compile=False)
        gender_classifier = load_model(gender_model_path,
compile=False)

        # getting input model shapes for inference
        emotion_target_size = emotion_classifier.input_shape[1:3]
        gender_target_size = gender_classifier.input_shape[1:3]

        # loading images
        image_array = np.fromstring(image, np.uint8)
        unchanged_image = cv2.imdecode(image_array,
cv2.IMREAD_UNCHANGED)

        rgb_image = cv2.cvtColor(unchanged_image, cv2.COLOR_BGR2RGB)
        gray_image = cv2.cvtColor(unchanged_image, cv2.COLOR_BGR2GRAY)

        faces = detect_faces(face_detection, gray_image)
        for face_coordinates in faces:
            x1, x2, y1, y2 = apply_offsets(face_coordinates,
gender_offsets)
            rgb_face = rgb_image[y1:y2, x1:x2]

            x1, x2, y1, y2 = apply_offsets(face_coordinates,
emotion_offsets)
            gray_face = gray_image[y1:y2, x1:x2]

            try:
                rgb_face = cv2.resize(rgb_face, (gender_target_size))
                gray_face = cv2.resize(gray_face,
(emotion_target_size))
            except:
                continue

            rgb_face = preprocess_input(rgb_face, False)
            rgb_face = np.expand_dims(rgb_face, 0)
            gender_prediction = gender_classifier.predict(rgb_face)
            gender_label_arg = np.argmax(gender_prediction)
            gender_text = gender_labels[gender_label_arg]
```

```
            gray_face = preprocess_input(gray_face, True)
            gray_face = np.expand_dims(gray_face, 0)
            gray_face = np.expand_dims(gray_face, -1)
            emotion_label_arg =
np.argmax(emotion_classifier.predict(gray_face))
            emotion_text = emotion_labels[emotion_label_arg]

            if gender_text == gender_labels[0]:
                color = (0, 0, 255)
            else:
                color = (255, 0, 0)

            draw_bounding_box(face_coordinates, rgb_image, color)
            draw_text(face_coordinates, rgb_image, gender_text, color,
0, -20, 1, 2)
            draw_text(face_coordinates, rgb_image, emotion_text,
color, 0, -50, 1, 2)
    except Exception as err:
        logging.error('Error in emotion gender processor:
"{0}"'.format(err))

    bgr_image = cv2.cvtColor(rgb_image, cv2.COLOR_RGB2BGR)

    dirname = 'result'
    if not os.path.exists(dirname):
        os.mkdir(dirname)

    cv2.imwrite(os.path.join(dirname, 'predicted_image.png'),
bgr_image)
```

datasets.py

```
from scipy.io import loadmat
import pandas as pd
import numpy as np
from random import shuffle
import os
import cv2

class DataManager(object):
    """Class for loading fer2013 emotion classification dataset or
        imdb gender classification dataset."""
    def __init__(self, dataset_name='imdb', dataset_path=None,
image_size=(48, 48)):

        self.dataset_name = dataset_name
```

```
        self.dataset_path = dataset_path
        self.image_size = image_size
        if self.dataset_path != None:
            self.dataset_path = dataset_path
        elif self.dataset_name == 'imdb':
            self.dataset_path = '../datasets/imdb_crop/imdb.mat'
        elif self.dataset_name == 'fer2013':
            self.dataset_path = '../datasets/fer2013/fer2013.csv'
        elif self.dataset_name == 'KDEF':
            self.dataset_path = '../datasets/KDEF/'
        else:
            raise Exception('Incorrect dataset name, please input imdb
or fer2013')

    def get_data(self):
        if self.dataset_name == 'imdb':
            ground_truth_data = self._load_imdb()
        elif self.dataset_name == 'fer2013':
            ground_truth_data = self._load_fer2013()
        elif self.dataset_name == 'KDEF':
            ground_truth_data = self._load_KDEF()
        return ground_truth_data

    def _load_imdb(self):
        face_score_treshold = 3
        dataset = loadmat(self.dataset_path)
        image_names_array = dataset['imdb']['full_path'][0, 0][0]
        gender_classes = dataset['imdb']['gender'][0, 0][0]
        face_score = dataset['imdb']['face_score'][0, 0][0]
        second_face_score = dataset['imdb']['second_face_score'][0,
0][0]
        face_score_mask = face_score > face_score_treshold
        second_face_score_mask = np.isnan(second_face_score)
        unknown_gender_mask = np.logical_not(np.isnan(gender_classes))
        mask = np.logical_and(face_score_mask, second_face_score_mask)
        mask = np.logical_and(mask, unknown_gender_mask)
        image_names_array = image_names_array[mask]
        gender_classes = gender_classes[mask].tolist()
        image_names = []
        for image_name_arg in range(image_names_array.shape[0]):
            image_name = image_names_array[image_name_arg][0]
            image_names.append(image_name)
        return dict(zip(image_names, gender_classes))

    def _load_fer2013(self):
        data = pd.read_csv(self.dataset_path)
        pixels = data['pixels'].tolist()
        width, height = 48, 48
        faces = []
        for pixel_sequence in pixels:
```

```
            face = [int(pixel) for pixel in pixel_sequence.split(' ')]
            face = np.asarray(face).reshape(width, height)
            face = cv2.resize(face.astype('uint8'), self.image_size)
            faces.append(face.astype('float32'))
        faces = np.asarray(faces)
        faces = np.expand_dims(faces, -1)
        emotions = pd.get_dummies(data['emotion']).as_matrix()
        return faces, emotions

    def _load_KDEF(self):
        class_to_arg = get_class_to_arg(self.dataset_name)
        num_classes = len(class_to_arg)

        file_paths = []
        for folder, subfolders, filenames in
os.walk(self.dataset_path):
            for filename in filenames:
                if filename.lower().endswith(('.jpg')):
                    file_paths.append(os.path.join(folder, filename))

        num_faces = len(file_paths)
        y_size, x_size = self.image_size
        faces = np.zeros(shape=(num_faces, y_size, x_size))
        emotions = np.zeros(shape=(num_faces, num_classes))
        for file_arg, file_path in enumerate(file_paths):
            image_array = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
            image_array = cv2.resize(image_array, (y_size, x_size))
            faces[file_arg] = image_array
            file_basename = os.path.basename(file_path)
            file_emotion = file_basename[4:6]
            # there are two file names in the dataset that don't match
the given classes
            try:
                emotion_arg = class_to_arg[file_emotion]
            except:
                continue
            emotions[file_arg, emotion_arg] = 1
        faces = np.expand_dims(faces, -1)
        return faces, emotions

def get_labels(dataset_name):
    if dataset_name == 'fer2013':
        return {0:'angry',1:'disgust',2:'fear',3:'happy',
                4:'sad',5:'surprise',6:'neutral'}
    elif dataset_name == 'imdb':
        return {0:'woman', 1:'man'}
    elif dataset_name == 'KDEF':
        return {0:'AN', 1:'DI', 2:'AF', 3:'HA', 4:'SA', 5:'SU',
6:'NE'}
    else:
```

```
        raise Exception('Invalid dataset name')

def get_class_to_arg(dataset_name='fer2013'):
    if dataset_name == 'fer2013':
        return {'angry':0, 'disgust':1, 'fear':2, 'happy':3, 'sad':4,
                'surprise':5, 'neutral':6}
    elif dataset_name == 'imdb':
        return {'woman':0, 'man':1}
    elif dataset_name == 'KDEF':
        return {'AN':0, 'DI':1, 'AF':2, 'HA':3, 'SA':4, 'SU':5,
'NE':6}
    else:
        raise Exception('Invalid dataset name')

def split_imdb_data(ground_truth_data, validation_split=.2,
do_shuffle=False):
    ground_truth_keys = sorted(ground_truth_data.keys())
    if do_shuffle == True:
        shuffle(ground_truth_keys)
    training_split = 1 - validation_split
    num_train = int(training_split * len(ground_truth_keys))
    train_keys = ground_truth_keys[:num_train]
    validation_keys = ground_truth_keys[num_train:]
    return train_keys, validation_keys

def split_data(x, y, validation_split=.2):
    num_samples = len(x)
    num_train_samples = int((1 - validation_split)*num_samples)
    train_x = x[:num_train_samples]
    train_y = y[:num_train_samples]
    val_x = x[num_train_samples:]
    val_y = y[num_train_samples:]
    train_data = (train_x, train_y)
    val_data = (val_x, val_y)
    return train_data, val_data
```

data_augmentation.py

```
import numpy as np
from random import shuffle
from .preprocessor import preprocess_input
from .preprocessor import _imread as imread
from .preprocessor import _imresize as imresize
from .preprocessor import to_categorical
import scipy.ndimage as ndi
import cv2

class ImageGenerator(object):
    """ Image generator with saturation, brightness, lighting,
contrast,
    horizontal flip and vertical flip transformations. It supports
    bounding boxes coordinates.

    TODO:
        - Finish support for not using bounding_boxes
            - Random crop
            - Test other transformations
    """
    def __init__(self, ground_truth_data, batch_size, image_size,
                 train_keys, validation_keys,
                 ground_truth_transformer=None,
                 path_prefix=None,
                 saturation_var=0.5,
                 brightness_var=0.5,
                 contrast_var=0.5,
                 lighting_std=0.5,
                 horizontal_flip_probability=0.5,
                 vertical_flip_probability=0.5,
                 do_random_crop=False,
                 grayscale=False,
                 zoom_range=[0.75, 1.25],
                 translation_factor=.3):

        self.ground_truth_data = ground_truth_data
        self.ground_truth_transformer = ground_truth_transformer
        self.batch_size = batch_size
        self.path_prefix = path_prefix
        self.train_keys = train_keys
        self.validation_keys = validation_keys
        self.image_size = image_size
        self.grayscale = grayscale
        self.color_jitter = []
        if saturation_var:
            self.saturation_var = saturation_var
            self.color_jitter.append(self.saturation)
```

```
        if brightness_var:
            self.brightness_var = brightness_var
            self.color_jitter.append(self.brightness)
        if contrast_var:
            self.contrast_var = contrast_var
            self.color_jitter.append(self.contrast)
        self.lighting_std = lighting_std
        self.horizontal_flip_probability = horizontal_flip_probability
        self.vertical_flip_probability = vertical_flip_probability
        self.do_random_crop = do_random_crop
        self.zoom_range = zoom_range
        self.translation_factor = translation_factor

    def _do_random_crop(self, image_array):
        """IMPORTANT: random crop only works for classification since
the
        current implementation does no transform bounding boxes"""
        height = image_array.shape[0]
        width = image_array.shape[1]
        x_offset = np.random.uniform(0, self.translation_factor *
width)
        y_offset = np.random.uniform(0, self.translation_factor *
height)
        offset = np.array([x_offset, y_offset])
        scale_factor = np.random.uniform(self.zoom_range[0],
                                         self.zoom_range[1])
        crop_matrix = np.array([[scale_factor, 0],
                                [0, scale_factor]])

        image_array = np.rollaxis(image_array, axis=-1, start=0)
        image_channel =
[ndi.interpolation.affine_transform(image_channel,
                         crop_matrix, offset=offset, order=0,
mode='nearest',
                         cval=0.0) for image_channel in image_array]

        image_array = np.stack(image_channel, axis=0)
        image_array = np.rollaxis(image_array, 0, 3)
        return image_array

    def do_random_rotation(self, image_array):
        """IMPORTANT: random rotation only works for classification
since the
        current implementation does no transform bounding boxes"""
        height = image_array.shape[0]
        width = image_array.shape[1]
        x_offset = np.random.uniform(0, self.translation_factor *
width)
        y_offset = np.random.uniform(0, self.translation_factor *
height)
```

```
        offset = np.array([x_offset, y_offset])
        scale_factor = np.random.uniform(self.zoom_range[0],
                                         self.zoom_range[1])
        crop_matrix = np.array([[scale_factor, 0],
                                [0, scale_factor]])

        image_array = np.rollaxis(image_array, axis=-1, start=0)
        image_channel =
[ndi.interpolation.affine_transform(image_channel,
                         crop_matrix, offset=offset, order=0,
mode='nearest',
                         cval=0.0) for image_channel in image_array]

        image_array = np.stack(image_channel, axis=0)
        image_array = np.rollaxis(image_array, 0, 3)
        return image_array

    def _gray_scale(self, image_array):
        return image_array.dot([0.299, 0.587, 0.114])

    def saturation(self, image_array):
        gray_scale = self._gray_scale(image_array)
        alpha = 2.0 * np.random.random() * self.brightness_var
        alpha = alpha + 1 - self.saturation_var
        image_array = alpha * image_array + (1 - alpha) *
gray_scale[:, :, None]
        return np.clip(image_array, 0, 255)

    def brightness(self, image_array):
        alpha = 2 * np.random.random() * self.brightness_var
        alpha = alpha + 1 - self.saturation_var
        image_array = alpha * image_array
        return np.clip(image_array, 0, 255)

    def contrast(self, image_array):
        gray_scale = (self._gray_scale(image_array).mean() *
                      np.ones_like(image_array))
        alpha = 2 * np.random.random() * self.contrast_var
        alpha = alpha + 1 - self.contrast_var
        image_array = image_array * alpha + (1 - alpha) * gray_scale
        return np.clip(image_array, 0, 255)

    def lighting(self, image_array):
        covariance_matrix = np.cov(image_array.reshape(-1,3) /
                                   255.0, rowvar=False)
        eigen_values, eigen_vectors =
np.linalg.eigh(covariance_matrix)
        noise = np.random.randn(3) * self.lighting_std
        noise = eigen_vectors.dot(eigen_values * noise) * 255
```

```
        image_array = image_array + noise
        return np.clip(image_array, 0 ,255)

    def horizontal_flip(self, image_array, box_corners=None):
        if np.random.random() < self.horizontal_flip_probability:
            image_array = image_array[:, ::-1]
            if box_corners != None:
                box_corners[:, [0, 2]] = 1 - box_corners[:, [2, 0]]
        return image_array, box_corners

    def vertical_flip(self, image_array, box_corners=None):
        if (np.random.random() < self.vertical_flip_probability):
            image_array = image_array[::-1]
            if box_corners != None:
                box_corners[:, [1, 3]] = 1 - box_corners[:, [3, 1]]
        return image_array, box_corners

    def transform(self, image_array, box_corners=None):
        shuffle(self.color_jitter)
        for jitter in self.color_jitter:
            image_array = jitter(image_array)

        if self.lighting_std:
            image_array = self.lighting(image_array)

        if self.horizontal_flip_probability > 0:
            image_array, box_corners =
self.horizontal_flip(image_array,

box_corners)

        if self.vertical_flip_probability > 0:
            image_array, box_corners = self.vertical_flip(image_array,

box_corners)
        return image_array, box_corners

    def preprocess_images(self, image_array):
        return preprocess_input(image_array)

    def flow(self, mode='train'):
            while True:
                if mode =='train':
                    shuffle(self.train_keys)
                    keys = self.train_keys
                elif mode == 'val' or  mode == 'demo':
                    shuffle(self.validation_keys)
                    keys = self.validation_keys
                else:
```

```
                    raise Exception('invalid mode: %s' % mode)

                inputs = []
                targets = []
                for key in keys:
                    image_path = self.path_prefix + key
                    image_array = imread(image_path)
                    image_array = imresize(image_array,
self.image_size)

                    num_image_channels = len(image_array.shape)
                    if num_image_channels != 3:
                        continue

                    ground_truth = self.ground_truth_data[key]

                    if self.do_random_crop:
                        image_array =
self._do_random_crop(image_array)

                    image_array = image_array.astype('float32')
                    if mode == 'train' or mode == 'demo':
                        if self.ground_truth_transformer != None:
                            image_array, ground_truth =
self.transform(

image_array,

ground_truth)
                            ground_truth = (

self.ground_truth_transformer.assign_boxes(

ground_truth))
                        else:
                            image_array =
self.transform(image_array)[0]

                    if self.grayscale:
                        image_array =
cv2.cvtColor(image_array.astype('uint8'),

cv2.COLOR_RGB2GRAY).astype('float32')
                        image_array = np.expand_dims(image_array, -1)

                    inputs.append(image_array)
                    targets.append(ground_truth)
                    if len(targets) == self.batch_size:
                        inputs = np.asarray(inputs)
```

```
                        targets = np.asarray(targets)
                        # this will not work for boxes
                        targets = to_categorical(targets)
                        if mode == 'train' or mode == 'val':
                            inputs = self.preprocess_images(inputs)
                            yield self._wrap_in_dictionary(inputs,
targets)
                        if mode == 'demo':
                            yield self._wrap_in_dictionary(inputs,
targets)
                        inputs = []
                        targets = []

    def _wrap_in_dictionary(self, image_array, targets):
        return [{'input_1':image_array},
                {'predictions':targets}]
```

Summary

We have proposed and tested a general building design for creating real-time CNNs. Our proposed architectures have been systematically built in order to reduce the amount of parameters. We began by eliminating completely the fully connected layers and by reducing the amount of parameters in the remaining convolutional layers via depth-wise separable convolutions. We have shown that our proposed models can be stacked for multi-class classifications while maintaining real-time inferences. Specifically, we have developed a vision system that performs face detection, gender classification and emotion classification in a single integrated module. We have achieved human-level performance in our classifications tasks using a single CNN that leverages modern architecture constructs. Our architecture reduces the amount of parameters 80x while obtaining favorable results.

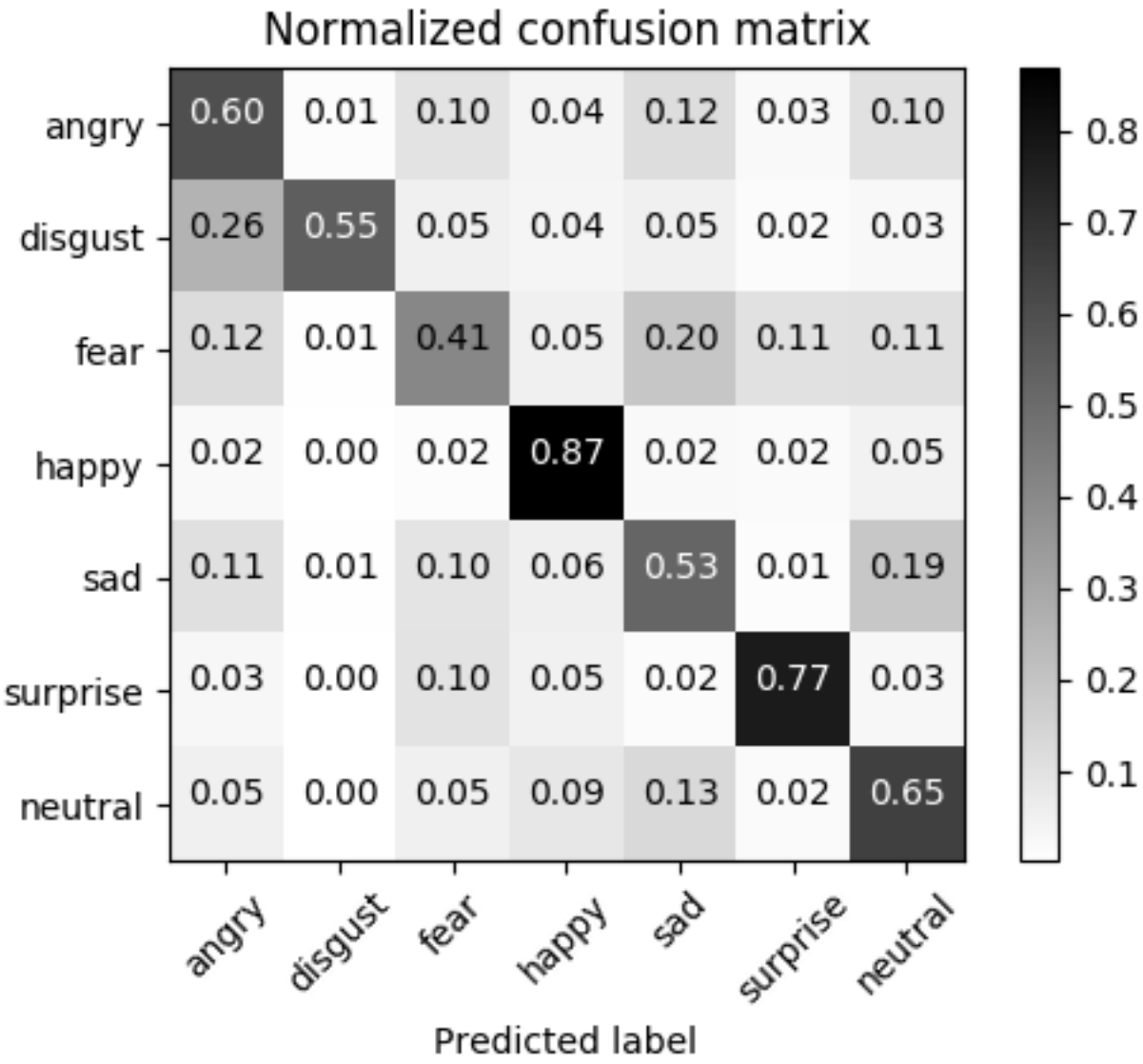


Figure 19: Confusion Matrix

Future Work

Machine learning models are biased in accordance to their training data. In our specific application we have empirically found that our trained CNNs for gender classification are biased towards western facial features and facial accessories. We hypothesize that this misclassification occurs since our training dataset consist of mostly western: actors, writers and cinematographers as observed.

Furthermore, as discussed previously, the use of glasses might affect the emotion classification by interfering with the features learned. However, the use of glasses can also interfere with the gender classification. This might be a result from the training data having most of the images of persons wearing glasses assigned with the label “man”. We believe that uncovering such behaviors is of extreme importance when creating robust classifiers, and that the use of the visualization techniques such as guided back-propagation will become invaluable when uncovering model biases.

- To increase the Accuracy & training it on cloud.
- Will try to Completely diminish the occlusions problem.
- Using Flask Libraries.

app/routes.py: Home page route

```
from app import app

@app.route('/')
@app.route('/index')
def index():
    return "Hello, World!"
```

Figure 20: Flask (python) Framework

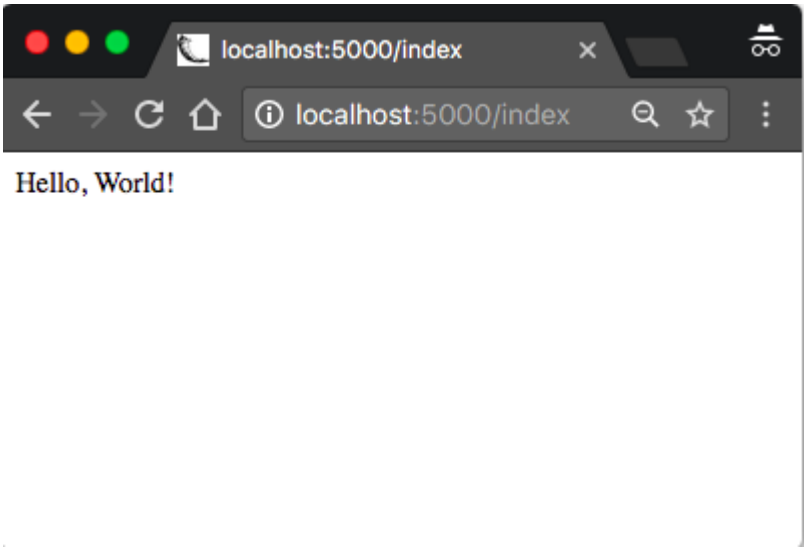


Figure 21: Output by Flask Library

Application

In recent years there has been a growing interest in improving all aspects of the interaction between humans and computers. This emerging field has been a research interest for scientists from several different scholastic tracks, i.e., computer science, engineering, psychology, and neuroscience. These studies focus not only on improving computer interfaces, but also on improving the actions the computer takes based on feedback from the user. Feedback from the user has traditionally been given through the keyboard and mouse. Other devices have also been developed for more application specific interfaces, such as joysticks, trackballs, data gloves, and touch screens. The rapid advance of 188 Application: Facial Expression Recognition technology in recent years has made computers cheaper and more powerful and has made the use of microphones and PC-cameras affordable and easily available. The microphones and cameras enable the computer to “see” and “hear,” and to use this information to act. A good example of this is the “SmartKiosk”. It is argued that to truly achieve effective human-

computer intelligent interaction (HCII), there is a need for the computer to be able to interact naturally with the user, similar to the way human-human interaction takes place. Human beings possess and express emotions in everyday interactions with others. Emotions are often reflected on the face, in hand and body gestures, and in the voice, to express our feelings or likings. While a precise, generally agreed upon definition of emotion does not exist, it is undeniable that emotions are an integral part of our existence. Facial expressions and vocal emotions are commonly used in everyday human-to-human communication, as one smiles to show greeting, frowns when confused, or raises one's voice when enraged. People do a great deal of inference from perceived facial expressions: "You look tired," or "You seem happy." The fact that we understand emotions and know how to react to other people's expressions greatly enriches the interaction. There is a growing amount of evidence showing that emotional skills are part of what is called "intelligence". Computers today, on the other hand, are still quite "emotionally challenged." They neither recognize the

user's emotions nor possess emotions of their own. Psychologists and engineers alike have tried to analyze facial expressions in an attempt to understand and categorize these expressions. This knowledge can be for example used to teach computers to recognize human emotions from video images acquired from built-in cameras. In some applications, it may not be necessary for computers to recognize emotions. For example, the computer inside an automatic teller machine or an airplane probably does not need to recognize emotions. However, in applications where computers take on a social role such as an "instructor," "helper," or even "companion," it may enhance their functionality to be able to recognize users' emotions. In her book, Picard suggested several applications where it is beneficial for computers to recognize human emotions. For example, knowing the user's emotions, the computer can become a more effective tutor. Synthetic speech with emotions in the voice would sound more pleasing than a monotonous voice. Computer "agents" could learn the user's preferences through the users' emotions. Another application

is to help the human users monitor their stress level. In clinical settings, recognizing a person's inability to express certain facial expressions may help diagnose early psychological disorders.

References

- Facial Expression Recognition using Convolutional Neural Networks: State of the Art, Pramerdorfer,Martin,2016.
- Facial Expression Recognition Using Salient Features and Convolutional Neural Network ,MD. ZIA UDDIN , (Senior Member, IEEE), WERIA KHAKSAR,AND JIM TORRESEN, (Senior Member, IEEE),2017
- Andree Pascu. Facial Expression Recognition .
- https://en.wikipedia.org/wiki/Facial_expression_databases
- https://en.wikipedia.org/wiki/Convolutional_neural_network
- https://keras.io/why-use-keras/#why-use-keras
- https://www.kaggle.com/ssrihari/keras-cnn-example-98-accuracy
- https://en.wikipedia.org/wiki/Facial_expression_databases
- https://en.wikipedia.org/wiki/Convolutional_neural_network
- https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9
- Tensorflow for poets.
- Udacity Deep learning Course by google.

- Andrew ng CNN Lectures.
- https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data
- Franc¸ois Chollet. Xception: Deep learning with depthwise separable convolutions. CoRR, abs/1610.02357, 2016.
- Andrew G. Howard et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. CoRR, abs/1704.04861, 2017.
- Dario Amodei et al. Deep speech 2: End-to-end speech recognition in english and mandarin. CoRR, abs/1512.02595, 2015.
- Ian Goodfellow et al. Challenges in Representation Learning: A report on three machine learning contests, 2013.
- Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pages 315–323,2011.