FACIAL RECOGNITION USING DEEP CONVOLUTIONAL NEURAL NETWORK

ARTIFICIAL INTELLIGENCE PROJECT

BY

USEN OSASUMWEN (171123022)

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE

IN

FACULTY OF NATURAL AND APPLIED SCIENCE

NIGERIAN TURKISH NILE UNIVERSITY

SUPERVISOR: PROF NNANNA NWOJO

ACKNOWLEDGEMENTS

Firstly, I would like to express a special appreciation to my advisor Professor Nnanna Nwojo. I would like to thank him for encouraging my research and for allowing me to grow as a research scientist. I would also like to thank all my classmates for their scientific and technical support.

TABLE OF CONTENTS

ABSTRACT

A method to produce classification models to automatically recognize a face in an image. The method takes advantage of a FaceNet facial classification model to extract feature embeddings corresponding to the individual facial features. The embeddings from the FaceNet model are a 128-dimensional vector. The L2 distance between the target embedding and the stored embedding is computed. By comparing two such vectors (target image and stored image), you can determine if the two pictures are of the same individual. A triplet loss is also used to "push" the encoding of the same person closer together, while "pulling" the encoding of two different people further apart. This method can be applied to face verification and face recognition tasks.

keywords: Deep Neural Networks (DNN), Convolutional Neural Networks (CNN), FaceNet, Triplet Loss

1. INTRODUCTION

Face recognition is the task of identifying an already detected face in an image as known or unknown. Face recognition tasks is often confused with face detection and face verification tasks, face detection identifies the presence of a face in an image. Face verification is a 1:1 matching problem where two faces are compared. On the other hand, face recognition is a 1:K matching problem that decides if a face is known or unknown, a database of faces is needed in order to validate the input face. Due to the rapid advancements in deep learning in natural language processing and more specifically computer vision tasks, the quality of image recognition and object detection has progressed at a dramatic pace. Face recognition technology can have major social, economic and cultural implications on our lives.

This paper discusses a method for developing face recognition systems using deep convolutional neural network. An implementation of FaceNet, Schroff et al. [1] is used to generate 128-dimensional vector embeddings of the input image. These embeddings are stored in a database and are used to compare with a target image if the face is someone known or not.

A. Deep Neural Network

Deep Neural Networks (DNN) are computational models composed of multiple non-linear layers that are able to learn functions that represents high level abstractions(i.e. speech, vision) [2]. DNN's are currently the foundation of many modern artificial intelligence applications like speech recognition, image recognition, self driving cars, playing complex games and are able to exceed human accuracy in many of these tasks. DNN are trained to find patterns and extract high level features from large amounts of datasets to obtain a meaningful representation of the feature space [3].

Neural networks are modeled after the human brain, consisting of processing units called *neurons* connected together and organized hierarchically. The connection between neurons are called *synapses*, synapses contain weights that determines how important the input is of the lower level neuron for the output of the higher level neuron. In addition, each neuron contain a bias that represents how likely in general a pattern exists. The values of this weights and biases are the parameters of the neural network that will be learned during the training process.

Afterwards, an activation function is applied to the output of the neural network to estimate complex functions by performing non-linear transformations.

B. Deep Convolutional Neural Network

A variant of standard artificial neural networks commonly used in image recognition tasks. In a CNN, the neurons of a layer are organized across three dimensions, height, width and depth, depth relating to the different color channels of an image. Instead of using fully connected dense layers, CNN introduce new network layers, *convolutional* and *pooling* layers. Each convolutional layer in the CNN are able to learn a high level abstraction of the input data called feature maps, the neuron are connected to a small subset of neurons in the previous layer across all three dimensions. In the pooling layer, computations are performed on the output feature map to reduce its dimensionality. Neurons in the pooling layer do not have weights and biases to be learned, instead they perform a fixed function on the input feature map.

C. FaceNet

In the paper [1], the researchers presented a system that learns a mapping from face images to an Euclidean space where distance is related to a measure of face similarity. This method uses a deep convolutional neural network architecture inspired by the inception model [4], trained to optimize the embeddings itself. During training the triplet loss function was applied, this enables faces of the same person to converge closer to a single point in the embedding space and diverge further away faces of different people. The loss that is being minimized is,

$$\sum_{i}^{N} \left[\| f(x_i^a) - f(x_i^p) \|_2^2 - \| f(x_i^a) - f(x_i^n) \|_2^2 + \alpha \right]_+$$

Where, x^a_i is the embedding of an image of a person. x^p_i is the embedding of an image of the same person, x^a_i is the embedding of any other person and α, the margin enforced between positive and negative pairs. The model was trained on multiple datasets and achieved an accuracy of 95.12-99.63% and cuts error down 30%. A trained FaceNet model is used to generate embeddings for the proposed system.

2. FORMULATION OF THE PROBLEM

Face recognition tasks commonly fall in two categories, face verification and face recognition. Face verification answers the question “is this the claimed person?”. An example of face verification can be seen at airports, where you scan your passport and the system tries to verify that you are who you say you are. Face verification answers the question “who is this person?”. For example, Nile staff entering offices or designated offices without needing to identify themselves. In this paper, a state of the art method for face recognition is presented using pre-trained CNN’s.

3. DESIGN

A. Load Packages

Firstly, the required packages and libraries are loaded to the application. Keras library, high-level neural networks API, written in Python for defining and configuring neural networks and their parameters. Tensorflow, a low-level open source machine learning library. Pandas is an open source library providing high performance data structures and data analysis tools in python. Numpy is a scientific computing package in python to facilitate advanced mathematical operation on large sets of data. The fr_utils and inception_blocks_v2 are helper functions for loading and defining the architecture of the neural network.

```
from keras.models import Sequential
from keras.layers import Conv2D, ZeroPadding2D, Activation, Input,
concatenate
from keras.models import Model
from keras.layers.normalization import BatchNormalization
from keras.layers.pooling import MaxPooling2D, AveragePooling2D
from keras.layers.merge import Concatenate
from keras.layers.core import Lambda, Flatten, Dense
from keras.initializers import glorot_uniform
from keras.engine.topology import Layer
from keras import backend as K
K.set_image_data_format('channels_first')
import cv2
import os
import numpy as np
from numpy import genfromtxt
import pandas as pd
import tensorflow as tf
from fr_utils import *
from inception_blocks_v2 import *
```

Code 1. Code snippet to load required libraries.

B. Load the Model

Due to the computation cost of training a CNN from scratch, i employ a common used technique in deep learning, Transfer Learning. This allows the use of already trained neural network models for other tasks, mostly used in computer vision tasks. The inception CNN architecture [4] is defined, then the pre-trained FaceNet weights are loaded. The function takes

an input parameter, input_shape is passed corresponding to the depth (RGB channels), height and width of the image.

```
FRmodel = faceRecoModel(input_shape=(3, 96, 96))
```

Code 2. Code snippet calling the implementation of the inception CNN architecture

```
def faceRecoModel(input_shape):
    """
    Implementation of the Inception model used for FaceNet
    Arguments:
    input_shape -- shape of the images of the dataset
   Returns:
    model -- a Model() instance in Keras
    """
   # Define the input as a tensor with shape input_shape
    X_input = Input(input_shape)
   # Zero-Padding
    X = ZeroPadding2D((3, 3))(X_input)
       # First Block
    X = Conv2D(64, (7, 7), strides = (2, 2), name = 'conv1')(X)
    X = BatchNormalization(axis = 1, name = 'bn1')(X)
    X = Activation('relu')(X)
   # Zero-Padding + MAXPOOL
    X = ZeroPadding2D((1, 1))(X)
    X = MaxPooling2D((3, 3), strides = 2)(X)
   # Second Block
    X = Conv2D(64, (1, 1), strides = (1, 1), name = 'conv2')(X)
    X = BatchNormalization(axis = 1, epsilon=0.00001, name = 'bn2')(X)
    X = Activation('relu')(X)
       # Zero-Padding + MAXPOOL
    X = ZeroPadding2D((1, 1))(X)
   # Second Block
    X = Conv2D(192, (3, 3), strides = (1, 1), name = 'conv3')(X)
    X = BatchNormalization(axis = 1, epsilon=0.00001, name = 'bn3')(X)
    X = Activation('relu')(X)
   # Zero-Padding + MAXPOOL
    X = ZeroPadding2D((1, 1))(X)
    X = MaxPooling2D(pool_size = 3, strides = 2)(X)
   # Inception 1: a/b/c
    X = inception_block_1a(X)
    X = inception_block_1b(X)
    X = inception_block_1c(X)
   # Inception 2: a/b
    X = inception_block_2a(X)
    X = inception_block_2b(X)
   # Inception 3: a/b
    X = inception_block_3a(X)
    X = inception_block_3b(X)
   # Top layer
    X = AveragePooling2D(pool_size=(3, 3), strides=(1, 1),
data_format='channels_first')(X)
    X = Flatten()(X)
    X = Dense(128, name='dense_layer')(X)
       # L2 normalization
```

```
    X = Lambda(lambda  x: K.l2_normalize(x,axis=1))(X)
   # Create model instance
    model = Model(inputs = X_input, outputs = X, name='FaceRecoModel')
           return model
```

Code 3. Function for implementing the inception CNN architecture

C. Compiling the Model

The model is compiled using the "Adam" optimizer, "accuracy" metrics for evaluation and "triplet loss function", the loss to minimize. The triplet loss function takes three parameters. "y_true", actual labelled values, required when defining losses in Keras. "y_pred", list containing the embedding for anchor image (image of a person), embedding for positive image (image of the same person) and embedding for negative image (image of a different person). Lastly, alpha, α as described in previous sections.

```
def triplet_loss(y_true, y_pred, alpha = 0.2):
    """
    Implementation of the triplet loss as defined by the formula
   Arguments:
    y_true -- true labels, required when you define a loss in Keras, you
don't need it in this function.
    y_pred -- python list containing three objects:
            anchor -- the encodings for the anchor images, of shape (None,
128)
            positive -- the encodings for the positive images, of shape
(None, 128)
            negative -- the encodings for the negative images, of shape
(None, 128)
   Returns:
    loss -- real number, value of the loss
    """
   anchor, positive, negative = y_pred[0], y_pred[1], y_pred[2]
   # Step 1: Compute the (encoding) distance between the anchor and the
positive, you will need to sum over axis=-1
    pos_dist =  tf.reduce_sum( tf.square( tf.subtract(anchor, positive)),
axis=-1)
    # Step 2: Compute the (encoding) distance between the anchor and the
negative, you will need to sum over axis=-1
    neg_dist = tf.reduce_sum( tf.square( tf.subtract(anchor,
negative)),axis=-1)
    # Step 3: subtract the two previous distances and add alpha.
    basic_loss = (pos_dist - neg_dist) + alpha
    # Step 4: Take the maximum of basic_loss and 0.0. Sum over the training
examples.
    loss = tf.reduce_sum(tf.maximum(basic_loss,0))
   return loss
```

Code 4. Function for implementing the triplet loss.

```
FRmodel.compile(optimizer = 'adam', loss = triplet_loss, metrics =
['accuracy'])
load_weights_from_FaceNet(FRmodel)
```

Code 5. Code snippet to compile the CNN model.

D. Define the Database

The database is defined as a python dictionary with keys corresponding to names of stored individual and values as 128-vector embedding of the individual's image. The image to encoding function takes two parameters, path to the image and CNN model to generate the embeddings.

```
database = {}
database["femi"] = img_to_encoding("images/femi.JPG", FRmodel)
database["osas"] = img_to_encoding("images/osas.JPG", FRmodel)
database["anita"] = img_to_encoding("images/anita.JPG", FRmodel)
database["oreva"] = img_to_encoding("images/oreva.jpg", FRmodel)
database["pelumi"] = img_to_encoding("images/pelumi.jpg", FRmodel)
```

Code 6. Code snippet to define the database dictionary.

```
def img_to_encoding(image_path, model):
    img1 = cv2.imread(image_path, 1)
    img = img1[...,::-1]
    img = np.around(np.transpose(img, (2,0,1))/255.0, decimals=12)
    x_train = np.array([img])
    embedding = model.predict_on_batch(x_train)
    return embedding
```

Code 7. Function to convert images to 128-dimensional vector.

E. Face Recognition

After the database has been loaded with encodings of known individuals, the face recognition function is defined for finding the presence of a known individual in an input image. This function takes three parameters, *image_path*, path to the target image, *database,* database containing image encodings of known individuals and model, the pre-trained FaceNet model. The function compares the target image and images in the database by computing the L2 distance between the encoding of the target image and encoding of images in the database. If the distance between encodings are less that 0.7, that means a face has been recognized.

```
def who_is_it(image_path, database, model):
    """
    Implements face recognition function for finding who is the person on the
image_path image.
       Arguments:
    image_path -- path to an image
    database -- database containing image encodings along with the name of
the person on the image
    model -- your Inception model instance in Keras
   Returns:
    min_dist -- the minimum distance between image_path encoding and the
encodings from the database
    identity -- string, the name prediction for the person on image_path
    """
   ## Step 1: Compute the target "encoding" for the image.
    encoding = img_to_encoding(image_path, model)
   ## Step 2: Find the closest encoding ##
   # Initialize "min_dist" to a large value, say 100
    min_dist = 100
   # Loop over the database dictionary's names and encodings.
    for (name, db_enc) in database.items():
       # Compute L2 distance between the target "encoding" and the current
"emb" from the database.
        dist = np.linalg.norm(db_enc - encoding)
       # If this distance is less than the min_dist, then set min_dist to
dist, and identity to name.
        if dist < min_dist:
            min_dist = dist
            identity = name
   if min_dist > 0.7:
        print("Not in the database.")
    else:
        print ("it's " + str(identity) + ", the distance is " +
str(min_dist))
   return min_dist, identity
```

Code 8. Function to compute L2 distance and to recognize individuals in images.

```
who_is_it("images/image.JPG", database, FRmodel)
```

Code 9. Code snippet to call the face recognition function.

4. ANALYSIS AND RESULT

The proposed face recognition system works well at identifying faces of similar individuals. The proposed system is currently the state-of-the-art face recognition system. More can be done to significantly increase the accuracy of the system, like, adding more pictures (under different lighting conditions) of individuals in the database, also, faces can also be cropped to contain just the face of the image, this removes some unnecessary pixel form the image making the algorithm more robust.

APPENDIX

[1]. Florian Schroff, D.k., J. p. (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering. *Arxiv, 1503.03832v3*.

[2]. Vivienne Sze, Y. c., T. y, J. e. (2017). Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Arxiv, 1703.09039v2*.

[3]. Felix Altenberger, C. l. (2018). A Non-Technical Survey on Deep Convolutional Neural Network Architectures. *Arxiv, 1803.02129v1*.

[4]. Christian Szegedy, W. l., Y. j., P. s., et.al. (2014). Going deeper with convolutions. *Arxiv, 1409.4842*.

main.py

```
from keras.models import Sequential
from keras.layers import Conv2D, ZeroPadding2D, Activation, Input,
concatenate
from keras.models import Model
from keras.layers.normalization import BatchNormalization
from keras.layers.pooling import MaxPooling2D, AveragePooling2D
from keras.layers.merge import Concatenate
from keras.layers.core import Lambda, Flatten, Dense
from keras.initializers import glorot_uniform
from keras.engine.topology import Layer
from keras import backend as K
K.set_image_data_format('channels_first')
import cv2
import os
import numpy as np
from numpy import genfromtxt
import pandas as pd
import tensorflow as tf
from fr_utils import *
from inception_blocks_v2 import *

np.set_printoptions(threshold=np.nan)

print('Building CNN Architecture...')
FRmodel = faceRecoModel(input_shape=(3, 96, 96))

print("Total Params:", FRmodel.count_params())

def triplet_loss(y_true, y_pred, alpha = 0.2):
    """
    Implementation of the triplet loss as defined by the formula

    Arguments:
    y_true -- true labels, required when you define a loss in Keras, you
don't need it in this function.
    y_pred -- python list containing three objects:
            anchor -- the encodings for the anchor images, of shape (None,
128)
```

```
            positive -- the encodings for the positive images, of shape
(None, 128)
            negative -- the encodings for the negative images, of shape
(None, 128)

    Returns:
    loss -- real number, value of the loss
    """

    anchor, positive, negative = y_pred[0], y_pred[1], y_pred[2]

    # Step 1: Compute the (encoding) distance between the anchor and the
positive, you will need to sum over axis=-1
    pos_dist =  tf.reduce_sum( tf.square( tf.subtract(anchor, positive)),
axis=-1)
    # Step 2: Compute the (encoding) distance between the anchor and the
negative, you will need to sum over axis=-1
    neg_dist = tf.reduce_sum( tf.square( tf.subtract(anchor,
negative)),axis=-1)
    # Step 3: subtract the two previous distances and add alpha.
    basic_loss = (pos_dist - neg_dist) + alpha
    # Step 4: Take the maximum of basic_loss and 0.0. Sum over the training
examples.
    loss = tf.reduce_sum(tf.maximum(basic_loss,0))

    return loss

FRmodel.compile(optimizer = 'adam', loss = triplet_loss, metrics =
['accuracy'])
load_weights_from_FaceNet(FRmodel)

database = {}
database["femi"] = img_to_encoding("images/femi.JPG", FRmodel)
database["osas"] = img_to_encoding("images/osas.JPG", FRmodel)
database["anita"] = img_to_encoding("images/anita.JPG", FRmodel)
database["oreva"] = img_to_encoding("images/oreva.jpg", FRmodel)
database["pelumi"] = img_to_encoding("images/pelumi.jpg", FRmodel)

def who_is_it(image_path, database, model):
    """
    Implements face recognition function for finding who is the person on the
image_path image.

    Arguments:
    image_path -- path to an image
    database -- database containing image encodings along with the name of
the person on the image
    model -- your Inception model instance in Keras

    Returns:
    min_dist -- the minimum distance between image_path encoding and the
encodings from the database
    identity -- string, the name prediction for the person on image_path
    """

    ## Step 1: Compute the target "encoding" for the image.
    encoding = img_to_encoding(image_path, model)
```

```
    ## Step 2: Find the closest encoding ##

    # Initialize "min_dist" to a large value, say 100
    min_dist = 100

    # Loop over the database dictionary's names and encodings.
    for (name, db_enc) in database.items():

        # Compute L2 distance between the target "encoding" and the current
"emb" from the database.
        dist = np.linalg.norm(db_enc - encoding)

        # If this distance is less than the min_dist, then set min_dist to
dist, and identity to name.
        if dist < min_dist:
            min_dist = dist
            identity = name

    if min_dist > 0.7:
        print("Not in the database.")
    else:
        print ("it's " + str(identity) + ", the distance is " +
str(min_dist))

    return min_dist, identity


who_is_it("images/image.JPG", database, FRmodel)
```

fr_utils.py

```
import tensorflow as tf
import numpy as np
import os
import cv2
from numpy import genfromtxt
from keras.layers import Conv2D, ZeroPadding2D, Activation, Input,
concatenate
from keras.models import Model
from keras.layers.normalization import BatchNormalization
from keras.layers.pooling import MaxPooling2D, AveragePooling2D
import h5py
# import matplotlib.pyplot as plt

_FLOATX = 'float32'

def variable(value, dtype=_FLOATX, name=None):
    v = tf.Variable(np.asarray(value, dtype=dtype), name=name)
    _get_session().run(v.initializer)
    return v

def shape(x):
    return x.get_shape()

def square(x):
    return tf.square(x)

def zeros(shape, dtype=_FLOATX, name=None):
```

```
    return variable(np.zeros(shape), dtype, name)

def concatenate(tensors, axis=-1):
    if axis < 0:
        axis = axis % len(tensors[0].get_shape())
    return tf.concat(axis, tensors)

def LRN2D(x):
    return tf.nn.lrn(x, alpha=1e-4, beta=0.75)

def conv2d_bn(x,
              layer=None,
              cv1_out=None,
              cv1_filter=(1, 1),
              cv1_strides=(1, 1),
              cv2_out=None,
              cv2_filter=(3, 3),
              cv2_strides=(1, 1),
              padding=None):
    num = '' if cv2_out == None else '1'
    tensor = Conv2D(cv1_out, cv1_filter, strides=cv1_strides,
data_format='channels_first', name=layer+'_conv'+num)(x)
    tensor = BatchNormalization(axis=1, epsilon=0.00001,
name=layer+'_bn'+num)(tensor)
    tensor = Activation('relu')(tensor)
    if padding == None:
        return tensor
    tensor = ZeroPadding2D(padding=padding, data_format='channels_first')
(tensor)
    if cv2_out == None:
        return tensor
    tensor = Conv2D(cv2_out, cv2_filter, strides=cv2_strides,
data_format='channels_first', name=layer+'_conv'+'2')(tensor)
    tensor = BatchNormalization(axis=1, epsilon=0.00001,
name=layer+'_bn'+'2')(tensor)
    tensor = Activation('relu')(tensor)
    return tensor

WEIGHTS = [
  'conv1', 'bn1', 'conv2', 'bn2', 'conv3', 'bn3',
  'inception_3a_1x1_conv', 'inception_3a_1x1_bn',
  'inception_3a_pool_conv', 'inception_3a_pool_bn',
  'inception_3a_5x5_conv1', 'inception_3a_5x5_conv2', 'inception_3a_5x5_bn1',
'inception_3a_5x5_bn2',
  'inception_3a_3x3_conv1', 'inception_3a_3x3_conv2', 'inception_3a_3x3_bn1',
'inception_3a_3x3_bn2',
  'inception_3b_3x3_conv1', 'inception_3b_3x3_conv2', 'inception_3b_3x3_bn1',
'inception_3b_3x3_bn2',
  'inception_3b_5x5_conv1', 'inception_3b_5x5_conv2', 'inception_3b_5x5_bn1',
'inception_3b_5x5_bn2',
  'inception_3b_pool_conv', 'inception_3b_pool_bn',
  'inception_3b_1x1_conv', 'inception_3b_1x1_bn',
  'inception_3c_3x3_conv1', 'inception_3c_3x3_conv2', 'inception_3c_3x3_bn1',
'inception_3c_3x3_bn2',
  'inception_3c_5x5_conv1', 'inception_3c_5x5_conv2', 'inception_3c_5x5_bn1',
'inception_3c_5x5_bn2',
  'inception_4a_3x3_conv1', 'inception_4a_3x3_conv2', 'inception_4a_3x3_bn1',
'inception_4a_3x3_bn2',
```

```
  'inception_4a_5x5_conv1', 'inception_4a_5x5_conv2', 'inception_4a_5x5_bn1',
'inception_4a_5x5_bn2',
  'inception_4a_pool_conv', 'inception_4a_pool_bn',
  'inception_4a_1x1_conv', 'inception_4a_1x1_bn',
  'inception_4e_3x3_conv1', 'inception_4e_3x3_conv2', 'inception_4e_3x3_bn1',
'inception_4e_3x3_bn2',
  'inception_4e_5x5_conv1', 'inception_4e_5x5_conv2', 'inception_4e_5x5_bn1',
'inception_4e_5x5_bn2',
  'inception_5a_3x3_conv1', 'inception_5a_3x3_conv2', 'inception_5a_3x3_bn1',
'inception_5a_3x3_bn2',
  'inception_5a_pool_conv', 'inception_5a_pool_bn',
  'inception_5a_1x1_conv', 'inception_5a_1x1_bn',
  'inception_5b_3x3_conv1', 'inception_5b_3x3_conv2', 'inception_5b_3x3_bn1',
'inception_5b_3x3_bn2',
  'inception_5b_pool_conv', 'inception_5b_pool_bn',
  'inception_5b_1x1_conv', 'inception_5b_1x1_bn',
  'dense_layer'
]
```

```
conv_shape = {
  'conv1': [64, 3, 7, 7],
  'conv2': [64, 64, 1, 1],
  'conv3': [192, 64, 3, 3],
  'inception_3a_1x1_conv': [64, 192, 1, 1],
  'inception_3a_pool_conv': [32, 192, 1, 1],
  'inception_3a_5x5_conv1': [16, 192, 1, 1],
  'inception_3a_5x5_conv2': [32, 16, 5, 5],
  'inception_3a_3x3_conv1': [96, 192, 1, 1],
  'inception_3a_3x3_conv2': [128, 96, 3, 3],
  'inception_3b_3x3_conv1': [96, 256, 1, 1],
  'inception_3b_3x3_conv2': [128, 96, 3, 3],
  'inception_3b_5x5_conv1': [32, 256, 1, 1],
  'inception_3b_5x5_conv2': [64, 32, 5, 5],
  'inception_3b_pool_conv': [64, 256, 1, 1],
  'inception_3b_1x1_conv': [64, 256, 1, 1],
  'inception_3c_3x3_conv1': [128, 320, 1, 1],
  'inception_3c_3x3_conv2': [256, 128, 3, 3],
  'inception_3c_5x5_conv1': [32, 320, 1, 1],
  'inception_3c_5x5_conv2': [64, 32, 5, 5],
  'inception_4a_3x3_conv1': [96, 640, 1, 1],
  'inception_4a_3x3_conv2': [192, 96, 3, 3],
  'inception_4a_5x5_conv1': [32, 640, 1, 1,],
  'inception_4a_5x5_conv2': [64, 32, 5, 5],
  'inception_4a_pool_conv': [128, 640, 1, 1],
  'inception_4a_1x1_conv': [256, 640, 1, 1],
  'inception_4e_3x3_conv1': [160, 640, 1, 1],
  'inception_4e_3x3_conv2': [256, 160, 3, 3],
  'inception_4e_5x5_conv1': [64, 640, 1, 1],
  'inception_4e_5x5_conv2': [128, 64, 5, 5],
  'inception_5a_3x3_conv1': [96, 1024, 1, 1],
  'inception_5a_3x3_conv2': [384, 96, 3, 3],
  'inception_5a_pool_conv': [96, 1024, 1, 1],
  'inception_5a_1x1_conv': [256, 1024, 1, 1],
  'inception_5b_3x3_conv1': [96, 736, 1, 1],
  'inception_5b_3x3_conv2': [384, 96, 3, 3],
  'inception_5b_pool_conv': [96, 736, 1, 1],
  'inception_5b_1x1_conv': [256, 736, 1, 1],
}
```

```
def load_weights_from_FaceNet(FRmodel):
    # Load weights from csv files (which was exported from Openface torch
model)
    weights = WEIGHTS
    weights_dict = load_weights()

    # Set layer weights of the model
    for name in weights:
        if FRmodel.get_layer(name) != None:
            FRmodel.get_layer(name).set_weights(weights_dict[name])
        elif model.get_layer(name) != None:
            model.get_layer(name).set_weights(weights_dict[name])

def load_weights():
    # Set weights path
    dirPath = './weights'
    fileNames = filter(lambda f: not f.startswith('.'), os.listdir(dirPath))
    paths = {}
    weights_dict = {}

    for n in fileNames:
        paths[n.replace('.csv', '')] = dirPath + '/' + n

    for name in WEIGHTS:
        if 'conv' in name:
            conv_w = genfromtxt(paths[name + '_w'], delimiter=',',
dtype=None)
            conv_w = np.reshape(conv_w, conv_shape[name])
            conv_w = np.transpose(conv_w, (2, 3, 1, 0))
            conv_b = genfromtxt(paths[name + '_b'], delimiter=',',
dtype=None)
            weights_dict[name] = [conv_w, conv_b]
        elif 'bn' in name:
            bn_w = genfromtxt(paths[name + '_w'], delimiter=',', dtype=None)
            bn_b = genfromtxt(paths[name + '_b'], delimiter=',', dtype=None)
            bn_m = genfromtxt(paths[name + '_m'], delimiter=',', dtype=None)
            bn_v = genfromtxt(paths[name + '_v'], delimiter=',', dtype=None)
            weights_dict[name] = [bn_w, bn_b, bn_m, bn_v]
        elif 'dense' in name:
            dense_w = genfromtxt(dirPath+'/dense_w.csv', delimiter=',',
dtype=None)
            dense_w = np.reshape(dense_w, (128, 736))
            dense_w = np.transpose(dense_w, (1, 0))
            dense_b = genfromtxt(dirPath+'/dense_b.csv', delimiter=',',
dtype=None)
            weights_dict[name] = [dense_w, dense_b]

    return weights_dict


def load_dataset():
    train_dataset = h5py.File('datasets/train_happy.h5', "r")
    train_set_x_orig = np.array(train_dataset["train_set_x"][:]) # your train
set features
    train_set_y_orig = np.array(train_dataset["train_set_y"][:]) # your train
set labels
```

```
    test_dataset = h5py.File('datasets/test_happy.h5', "r")
    test_set_x_orig = np.array(test_dataset["test_set_x"][:]) # your test set
features
    test_set_y_orig = np.array(test_dataset["test_set_y"][:]) # your test set
labels

    classes = np.array(test_dataset["list_classes"][:]) # the list of classes

    train_set_y_orig = train_set_y_orig.reshape((1,
train_set_y_orig.shape[0]))
    test_set_y_orig = test_set_y_orig.reshape((1, test_set_y_orig.shape[0]))

    return train_set_x_orig, train_set_y_orig, test_set_x_orig,
test_set_y_orig, classes

def img_to_encoding(image_path, model):
    img1 = cv2.imread(image_path, 1)
    img = img1[...,::-1]
    img = np.around(np.transpose(img, (2,0,1))/255.0, decimals=12)
    x_train = np.array([img])
    embedding = model.predict_on_batch(x_train)
    return embedding
```

inception_blocks_v2.py

```
import tensorflow as tf
import numpy as np
import os
from numpy import genfromtxt
from keras import backend as K
from keras.layers import Conv2D, ZeroPadding2D, Activation, Input,
concatenate
from keras.models import Model
from keras.layers.normalization import BatchNormalization
from keras.layers.pooling import MaxPooling2D, AveragePooling2D
import fr_utils
from keras.layers.core import Lambda, Flatten, Dense

def inception_block_1a(X):
    """
    Implementation of an inception block
    """

    X_3x3 = Conv2D(96, (1, 1), data_format='channels_first', name
='inception_3a_3x3_conv1')(X)
    X_3x3 = BatchNormalization(axis=1, epsilon=0.00001, name =
'inception_3a_3x3_bn1')(X_3x3)
    X_3x3 = Activation('relu')(X_3x3)
    X_3x3 = ZeroPadding2D(padding=(1, 1), data_format='channels_first')
(X_3x3)
    X_3x3 = Conv2D(128, (3, 3), data_format='channels_first',
name='inception_3a_3x3_conv2')(X_3x3)
    X_3x3 = BatchNormalization(axis=1, epsilon=0.00001,
name='inception_3a_3x3_bn2')(X_3x3)
    X_3x3 = Activation('relu')(X_3x3)

    X_5x5 = Conv2D(16, (1, 1), data_format='channels_first',
name='inception_3a_5x5_conv1')(X)
```

```
    X_5x5 = BatchNormalization(axis=1, epsilon=0.00001,
name='inception_3a_5x5_bn1')(X_5x5)
    X_5x5 = Activation('relu')(X_5x5)
    X_5x5 = ZeroPadding2D(padding=(2, 2), data_format='channels_first')
(X_5x5)
    X_5x5 = Conv2D(32, (5, 5), data_format='channels_first',
name='inception_3a_5x5_conv2')(X_5x5)
    X_5x5 = BatchNormalization(axis=1, epsilon=0.00001,
name='inception_3a_5x5_bn2')(X_5x5)
    X_5x5 = Activation('relu')(X_5x5)

    X_pool = MaxPooling2D(pool_size=3, strides=2,
data_format='channels_first')(X)
    X_pool = Conv2D(32, (1, 1), data_format='channels_first',
name='inception_3a_pool_conv')(X_pool)
    X_pool = BatchNormalization(axis=1, epsilon=0.00001,
name='inception_3a_pool_bn')(X_pool)
    X_pool = Activation('relu')(X_pool)
    X_pool = ZeroPadding2D(padding=((3, 4), (3, 4)),
data_format='channels_first')(X_pool)

    X_1x1 = Conv2D(64, (1, 1), data_format='channels_first',
name='inception_3a_1x1_conv')(X)
    X_1x1 = BatchNormalization(axis=1, epsilon=0.00001,
name='inception_3a_1x1_bn')(X_1x1)
    X_1x1 = Activation('relu')(X_1x1)

    # CONCAT
    inception = concatenate([X_3x3, X_5x5, X_pool, X_1x1], axis=1)

    return inception

def inception_block_1b(X):
    X_3x3 = Conv2D(96, (1, 1), data_format='channels_first',
name='inception_3b_3x3_conv1')(X)
    X_3x3 = BatchNormalization(axis=1, epsilon=0.00001,
name='inception_3b_3x3_bn1')(X_3x3)
    X_3x3 = Activation('relu')(X_3x3)
    X_3x3 = ZeroPadding2D(padding=(1, 1), data_format='channels_first')
(X_3x3)
    X_3x3 = Conv2D(128, (3, 3), data_format='channels_first',
name='inception_3b_3x3_conv2')(X_3x3)
    X_3x3 = BatchNormalization(axis=1, epsilon=0.00001,
name='inception_3b_3x3_bn2')(X_3x3)
    X_3x3 = Activation('relu')(X_3x3)

    X_5x5 = Conv2D(32, (1, 1), data_format='channels_first',
name='inception_3b_5x5_conv1')(X)
    X_5x5 = BatchNormalization(axis=1, epsilon=0.00001,
name='inception_3b_5x5_bn1')(X_5x5)
    X_5x5 = Activation('relu')(X_5x5)
    X_5x5 = ZeroPadding2D(padding=(2, 2), data_format='channels_first')
(X_5x5)
    X_5x5 = Conv2D(64, (5, 5), data_format='channels_first',
name='inception_3b_5x5_conv2')(X_5x5)
    X_5x5 = BatchNormalization(axis=1, epsilon=0.00001,
name='inception_3b_5x5_bn2')(X_5x5)
    X_5x5 = Activation('relu')(X_5x5)
```

```
    X_pool = AveragePooling2D(pool_size=(3, 3), strides=(3, 3),
data_format='channels_first')(X)
    X_pool = Conv2D(64, (1, 1), data_format='channels_first',
name='inception_3b_pool_conv')(X_pool)
    X_pool = BatchNormalization(axis=1, epsilon=0.00001,
name='inception_3b_pool_bn')(X_pool)
    X_pool = Activation('relu')(X_pool)
    X_pool = ZeroPadding2D(padding=(4, 4), data_format='channels_first')
(X_pool)

    X_1x1 = Conv2D(64, (1, 1), data_format='channels_first',
name='inception_3b_1x1_conv')(X)
    X_1x1 = BatchNormalization(axis=1, epsilon=0.00001,
name='inception_3b_1x1_bn')(X_1x1)
    X_1x1 = Activation('relu')(X_1x1)

    inception = concatenate([X_3x3, X_5x5, X_pool, X_1x1], axis=1)

    return inception

def inception_block_1c(X):
    X_3x3 = fr_utils.conv2d_bn(X,
                               layer='inception_3c_3x3',
                               cv1_out=128,
                               cv1_filter=(1, 1),
                               cv2_out=256,
                               cv2_filter=(3, 3),
                               cv2_strides=(2, 2),
                               padding=(1, 1))

    X_5x5 = fr_utils.conv2d_bn(X,
                               layer='inception_3c_5x5',
                               cv1_out=32,
                               cv1_filter=(1, 1),
                               cv2_out=64,
                               cv2_filter=(5, 5),
                               cv2_strides=(2, 2),
                               padding=(2, 2))

    X_pool = MaxPooling2D(pool_size=3, strides=2,
data_format='channels_first')(X)
    X_pool = ZeroPadding2D(padding=((0, 1), (0, 1)),
data_format='channels_first')(X_pool)

    inception = concatenate([X_3x3, X_5x5, X_pool], axis=1)

    return inception

def inception_block_2a(X):
    X_3x3 = fr_utils.conv2d_bn(X,
                               layer='inception_4a_3x3',
                               cv1_out=96,
                               cv1_filter=(1, 1),
                               cv2_out=192,
                               cv2_filter=(3, 3),
                               cv2_strides=(1, 1),
                               padding=(1, 1))
```

```
    X_5x5 = fr_utils.conv2d_bn(X,
                               layer='inception_4a_5x5',
                               cv1_out=32,
                               cv1_filter=(1, 1),
                               cv2_out=64,
                               cv2_filter=(5, 5),
                               cv2_strides=(1, 1),
                               padding=(2, 2))

    X_pool = AveragePooling2D(pool_size=(3, 3), strides=(3, 3),
data_format='channels_first')(X)
    X_pool = fr_utils.conv2d_bn(X_pool,
                               layer='inception_4a_pool',
                               cv1_out=128,
                               cv1_filter=(1, 1),
                               padding=(2, 2))
    X_1x1 = fr_utils.conv2d_bn(X,
                               layer='inception_4a_1x1',
                               cv1_out=256,
                               cv1_filter=(1, 1))
    inception = concatenate([X_3x3, X_5x5, X_pool, X_1x1], axis=1)

    return inception

def inception_block_2b(X):
    #inception4e
    X_3x3 = fr_utils.conv2d_bn(X,
                               layer='inception_4e_3x3',
                               cv1_out=160,
                               cv1_filter=(1, 1),
                               cv2_out=256,
                               cv2_filter=(3, 3),
                               cv2_strides=(2, 2),
                               padding=(1, 1))
    X_5x5 = fr_utils.conv2d_bn(X,
                               layer='inception_4e_5x5',
                               cv1_out=64,
                               cv1_filter=(1, 1),
                               cv2_out=128,
                               cv2_filter=(5, 5),
                               cv2_strides=(2, 2),
                               padding=(2, 2))

    X_pool = MaxPooling2D(pool_size=3, strides=2,
data_format='channels_first')(X)
    X_pool = ZeroPadding2D(padding=((0, 1), (0, 1)),
data_format='channels_first')(X_pool)

    inception = concatenate([X_3x3, X_5x5, X_pool], axis=1)

    return inception

def inception_block_3a(X):
    X_3x3 = fr_utils.conv2d_bn(X,
                               layer='inception_5a_3x3',
                               cv1_out=96,
                               cv1_filter=(1, 1),
                               cv2_out=384,
```

```
                            cv2_filter=(3, 3),
                            cv2_strides=(1, 1),
                            padding=(1, 1))
    X_pool = AveragePooling2D(pool_size=(3, 3), strides=(3, 3),
data_format='channels_first')(X)
    X_pool = fr_utils.conv2d_bn(X_pool,
                            layer='inception_5a_pool',
                            cv1_out=96,
                            cv1_filter=(1, 1),
                            padding=(1, 1))
    X_1x1 = fr_utils.conv2d_bn(X,
                            layer='inception_5a_1x1',
                            cv1_out=256,
                            cv1_filter=(1, 1))

    inception = concatenate([X_3x3, X_pool, X_1x1], axis=1)

    return inception

def inception_block_3b(X):
    X_3x3 = fr_utils.conv2d_bn(X,
                            layer='inception_5b_3x3',
                            cv1_out=96,
                            cv1_filter=(1, 1),
                            cv2_out=384,
                            cv2_filter=(3, 3),
                            cv2_strides=(1, 1),
                            padding=(1, 1))
    X_pool = MaxPooling2D(pool_size=3, strides=2,
data_format='channels_first')(X)
    X_pool = fr_utils.conv2d_bn(X_pool,
                            layer='inception_5b_pool',
                            cv1_out=96,
                            cv1_filter=(1, 1))
    X_pool = ZeroPadding2D(padding=(1, 1), data_format='channels_first')
(X_pool)

    X_1x1 = fr_utils.conv2d_bn(X,
                            layer='inception_5b_1x1',
                            cv1_out=256,
                            cv1_filter=(1, 1))
    inception = concatenate([X_3x3, X_pool, X_1x1], axis=1)

    return inception

def faceRecoModel(input_shape):
    """
    Implementation of the Inception model used for FaceNet

    Arguments:
    input_shape -- shape of the images of the dataset

    Returns:
    model -- a Model() instance in Keras
    """

    # Define the input as a tensor with shape input_shape
    X_input = Input(input_shape)
```

```
    # Zero-Padding
    X = ZeroPadding2D((3, 3))(X_input)

    # First Block
    X = Conv2D(64, (7, 7), strides = (2, 2), name = 'conv1')(X)
    X = BatchNormalization(axis = 1, name = 'bn1')(X)
    X = Activation('relu')(X)

    # Zero-Padding + MAXPOOL
    X = ZeroPadding2D((1, 1))(X)
    X = MaxPooling2D((3, 3), strides = 2)(X)

    # Second Block
    X = Conv2D(64, (1, 1), strides = (1, 1), name = 'conv2')(X)
    X = BatchNormalization(axis = 1, epsilon=0.00001, name = 'bn2')(X)
    X = Activation('relu')(X)

    # Zero-Padding + MAXPOOL
    X = ZeroPadding2D((1, 1))(X)
```

```
    # Second Block
    X = Conv2D(192, (3, 3), strides = (1, 1), name = 'conv3')(X)
    X = BatchNormalization(axis = 1, epsilon=0.00001, name = 'bn3')(X)
    X = Activation('relu')(X)

    # Zero-Padding + MAXPOOL
    X = ZeroPadding2D((1, 1))(X)
    X = MaxPooling2D(pool_size = 3, strides = 2)(X)

    # Inception 1: a/b/c
    X = inception_block_1a(X)
    X = inception_block_1b(X)
    X = inception_block_1c(X)

    # Inception 2: a/b
    X = inception_block_2a(X)
    X = inception_block_2b(X)

    # Inception 3: a/b
    X = inception_block_3a(X)
    X = inception_block_3b(X)

    # Top layer
    X = AveragePooling2D(pool_size=(3, 3), strides=(1, 1),
data_format='channels_first')(X)
    X = Flatten()(X)
    X = Dense(128, name='dense_layer')(X)

    # L2 normalization
    X = Lambda(lambda  x: tf.nn.l2_normalize(x,dim=1))(X)
```

```
    # Create model instance
    model = Model(inputs = X_input, outputs = X, name='FaceRecoModel')

    return model
```