AI & Python

The AI part consists of two machine learning models for recognizing facial expressions and activities in order to understand the human behaviour, expressions, movements, monitor patients and their environment.

Model 1: Facial expression recognition

Using convolutional neural networks build from scratch, this model classify faces in a given sequence of video frames. For the moment, we have focused only on 7 basic emotions (anger, disgust, fear, happiness, sadness, surprise, neutral) but they can be extended.

For this approach, it was prepared two scripts:

- *flaskServer/ai_module/facial_expression/facial_expression_model_train.ipynb*: used to prepare and to train the model
- *flaskServer/ai_module/facial_expression/facial_expression_model_inference.py*: used to test the model on new data

I. Script 1: *facial_expression_model_train.ipynb*

1. Prerequisites

pip install numpy
pip install sklearn
pip install pandas
pip install matplotlib
pip install tensorflow
pip install keras
pip install opencv-python

Also, for Google Colaboratory are required:
from google.colab import drive
from google.colab.patches import cv2_imshow

2. Download, explore and prepare the data

After different testing approaches, we combined 2 publicly available facial expression datasets: Fer2013 (see link: https://www.kaggle.com/deadskull7/fer2013) and FERG-DB (http://grail.cs.washington.edu/projects/deepexpr/ferg-db.html).

- Fer2013 consists of 48x48 pixel grayscale images of faces and contains a number of 35.887 images. It presents different variabilities in illumination, age, pose, expression

intensity and occlusions that occur under realistic conditions. However, the database is not balanced enough and contains faces that are not properly labeled or do not contain the entire face.

- FERG-DB contains 54.977 annotated face images of six stylized characters modeled using the MAYA software and rendered out in 2D to create the images.

Both databases contain seven emotion categories (anger, disgust, fear, happiness, sadness, surprise, neutral). The final database contains 90.864 images being balanced enough for what we need. After that, it was loaded, explored and prepared for training.

3. Perform cross-validation

After preparing the numpy arrays, it was split into training (80%) and testing sets (20%).

4. Build the model architecture

The layers in the Convolution Neural Network used in implementing this classifier can be summarized as described below:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 46, 46, 64)        640
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 46, 46, 64)        36928
_________________________________________________________________
batch_normalization_1 (Batch (None, 46, 46, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 23, 23, 128)       73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 23, 23, 128)       512
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 23, 23, 128)       147584
_________________________________________________________________
batch_normalization_3 (Batch (None, 23, 23, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 11, 11, 128)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 11, 11, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 11, 11, 256)       295168
_________________________________________________________________
batch_normalization_4 (Batch (None, 11, 11, 256)       1024
_________________________________________________________________
```

```
conv2d_6 (Conv2D)            (None, 11, 11, 256)       590080
_________________________________________________________________
batch_normalization_5 (Batch (None, 11, 11, 256)       1024
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 256)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 5, 5, 256)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 5, 5, 512)         1180160
_________________________________________________________________
batch_normalization_6 (Batch (None, 5, 5, 512)         2048
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 5, 5, 512)         2359808
_________________________________________________________________
batch_normalization_7 (Batch (None, 5, 5, 512)         2048
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 2, 512)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 2, 2, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1049088
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_6 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 7)                 903
=================================================================
Total params: 5,905,863
Trainable params: 5,902,151
Non-trainable params: 3,712
```

5. Compile the model

After several testing, the model was compiled using Adam optimizer and categorical cross-entropy loss.

6. Define callbacks

To get a view on different internal states and statistics of the model during training we defined callbacks.

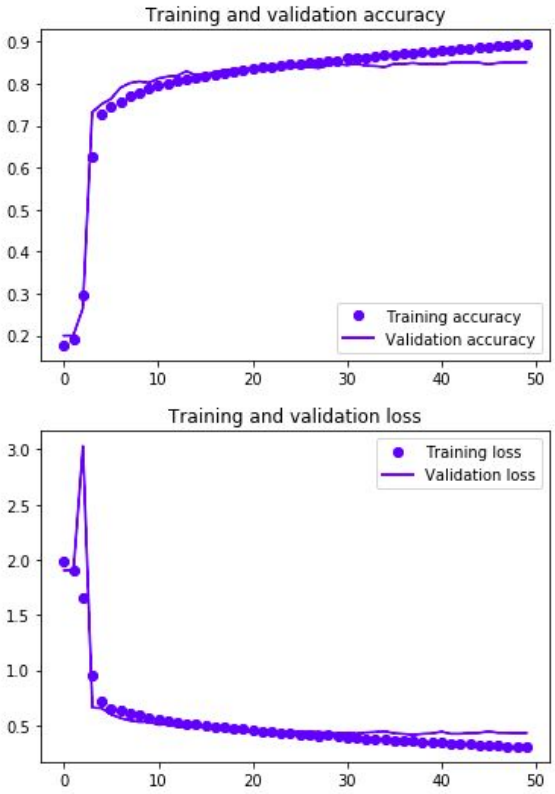
7. Train the model

After adding the second dataset, the accuracy increases from 70% to 84%.

8. Save the model

The model saved for the inference parts: *facial_expression_model_train.h5* .

9. Plot the training and validation loss and accuracy



II. Script 2: *facial_expression_model_inference.py*

1. Prerequisites

pip install numpy
pip install keras
pip install face_recognition
pip install opencv-python

Also, for Google Colaboratory are required:
from google.colab.patches import cv2_imshow

2. Generate predictions

The network managed to learn patterns from the training samples and to generalize very well when giving new samples never seen before.

Model 2: Human activity recognition

Using 3D convolutional neural networks build from scratch, this approach aims to detect and monitor people's activities in different environments. Here we are particularly interested in those actions which imply negative consequences for their health: e.g. falls.

For this approach, it was prepared four scripts:

- ai_module/activity_recognition/3dcnn.py
- ai_module/activity_recognition/utils.py
- ai_module/activity_recognition/videobuffering.py
- ai_module/activity_recognition/activity_inferance.py

Prerequisites

pip install numpy
pip install sklearn
pip install pandas
pip install matplotlib
pip install tensorflow
pip install keras
pip install opencv-python

Download, explore and prepare the data

We combined 2 publicly available fall detection datasets: UR Fall Dataset - URFD (see link: http://fenix.univ.rzeszow.pl/~mkepski/ds/uf.html) and Fall Detection dataset - FDD (http://le2i.cnrs.fr/Fall-detection-Dataset?lang=fr). The final dataset contains 859 fall cases and 918 not-fall cases.

Train/test split

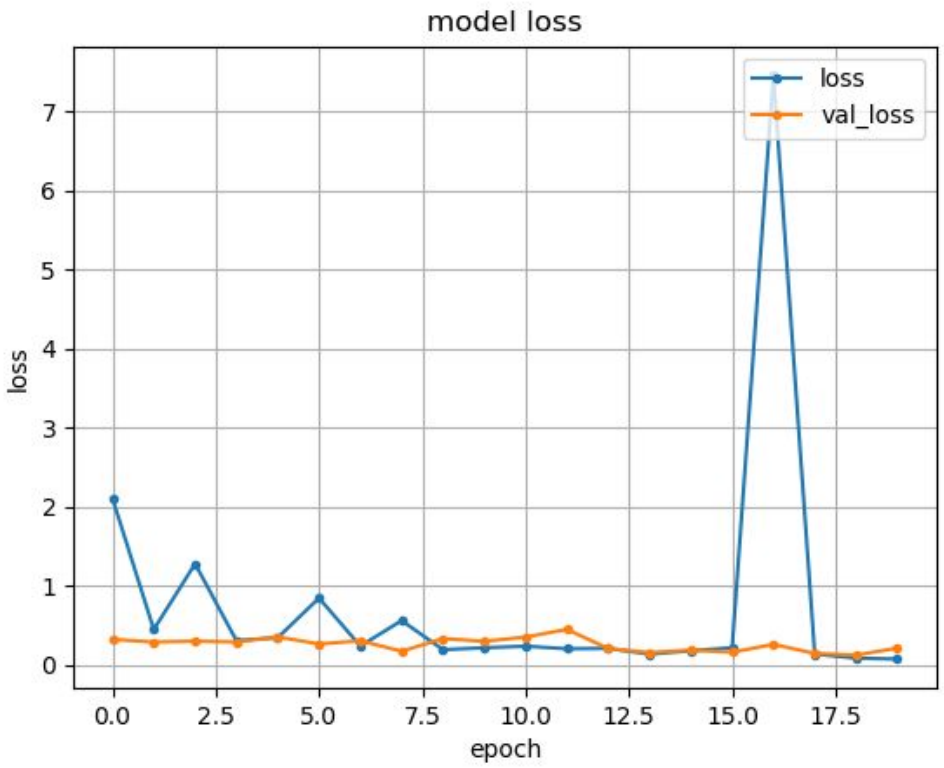
The dataset was split into training (80%) and testing sets (20%).

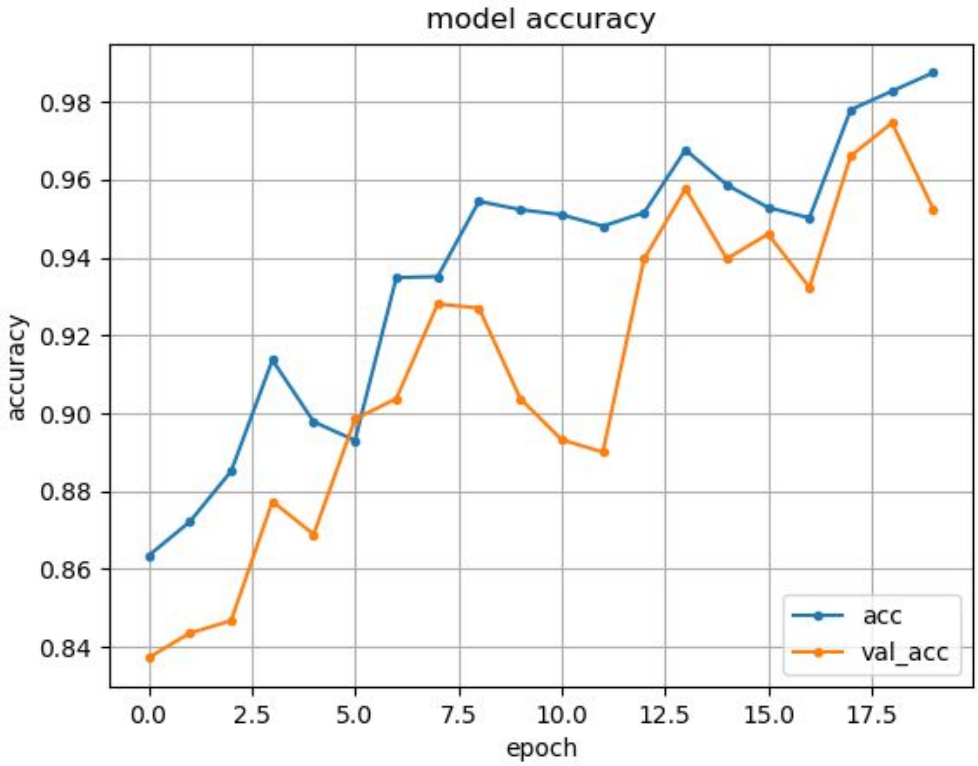
Build the model architecture

input_1 (InputLayer)	(None, 224, 224, 5,	0	
conv3d_1 (Conv3D)	(None, 224, 224, 5,	1312	input_1[0][0]
leaky_re_lu_1 (LeakyReLU)	(None, 224, 224, 5,	0	conv3d_1[0][0]
conv3d_2 (Conv3D)	(None, 224, 224, 5,	13856	leaky_re_lu_1[0][0]
leaky_re_lu_2 (LeakyReLU)	(None, 224, 224, 5,	0	conv3d_2[0][0]
conv3d_3 (Conv3D)	(None, 224, 224, 5,	528	leaky_re_lu_2[0][0]
conv3d_6 (Conv3D)	(None, 224, 224, 5,	264	leaky_re_lu_2[0][0]
conv3d_9 (Conv3D)	(None, 224, 224, 5,	132	leaky_re_lu_2[0][0]
leaky_re_lu_3 (LeakyReLU)	(None, 224, 224, 5,	0	conv3d_3[0][0]
leaky_re_lu_6 (LeakyReLU)	(None, 224, 224, 5,	0	conv3d_6[0][0]
leaky_re_lu_9 (LeakyReLU)	(None, 224, 224, 5,	0	conv3d_9[0][0]
max_pooling3d_1 (MaxPooling3D)	(None, 112, 112, 3,	0	leaky_re_lu_3[0][0]
max_pooling3d_3 (MaxPooling3D)	(None, 112, 112, 3,	0	leaky_re_lu_6[0][0]
max_pooling3d_5 (MaxPooling3D)	(None, 112, 112, 3,	0	leaky_re_lu_9[0][0]
conv3d_4 (Conv3D)	(None, 112, 112, 3,	6928	max_pooling3d_1[0][0]
conv3d_7 (Conv3D)	(None, 112, 112, 3,	1736	max_pooling3d_3[0][0]
conv3d_10 (Conv3D)	(None, 112, 112, 3,	436	max_pooling3d_5[0][0]
leaky_re_lu_4 (LeakyReLU)	(None, 112, 112, 3,	0	conv3d_4[0][0]
leaky_re_lu_7 (LeakyReLU)	(None, 112, 112, 3,	0	conv3d_7[0][0]
leaky_re_lu_10 (LeakyReLU)	(None, 112, 112, 3,	0	conv3d_10[0][0]
max_pooling3d_2 (MaxPooling3D)	(None, 56, 56, 2, 16	0	leaky_re_lu_4[0][0]
max_pooling3d_4 (MaxPooling3D)	(None, 56, 56, 2, 8)	0	leaky_re_lu_7[0][0]
max_pooling3d_6 (MaxPooling3D)	(None, 56, 56, 2, 4)	0	leaky_re_lu_10[0][0]
conv3d_5 (Conv3D)	(None, 56, 56, 2, 1)	17	max_pooling3d_2[0][0]
conv3d_8 (Conv3D)	(None, 56, 56, 2, 1)	9	max_pooling3d_4[0][0]
conv3d_11 (Conv3D)	(None, 56, 56, 2, 1)	5	max_pooling3d_6[0][0]
leaky_re_lu_5 (LeakyReLU)	(None, 56, 56, 2, 1)	0	conv3d_5[0][0]

```
leaky_re_lu_8 (LeakyReLU)       (None, 56, 56, 2, 1) 0           conv3d_8[0][0]
leaky_re_lu_11 (LeakyReLU)      (None, 56, 56, 2, 1) 0           conv3d_11[0][0]
add_1 (Add)                     (None, 56, 56, 2, 1) 0           leaky_re_lu_5[0][0]
                                                                 leaky_re_lu_8[0][0]
                                                                 leaky_re_lu_11[0][0]
max_pooling3d_7 (MaxPooling3D)  (None, 28, 28, 1, 1) 0           add_1[0][0]
flatten_1 (Flatten)             (None, 784)          0           max_pooling3d_7[0][0]
dense_1 (Dense)                 (None, 784)          615440      flatten_1[0][0]
dense_2 (Dense)                 (None, 2)            1570        dense_1[0][0]
==================================================================================================
Total params: 642,233
Trainable params: 642,233
Non-trainable params: 0
__________________________________________________________________________________________________
Train data:  (280, 224, 224, 5, 3)
Test data:   (70, 224, 224, 5, 3)
```

Train the model





Save the model

- ai_module/activity_recognition/fall_3dcnnmodel-gpu.hd5

React-Native & Python

Prerequisites

Python 3.7.+
Node.js
Android Studio and Android SDK
Visual Studio Code
JDK > 8

Installing

- Run the python server
- Run npm install
- Run react-native run-android

Built With

- React-native - The library used
- Android Studio - for dependencies

Folder Structure

1. ***React Native***

- assets: contains folders for resources (e.g. fonts, images etc.)
- src - folder that is the main container of the files in the application
    - components - folder that contains the project's components
    - constants - folder that contains the constants of the project
    - data - folder that contains some mock data for populating the screens
    - navigators - folder in which the logic for navigation can be found
    - screens - folder that contains the main screens of the application

2. ***Python***

- flask_server - folder that contains the whole structure and files
    - ai_module - folder that contains the algorithms for facial and activity recognition
    - resources - folder that contains the resources for this application
    - flask server logic - represent all the scripts the build the server's logic for linking the client app to the AI side

Acknowledgments

Modern Nursing Home is a safe and secure Nursing Home system that is constantly monitoring patients and their environment, day and night, through video cameras, sensors, smartphones, wristbands and beacons, and provides feedback to nurses and personnel via wristbands and companion phone and desktop app.

The scope is to offer immediate help to elder people, prevent injuries, monitor health state and provide accurate treatment, offer personalised entertainment and services so that life feels good and safe even when old and amongst strangers. Provide nurses, medical staff and personnel accurate and real-time information about patients, their wellbeing and state, so that they can offer help exactly when it’s needed, save lives, increase quality of life, provide more accurate treatments and better understand patients.