EmAUtion Analysis Wiki

EmAUtion Analysis (emaution-analysis) is the new PyPI package library developed from Emotiva s.r.l. to perform emotion and action units (AUs) recognition from images. This documentation will provide you with instructions on how to install and use the package and information about known issues.

current version: v1.1.2 - 3/12/2021

🔥 Quickstart

To start using this package, just instantiate the class StaticModel of the package and pass an image to the functions predict_emotions() or predict_aus() to get the emotions or action units predicted in the image.

```
import cv2
from emautionanalysis import StaticModel

img_path = "happy_sample.jpg"
img = cv2.imread(img_path)

model = StaticModel(model_name="cnn_model")
print(model.predict_emotions(img=img))
```

```
Face 0:
{
'anger': 1.9467853462629137e-07,
'disgust': 9.477326966589317e-05,
'fear': 8.966335371951573e-07,
'happiness': 0.9734345078468323,
'sadness': 6.467414398988088e-05,
'surprise': 1.3646038799682476e-06
}
```

Installation and Setup

Requirements

- Python >= 3.8

Install from AWS

Currently package `emaution-analysis` is not publicly available. The only way to access this package is through private versions hosted in AWS. To do this, you will need valid credentials: if needed, contact info@emotiva.it to request access keys.

To install the package via pip, please follow these instructions:

1. Install AWS CLI
2. Configure AWS profile with the credentials you have been provided.
3. From shell, run these commands:

```
CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
domain emotiva --domain-owner 738822722489 --query authorizationToken
--output text`
pip config set global.extra-index-url
https://aws:$CODEARTIFACT_AUTH_TOKEN@emotiva-
738822722489.d.codeartifact.eu-central-
1.amazonaws.com/pypi/emautionanalysis/simple/
```

4. (*Optional*) Activate a virtual environment:

```
python -m venv venv
source venv/bin/activate
```

5. Install package with `pip`:

```
(venv) pip install emaution-analysis
```

OpenCV fix (best practice)

It has been reported that two packages in requirements (`mediapipe` and `face-align`) install different versions of `opencv`: `opencv-python` and `opencv-contrib-python`. Despite the code should work with both, it is a best practice to use only one of them. Please uninstall both and then install `opencv-contrib-python==4.5.3.56`:

```
pip uninstall opencv-python opencv-contrib-python -y
pip install opencv-contrib-python==4.5.3.56
```

```

```

Pytorch GPU support

This package contains in requirements `torch==1.9.0` and `torchvision==0.10.0`, but you could get different issue using GPU depending on your graphic card and CUDA version. Please check your Pytorch compatibility and installed required version to run code in gpu mode (link).

Environment Setup

To download weights and models on first usage (and to check for weights updates), credentials are required. Before to use the package please ensure to include the provided credentials into environment variables `EMOTIVA_ACCESS_KEY` and `EMOTIVA_SECRET_KEY`.

Documentation

StaticModel

The main module of the `emaution-analysis` package, to perform emotions and action units recognition.

StaticModel Object

```
class StaticModel:
    def __init__(self, model_name='cnn_model', extra_cfg: dict = None):
```

Parameters:

- `model_name`: string (*Optional*). Model to use for emotion/aus recognition; it can be one of "old_model", "single_aus_collection" or "cnn_model" (default). "cnn_model" is the newest model; "old_model" will be deprecated in future.
- `extra_cfg`: dictionary (*Optional*). Extra parameters to be passed, to configure the model. Currently, the supported parameters are:

```
##################################################
##  PARAMETERS FOR MODEL single_aus_collection  ##
##################################################

# model directory, containing files 'single_aus_collection.pkl' with
trained model and 'cfg.yml' with other parameters
# used in train
model_dir: null

######################################
##  PARAMETERS FOR MODEL cnn_model  ##
```

```
#######################################

# path to the weights to be loaded in the model
weight_path: null
# face detector type: ['dlib', 'mp', 'facealign']
face_detector: null
# device [cpu, gpu]
device: null
# model selection for mediapipe face detector: [0, 1]: 0 optimized within 2
mt from camera;
#                                                     1 optimized within 5
mt from camera.
mp_model_selection: null
# minimum confidence threshold for face detection with mediapipe face
detector
mp_min_detection_confidence: null
```

The parameter `face_detector` permits to specify which library to use to perform face detection between three options:

- `dlib`: Dlib frontal face detector; standard library.
- `mp` or `mediapipe`: Google Mediapipe face detector. Fast face detection method based on BlazeFace, optimized for frontal subjects.
- `facealign` or `face_align`: Face Align based on S3FD. Good face detection success also with lateral faces. Slow on CPU, but can run in real-time on GPU.

predict_aus()

```
def predict_aus(self, img, return_landmarks=False, detected_faces:
List[Tuple] = None):
```

Parameters

- `img`: **Required**. It can be both ndarray image data (i.e. loaded with `opencv`) or a valid image path; in case of image data, it must have shape `(height, width, channels)` (channel last) in BGR; in case of image data, it will be loaded using `opencv`.
- `return_landmark`: boolean (*Optional*, default False). If true, also face bounding boxes and facial landmarks will be returned.
- `detected_faces`: *Optional* list of bounding boxes indicating the position of faces in the image provided. The format of each bounding box must match the format returned by this function when `return_landmark = True`.

Returns

If `return_landmarks=False` only a list is returned, containing a dictionary for each face detected in the images with the predicted AUs. The AUs predicted depend on the model used:

- `old_model` (*deprecated*) and `single_aus_collection` return 15 AUs: 'AU1', 'AU2', 'AU4', 'AU5', 'AU6', 'AU7', 'AU9', 'AU12', 'AU15', 'AU17', 'AU20', 'AU23', 'AU24', 'AU25', 'AU27'.
- `cnn_model` (default) return 17 AUs: 'AU1', 'AU2', 'AU4', 'AU5', 'AU6', 'AU9', 'AU10', 'AU12', 'AU15', 'AU17', 'AU18', 'AU20', 'AU24', 'AU25', 'AU26', 'AU28', 'AU43'.

If `return_landmarks=True` also a list with bounding boxes and facial landmarks is returned, for each face detected in the image:

- Bounding boxes are represented as a tuple `((min_x, min_y), (max_x, max_y))`;
- Facial landmarks are represented as (68, 2) dimensional numpy array: 68 points with x any coordinates. All the coordinates (of both bounding boxes and facial landmarks) are normalized in the range [0,1] with respect to the image width and height.

In case of any error (i.e. no face is found) `None` is returned.

predict_emotions()

```
def predict_emotions(self, img, return_aus=False, return_landmarks=False,
detected_faces: List[Tuple] = None):
```

This function wrap `predict_aus()` and aggregates its result into 6 basic emotions: anger, disgust, fear, happiness, sadness and surprise. Many parameters are shared between these two functions.

Parameters

- `img`: same as in `predict_aus()`.
- `return_aus`: boolean (*Optional*, default False); if True, also the activation of the single AUs are returned.
- `return_landmarks`: same as in `predict_aus()`.
- `detected_faces`: same as in `predict_aus()`.

Returns

If `return_aus=False` and `return_landmarks=False` only a list is returned, with a dictionary per face detected including predictions for each of the six basic emotions.

If `return_aus=True` and `return_landmarks=False` emotions and single aus results are returned (same format as in `predict_aus()`).

If `return_aus=False` and `return_landmarks=True` emotions and bounding boxes with facial landmarks are returned (same format as in `predict_aus()`).

If `return_aus=True` and `return_landmarks=True` three lists are returned: one for emotions, one for AUs and one for bounding boxes and landmarks.

In case of any error (i.e. no face is found) `None` is returned.

Usage example with visualization

In the following example, an image is analyzed and results are visualized, plotting face landmarks and face bounding box.

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.gridspec as gridspec
from emautionanalysis import StaticModel

model_name = 'cnn_model'
extra_params = {
        'face_detector':'mp',
    }
model = StaticModel(model_name=model_name, extra_cfg=extra_params)

res = model.predict_emotions(img=test_image, return_aus=True,
return_landmarks=True)

if res is None:
    print('No face detected')

emotions_list, aus_list, global_landmarks_list = res

for i, _ in enumerate(aus_list):

    gs = gridspec.GridSpec(2, 2)
    plt.figure(figsize=(10, 5))
    ax = plt.subplot(gs[0, 0])

    plt.title(f'{model_name}, subj {i}')
    ax.set_xticks([])
    ax.set_yticks([])
    plt.imshow(test_image[:, :, ::-1])

    face_rect, gl = global_landmarks_list[i]
    h, w, c = test_image.shape
    gl = np.array(gl)
    tl, br = face_rect
    tl = (tl[0] * w, tl[1] * h)
    br = (br[0] * w, br[1] * h)
    rect = patches.Rectangle(tl, br[0] - tl[0], br[1] - tl[1], linewidth=1,
edgecolor='r', facecolor='none')
    ax.add_patch(rect)
    plt.scatter(gl[:, 0] * w, gl[:, 1] * h, c='y', s=3.2)

    ax = plt.subplot(gs[0, 1])  # row 0, col 1
    plt.bar(range(len(emotions_list[i])), list(emotions_list[i].values()),
align='center',
                    tick_label=list(emotions_list[i].keys()))
    plt.ylim([0, 1])

    ax = plt.subplot(gs[1, :])  # row 1, span all columns
    plt.bar(range(len(aus_list[i])), list(aus_list[i].values()),
```

```
align='center',
                            tick_label=list(aus_list[i].keys()))
      plt.ylim([0, 1])

      plt.show()
```

Output: