Philipp Hofer
Institute of
Networks and Security

@ philipp.hofer@ins.jku.at
https://www.digidow.eu/

August 2021

Face recognition: Combining embeddings

Technical Report

Christian Doppler Laboratory for
Private Digital Authentication in the Physical World

This work has been carried out within the scope of Digidow, the Christian Doppler Laboratory for Private Digital Authentication in the Physical World, funded by the Christian Doppler Forschungsgesellschaft, 3 Banken IT GmbH, Kepler Universitätsklinikum GmbH, NXP Semiconductors Austria GmbH, and Österreichische Staatsdruckerei GmbH.

INSTITUTE
OF NETWORKS
AND SECURITY

**JOHANNES KEPLER
UNIVERSITY LINZ**
Altenberger Straße 69
4040 Linz, Austria
jku.at

Contents

1. Motivation

In order to increase the accuracy of SOTA face recognition pipelines, intuitively it would make sense to not only use a single image as reference embedding (*template*), but combine multiple embeddings from different images (different pose, angle, setting) to create a more accurate and robust template. In order to objectively evaluate our different proposed combinations of embeddings, we would benefit from having a single metric to tell how well the template is performing on our dataset. For certain applications (e.g. opening doors) a low false-positive rate is required, while in other situations (e.g. sensor contacting PIA's) a low false-negative rate is required. Therefore, in this document we try to balance these different approaches by using the harmonic mean of recall and precision. There are multiple ways of combining different embeddings:

1. Average (as proposed by e.g. ArcFace)
2. Voting (perform comparison on n different images, everyone votes whether there is a match; if $m <= n$ votes say *match*, it is actually counted as match)

2. Dataset

For evaluation, we created a dataset by mounting a camera in our printer room and collecting images whenever a person is detected in front of the camera.

- There are 13 different people in the dataset.
- There are between 5 and 210 images of every person, with an average of 79.3 images/person.

From this dataset, we pick a person and copy the images to a new location:

```
person = "hofer"
person_path = os.path.join(base_path, person)
new_location = os.path.expanduser("~/datasets")
new_location = os.path.join(new_location, person)
!rm -rf $new_location
!cp -r $person_path $new_location
```

As discussed in *Simple heuristics to reduce false-positives in face recognition*, we remove all images where the head rotation is bad:

```
for file in os.listdir(new_location):
    img_path = os.path.join(new_location, file)
    if os.path.isfile(img_path):
        if rec.get_eye_distance(img_path) < eye_distance:
            ! rm $img_path
        elif rec.get_eyes_mouth_distance(img_path) < eye_mouth_distance:
            ! rm $img_path
```

We have 166 images of this person.

3. Average

In this setting, we simply calculate the embedding of multiple images, take their average and use this average as template.

3.1 Baseline - Face image

In order to be able to compare the performance of different ways of combining embeddings, we first calculate the baseline where we use a single, full-frontal image as template.

```python
template_single_path = os.path.join(base_path, "atemplates_variations/hofer/2_face
    -5/2021-05-25-144227_30.jpg")
template_single = rec.get_emb(template_single_path)
```

After processing the image, the face looks like this:

```python
from PIL import Image
# Image.fromarray(rec.vis._get_img(template_single_path, 1, "", True), 'RGB')
```

Next, we need to calculate the harmonic mean of a given template with respect to the whole dataset.

```python
from prettytable import PrettyTable
def get_confusion(threshold, same_person, different_person):
    tp, tn, fp, fn = 0, 0, 0, 0

    for p in same_person:
        if p > threshold:
            fn += 1
        else:
            tp += 1
    for p in different_person:
        if p <= threshold:
            fp += 1
        else:
            tn += 1
    return fp/len(different_person), fn/len(same_person), tp/len(same_person), tn/
    len(different_person)
def print_confusion(fp, fn, tp, tn):
    x = PrettyTable()
    x.title = 'Predicted␣condition'
    x.field_names = ["␣", "", "Positive", "Negative"]
    x.add_row(["Actual","Positive", "{:.2f}".format(tp), "{:.2f}".format(fn)])
    x.add_row(["Condition", "Negative", "{:.2f}".format(fp), "{:.2f}".format(tn)])
    print(x)

def get_harmonic_mean(threshold, same, different):
    fp, fn, tp, tn = get_confusion(threshold, same, different)
    if tp+fp == 0:
        return 0
    if tp + fn == 0:
        return 0
    prec = tp / (tp + fp)
    recall = tp / (tp + fn)
    harmonic_mean = 2*(prec*recall/(prec+recall))
    return harmonic_mean
```

```python
def optimize_harmonic_mean(same, different):
    highest_h_mean = 0
    threshold_highest_h_mean = -1
    upper_limit = int(max(max(same), max(different))*1000
    for i in range(0,upper_limit):
        i = i/1000
        harmonic_mean = get_harmonic_mean(i, same, different)
        if harmonic_mean > highest_h_mean:
            highest_h_mean = harmonic_mean
            threshold_highest_h_mean = i
    return threshold_highest_h_mean, highest_h_mean
def harmonic_mean(template):
    same = []
    for file in os.listdir(new_location):
        img_path = os.path.join(new_location, file)
        if os.path.isfile(img_path):
            emb = rec.get_emb(img_path)[0]
            same.append(rec.get_score(template, emb))
    different = []
    for file in os.listdir(base_path):
        person_path = os.path.join(base_path, file)
        if os.path.isdir(person_path) and file != person and not file in
 exclude_dirs:
            for img in os.listdir(person_path):
                img_path = os.path.join(person_path, img)
                if rec.get_eye_distance(img_path) < eye_distance:
                    continue
                if rec.get_eyes_mouth_distance(img_path) < eye_mouth_distance:
                    continue
                emb = rec.get_emb(img_path)[0]
                different.append(rec.get_score(template, emb))
    threshold_highest_h_mean, highest_h_mean = optimize_harmonic_mean(same,
 different)
    return highest_h_mean, same, different, threshold_highest_h_mean
def print_harmonic_mean(h_mean, threshold):
    print("Harmonic mean: {:.3f} (with a threshold of {})".format(h_mean,
 threshold))
```

```python
h_mean, same, different, threshold = harmonic_mean(template_single)
print_harmonic_mean(h_mean, threshold)
```

```
    Harmonic mean: 0.852 (with a threshold of 1.54)
```

We can also plot our confusion matrix:

```
    +---------------------------------------------+
    |             Predicted condition             |
    +-----------+----------+----------+----------+
    |           |          | Positive | Negative |
    +-----------+----------+----------+----------+
    |   Actual  | Positive |   0.89   |   0.11   |
    | Condition | Negative |   0.20   |   0.80   |
    +-----------+----------+----------+----------+
```

In our current setup, we want to minimize FP, thus we lowered the threshold to 1.2. This would result in the following confusion matrix:

```
    +---------------------------------------------+
    |              Predicted condition              |
    +-----------+----------+----------+----------+
    |           |          | Positive | Negative |
    +-----------+----------+----------+----------+
    |   Actual  | Positive |   0.30   |   0.70   |
    | Condition | Negative |   0.00   |   1.00   |
    +-----------+----------+----------+----------+
```

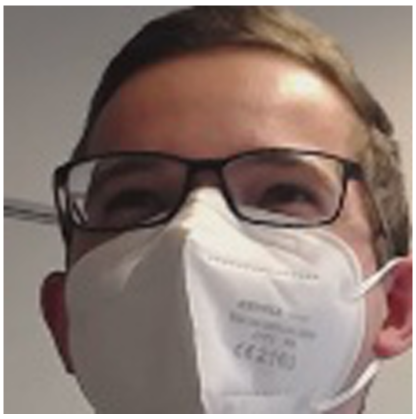
3.2 Baseline - mask image

Due to the current pandemic, most of the images in the dataset depicts a person wearing a face mask. Thus, it might be interesting, if using an image where the person wears a face mask increases our performance. Even though the template image more closely resembles the test images, we expect the performance to decrease, as the mask hides more than half of the face, and thus information is lost.

```
template_singlemask_path = os.path.join(base_path, "atemplates_variations/hofer/1
    _different-setting-50/2021-05-25-145821_26.jpg")
template_singlemask = rec.get_emb(template_singlemask_path)
```

After processing the image, the face looks like this:

```
from PIL import Image
Image.fromarray(rec.vis._get_img(template_singlemask_path, 1, "", True), 'RGB')
```



```
h_mean_mask, same_mask, different_mask, threshold_mask = harmonic_mean(
    template_singlemask)
print_harmonic_mean(h_mean_mask, threshold_mask)
```

```
    Harmonic mean: 0.850 (with a threshold of 1.637)
```

```
    +--------------------------------------------+
    |              Predicted condition              |
    +-----------+----------+----------+----------+
    |           |          | Positive | Negative |
    +-----------+----------+----------+----------+
    |   Actual  | Positive |   0.88   |   0.12   |
    | Condition | Negative |   0.19   |   0.81   |
    +-----------+----------+----------+----------+
```

The harmonic mean decreased by 0.2 percent points, which supports our expectation of reduced accuracy due to lower amount of information.

3.3 Combination of mask + face image

```
emb_combined = np.mean( np.array([ template_single, template_singlemask ]), axis=0
    )
```

```
h_mean_comb, same_comb, different_comb, threshold_comb = harmonic_mean(
    emb_combined)
print_harmonic_mean(h_mean_comb, threshold_comb)
```

```
    Harmonic mean: 0.870 (with a threshold of 1.267)
```

Face: 0.85 Mask: 0.85 Face+Mask: 0.87

3.4 Different arithethic methods of combining embeddings

RoyChowdhury et al. proposed in *One-to-many face recognition with bilinear CNNs* instead of the classical usage of the average for each dimension of an embedding the use of the maximum. They argue, that this will reduce the overfit on the dominant angle and thus create a more robust embedding. Based on this, we evaluated four different arithmetic methods of combining embeddings.

```
embs = []
for file in os.listdir("/home/philipp/Nextcloud/printerroom-dataset/
    atemplates_variations/hofer/1_different-setting-50"):
    abs_path = os.path.join("/home/philipp/Nextcloud/printerroom-dataset/
    atemplates_variations/hofer/1_different-setting-50", file)
    if len(rec.get_emb(abs_path)) == 1:
        embs.append(rec.get_emb(abs_path)[0])
```

```
print("Mean")
emb_combined = np.mean( np.array(embs), axis=0 )
h_mean_comb, same_comb, different_comb, threshold_comb = harmonic_mean(
    emb_combined)
print_harmonic_mean(h_mean_comb, threshold_comb)
```

```
    Mean
    Harmonic mean: 0.914 (with a threshold of 0.996)
```

```
print("Max")
emb_combined = np.max( np.array(embs), axis=0 )
h_mean_comb, same_comb, different_comb, threshold_comb = harmonic_mean(
    emb_combined)
print_harmonic_mean(h_mean_comb, threshold_comb)
```

```
    Max
    Harmonic mean: 0.868 (with a threshold of 4.178)
```

```
print("Median")
emb_combined = np.median( np.array(embs), axis=0 )
h_mean_comb, same_comb, different_comb, threshold_comb = harmonic_mean(
    emb_combined)
print_harmonic_mean(h_mean_comb, threshold_comb)
```

```
    Median
    Harmonic mean: 0.899 (with a threshold of 0.955)
```

```
print("Min")
emb_combined = np.min( np.array(embs), axis=0 )
h_mean_comb, same_comb, different_comb, threshold_comb = harmonic_mean(
    emb_combined)
print_harmonic_mean(h_mean_comb, threshold_comb)
```

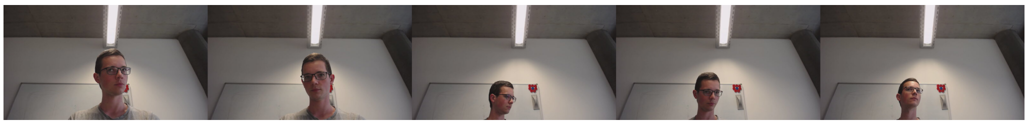
```
    Min
    Harmonic mean: 0.764 (with a threshold of 3.999)
```

3.5 Multiple angles

3.5.1 5 images

Let's see if the accuracy improves if we use 5 images with various angles.

```
comb_images(os.path.join(base_path, "atemplates_variations/hofer/2_face-5"))
```



```
folder_2 = os.path.join(base_path, "atemplates_variations/hofer/2_face-5")
emb_2 = rec.get_combined_embedding(folder_2)
h_mean_2, same_2, different_2, threshold_2 = harmonic_mean(emb_2)
print_harmonic_mean(h_mean_2, threshold_2)
```

```
    Harmonic mean: 0.892 (with a threshold of 1.166)
```

Getting a higher accuracy with multiple pictures seems promising. What happens if we increase the amount of images?

3.5.2 25 images

```
folder_3 = os.path.join(base_path, "atemplates_variations/hofer/3_face-25")
emb_3 = rec.get_combined_embedding(folder_3)
h_mean_3, same_3, different_3, threshold_3 = harmonic_mean(emb_3)
print_harmonic_mean(h_mean_3, threshold_3)
```

```
    Harmonic mean: 0.919 (with a threshold of 0.947)
```

```
print_harmonic_mean(h_mean_3, threshold_3)
```

```
    Harmonic mean: 0.919 (with a threshold of 0.947)
```

Accuracy is increased again (by 2.63 percentage points).

3.6 Amount of images limit

So far, there has been a positive correlation between amount of images used for the template and its harmonic mean. In this section we test its limit. For this we need a lot of images of the same person, thus we decided to go with the LFW dataset. Let's look at some stats from the dataset.

```
from collections import defaultdict
amount_images = defaultdict(lambda: 0)
paths_many_images = []
path = "/home/philipp/datasets/lfw"
for file in os.listdir(path):
    person_path = os.path.join(path, file)
    amount_images[len(os.listdir(person_path))] += 1
    if len(os.listdir(person_path)) > 100:
        paths_many_images.append(person_path)
```

```
amount_people_below_50_images = 0
for i in sorted(amount_images.keys(), reverse=True):
    if i < 50:
        amount_people_below_50_images += amount_images[i]
    else:
        print("{}␣{}␣{}␣images".format(amount_images[i], "people␣have" if
     amount_images[i] > 1 else "person␣has", i))
print("{}␣people␣have␣less␣than␣50␣images".format(amount_people_below_50_images))
```

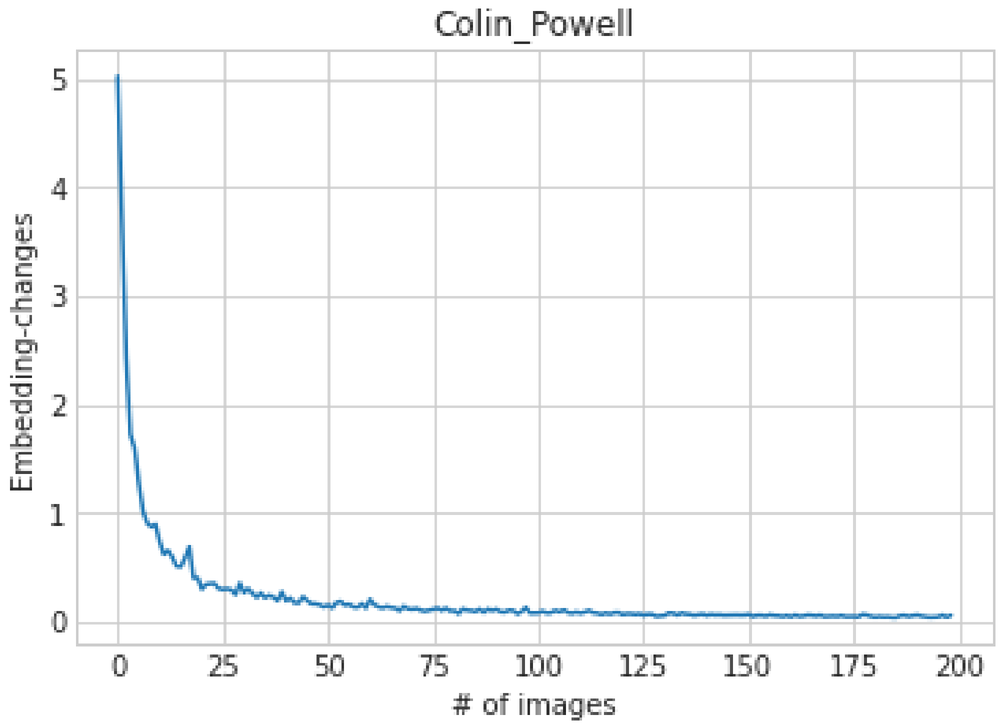
```
    1 person has 530 images
    1 person has 236 images
    1 person has 144 images
    1 person has 121 images
    1 person has 109 images
    1 person has 77 images
    1 person has 71 images
    1 person has 60 images
    1 person has 55 images
    1 person has 53 images
    2 people have 52 images
    5737 people have less than 50 images
```

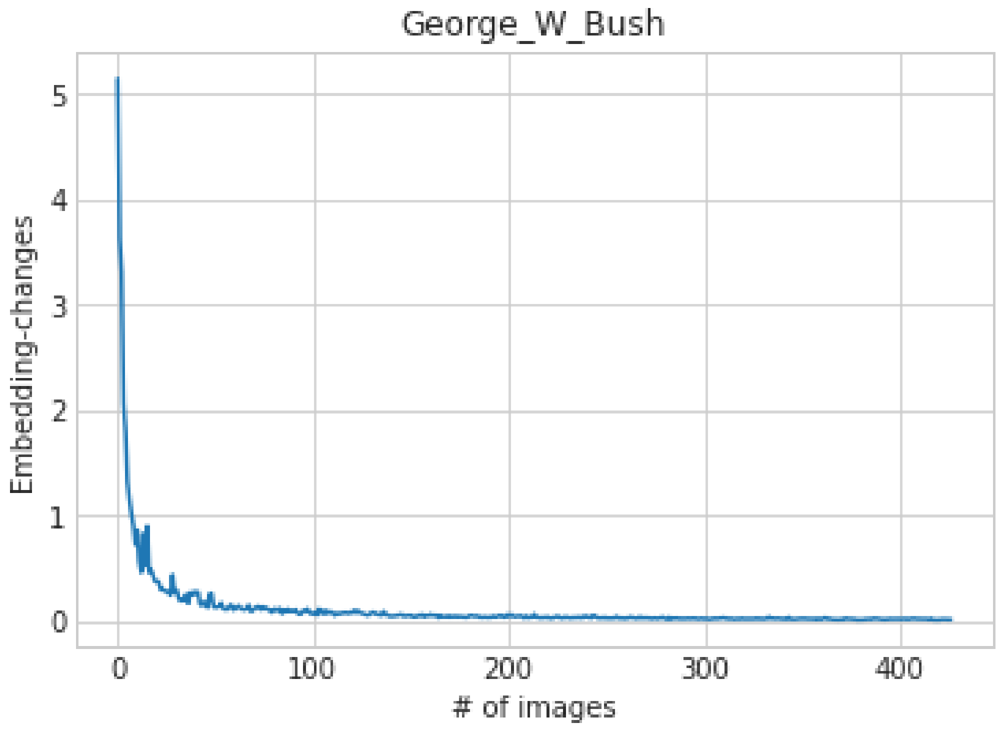
We use the 5 people with more than 100 images and plot the amount of difference in their embeddings.

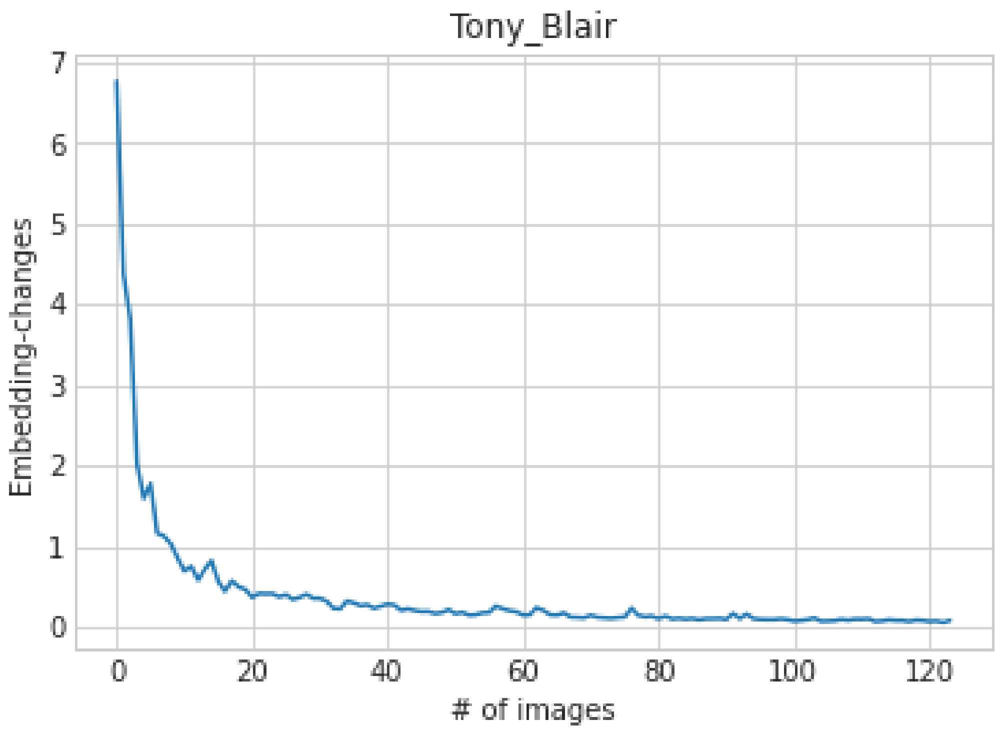
```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
def calc_distances(images, print_only_last_change = False, dont_print=False):
    ret = []
    embs = []
    for image in images:
        emb = rec.get_emb(image)
```

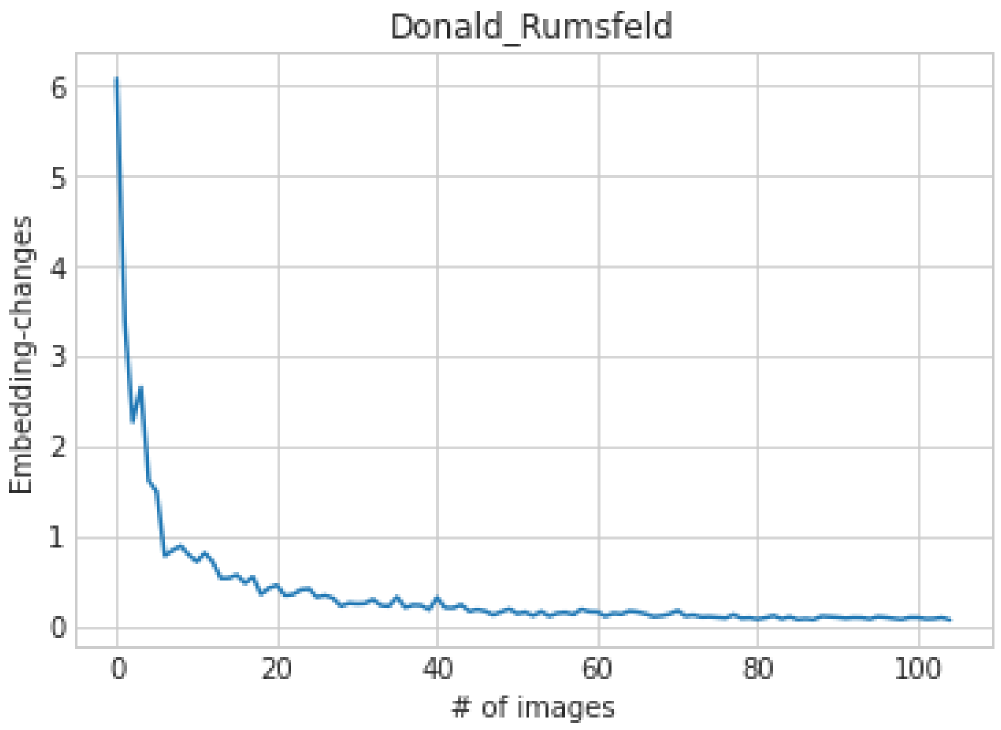
```python
        if len(emb) != 1:
            if not dont_print:
                print("skip␣{}␣due␣to␣{}␣people␣detected".format(image, len(emb)))
        else:
            embs.append(np.array(emb[0]))
    combs = [embs[0]]
    for i in range(1,len(embs)):
        cur = []
        for j in range(i+1):
            cur.append(embs[j])
        emb_combined = np.mean( np.array(cur), axis=0 )
        prev_combined = combs[-1]
        combs.append(emb_combined)
        dist = np.sum(np.absolute(emb_combined - prev_combined))
        if not print_only_last_change or i == len(embs)-1:
            if not dont_print:
                print("After␣{}␣images:␣Distance␣to␣previous:␣{}".format(i+1, dist
    ))
            ret.append(dist)
    return ret
def emb_change(path):
    imgs = []
    for img in sorted(os.listdir(path)):
        img_path = os.path.join(path, img)
        imgs.append(img_path)

    dist = calc_distances(imgs, dont_print=True)
    fig = plt.figure()
    ax = plt.axes()
    plt.title(os.path.basename(path))
    plt.xlabel("#␣of␣images")
    plt.ylabel("Embedding-changes")
    ax.plot(range(len(dist)), dist)
```

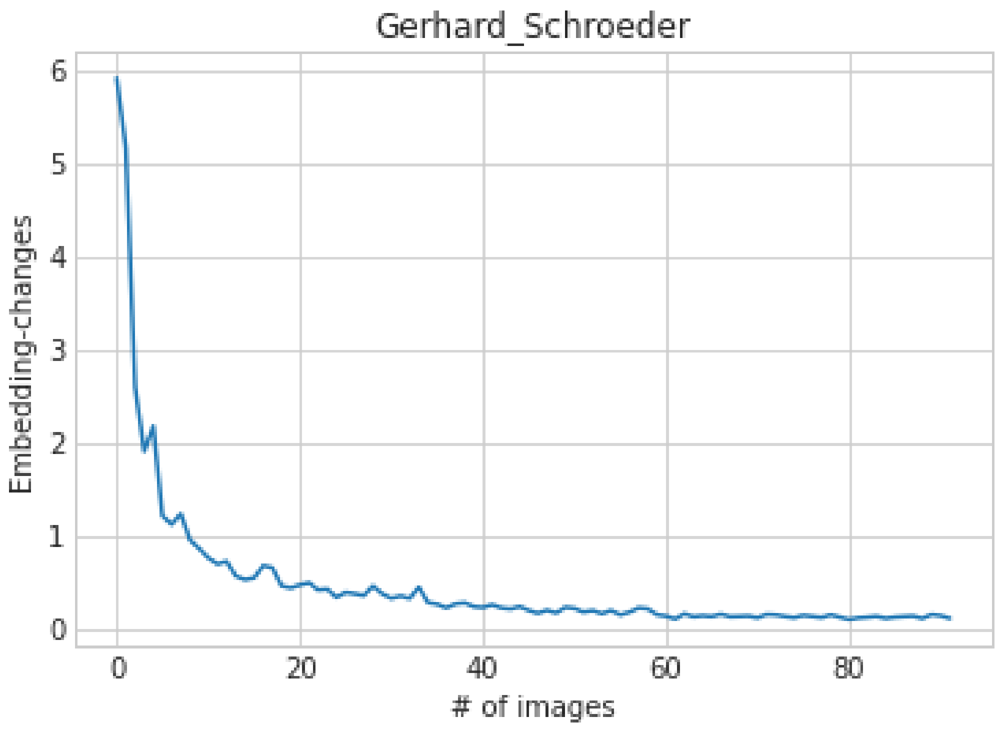
```python
for path in paths_many_images:
    emb_change(path)
```


Colin_Powell
Embedding-changes
# of images
0
1
2
3
4
5
0
25
50
75
100
125
150
175
200


George_W_Bush
Embedding-changes
# of images
0
1
2
3
4
5
0
100
200
300
400

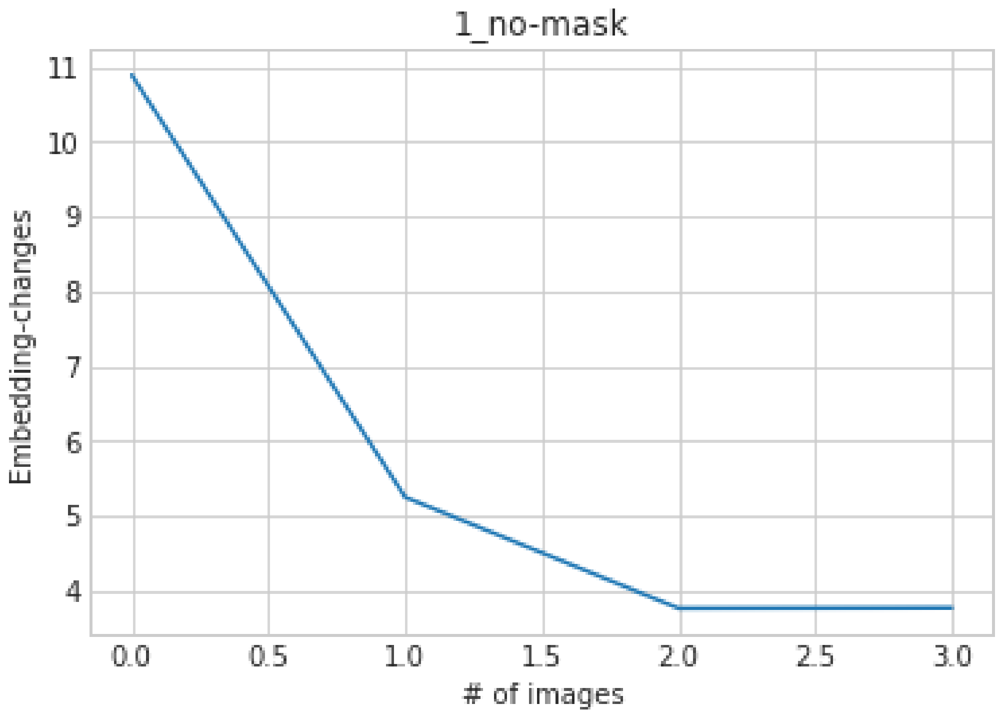

Tony_Blair
Embedding-changes
# of images
0
1
2
3
4
5
6
7
0
20
40
60
80
100
120


Donald_Rumsfeld
Embedding-changes
# of images
0
1
2
3
4
5
6
0
20
40
60
80
100

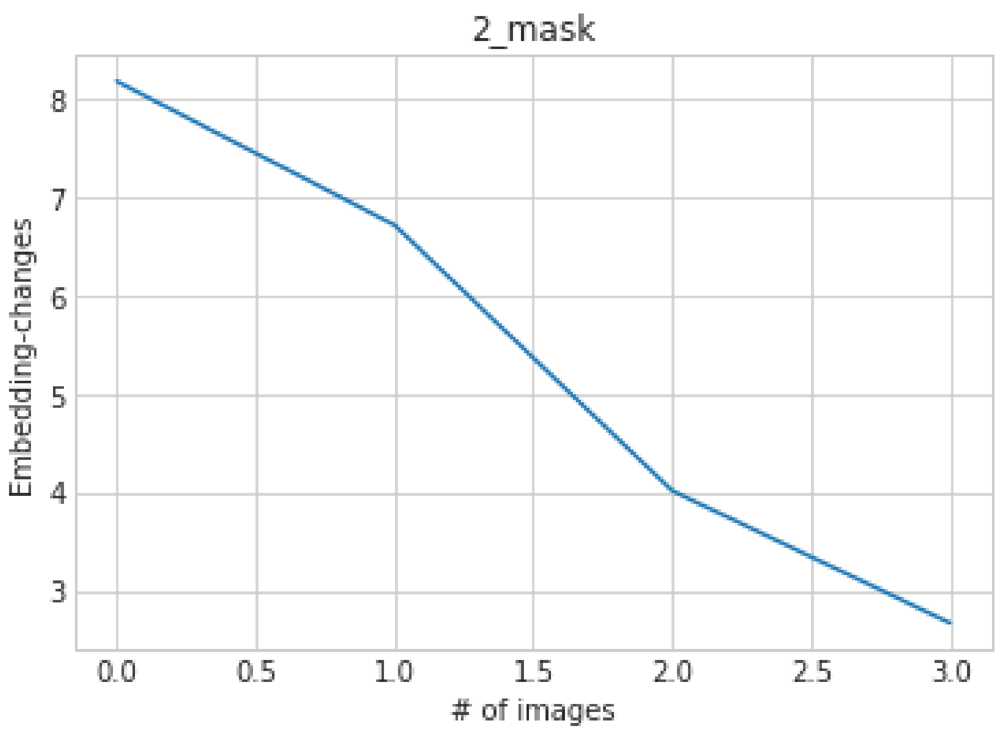


It is clearly visible, that the amount of change decreases with an increased amount of images. After roughly 20 images the combined embedding is not getting better anymore. How is the embedding changing with our templates?

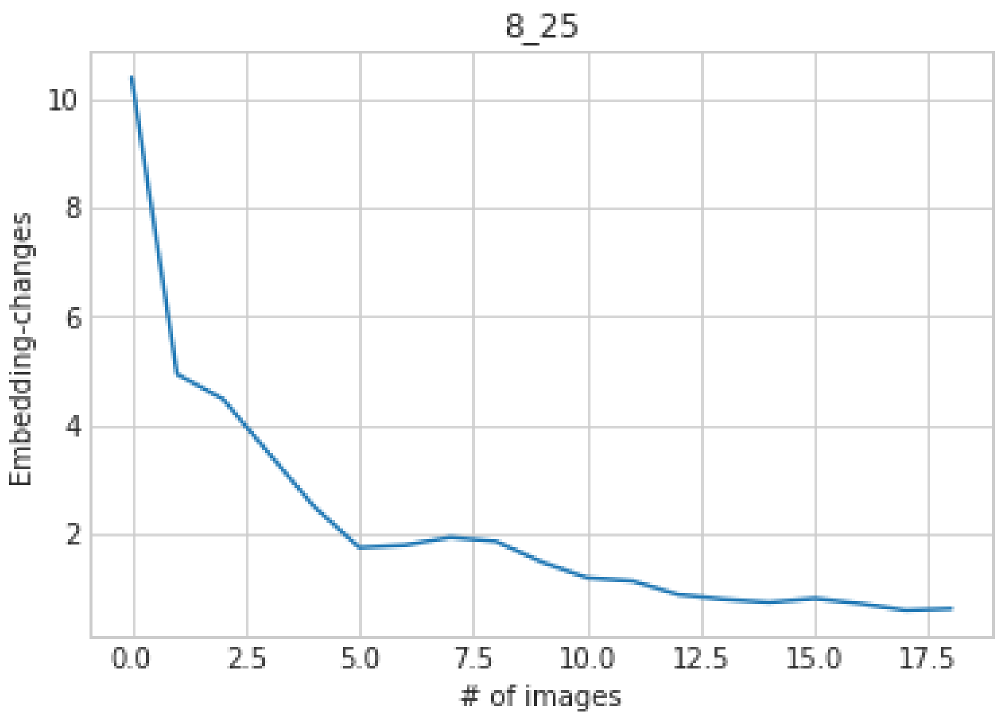
```
emb_change(os.path.join(base_path, "atemplates_variations/hofer2/1_no-mask"))
```



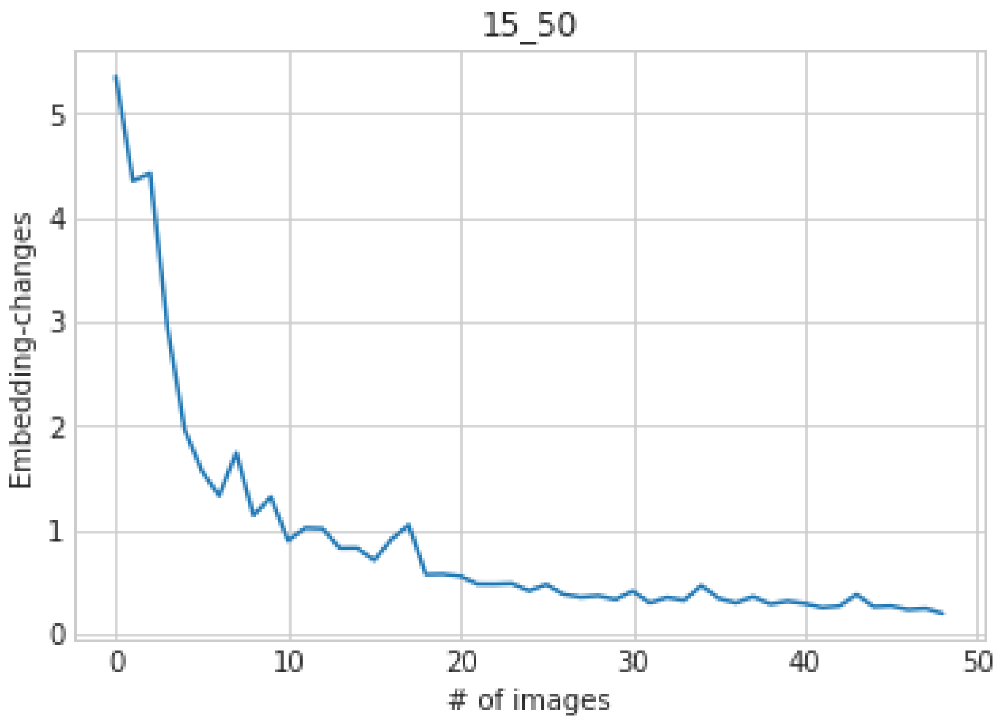
```
emb_change(os.path.join(base_path, "atemplates_variations/hofer2/2_mask"))
```



```
emb_change(os.path.join(base_path, "atemplates_variations/hofer2/8_25"))
```



```
emb_change(os.path.join(base_path, "atemplates_variations/hofer2/15_50"))
```



4. Voting

What happens if we switch from doing 1 comparison with the average template (created using *n* images) to doing *n* comparisons with the original images?

```
def harmonic_mean_from_conf(tp, tn, fp, fn):
    if tp+fp == 0:
        return 0
    if tp + fn == 0:
        return 0
    prec = tp / (tp + fp)
    recall = tp / (tp + fn)
    harmonic_mean = 2*(prec*recall/(prec+recall))
    return harmonic_mean
def optimize_amount_votes(same, different):
    glob_min = min(min(same), min(different))
    glob_max = max(max(same), max(different))

    max_h_mean = 0
    votesneeded_max_h_mean = 0

    for votes_needed in range(glob_min, glob_max+1):
        tp, tn, fp, fn = 0,0,0,0
        for s in same:
            if s >= votes_needed:
                tp += 1
            else:
                fn += 1
        for d in different:
            if d >= votes_needed:
                fp += 1
            else:
                tn += 1
        h_mean = harmonic_mean_from_conf(tp, tn, fp, fn)
        if h_mean > max_h_mean:
            max_h_mean = h_mean
            votesneeded_max_h_mean = votes_needed
    return max_h_mean, votesneeded_max_h_mean
```

As a baseline, let's calculate the harmonic mean if we always predict match/non-match:

```
amount_same = 0
for file in os.listdir(new_location):
    img_path = os.path.join(new_location, file)
    if os.path.isfile(img_path):
        amount_same += 1
amount_different = 0
for file in os.listdir(base_path):
    person_path = os.path.join(base_path, file)
    if os.path.isdir(person_path) and file != person and not file in exclude_dirs:
        for img in os.listdir(person_path):
            img_path = os.path.join(person_path, img)

            if rec.get_eye_distance(img_path) >= eye_distance and rec.
    get_eyes_mouth_distance(img_path) >= eye_mouth_distance:
                amount_different += 1
# always predict true
tp=amount_same
fp=amount_different
fn,tn = 0,0
print("Always␣predicting␣true␣would␣yield␣a␣harmonic␣mean␣of␣{}".format(
    harmonic_mean_from_conf(tp, tn, fp, fn)))
# always predict false
tn = amount_different
fn = amount_same
tp, fp = 0, 0
print("Always␣predicting␣false␣would␣yield␣a␣harmonic␣mean␣of␣{}".format(
    harmonic_mean_from_conf(tp, tn, fp, fn)))
```

```
    Always predicting true would yield a harmonic mean of 0.3528161530286929
    Always predicting false would yield a harmonic mean of 0
```

```
local_cache = dict()
```

```
folder = os.path.join(base_path, "atemplates_variations/hofer/3_face-25")
def get_embs_from_path(path):
    if path in local_cache:
        return local_cache[path]

    embs = []
    for file in sorted(os.listdir(path)):
        img_path = os.path.join(path, file)
        embs.append(rec.get_emb(img_path)[0])

    local_cache[path] = embs
    return embs
def get_amount_votes(img, template_path, threshold):
    amount_votes = 0
    probe = rec.get_emb(img)[0]
    count = 0
    for emb in get_embs_from_path(template_path):
        score = rec.get_score(probe, emb)
        if score < threshold:
            amount_votes += 1
        count += 1
    return amount_votes
```

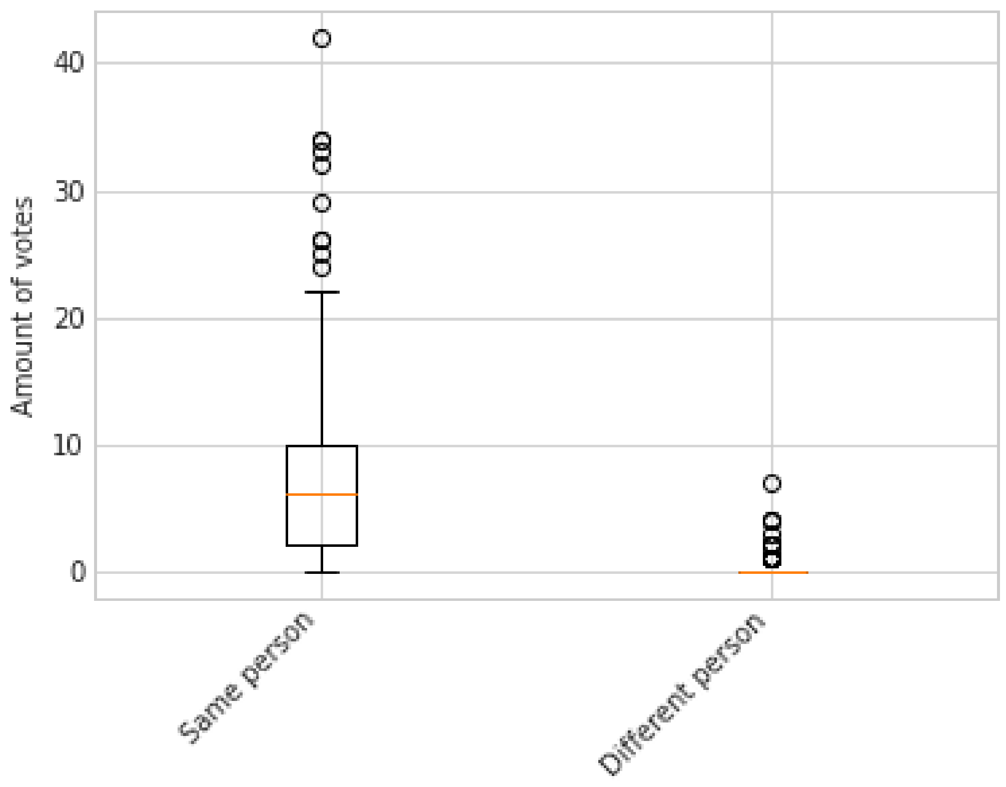
```
output = []
h_means = []
for threshold in np.arange(0.5,2,0.1):
    same = []
    for file in os.listdir(new_location):
        img_path = os.path.join(new_location, file)
        if os.path.isfile(img_path):
            same.append(get_amount_votes(img_path, folder, threshold))
    different = []
    for file in os.listdir(base_path):
        person_path = os.path.join(base_path, file)
        if os.path.isdir(person_path) and file != person and not file in exclude_dirs:
            for img in os.listdir(person_path):
                img_path = os.path.join(person_path, img)
                if rec.get_eye_distance(img_path) < eye_distance:
                    continue
                if rec.get_eyes_mouth_distance(img_path) < eye_mouth_distance:
                    continue
                different.append(get_amount_votes(img_path, folder, threshold))
    h_mean, amount_votes = optimize_amount_votes(same,different)
    h_means.append(h_mean)
    output.append("Threshold={:.2f}:␣h_mean:␣{:.4f}␣(with␣{}␣{}␣needed)".format(threshold, h_mean, amount_votes, "vote" if amount_votes==1 else "votes"))
```

```
    - Threshold=0.50: h_mean: 0.3528 (with 0 votes needed)
    - Threshold=0.60: h_mean: 0.3528 (with 0 votes needed)
    - Threshold=0.70: h_mean: 0.3528 (with 0 votes needed)
    - Threshold=0.80: h_mean: 0.3528 (with 0 votes needed)
    - Threshold=0.90: h_mean: 0.6033 (with 1 vote needed)
    - Threshold=1.00: h_mean: 0.8317 (with 1 vote needed)
    - **Threshold=1.10: h_mean: 0.8580 (with 2 votes needed)**
    - Threshold=1.20: h_mean: 0.8520 (with 6 votes needed)
    - Threshold=1.30: h_mean: 0.8563 (with 15 votes needed)
    - Threshold=1.40: h_mean: 0.8333 (with 28 votes needed)
    - Threshold=1.50: h_mean: 0.8474 (with 40 votes needed)
    - Threshold=1.60: h_mean: 0.8446 (with 45 votes needed)
    - Threshold=1.70: h_mean: 0.8078 (with 48 votes needed)
    - Threshold=1.80: h_mean: 0.7481 (with 49 votes needed)
    - Threshold=1.90: h_mean: 0.6256 (with 50 votes needed)
```

As expected, the higher the threshold is set, the more votes are needed for optimal performance. Interestingly, the highest-performing combination of hyper-parameters sets the threshold quite low (1.05) and requires only a single vote. With the previous average embedding, we achieve an 94.3% harmonic mean with this template. With the best hyper-parameters (threshold=1.05; votes_needed=1) we achieve about the same harmonic mean (93%). Thus, we conclude that the additional processing power during inference is not significantly impacting the harmonic mean. To get more intuition, we can plot a box plot of the amount of votes images get, grouped by if face recognition successfully detected the pair correctly. We use the hyper-parameters of our best output:

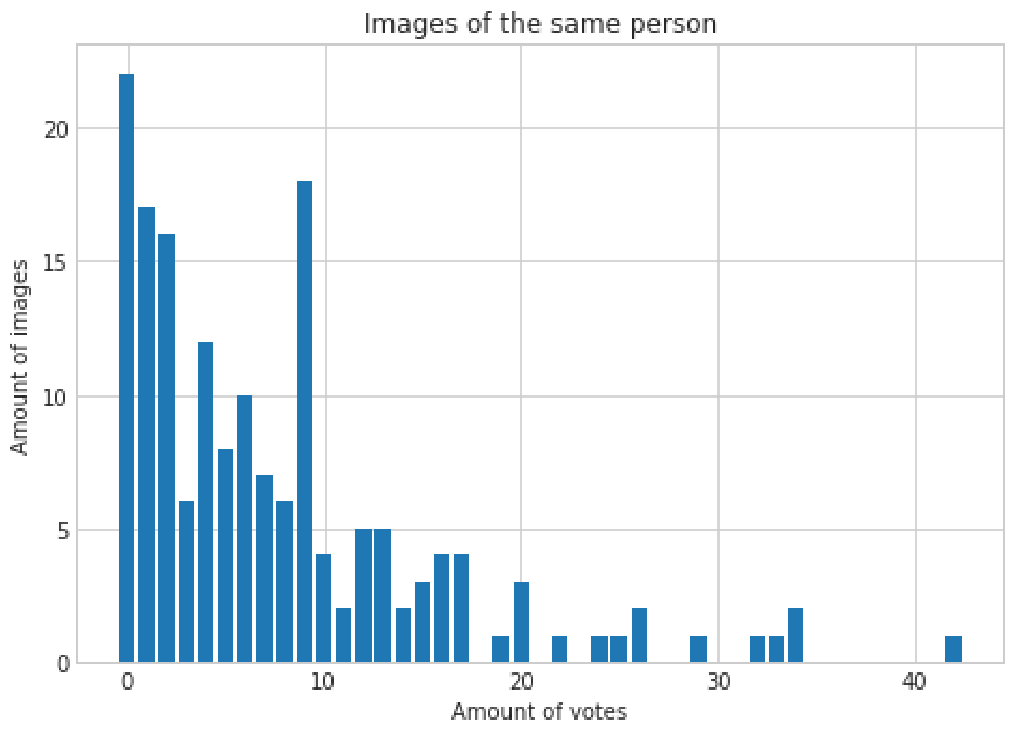
```python
threshold = 1.05
same = []
for file in os.listdir(new_location):
    img_path = os.path.join(new_location, file)
    if os.path.isfile(img_path):
        same.append(get_amount_votes(img_path, folder, threshold))
different = []
for file in os.listdir(base_path):
    person_path = os.path.join(base_path, file)
    if os.path.isdir(person_path) and file != person and not file in exclude_dirs:
        for img in os.listdir(person_path):
            img_path = os.path.join(person_path, img)
            if rec.get_eye_distance(img_path) < eye_distance:
                continue
            if rec.get_eyes_mouth_distance(img_path) < eye_mouth_distance:
                continue
            different.append(get_amount_votes(img_path, folder, threshold))
```

```python
import matplotlib.pyplot as plt
data = [same, different]
axes = plt.boxplot(data)
plt.xticks(np.arange(len(data))+1, ["Same␣person", "Different␣person"], rotation
    =45, ha="right")
plt.ylabel("Amount␣of␣votes")
plt.show()
```



As expected, for the *same person* class most faces receive many votes. Although, there seems to be quite a few faces where only a few (or a single) vote matches the template. Let's plot this as bar chart:

```python
from collections import defaultdict
amount_votes = defaultdict(lambda: 0)
for s in same:
    amount_votes[s] += 1
data = []
for i in range(0,max(amount_votes.keys())+1):
    data.append(amount_votes[i])
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(range(len(data)),data)
ax.set_xlabel("Amount␣of␣votes")
ax.set_ylabel("Amount␣of␣images")
ax.set_title("Images␣of␣the␣same␣person")
plt.show()
```



(Interpretation: e.g. there are 22 images of the same person, which receive 0 votes) Let's look at the images with 0 positive votes:

```python
zero_votes = []
for file in os.listdir(new_location):
    img_path = os.path.join(new_location, file)
    if get_amount_votes(img_path, folder, threshold) == 0:
        zero_votes.append(img_path)
comb_images(zero_votes)
```

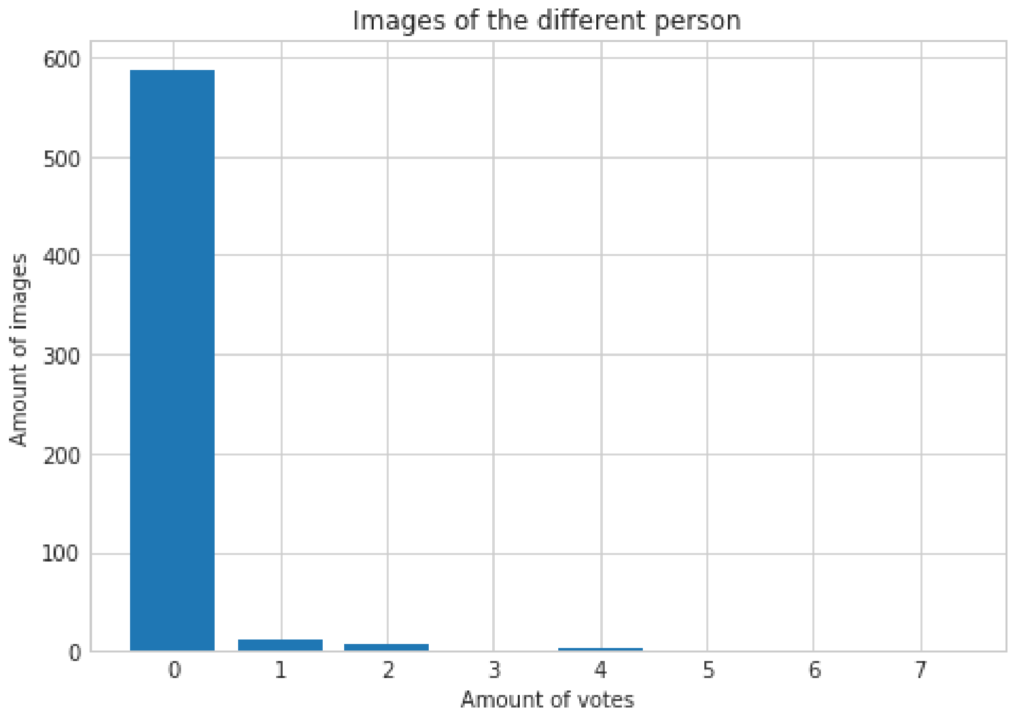
In all these images, the face is not really visible. Thus, with an additional pipeline step between face detection and face recognition which calculates the quality of the face, we might get rid of this example.

```python
def conf_matrix(votes_needed):
    tp, fn, fp, tn = 0,0,0,0
    for s in same:
        if s >= votes_needed:
            tp += 1
        else:
            fn += 1
    for d in different:
        if d >= votes_needed:
            fp += 1
        else:
            tn += 1
    print_confusion(fp, fn, tp, tn)
    return tp, tn, fp, fn
for votes_needed in range(10):
    res = conf_matrix(votes_needed)
    print("(votes_needed={};␣h_mean={:.3f})".format(votes_needed,
     harmonic_mean_from_conf(*res)))
    print()
```

```
+--------------------------------------------+
|              Predicted condition            |
+-----------+----------+----------+----------+
|           |          | Positive | Negative |
+-----------+----------+----------+----------+
|   Actual  | Positive |  166.00  |   0.00   |
| Condition | Negative |  609.00  |   0.00   |
+-----------+----------+----------+----------+
(votes_needed=0; h_mean=0.353)

+--------------------------------------------+
|              Predicted condition            |
+-----------+----------+----------+----------+
|           |          | Positive | Negative |
+-----------+----------+----------+----------+
|   Actual  | Positive |  144.00  |  22.00   |
| Condition | Negative |  22.00   |  587.00  |
+-----------+----------+----------+----------+
(votes_needed=1; h_mean=0.867)

+--------------------------------------------+
|              Predicted condition            |
+-----------+----------+----------+----------+
|           |          | Positive | Negative |
+-----------+----------+----------+----------+
|   Actual  | Positive |  127.00  |  39.00   |
| Condition | Negative |  11.00   |  598.00  |
+-----------+----------+----------+----------+
(votes_needed=2; h_mean=0.836)

+--------------------------------------------+
|              Predicted condition            |
+-----------+----------+----------+----------+
|           |          | Positive | Negative |
+-----------+----------+----------+----------+
|   Actual  | Positive |  111.00  |  55.00   |
| Condition | Negative |   5.00   |  604.00  |
+-----------+----------+----------+----------+
(votes_needed=3; h_mean=0.787)

+--------------------------------------------+
|              Predicted condition            |
+-----------+----------+----------+----------+
|           |          | Positive | Negative |
+-----------+----------+----------+----------+
|   Actual  | Positive |  105.00  |  61.00   |
| Condition | Negative |   4.00   |  605.00  |
+-----------+----------+----------+----------+
(votes_needed=4; h_mean=0.764)

+--------------------------------------------+
|              Predicted condition            |
+-----------+----------+----------+----------+
|           |          | Positive | Negative |
+-----------+----------+----------+----------+
|   Actual  | Positive |  93.00   |  73.00   |
| Condition | Negative |   1.00   |  608.00  |
+-----------+----------+----------+----------+
(votes_needed=5; h_mean=0.715)

+-------------------------------------------+
|            Predicted condition            |
+-----------+----------+----------+----------+
|           |          | Positive | Negative |
+-----------+----------+----------+----------+
|   Actual  | Positive |  85.00   |  81.00   |
| Condition | Negative |   1.00   |  608.00  |
+-----------+----------+----------+----------+
(votes_needed=6; h_mean=0.675)

+-------------------------------------------+
|            Predicted condition            |
+-----------+----------+----------+----------+
|           |          | Positive | Negative |
+-----------+----------+----------+----------+
|   Actual  | Positive |  75.00   |  91.00   |
| Condition | Negative |   1.00   |  608.00  |
+-----------+----------+----------+----------+
(votes_needed=7; h_mean=0.620)

+-------------------------------------------+
|            Predicted condition            |
+-----------+----------+----------+----------+
|           |          | Positive | Negative |
+-----------+----------+----------+----------+
|   Actual  | Positive |  68.00   |  98.00   |
| Condition | Negative |   0.00   |  609.00  |
+-----------+----------+----------+----------+
(votes_needed=8; h_mean=0.581)

+-------------------------------------------+
|            Predicted condition            |
+-----------+----------+----------+----------+
|           |          | Positive | Negative |
+-----------+----------+----------+----------+
|   Actual  | Positive |  62.00   |  104.00  |
| Condition | Negative |   0.00   |  609.00  |
+-----------+----------+----------+----------+
(votes_needed=9; h_mean=0.544)
```

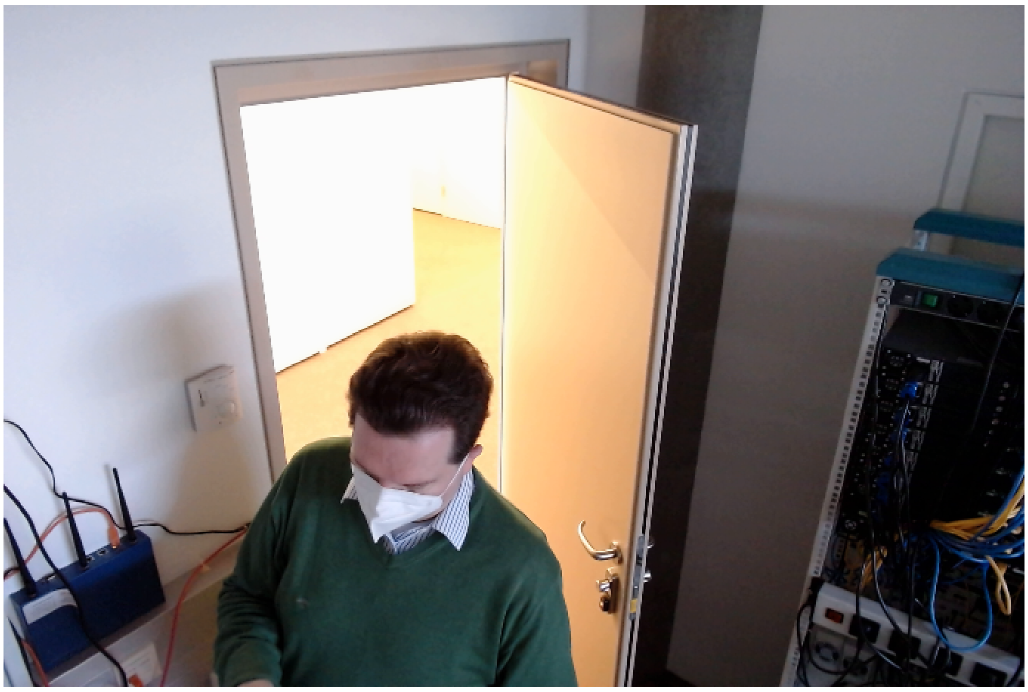
Similar to the previous analysis, we can create the same plot for all different people:

```python
amount_votes = defaultdict(lambda: 0)
for d in different:
    amount_votes[d] += 1
data = []
for i in range(0,max(amount_votes.keys())+1):
    data.append(amount_votes[i])
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(range(len(data)),data)
ax.set_xlabel("Amount␣of␣votes")
ax.set_ylabel("Amount␣of␣images")
ax.set_title("Images␣of␣the␣different␣person")
plt.show()
```



For 96.4% of the images, no single template image has a distance smaller than the chosen threshold. There are 1 images where 7 template images have a distance smaller than the threshold. Thus, if we set votes_needed to 8 we would not get a single FP (in our dataset). Let's look at these 1 examples with 7 votes:

```
highest_errors = []
max_votes = len(data)-1
for file in os.listdir(base_path):
    person_path = os.path.join(base_path, file)
    if os.path.isdir(person_path) and file != person and not file in exclude_dirs:
        for img in os.listdir(person_path):
            img_path = os.path.join(person_path, img)
            if rec.get_eye_distance(img_path) < eye_distance:
                continue
            if rec.get_eyes_mouth_distance(img_path) < eye_mouth_distance:
                continue
            if get_amount_votes(img_path, folder, threshold) == max_votes:
                highest_errors.append(img_path)
```

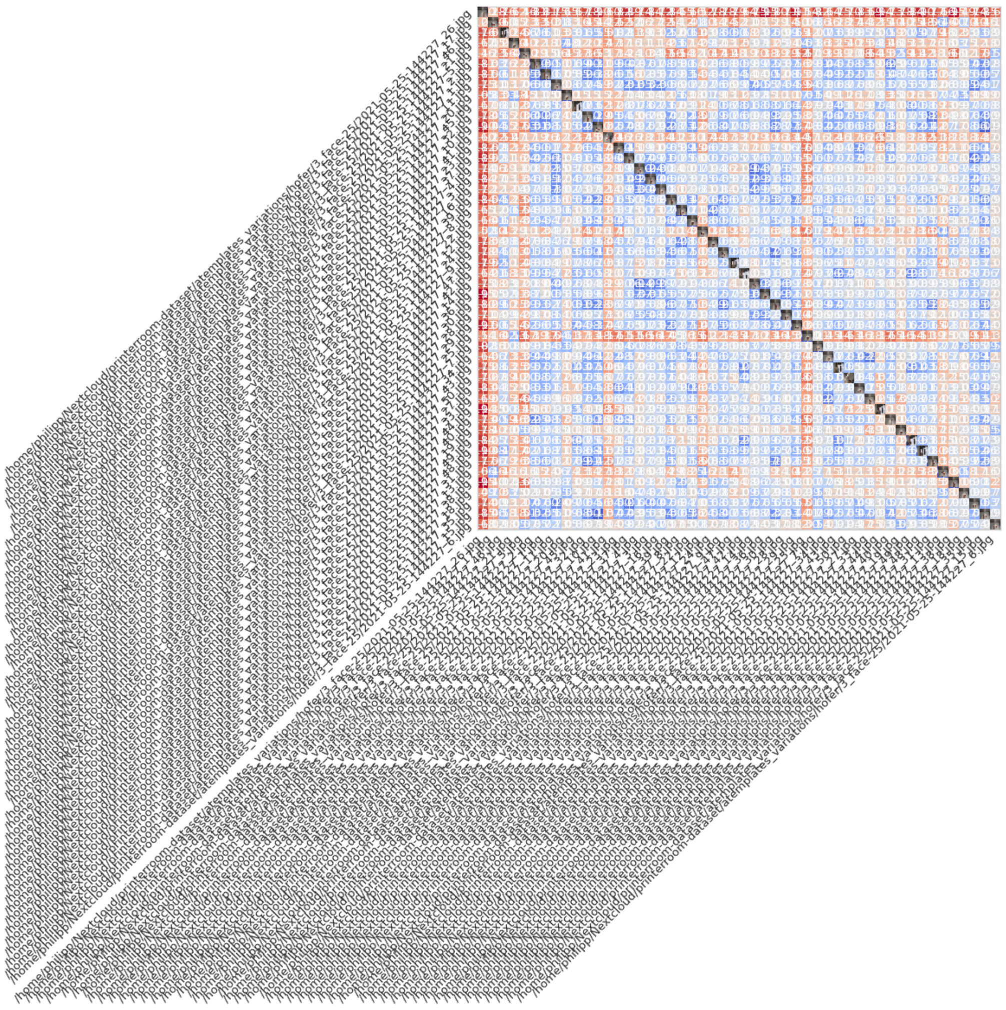


```
highest_errors
```

```
    ['/home/philipp/Nextcloud/printerroom-dataset/sonntag/2021-06-14T15-20-44.png
    ']
```

```
for error in highest_errors:
    get_amount_votes(error, folder, 1.05)
```

```
data = []
for file in os.listdir(folder):
    file_path = os.path.join(folder, file)
    data.append(file_path)
rec.vis.get_confusion_vis(data)
```



5. KMeans

```
from sklearn.cluster import KMeans
import numpy as np
```

```
kmeans = KMeans(n_clusters=1, random_state=42).fit(get_embs_from_path(folder))
```

```
template = kmeans.cluster_centers_
```

```
h_mean_onecluster, same_onecluster, different_onecluster, threshold_onecluster =
    harmonic_mean(template)
print_harmonic_mean(h_mean_onecluster, threshold_onecluster)
```

```
    Harmonic mean: 0.919 (with a threshold of 0.947)
```

```
kmeans = KMeans(n_clusters=5, random_state=42).fit(get_embs_from_path(folder))
```

```
emb_combined = np.mean( np.array(kmeans.cluster_centers_), axis=0 )
h_mean_fiveclustersavg, same_fiveclustersavg, different_fiveclustersavg,
    threshold_fiveclustersavg = harmonic_mean(emb_combined)
print_harmonic_mean(h_mean_fiveclustersavg, threshold_fiveclustersavg)
```

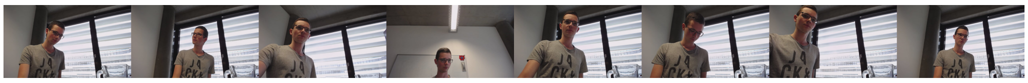
```
    Harmonic mean: 0.932 (with a threshold of 0.93)
```

```
for amount_clusters in range(1,20+1):
    print("{}␣cluster{}:".format(amount_clusters,"" if amount_clusters == 1 else "
    s"))
    kmeans = KMeans(n_clusters=amount_clusters, random_state=42).fit(
    get_embs_from_path(folder))
    emb_combined = np.mean( np.array(kmeans.cluster_centers_), axis=0 )
    h_mean_fiveclustersavg, same_fiveclustersavg, different_fiveclustersavg,
    threshold_fiveclustersavg = harmonic_mean(emb_combined)
    print_harmonic_mean(h_mean_fiveclustersavg, threshold_fiveclustersavg)
    print("")
```

```
    1 cluster:
    Harmonic mean: 0.919 (with a threshold of 0.947)

    2 clusters:
    Harmonic mean: 0.919 (with a threshold of 0.944)

    3 clusters:
    Harmonic mean: 0.932 (with a threshold of 0.958)

    4 clusters:
    Harmonic mean: 0.925 (with a threshold of 0.943)

    5 clusters:
    Harmonic mean: 0.932 (with a threshold of 0.93)

    6 clusters:
    Harmonic mean: 0.928 (with a threshold of 0.945)

    7 clusters:
    Harmonic mean: 0.929 (with a threshold of 0.932)

    8 clusters:
    Harmonic mean: 0.929 (with a threshold of 0.91)

    9 clusters:
    Harmonic mean: 0.920 (with a threshold of 0.918)

    10 clusters:
    Harmonic mean: 0.927 (with a threshold of 0.906)

    11 clusters:
    Harmonic mean: 0.932 (with a threshold of 0.907)

    12 clusters:
    Harmonic mean: 0.929 (with a threshold of 0.909)

    13 clusters:
    Harmonic mean: 0.932 (with a threshold of 0.918)

    14 clusters:
    Harmonic mean: 0.927 (with a threshold of 0.921)

    15 clusters:
    Harmonic mean: 0.936 (with a threshold of 0.923)

```

```
16 clusters:
Harmonic mean: 0.931 (with a threshold of 0.937)

17 clusters:
Harmonic mean: 0.931 (with a threshold of 0.941)

18 clusters:
Harmonic mean: 0.935 (with a threshold of 0.938)

19 clusters:
Harmonic mean: 0.934 (with a threshold of 0.946)

20 clusters:
Harmonic mean: 0.929 (with a threshold of 0.942)
```

```python
kmeans = KMeans(n_clusters=4, random_state=42).fit(get_embs_from_path(folder))
```



```python
import sys
from PIL import Image
def comb_images_face(path):
    images = []
    if isinstance(path, list):
        for p in path:
            images.append(Image.open(p))
    else:
        for file in os.listdir(path):
            images.append(Image.open(os.path.join(path, file)))
    faces = []
    for image in images:
        faces.append(rec.extractor.get_faces(np.array(image))[0])

    images = faces

    widths, heights = zip(*(i.size for i in images))
    total_width = sum(widths)
    max_height = max(heights)
    new_im = Image.new('RGB', (total_width, max_height))
    x_offset = 0
    for im in images:
        new_im.paste(im, (x_offset,0))
        x_offset += im.size[0]
    return new_im
```

5.1 Summary

91.9% for average embedding, 85.8% for voting; 93.6% for k_means (with 15 clusters)