# Mimicking Face Pareidolia with a CNN

Roman Kessler

July 10, 2020

## Abstract

Face pareidolia describes the phenomenon of seeing a face in random noise. Previous studies have shown, face pareidolia is associated with higher activation of neuronal areas specialized on face perception [1, 2, 3]. However, it is unclear if this phenomenon is a top-down task effect, caused by the attempt of finding a face in random noise, or if it is a bottom-up stimulus effect, caused by a random beneficial configuration of the random pattern – which renders the stimulus more face-like. Most likely, it is a mixture of both effects. However, we here asked ourselves, if we could build a machine, that is able to mimic the pareidolia experienced by a subject. Finding such a machine could help interpreting face pareidolia in humans. For this aim, we trained a CNN on artificially noisy face and non-face images to retrieve a binary face classifier. We then compared face predictions of this machine on random noise stimuli to face predictions of human subjects on the same noise stimuli. In our case, we could not find an overlap in the predictions of the machine and the predictions of the subject. We discuss possible explanations and implications for further research.

## 1 Introduction

Recognizing patterns is a noisy environment is a fundamental human ability and important for learning, reasoning, and creativity. Nearly everyone has experienced a phenomenon called *face pareidolia*, i.e. seeing a face in a random pattern. Typically, face pareidolia (or sometimes different forms of pareidolia) occur when looking at clouds or random rock formations. Another famous example is the face on mars illusion (Fig. 1, [4]). Whereas face pareidolia seems like a side effect of important human abilities and is typically harmless, more severe consequences occur when random patterns get over-interpreted by conspiracy ideologists, as in the absurd history of the face on mars illusion [5].

Figure 1: The *face on mars* illusion [4].

On a neural system level, an interesting series of experiments has showed, that imagining faces in random noise is associated with higher activation in neuronal areas for face perception [3, 1, 2]. However, the nature of the face perception in random noise is of debate. Do images, labelled as face by a subject, contain some physical properties that may drive face perception (*bottom-up pareidolia*). Or alternatively, is the task - the instruction to find faces - that strong, that the subject randomly "chooses" some images to contain faces (*top-down pareidolia*)? The answer is probably a combination of both, alongside with a vast variety of confounding effects.

The aim of this study is to find any physical correlate of a face in random noise. For this we used a deep convolutional neural network which we trained as binary face-classifier. We then compared the classifiers predictions to the subjects' predictions.

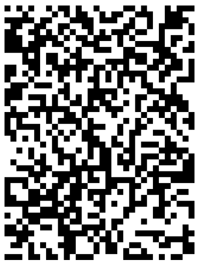

Figure 2: Example stimulus. All stimuli consisted of random black and white pixels.

## 2 Material & Methods

### 2.1 Subjects and data

12 subject were recruited at the University of Marburg. Written informed consent was obtained from all participants before enrolment, and the study protocol followed the tenets of the declaration of Helsinki. Two subjects labeled each 3295 random noise images with the extend of $43 * 32$px (Fig. 2). The remaining ten subjects labeled each 1000 random noise images. The subjects were convinced by a cover-story that around 50% of images contain faces, which were barely visible. The appearing face should cover most part of the image and view towards the camera, similar to a passport photo. The subjects each labeled around 26% to 47% of the images as face, and the rest as non-face. The pixels of each stimulus were randomly drawn from a Bernoulli distribution $B(1, 0.5)$ at the beginning of a trial. Therefore it was very unlikely that two subjects rated the same random noise stimulus ($p \equiv \frac{1}{10^{414}}$).

### 2.2 Binary face classifier

To disentangle if there may be some physical correlate of a face in the stimuli, the subject labeled as "face", we constructed a binary face classifier. As a core network, we used the MobileNet architecture [6]. We cut the fully connected layers at the end of the network, and replaced them by two fully connected layers, whereas the very last layer produced binary probabilities for two classes (Fig. 3). We allowed the full model to be trained (no frozen layers).

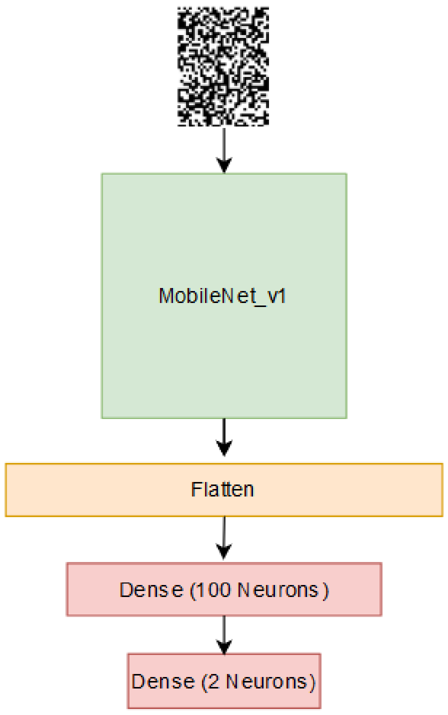

Figure 3: Coarse model architecture. The MobileNet framework [6] served as base network (here displayed as layer 1), whereas the other layers were added manually. All layers were trainable, including those within the MobileNet architecture. The network was first trained on abstracted face- and non-face stimuli (Fig. 5 & 6), and then tested with pure noise stimuli (Fig. 2).

Figure 4: Example training images. Displayed are exemplary original images from the respective data bases. For model training, those images were thoroughly preproccessed. The results of preprocessed images can be seen in Fig. 5 and Fig. 6

### 2.2.1 Training categories

To create a binary face classifier we needed face and non-face training images. For the non-face class, we chose 8000 cat and dog images from a public repository (`https://www.kaggle.com/chetankv/dogs-cats-image`). For the face-class, we randomly chose 32000 images from the *CelebFaces Attributes (CelebA)* dataset [7], which contain faces of celebrities covering most part of the image area.

### 2.2.2 Training image preprocessing

We trained a model which shall later be tested on the noise images, rated by the subjects (Fig. 2) to delineate if there are any similarities in face-no-face-decision between subjects and a binary face classifier. Therefore we needed to render the images for training the network similar to the random noise images, which we want to predict in a later step.

The images were first converted to gray-scale, cropped at the borders, and re-scaled to $43x32$px to be in the same size than the random noise stimuli, which serve as test stimuli in a later step. The cropping was performed to get get the faces more centered and covering the entire picture. Because the random noise stimuli had an approximate 50:50 distribution of black and white pixels, we determined the median pixel value of each training image and then applied binarization to the whole image, with pixels below the median pixel converted to 0 and pixels above to 255. This killed two birds with one stone: we had (1) a histogram equalization and (2) sharper edges which within the images which correspond better to the random noise stimuli. We then pseudo-converted the image to RGB because

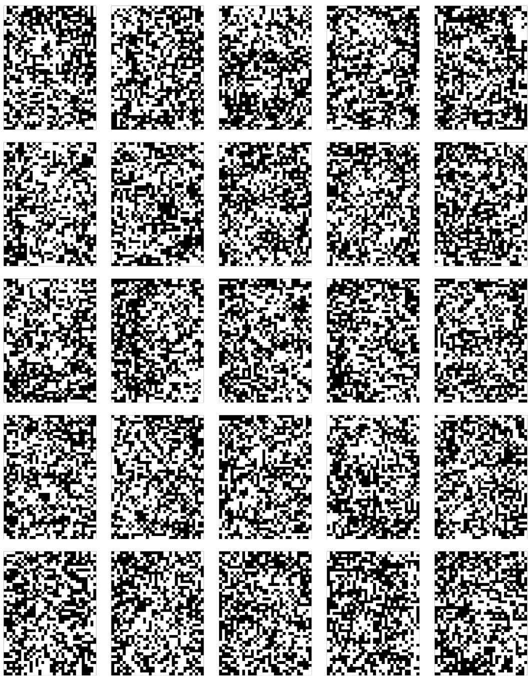

Figure 5: Example face-class training images after preprocessing and the overlay of artificial noise.

MobileNet expects 3 input channels. Because we had a limited amount of images for the non-face category, we introduced a simple image augmentation. Therefore, we added an inverted version of all non-face images, and further horizontally mirrored all stimuli (original and inverted). This effectively quadrupled the amount of images in the non-face class, and ensured equal amount of images for both classes (32.000 each). In a last step, we overlayed random noise to the training images, by randomly flipping a pixel value from black to white or vise versa with a probability of $p = 0.4$, which corresponds to a noise level of 80%. The resulting training images were noisy enough so human observers may have difficulty identify the category (Fig. 5 & 6).

### 2.2.3 Training hyperparameters

The training was conducted using Adam optimizer [8] and binary cross-entropy as loss function. Training was conducted for 20 epochs with batch size of 25 images. Further information can be found in the supplementary material.

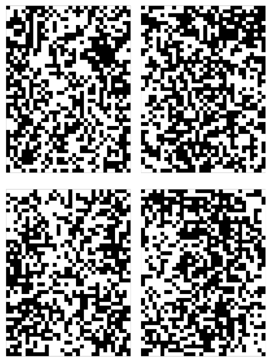
Figure 6: Example non-face-class training images after preprocessing, augmentation, and the overlay of artificial noise. Displayed is the same non-face stimulus as in Fig. 4

### 2.3 Face classification in pure noise

We then used the model to predict the classes of the random noise images (Fig. 2) for all twelve subjects. A prediction contains probabilities for both classes for each image, summing up to 1. To test, if there is a difference (which we expected to be very tiny), we conducted a independent sample t-test on the probability of being categorized as face by the model. Therefore we split the images into two groups: either being labeled as a face by the subject, or labeled as no-face by the subject. We then performed an independent-sample t-test on the probabilities of being a face, according to the prediction by the model, grouped by the prediction of the subject. Because the face prediction of the model was $< 0.5$ for a vast majority of noise images, we first calculated the proportion of images belonging to the face class in each subject. Then we used that proportion (somewhere between 26% and 47%) and identified the corresponding Quantile in the face prediction values as calculated by the model. Therefore, the number of images in the face and non-face classes were consistent between subject and model. A positive T-statistics would indicate an agreement between subject an model. We performed this for every subject and therefore applied Bonferroni Correction ($p = \frac{0.05}{12}$) for null-hypothesis significance testing.

### 2.4 Revealing the "internal" face template

To reveal a possible face pattern within the noise stimuli, which may indicate the internal "template" of a face, or a visualization of an average face prediction of the subject or of the model, we again divided all noise images in two groups. First, for each subject, we split the noise images according to the subject label into a non-face class and a face-class. Then we averaged pixel-wise within a class. Then, we subtracted the non-face-class average image from the face-class average image. This was performed for each subject and her corresponding noise images.

Further to dividing groups by subject label, we also divided by the labels provided by the model, to get the "internal" face template of the model, tested on each subject's noise stimuli. Therefore we first determined the proportion of images that a respective subject labeled as face. Then we split the model predictions of images belonging to the face-class – which spread between 0 and 1 – according to this ratio to ensure a similar split of face- and non-face-classes between subject and model, similar to the proceeding in Section 2.3.

## 3 Results

### 3.1 Training the modified MobileNet

The training accuracy after the 20th epoch exceeded 97%. Cross-validation with a holdout data set of 4000 images revealed a 86% testing accuracy.

### 3.2 Predicting faces in pure noise images

We used the trained model to predict faces in the random noise images which were rated by subjects. The predictions of being a face varied between 0 and 1 ($\tilde{1}00\%$) in most subjects (Fig 7). We then evaluated, if the stimuli which were labelled as face by the subject, exceeded a higher probability of being a face, as predicted by the model, than those the subject labelled as non-face. This relationship was not given (p=0.05, Bonferroni corrected). The

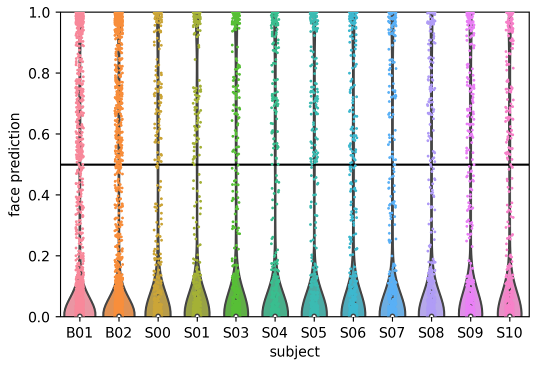

Figure 7: Face predictions of the trained model on the pure noise images of each subject's dataset. Most predictions accumulated around zero, with a considerable amount of predictions spreading the whole range towards 1 (̃100%). There was no significant overlap between the labels of the subjects and the predictions of the model (p¿0.05, Bonferroni corrected).

T-statistics was however positive in most cases, indicating a trend at least in the expected direction.

### 3.3 Internal face template

We then calculated internal face templates on each subject's data for both the subject's behavior and the model's behavior on the subject's data (cf. 2.4). Figure 8 shows the internal templates for all subjects (top row) and for the model predicted on each subject's data (bottom row). The internal templates of the first subject (top left) shows a face-like pattern. This may be caused by the higher amount of images labelled by this subject, and a consistency within this subject's behavior in searching for dark, eye-like patterns in the top region of the image. However, in the second subject, which rated a similar amount of noise images, such a pattern was not visible. Face patterns are however nicely visible in the internal templates of the model (Fig. 8, bottom row). For the different datasets, the internal template of the model seems quite consistent.

## 4 Discussion

In the present article, we were successfully able to train a network on highly noisy train categories. The performance exceeded 86% accuracy on a hold-out test dataset. However, future experiments should investigate the impact of the choice if input image categories (i.e. the non-face category). One could try a more diverse category. Similarly, the images of the face category were quite homogeneous, in particular with respect to location of the faces in the image. This fits to the instruction given to the subjects [2]. The subject's were told to search a face in a passport-like size. However, we do not know if subjects followed this order, as they solely indicated a "face" or "no-face" decision. Similarly, one may chose another network architecture. We chose the MobilNet_v1 [6] due to its lightweight, performance and computational efficiency [6]. Another important point that may be considered in future analyses are the image labels given by the subjects. Subjects had a two-alternative-forced-choice task, namely they just labelled an image as "face" or "non-face". However, a built machine outputs a continuous probability. One may include a function of reaction time of a subject as a substitute for face probability. For example, if the reaction time for a particular image is very high, the subject may be in-confident and therefore the face prediction should be far lower than 100%. On the contrary, if a subject responds very fast, the face prediction may be more confident. Such a fine-grained scale may help splitting the images to "face" and "non-face" groups more reliably, and may improve the accordance of subject and machine in the future.

## Data availability statement

All used data, analysis pipelines, as well as further data of more subjects and further experiments can be retrieved from the GitHub repository of the first author `https://github.com/kesslerr/facepareidolia`.

## Acknowledgement

I want to thank Ina Thome for the stimulus material, which was kindly provided.

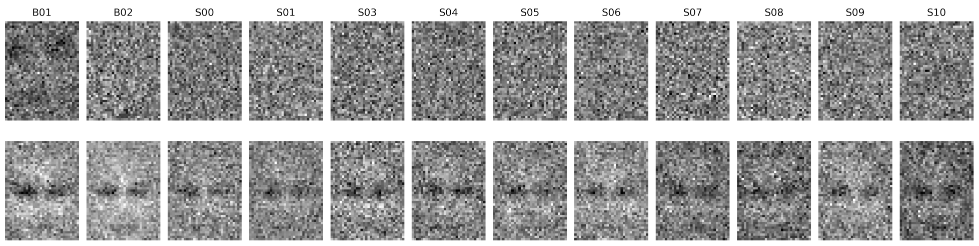

Figure 8: "Internal face templates". After subtracting the average of all noise images which were labelled non-face from those which were labelled face, we were able to disentangle a face-like pattern in one subject (top left). The data of the other subjects did not express such a pattern. The predictions of the machine (bottom row) however revealed a pretty consistent, face-like pattern.

## References

[1] D. Hohmann, I. Thome, and A. Jansen, "I spy with my little eye...: Connectivity analyses of the illusory face perception network," *Pharmacopsychiatry*, vol. 53, no. 02, pp. P5–3, 2020.

[2] I. Thome, K. M. Zimmermann, M. L. Smith, and A. Jansen, "I spy with my little eye, something that is a face...: A brain network for illusory face perception," 2020.

[3] K. M. Zimmermann, A.-S. Stratil, I. Thome, J. Sommer, and A. Jansen, "Illusory face detection in pure noise images: The role of interindividual variability in fmri activation patterns," *Plos one*, vol. 14, no. 1, p. e0209310, 2019.

[4] M. J. Carlotto, "Digital imagery analysis of unusual martian surface features," *Applied Optics*, vol. 27, no. 10, pp. 1926–1933, 1988.

[5] B. Dunning, "The face on mars revealed," *Skeptoid Podcast*, 2008.

[6] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[7] S. Yang, P. Luo, C.-C. Loy, and X. Tang, "From facial parts responses to face detection: A deep learning approach," in *Proceedings of the IEEE international conference on computer vision*, pp. 3676–3684, 2015.

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

## Supplementary Material

### importing libraries

```
import os
from glob import glob
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
np.random.seed(42)
from PIL import Image # to handle images
from scipy.ndimage.filters import gaussian_filter
from scipy.stats import ttest_ind
from keras.models import Sequential
from keras.layers import Dense,GlobalAveragePooling2D
from keras.applications import MobileNet
from keras.preprocessing import image
from keras.applications.mobilenet import preprocess_input
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Model
from keras.optimizers import Adam
from keras.layers import Dense, Dropout, Conv2D, Flatten, MaxPool2D, BatchNormalization
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
```

### functions for preprocessing

```
def iter_binarize(imgArr): # input: an image array of format X,Y,1 (non-color)
    # find median value as threshold
    threshold = np.median(imgArr)
    imgArr = cv2.threshold(imgArr,threshold,255,cv2.THRESH_BINARY)
    return imgArr

# load images and process to the format
def load_and_preproc(file, crop = False):
    img = cv2.imread(file,0)
    if crop:
        img = img[50:168, 40:140].copy() # crop, original shape: 218x178
    img = cv2.resize(img,(32,43)) # reshape it
    img = iter_binarize(img)
    img = cv2.cvtColor(img[1],cv2.COLOR_GRAY2RGB)
    return img

def augment_image(images4D):
    ori_length = images4D.shape[0]
    images4DAug = np.concatenate( (images4D, images4D, images4D, images4D), axis = 0 ) #
         quadruple it
    images4DAug[ori_length*2:,:,:,:] = 255 - images4DAug[ori_length*2:,:,:,:] #invert
        the last half
    images4DAug[ori_length:ori_length*2,:,:,:] = np.flip(images4DAug[ori_length:
        ori_length*2,:,:,:], axis = 2) # flip the second quarter
    images4DAug[ori_length*3:ori_length*4,:,:,:] = np.flip(images4DAug[ori_length*3:
        ori_length*4,:,:,:], axis = 2)# flip the fourth quarter
    return images4DAug

def render_noisy(images4D, p=0.2):
    na,nb,nc,nd = images4D.shape
    for a in range(na):
        for b in range(nb):
```

```
        for c in range(nc):
            r = np.random.binomial(1,p)
            if r == 1:
                images4D[a,b,c,0] = 255-images4D[a,b,c,0]
                images4D[a,b,c,1] = 255-images4D[a,b,c,1]
                images4D[a,b,c,2] = 255-images4D[a,b,c,2]
    return images4D
```

## MobileNet adaption

```
def transfer_learning():
    # get mobilenet and add some layers at the end, 2 output classes
    model = Sequential()
    model.add(MobileNet(weights='imagenet',include_top=False, input_shape = (43, 32, 3)
        )) #imports the mobilenet model and discards the last 1000 neuron layer.
    model.add(Flatten())
    model.add(Dropout(0.2))
    model.add(Dense(100, activation='relu'))
    model.add(Dropout(0.01))
    model.add(Dense(2, activation='softmax'))
    metrics=['accuracy'])
    return model
```

## functions for evaluating the model

```
def predictions_noise(model, X_test):
    predictions = model.predict_proba(X_test)
    facePredictions = predictions[:,1]
    plt.plot(facePredictions)
    plt.show()

def load_labels_and_stim(s, sigma = 1): # s: subject identifier
    # load behavior file
    behaviorFile = 'D:/paredolia/facepareidolia/labels/' + s + 'df.csv'
    df = pd.read_csv(behaviorFile)
    # load stimuli
    files = sorted(glob('D:/paredolia/facepareidolia/stimuli/'+ s +'/' + '*.tif'))
    X = np.ndarray((len(files),43,32,3))
    yn = []
    c = 0
    for j, file in enumerate(files):
        fname = os.path.split(os.path.normpath(file))[-1] # file name without path
        # check if face or noface
        try:
            if df.loc[df['picture'] == fname, 'category'].iloc[0] == "Face":
                yn.append(1)
            elif df.loc[df['picture'] == fname, 'category'].iloc[0] == "noFace":
                yn.append(0)
        except:
            #print(fname + "not rated !")
            c += 1
            continue
        img = cv2.imread(file,0)
        img = cv2.resize(img,(32,43)) # reshape it
        img = np.flip(img, axis = 0) # faces are upside down
        img = cv2.GaussianBlur(img,(sigma,sigma),0)
        img = cv2.threshold(img,127,255,cv2.THRESH_BINARY) #TTT
        img = cv2.cvtColor(img[1],cv2.COLOR_GRAY2RGB) # to rgb
        img = np.expand_dims(img, 0) # new dimension
        X[j-c,:,:,:] = img.astype("int32")
    X = np.delete(X, list(range(X.shape[0]-c , X.shape[0])), axis = 0) # delete the
        images which were not rated
    return X, yn

def test_model(model, sigma = 1):
    # determine subjects
    subs = glob("D:/paredolia/facepareidolia/stimuli/*")
    subs = [os.path.split(subs[i])[1] for i,s in enumerate(subs)]
    # loop over subjects
    allFacePrediction = np.ndarray(1)
    allNoFacePrediction = np.ndarray(1)
    for i,s in enumerate(subs):
        # load stimuli and labels
        X, yn = load_labels_and_stim(s, sigma = sigma)
        # predict
        predictions = model.predict_proba(X)
        facePredictions = predictions[:,1]
        indices_face   = [i for i, x in enumerate(yn) if x == 1]
        indices_noface = [i for i, x in enumerate(yn) if x == 0]
        # test significance
        print(ttest_ind(facePredictions[indices_face], facePredictions[indices_noface],
            equal_var=True))
        allFacePrediction = np.concatenate([allFacePrediction,facePredictions[
            indices_face]],axis=0)
        allNoFacePrediction = np.concatenate([allNoFacePrediction,facePredictions[
            indices_noface]],axis=0)
    # test significance across subjects (not math. correct)
    print("across all subjects:")
    print(ttest_ind(allFacePrediction, allNoFacePrediction, equal_var=True))

def compare_prediction_similarities(model, sigma = 1):
    # determine subjects
    subs = glob("D:/paredolia/facepareidolia/stimuli/*")
    subs = [os.path.split(subs[i])[1] for i,s in enumerate(subs)]
    # loop over subjects
    predAllSub = []
    labelAllSub = []
    for i,s in enumerate(subs):
        print('subject ' + s)
        # load stimuli and labels
        X, yn = load_labels_and_stim(s, sigma = sigma)
        # predict
        predictions = model.predict_proba(X)
        facePredictions = predictions[:,1]
        predAllSub.append(facePredictions)
        labelAllSub.append(yn)
    return predAllSub, labelAllSub
```

## function for internal face template

```
def internal_template(model, gauss = 0, sigma = 1):
    from scipy.ndimage.filters import gaussian_filter
    # determine subjects
    subs = glob("D:/paredolia/facepareidolia/stimuli/*")
    subs = [os.path.split(subs[i])[1] for i,s in enumerate(subs)]
    fig, ax = plt.subplots(nrows=2,ncols=12, figsize = (20,5))
    # loop over subjects
    for j,s in enumerate(subs):
        # load stimuli and labels
        X, yn = load_labels_and_stim(s, sigma = sigma)
        # predict
        predictions = model.predict_proba(X)
        facePredictions = predictions[:,1]
        # find proportion of face/noFace for the subject
        proportion = sum(yn)/len(yn)
        # find the same quantil in the machine
        quantile = np.quantile(facePredictions, proportion)
        # convert to binary predictions, thresholded
        facePredictions_binary = np.where(facePredictions>=quantile, 1, 0)
        # calculate "internal templates"
        ## 1. subject template
        ind_face_sub   = [i for i, x in enumerate(yn) if x == 1]
        ind_noface_sub = [i for i, x in enumerate(yn) if x == 0]
        template_sub = np.mean(X[ind_face_sub,:,:,0],0) - np.mean(X[ind_noface_sub
            ,:,:,0],0)
        ## 2. machine template
        ind_face_mac   = [i for i, x in enumerate(facePredictions_binary) if x == 1]
        ind_noface_mac = [i for i, x in enumerate(facePredictions_binary) if x == 0]
        template_mac = np.mean(X[ind_face_mac,:,:,0],0) - np.mean(X[ind_noface_mac
            ,:,:,0],0)
        # plot 1. subject template 2. machine template
        im1 = ax[0][j].imshow(gaussian_filter(template_sub, sigma=gauss), cmap="gray");
            ax[0][j].axis('off');
        im2 = ax[1][j].imshow(gaussian_filter(template_mac, sigma=gauss), cmap="gray");
            ax[1][j].axis('off');
        ax[0][j].set_title(s);
    fig.subplots_adjust(wspace = 0.1, hspace = 0.1, left = 0.1, right = 0.9)
```

## run: preprocessing

```
# cat dog images: https://www.kaggle.com/chetankv/dogs-cats-images?
catdogimages = glob("D:/paredolia/facepareidolia/training/cats_dogs/*.jpg")
catdogTrain = np.ndarray((len(catdogimages),43,32,3))
for j, i in enumerate(catdogimages):
    try:
        catdogTrain[j,:,:,:] = load_and_preproc(i, crop = False)
    except:
        print("error␣in␣image␣#"+str(j))
        catdogTrain[j,:,:,:] = load_and_preproc(catdogimages[j-1])

catdogTrain = augment_image(catdogTrain)
catdogTrain = render_noisy(catdogTrain, p = 0.4)

# plot
fig, ax = plt.subplots(2,2, figsize = (6,8))
im1 = ax[0,0].imshow(catdogTrain[0,:,:,:].astype("int32")); ax[0,0].axis('off');
im2 = ax[1,0].imshow(catdogTrain[8000,:,:,:].astype("int32")); ax[1,0].axis('off')
im3 = ax[0,1].imshow(catdogTrain[16000,:,:,:].astype("int32")); ax[0,1].axis('off')
im4 = ax[1,1].imshow(catdogTrain[24000,:,:,:].astype("int32")); ax[1,1].axis('off')
fig.subplots_adjust(wspace = 0, hspace = 0.1)
#fig.suptitle('augmented non-face-class training images', fontsize=10)
plt.savefig("./paper/cats_noisy.png", dpi=250, transparent=True)

# celeb faces: https://www.kaggle.com/jessicali9530/celeba-dataset?
Nceleb = catdogTrain.shape[0]
celebimages = glob("D:/paredolia/facepareidolia/training/celebs/*.jpg")
celebTrain = np.ndarray((Nceleb,43,32,3))
for j, i in enumerate(celebimages):
    if j >= Nceleb: # catdogimages are the limiting factor
        break
    celebTrain[j,:,:,:] = load_and_preproc(i, crop = True)
plt.imshow(celebTrain[1,:,:,:].astype("int32"))
# for now, i do not augment the celeb database, because we have already enough pictures
celebTrain = render_noisy(celebTrain, p = 0.4)

# plot
```

```
fig, ax = plt.subplots(5,5, figsize=(10,13))
for i in range(5):
    for j in range(5):
        r = np.random.randint(Nceleb)
        im1 = ax[i,j].imshow(celebTrain[r,:,:,:].astype("int32")); ax[i,j].axis('off');
fig.subplots_adjust(wspace = 0.1, hspace = 0.1)
#fig.suptitle('example face-class training images', fontsize=20)
plt.savefig("./paper/celebs_noisy.png", dpi=250, transparent=True)
```

## train test split

```
y = np.repeat((0,1), Nceleb) # labels
cdc = np.concatenate( (catdogTrain, celebTrain), axis = 0 ) # concatenate celebs and
    cats / dogs
y_binary = to_categorical(y) # one-hot encode labels
X_train, X_test, y_train, y_test = train_test_split(cdc, y_binary, test_size = 0.25,
    random_state = 42)
```

## modeling

```
model = transfer_learning()
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy']) # adam
     #categorical_crossentropy
model.fit(X_train, y_train, epochs=20, batch_size=25)
```

## benchmarking

```
scores = model.evaluate(X_test, y_test, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

test_model(model)
```

## reveal internal templates

```
internal_template(model, gauss = 0)
plt.savefig("./paper/template_gauss0.png", dpi=250, transparent = True)
```

## plot model predictions

```
preds, labs = compare_prediction_similarities(model, sigma = 1)#

subs = glob("D:/paredolia/facepareidolia/stimuli/*")
subs = [os.path.split(subs[i])[1] for i,s in enumerate(subs)]

# to tidy df
d = []
sub = []
lab = []
for s, i in enumerate(preds):
    l = labs[s]
    for j,k in zip (i,l):
        d.append(j)
        sub.append(subs[s])
        lab.append(k)
di = {"subject":sub,"face_prediction":d,"label":lab}

df = pd.DataFrame(di)
df.head()

sns.stripplot(x="subject", y = "face_prediction", data = df, size=2)
sns.violinplot(x="subject", y = "face_prediction", data = df)
```

```
plt.axhline(y=0.5, color = "black")
plt.ylim(0,1)
plt.savefig("./paper/face_predictions.png", dpi=250, transparency = True)
```