

Programmierung für KI  
Projektausarbeitung - Haar Cascades

Onur Yilmaz, Alexander Fuchs, Johannes-Peter Kübert, Peter Spanke

15. Januar 2023

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung - Haar Cascades</b>	<b>3</b>
1.1	Funktionsweise: Haar-like Features . . . . .	3
1.2	Umsetzung in OpenCV . . . . .	5
<b>2</b>	<b>Implementation in Tkinter</b>	<b>5</b>
2.1	Standardaufbau einer Tkinter-App . . . . .	5
2.2	Haar-Cascade - Tkinter-App . . . . .	6
2.2.1	Packages . . . . .	6
2.2.2	Initialisierung und Geometrie . . . . .	7
2.2.3	Bild öffnen und Klassifizierer anwenden . . . . .	7
2.2.4	Slider und Button . . . . .	9
<b>3</b>	<b>Implementierung mit Streamlit</b>	<b>12</b>
3.1	Eingabequellen . . . . .	12
3.2	Optionen . . . . .	13
3.3	Code . . . . .	13
<b>4</b>	<b>Organisation</b>	<b>28</b>
<b>5</b>	<b>Continuous Integration</b>	<b>29</b>

## Einleitung

Im Rahmen der Veranstaltung - *Programmierung für KI*, haben wir das Thema **Haar Cascades**, als gemeinsam zu bearbeitendes Projekt erhalten. Haar Cascades beschreiben ein Verfahren bzw. einen Algorithmus, welches bestimmte Merkmale oder Muster in Bildern oder Videos erkennen kann.

Ziel dieses Projektes ist es nun ein solches Verfahren zu implementieren und eine passende graphische Benutzeroberfläche in Python bereitzustellen. Nach einer kurzen theoretischen Einführung in das Thema, teilt sich die Arbeit in zwei Teile auf. Der erste Teil beschäftigt sich mit der Erstellung einer graphischen Benutzeroberfläche mit Hilfe von **Tkinter** und der zweite Teil wiederum mit **streamlit**.

Schrittweise werden wir die einzelnen Python Codes hier darstellen und kennzeichnen, welche Codes genau, welchem Projektteilnehmer zuzuordnen sind. Für die Funktionalitäten der Programmcodes sind auf die Kommentare des beigefügten Programm Codes hingewiesen.

## 1 Einführung - Haar Cascades

Wie bereits oben erwähnt, sind Haar Cascades <sup>1</sup> ein Verfahren zur Erkennung von Objekten in Bildern und Videos. Genauer handelt es sich hierbei um einen Klassifikationsalgorithmus, welches Beispielbilder trainiert, die entweder das zu erkennende Muster bzw. Objekt enthalten oder nicht. Anschließend kann der Haar Cascade dann auf neue Bilder oder Videos angewendet werden und versuchen, das gesuchte Muster oder Objekt darin wieder zu erkennen (siehe [Ada19] und [Phi]).

### 1.1 Funktionsweise: Haar-like Features

Aus den Bildern der Eingangsdaten, werden für den Klassifikationsalgorithmus nun die richtigen Merkmale (im engl. *Feature*) extrahiert und trainiert. Hierbei benötigt man eine relativ große Trainingsmenge an positiven und negativen Graustufenbildern.<sup>2</sup>

Für die Merkmals-Extraktion werden nun rechteckige Bereiche (siehe **Abbildung 1**) an einem Bild abgetastet. Hierfür gibt es sogenannte Kantenmerkmale ((1) und (2)), Linienmerkmale (3) und vier-rechteckige Merkmale (4).

---

<sup>1</sup>Rapid object detection using a boosted cascade of simple features und Viola, Jones: Robust Real-time Object Detection, IJCV 2001, zurück zu führen auf *Paul Viola* und *Michael Jones*

<sup>2</sup>Genauer handelt es sich hierbei um normierte Graustufenbilder

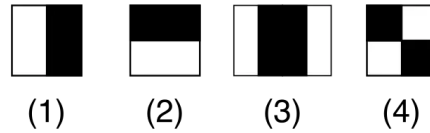


Abbildung 1: Haar-like Features

Um z.B. die Augen einer Person zu detektieren, stellen wir zunächst einmal fest, dass die Augen öfters dunkler sind, als andere Regionen im Gesicht. Hierzu verwenden wir dann das Feature (2) aus [Abbildung 1](#).



Abbildung 2: Anwendung eines Haar-like Features

Für alle Pixelwerte, z.B. normierte Helligkeitswerte, eines hellen bzw. dunklen Rechtecks wird der Durchschnitt gebildet und darauf hin die Werte der hellen und dunklen Rechtecke voneinander abgezogen [\[Neu\]](#).

Um zu ermitteln, welche Haar-like Features mit Form und Größe nun am aussagekräftigsten sind, gibt es den sogenannten **Adaboost**-Algorithmus<sup>3</sup>. Es wählt aus den vielen möglichen Merkmalen diejenigen aus, die die beste Unterscheidung zwischen einer Menge an Positiv- und Negativbeispielen von Mustern liefern und trainiert gleichzeitig den Klassifikator [\[Neu\]](#).

---

<sup>3</sup>AdaBoost (adapatives Boosting) ist ein Algorithmus für das Ensemble-Lernen, der für die Klassifikation oder Regression verwendet werden kann

## 1.2 Umsetzung in OpenCV

Für unseren Anwendungsfall ist es mit Hilfe von OpenCV nicht notwendig einen eigenen Haar Cascade Algorithmus anzulernen. Wir laden die XML-Datei mit den zugehörigen *Haar-like-Features*, die wir verwenden möchten. Diese XML-Dateien enthalten dann anschließend Informationen darüber, wie die Features aussehen und wie sie auf Bilder angewendet werden sollen.

## 2 Implementation in Tkinter

### 2.1 Standardaufbau einer Tkinter-App

[Onur]

```
1 import tkinter as tk
2
3 def druecke_knopf():
4     label.config(text="Button gedrueckt!")
5
6 root = tk.Tk()
7 root.geometry("1200x800")
8 root.title('Projekt Gruppe a2-2 - Thema: Bilderkennung Haar-
9           Cascades')
10 button = tk.Button(root, text="Drueck mich", command=
11                druecke_knopf)
12 button.pack()
13 label = tk.Label(root, text="Hallo Welt!")
14 label.pack()
15
16 root.mainloop()
```

Der obige Code erzeugt die recht simple Tkinter-Applikation mit einem Button und einem Label. Es sei hier auf [\[Mar\]](#) verwiesen.

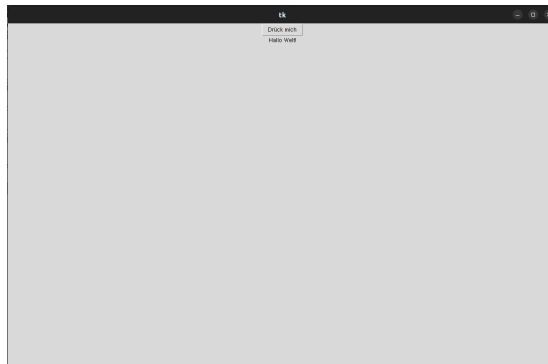


Abbildung 3: Einfache Tkinter App

Zunächst einmal müssen wir Tkinter-Bibliothek importieren, welche essentiell für unsere App ist.

Das Hauptfenster wird mit dem Befehl

```
1 root = tk.Tk()
```

erzeugt und mit

```
1 root.mainloop()
```

'beendet'.

Letzteres trifft jedoch in Wirklichkeit nicht ganz zu, da wir uns in einer Endlosschleife befinden, welche nur dafür zuständig ist, darauf zu warten, bis der Benutzer auf etwas klickt [Mar]. Wenn der Benutzer schließlich dann auf einen Button klickt, wird ein Ereignis bzw. eine Funktion durchgeführt und anschließend wieder in die Endlosschleife versetzt.

Darüber hinaus können wir die Geometrie und Titel der App anpassen (siehe Zeile 7 und 8).

In der obigen Anwendung haben wir einen Button und ein Label Modul aus der Tkinter-Bibliothek eingebaut. Die Referenzierung zu der Funktion erfolgt dann anschließend durch die *command-Option*. Die Methode *.pack()* von Tkinter kümmert sich um eine automatische Anordnung von Widgets, sprich Button, Labels, etc. im Fenster (root) zu verwalten.

## 2.2 Haar-Cascade - Tkinter-App

Um den Code möglichst lesbar zu gestalten, teilen wir im Folgenden, diesen in mehreren kleinen Blöcke auf.

### 2.2.1 Packages

[Alle]

```
1 import tkinter as tk
2 from tkinter import ttk
3 from tkinter import filedialog
4 import numpy as np
5 import cv2
6 from PIL import Image, ImageTk
7 import random
8 import os
9
10 from pki_a22_app.utils.file_loader import get_classifiers
11
12 path_haarcascade = "resources/haarcascades/haarcascade_"
13
```

```

14
15 classifier_list = get_classifiers()

```

### 2.2.2 Initialisierung und Geometrie

[Alle]

```

1 root = tk.Tk()
2 root.title('Projekt Gruppe a2-2 - Thema: Bilderkennung Haar-
   Cascades')
3
4
5 screen_width = root.winfo_screenwidth()
6 screen_height = root.winfo_screenheight()
7
8
9 if screen_width/screen_height < 1.8:
10     window_width = int(screen_width * 0.8)
11 else:
12     window_width = int(screen_height * (screen_width/2/
   screen_height)*0.8)
13     window_height = int(screen_height * 0.8)
14
15 img_max_width = int(window_width/2-75)
16 img_max_height = int(window_height-300)
17
18
19 center_x = int(screen_width/2 - window_width / 2)
20 center_y = int(screen_height/2 - window_height / 2)
21
22
23
24 root.geometry(f'{window_width}x{window_height}+{center_x}+{
   center_y}')
25 root.resizable(False, False)

```

### 2.2.3 Bild öffnen und Klassifizierer anwenden

[Peter]

```

1 def file_open():
2     global input_image
3     global output_image
4     filename = filedialog.askopenfilename(filetypes=(("jpg files",
   "*.jpg"), ("png files", "*.png")))
5     input_image = Image.open(filename)
6     width, height = input_image.size
7     aspect_ratio = width / height
8     if aspect_ratio > 1:
9         new_width = img_max_width
10        new_height = int(new_width / aspect_ratio)
11    else:
12        new_height = img_max_height
13        new_width = int(new_height * aspect_ratio)
14    input_image = input_image.resize((new_width, new_height), Image.
   ANTIALIAS)

```

```

15 input_image_tk = ImageTk.PhotoImage(input_image)
16 input_img_label.config(image=input_image_tk)
17 input_img_label.image = input_image_tk
18
19 output_image = Image.open(filename)
20 output_image = output_image.resize((new_width,new_height),
    Image.ANTIALIAS)
21 output_image_tk = ImageTk.PhotoImage(output_image)
22 output_img_label.config(image=output_image_tk)
23 output_img_label.image = output_image_tk

```

### [Peter und Alexander]

```

1 def img_change(classifier):
2     global input_image
3     global output_image
4     cascade = cv2.CascadeClassifier(path_haarcascade + classifier +
    ".xml")
5     output_image_cv = np.array(output_image.convert('RGB'))
6     output_image_cv_gray = cv2.cvtColor(output_image_cv, cv2.
    COLOR_BGR2GRAY)
7     cascade_results = cascade.detectMultiScale(output_image_cv_gray
    , scaleFactor=s1.get_val(), minNeighbors = s2.get_val(),
    minSize=(s3.get_val(), s3.get_val()))
8     iterations = 0
9
10    if len(cascade_results) > 0:
11        color = (random.randint(0,255),random.randint(0,255),random.
    randint(0,255))
12        for (x,y,w,h) in cascade_results:
13            cv2.rectangle(output_image_cv,(x,y),(x+w,y+h),(color),2)
14            roi_gray = output_image_cv_gray[y:y+h, x:x+w]
15            roi_color = output_image_cv[y:y+h, x:x+w]
16            output_image = Image.fromarray(output_image_cv)
17            output_img_label.image.paste(output_image)
18    else:
19        flag = False
20        for i1 in np.arange (1.5, 0.9, -0.1):
21            for i2 in range (6, 2, -1):
22                for i3 in range (50, 9, -10):
23                    s1.s.set(i1)
24                    s2.s.set(i2)
25                    s3.s.set(i3)
26                    iterations = iterations + 1
27                    cascade_results = cascade.detectMultiScale(
    output_image_cv_gray, scaleFactor=s1.get_val(), minNeighbors
    = s2.get_val(), minSize=(s3.get_val(), s3.get_val()))
28                    if len(cascade_results) > 0:
29                        color = (random.randint(0,255),random.randint(0,255),
    random.randint(0,255))
30                        for (x,y,w,h) in cascade_results:
31                            cv2.rectangle(output_image_cv,(x,y),(x+w,y+h),(
    color),2)
32                            roi_gray = output_image_cv_gray[y:y+h, x:x+w]
33                            roi_color = output_image_cv[y:y+h, x:x+w]
34                            output_image = Image.fromarray(output_image_cv)
35                            output_img_label.image.paste(output_image)

```



```

36         flag = True
37         break
38     if flag: break
39     if flag: break
40     if len(cascade_results) == 0:
41         print(f"Kein Ergebnis nach {iterations} Iterationen")

```

[Peter]

```

1 def output_image_restart():
2     global input_image
3     global output_image
4     output_image = input_image
5     output_img_label.image.paste(output_image)

```

[Peter]

## 2.2.4 Slider und Button

```

1 class slider:
2     def __init__(self, name, x_pos=0, y_pos=0, scale_from=0, scale_to
3         =100, typ=int):
4         self.name = name
5         self.x_pos = x_pos
6         self.y_pos = y_pos
7         self.scale_from = scale_from
8         self.scale_to = scale_to
9         if typ==int:
10             self.val = tk.IntVar()
11         else:
12             self.val = tk.DoubleVar()
13
14         self.lbl = ttk.Label(root, text=name)
15         self.lbl.place(x=self.x_pos, y=self.y_pos)
16
17         self.s = ttk.Scale(root, from_=self.scale_from, to=self.
18             scale_to, orient="horizontal", command=self.slider_change,
19             variable=self.val)
20         self.s.place(x=self.x_pos+100, y=self.y_pos)
21
22         self.v = ttk.Label(root, text=f"{self.get_val():.1f}")
23         self.v.place(x=self.x_pos+210, y=self.y_pos)
24     def get_val(self):
25         return self.val.get()
26     def slider_change(self, event):
27         self.v.configure(text=f"{self.get_val():.1f}")

```

[Alexander]

```

1 def blur_rectangle(classifier):
2     global input_image
3     global output_image
4     cascade = cv2.CascadeClassifier(path_haarcascade + classifier +
5         ".xml")
6     output_image_cv = np.array(output_image.convert('RGB'))

```

```

6   output_image_cv_gray = cv2.cvtColor(output_image_cv, cv2.
   COLOR_BGR2GRAY)
7   cascade_results = cascade.detectMultiScale(output_image_cv_gray
   , scaleFactor=s1.get_val(), minNeighbors = s2.get_val(),
   minSize=(s3.get_val(), s3.get_val()))
8   if len(cascade_results) > 0:
9       for (x,y,w,h) in cascade_results:
10          face = output_image_cv[y:y+h, x:x+w]
11          face = cv2.GaussianBlur(face, (23, 23), 30)
12          output_image_cv[y:y+h, x:x+w] = face
13
14          output_image = Image.fromarray(output_image_cv)
15          output_img_label.image.paste(output_image)
16
17 def save_jpg():
18     file_path = filedialog.asksaveasfilename(initialfile="
   output_image", filetypes=(("jpg files", "*.jpg"),("png files"
   , "*.png")), defaultextension=".jpeg")
19     if file_path:
20         output_image.save(file_path)

```

[Peter]

```

1 tk.Button(root, text="Bild aus Datei oeffnen", command=file_open)
   .place(x=window_width/2-window_width/4,y=150)
2 tk.Button(root, text="Classifier anwenden", command=lambda:
   img_change(dropdown.get())).place(x=window_width/2+
   window_width/4,y=150)
3 tk.Button(root, text=="=>", command=output_image_restart).place(x
   =window_width/2-12.5,y=window_height/2)
4 tk.Button(root, text="Weichzeichnen",command=lambda:
   blur_rectangle(dropdown.get())).place(x=window_width/2+
   window_width/4+122,y=150)
5 tk.Button(root, text="Bild speichern",command=save_jpg).place(x=
   window_width/2+window_width/4+220,y=150)
6
7 dropdown = tk.StringVar(root)
8 dropdown.set(classifier_list[2])
9 dropdown_label = tk.OptionMenu(root, dropdown, *classifier_list)
10 dropdown_label.place(x=window_width/2-75,y=20)
11
12
13 xpos_slider_window = window_width/2 -75
14 ypos_slider_window = 60
15 s1 = slider("ScaleFactor",xpos_slider_window,ypos_slider_window
   ,1.01,1.5,float)
16 s1.s.set(1.1)
17 s2 = slider("MinNeighbors",xpos_slider_window,ypos_slider_window
   +30,3,6)
18 s2.s.set(4)
19 s3 = slider("minSize",xpos_slider_window,ypos_slider_window
   +60,10,50)
20 s3.s.set(30)
21
22
23 input_img_label = ttk.Label(root)
24 input_img_label.place(x=25,y=200)

```

```
25
26 output_img_label = ttk.Label(root)
27 output_img_label.place(x=window_width/2+50,y=200)
28
29 fh_logo = Image.open("resources/images/Logo.jpg")
30 fh_logo = fh_logo.resize((300,100), Image.ANTIALIAS)
31 fh_logo_tk = ImageTk.PhotoImage(fh_logo)
32 ttk.Label(root, image=fh_logo_tk).place(x=0,y=0)
33
34 root.mainloop()
```

## 3 Implementierung mit Streamlit

[Johannes-Peter]

Eine weitere Implementierung des Haar Cascade Klassifizierers wurde mit dem Dashboard Framework **Streamlit** als **Web-Anwendung** umgesetzt. Das Dashboard enthält zum einen einen Bereich mit unterschiedlichen Parametern für den Haar Cascade Klassifizierer, und zum anderen einen Bereich, in dem verschieden Bild-Quellen ausgewählt werden können. Dazu gehört neben eigenen oder vorinstallierten Bildern auch die Möglichkeit, ein Video (Siehe [Abbildung 4](#)) oder das eigene Webcam-Bild zu verwenden. Für die Übertragung von Video und Webcam Bildern wurde die **WebRTC-Erweiterung** für Streamlit verwendet .

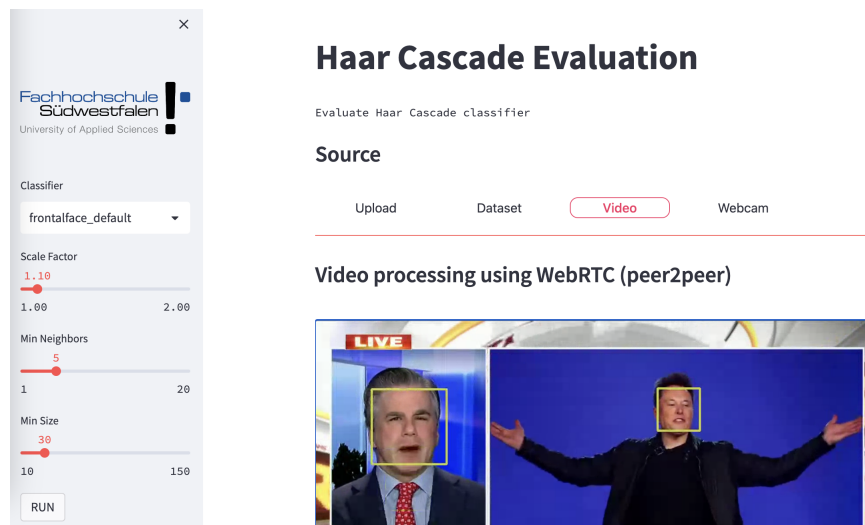


Abbildung 4: Streamlit Dashboard mit Haar Cascade Klassifizierer

Das Dashboard kann auch als **Demo-Anwendung** auf der Streamlit-Cloud ausprobiert werden.

### 3.1 Eingabequellen

Die Anwendung unterstützt diese bereits angesprochenen Eingabemöglichkeiten:

#### Upload

Hier kann ein einzelnes Bild hochgeladen und verwendet werden

#### Dataset

Unter diesem Punkt können mehrere vorkonfigurierte Bilder auf einmal verwendet werden

## Video

Bei dieser Option wird ein vorkonfiguriertes Video abgespielt, auf dem der ausgewählte Klassifizierer angewandt wird

## Webcam

Hier besteht die Möglichkeit, den Klassifizierer auf die eigene Webcam-Bilder anzuwenden

## 3.2 Optionen

Die verwendeten Optionen zur Konfiguration des Haar Cascade Klassifizierers **Scale Factor**, **Min Neighbors** und **Min Size** wurden bereits in den vorherigen Abschnitten im Rahmen der Tkinter Anwendung beschrieben und können hier nachgelesen werden: [\[han\]](#)

## 3.3 Code

Der Programmcode für die Implementierung des Streamlit-Dashboards ist auf die folgenden Quelldateien verteilt:

```
1 # Hauptanwendung und Einstiegspunkt
2 app.py
3 # Funktion zur Ausfuehrung des eigentlichen Klassifizierers
4 pki_a22_app/haarcascades/haarcascades.py
5 # Interface fuer die unterschiedlichen Eingabequellen
6 pki_a22_app/dashboard/sources.py
7 # Eingabequelle mit einem einzelnen hochladbarem Bild
8 pki_a22_app/dashboard/source_upload.py
9 # Eingabequelle mit Auswahl eines Datensatzes mit mehreren
  Bildern
10 pki_a22_app/dashboard/source_dataset.py
11 # Eingabequelle zum Abspielen eines voreingestellten Videos
12 pki_a22_app/dashboard/source_video.py
13 # Eingabequelle bei der die eigene Webcam verwendet wird
14 pki_a22_app/dashboard/source_webcam.py
15 # Utility Modul, das vor allem Funktionen fuer Datei-Operationen
  enthaelt
16 pki_a22_app/utils/file_loader.py
```

Im folgenden Abschnitt wird der Programmcode im Detail aufgelistet. Die Dokumentation zu den einzelnen Abschnitten befindet sich im Code selbst:

```
1 """Start a streamlit dashboard showing some haarcascade
  operations with Opencv"""
2
3 # Core Pkgs
4
5 import extra_streamlit_components as stx
6 import streamlit as st
7 from pki_a22_app.dashboard.source_dataset import DatasetSource
8 from pki_a22_app.dashboard.source_upload import UploadSource
9 from pki_a22_app.dashboard.source_video import VideoSource
```

```

10 from pki_a22_app.dashboard.source_webcam import WebcamSource
11
12 from pki_a22_app.utils.file_loader import (get_classifiers)
13
14
15 def main():
16     """
17     Main function that initiates the Streamlit dashboard
18     """
19
20     st.title("Haar Cascade Evaluation")
21     st.text("Evaluate Haar Cascade classifier")
22
23     # Configure the sources section
24     st.subheader("Source")
25     chosen_id = stx.tab_bar(data=[
26         stx.TabBarItemData(id=1, title="Upload", description=""),
27         stx.TabBarItemData(id=2, title="Dataset", description="")
28     ],
29     ,
30     stx.TabBarItemData(id=3, title="Video", description=""),
31     stx.TabBarItemData(id=4, title="Webcam", description=""),
32     ], default=1)
33
34     st.sidebar.image("resources/images/logo-fh-swf-300x93.png")
35     st.sidebar.markdown('#')
36
37     # Configure the algorithm controls
38     classifiers: list = get_classifiers()
39     classifier_id = st.sidebar.selectbox("Classifier",
40     classifiers, index=5)
41     scale_factor = st.sidebar.slider('Scale Factor', 1.01, 2.0,
42     1.1, 0.01)
43     min_neighbors = st.sidebar.slider('Min Neighbors', 1, 20, 5,
44     1)
45     min_size = st.sidebar.slider('Min Size', 10, 150, 30, 5)
46
47     show_results = st.sidebar.button("RUN")
48
49     # Define different sources that render the input and output
50     accordingly
51     sources = {
52         "1": UploadSource(),
53         "2": DatasetSource(),
54         "3": VideoSource(),
55         "4": WebcamSource()
56     }
57
58     # Execute selected source
59     sources[chosen_id].load_source(
60         classifier_id, scale_factor, min_neighbors, min_size,
61         show_results)
62
63 if __name__ == '__main__':
64     main()

```

Listing 1: app.py

```

1 """This module contains haar cascade core functionality"""
2
3 import cv2
4 from cv2 import Mat
5 import numpy.typing as npt
6
7
8 def detect_objects(np_input_image: npt.ArrayLike, classifier_id:
9     str, scale_factor: float = 1.1, min_neighbors: int = 4,
10     min_size: int = 10) -> Mat:
11     """
12     Detects objects in an image using Haar Cascades classifiers.
13     It can use a preset of trained classifiers in resources/
14     haarcascades/*.
15     The found objects get highlighted by a rectangle around it.
16
17     The parameter description was taken from:
18     https://hackaday.io/project/12384-autofan-automated-control-
19     of-air-flow/log/41956-face-detection-using-a-haar-cascade-
20     classifier
21
22     Parameters
23     -----
24     np_input_image : npt.ArrayLike
25         The input image
26     classifier_id : str
27         the classifier name. E.g. when you set it to "
28         frontalface_default", the configuration file
29         "haarcascade_frontalface_default.xml" will be loaded
30     scale_factor : float, optional, by default 1.1
31         determines the factor by which the detection window of
32         the classifier is scaled down per detection pass. A factor of
33         1.1
34         corresponds to an increase of 10%. Hence, increasing the
35         scale factor increases performance, as the number of
36         detection
37         passes is reduced. However, as a consequence the
38         reliability by which a face is detected is reduced. See:
39     min_neighbors : int, optional, by default 4
40         determines the minimum number of neighboring facial
41         features that need to be present to indicate the detection of
42         a face by the
43         classifier. Decreasing the factor increases the amount of
44         false positive detections. Increasing the factor might lead
45         to missing
46         faces in the image. The argument seems to have no
47         influence on the performance of the algorithm. See:
48     min_size: int
49         determines the minimum size of the detection window in
50         pixels. Increasing the minimum detection window increases
51         performance.
52         However, smaller faces are going to be missed then. In
53         the scope of this project however a relatively big detection
54         window can
55         be used, as the user is sitting directly in front of the
56         camera.

```

```

36
37     Returns
38     -----
39     cv2.Mat
40         The image in Mat representation
41     list
42         A list of found objects in the image
43     """
44
45     classifier = cv2.CascadeClassifier(
46         f"resources/haarcascades/haarcascade_{classifier_id}.xml"
47     )
48
49     img = cv2.cvtColor(np_input_image, 1)
50     gray = cv2.cvtColor(np_input_image, cv2.COLOR_BGR2GRAY)
51     objects = classifier.detectMultiScale(
52         gray, scaleFactor=scale_factor, minNeighbors=
53         min_neighbors, minSize=(min_size, min_size))
54     # pylint: disable=invalid-name
55     for (x, y, w, h) in objects:
56         cv2.rectangle(img, (x, y), (x + w, y + h), (72, 209, 204)
57             , 4)
58     return img, objects

```

Listing 2: haarcascades/haarcascades.py



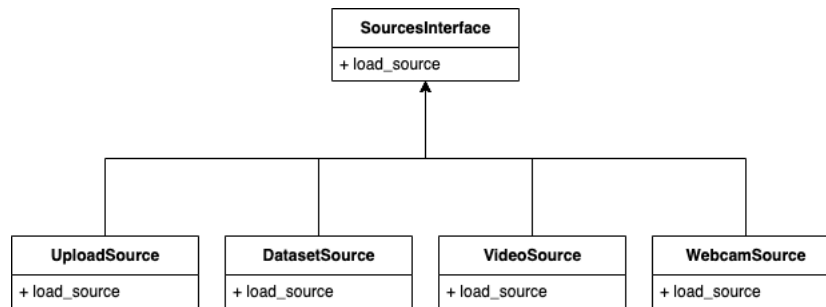


Abbildung 5: UML Diagramm für unterschiedliche Source-Typen

```

1 """
2 Contains an interface definition to use different source-types
3 in the Streamlit dashboard
4 """
5 import abc
6
7
8 class SourcesInterface(metaclass=abc.ABCMeta):
9     """
10     Interface for handling different input source types
11
12     Methods
13     -----
14     load_source(classifier_id: str, scale_factor: int,
15                 min_neighbors: int, show_results: bool = False)
16         Load the sources and outputs with respect to the given
17         parameters
18     """
19
20     @classmethod
21     def __subclasshook__(cls, subclass):
22         """
23         Ensures that needed methods are implemented in your
24         derivatives
25
26         Parameters
27         -----
28         subclass : Any
29             subclass that was created
30
31         Returns
32         -----
33         Bool
34             True if it is a valid subclass
35         """
36         return (hasattr(subclass, 'load_source') and
37                 callable(subclass.load_source) or
38                 NotImplemented)
39
40     @abc.abstractmethod

```

```

39 def load_source(self, classifier_id: str, scale_factor: int,
40 min_neighbors: int, min_size: int, show_results: bool = False
41 ):
42     """
43     This method handles the display of different Streamlit-UI
44     elements. Implement these in your
45     concrete derivatives
46
47     The parameter description was taken from:
48     https://hackaday.io/project/12384-autofan-automated-
49     control-of-air-flow/log/41956-face-detection-using-a-haar-
50     cascade-classifier
51
52     Parameters
53     -----
54     classifier_id : str
55         the classifier name. E.g. when you set it to "
56         frontalface_default", the configuration file
57         "haarcascade_frontalface_default.xml" will be loaded
58     scale_factor : float, optional, by default 1.1
59         determines the factor by which the detection window
60         of the classifier is scaled down per detection pass. A factor
61         of 1.1
62         corresponds to an increase of 10%. Hence, increasing
63         the scale factor increases performance, as the number of
64         detection
65         passes is reduced. However, as a consequence the
66         reliability by which a face is detected is reduced. See:
67         min_neighbors : int, optional, by default 4
68         determines the minimum number of neighboring facial
69         features that need to be present to indicate the detection of
70         a face by the
71         classifier. Decreasing the factor increases the
72         amount of false positive detections. Increasing the factor
73         might lead to missing
74         faces in the image. The argument seems to have no
75         influence on the performance of the algorithm. See:
76         min_size: int
77         determines the minimum size of the detection window
78         in pixels. Increasing the minimum detection window increases
79         performance.
80         However, smaller faces are going to be missed then.
81         In the scope of this project however a relatively big
82         detection window can
83         be used, as the user is sitting directly in front of
84         the camera.
85         show_results : bool, optional
86         If true we want to display the original images AND
87         the results, by default False
88
89     Raises
90     -----
91     NotImplementedError
92         Error that gets thrown if you do not implement this
93         function
94     """

```

72 `raise NotImplementedError`

Listing 3: dashboard/sources.py

```
1 """Implementation of a source-type to upload an image and apply
   the Haar Cascade classifier on it"""
2 # Core Pkgs
3
4 import numpy as np
5 import streamlit as st
6 from PIL import Image
7
8 from pki_a22_app.dashboard.sources import SourcesInterface
9 from pki_a22_app.haarcascades.haarcascades import detect_objects
10
11
12 class UploadSource(SourcesInterface):
13     def load_source(self, classifier_id: str, scale_factor: int,
14                    min_neighbors: int, min_size: int, show_results: bool = False
15                    ):
16         """
17         This method creates a form to upload an image that can be
18         processed by the haar cascade classifier afterwards.
19
20         The parameter description was taken from:
21         https://hackaday.io/project/12384-autofan-automated-
22         control-of-air-flow/log/41956-face-detection-using-a-haar-
23         cascade-classifier
24
25         Parameters
26         -----
27         classifier_id : str
28             the classifier name. E.g. when you set it to "
29             frontalface_default", the configuration file
30             "haarcascade_frontalface_default.xml" will be loaded
31             scale_factor : float, optional, by default 1.1
32                 determines the factor by which the detection window
33                 of the classifier is scaled down per detection pass. A factor
34                 of 1.1
35                 corresponds to an increase of 10%. Hence, increasing
36                 the scale factor increases performance, as the number of
37                 detection
38                 passes is reduced. However, as a consequence the
39                 reliability by which a face is detected is reduced. See:
40                 min_neighbors : int, optional, by default 4
41                     determines the minimum number of neighboring facial
42                     features that need to be present to indicate the detection of
43                     a face by the
44                     classifier. Decreasing the factor increases the
45                     amount of false positive detections. Increasing the factor
46                     might lead to missing
47                     faces in the image. The argument seems to have no
48                     influence on the performance of the algorithm. See:
49                     min_size: int
50                         determines the minimum size of the detection window
51                         in pixels. Increasing the minimum detection window increases
52                         performance.
```

```

35         However, smaller faces are going to be missed then.
        In the scope of this project however a relatively big
        detection window can
36         be used, as the user is sitting directly in front of
        the camera.
37         show_results : bool, optional
38         If true we want to display the original images AND
        the results, by default False
39         """
40         st.subheader("Upload an image")
41
42         # Show the upload form
43         image_file = st.file_uploader(
44             "Upload Image", type=["jpg", "png", "jpeg"])
45         our_image = None
46
47         # If a image was uploaded, display it
48         if image_file is not None:
49             our_image = Image.open(image_file)
50             our_np_image = np.array(our_image.convert("RGB"))
51             st.text("Original Image")
52             st.image(our_image)
53             image_location = st.empty()
54
55         # Show processed image
56         if show_results and classifier_id:
57             # Actual object detection
58             result_img, result_faces = detect_objects(
59                 our_np_image, classifier_id, scale_factor,
        min_neighbors, min_size)
60             image_location.image(result_img)
61             st.text("Resulting Image")
62             st.success(f"Found {len(result_faces)} Objects")

```

Listing 4: dashboard/source upload.py

```

1  """Implementation of a source-type that show multiple images at
    once"""
2
3  # Core Pkgs
4  from itertools import cycle
5
6  import numpy as np
7  import streamlit as st
8  from PIL import Image
9
10 from pki_a22_app.dashboard.sources import SourcesInterface
11 from pki_a22_app.haarcascades.haarcascades import detect_objects
12 from pki_a22_app.utils.file_loader import get_datasets,
    get_images_of_dataset
13
14
15 class DatasetSource(SourcesInterface):
16     """
17     Implementation for handling multiple images, also called
    dataset here
18
19     Methods

```

```

20 -----
21 load_source(classifier_id: str, scale_factor: int,
22             min_neighbors: int, show_results: bool = False)
23     """
24     Load multiple images and display results
25     """
26
27     def load_source(self, classifier_id: str, scale_factor: int,
28                   min_neighbors: int, min_size: int, show_results: bool = False
29                   ):
30         """
31         This method shows a selectbox to select an existing
32         dataset. A dataset contains multiple images. The datasets are
33         located
34         at resource/datasets/. The Haar Cascade classifier will
35         then be applied to all images of the dataset.
36
37         The parameter description was taken from:
38         https://hackaday.io/project/12384-autofan-automated-
39         control-of-air-flow/log/41956-face-detection-using-a-haar-
40         cascade-classifier
41
42         Parameters
43         -----
44         classifier_id : str
45             the classifier name. E.g. when you set it to "
46             frontalface_default", the configuration file
47             "haarcascade_frontalface_default.xml" will be loaded
48             scale_factor : float, optional, by default 1.1
49             determines the factor by which the detection window
50             of the classifier is scaled down per detection pass. A factor
51             of 1.1
52             corresponds to an increase of 10%. Hence, increasing
53             the scale factor increases performance, as the number of
54             detection
55             passes is reduced. However, as a consequence the
56             reliability by which a face is detected is reduced. See:
57             min_neighbors : int, optional, by default 4
58             determines the minimum number of neighboring facial
59             features that need to be present to indicate the detection of
60             a face by the
61             classifier. Decreasing the factor increases the
62             amount of false positive detections. Increasing the factor
63             might lead to missing
64             faces in the image. The argument seems to have no
65             influence on the performance of the algorithm. See:
66             min_size: int
67             determines the minimum size of the detection window
68             in pixels. Increasing the minimum detection window increases
69             performance.
70             However, smaller faces are going to be missed then.
71             In the scope of this project however a relatively big
72             detection window can
73             be used, as the user is sitting directly in front of
74             the camera.
75             show_results : bool, optional
76             If true we want to display the original images AND
77             the results, by default False

```

```

52     """
53     st.subheader("Load an existing dataset")
54
55     # Setup dataset selector
56     datasets = get_datasets()
57     datasets.insert(0, "None")
58     dataset_selection = st.selectbox("Dataset", datasets)
59
60     img_width = 300
61     img_columns = 2
62
63     # When a dataset was selected start presenting it
64     if dataset_selection and dataset_selection != "None":
65         images_in_ds = get_images_of_dataset(
66             dataset_selection)
67
68         # Show images in max n columns
69         cols = cycle(st.columns(img_columns))
70         for idx, filtered_image in enumerate(images_in_ds):
71             next(cols).image(str(filtered_image), width=
72                 img_width)
73
74         # We were requested to show the results
75         if show_results and dataset_selection:
76             images_in_ds = get_images_of_dataset(
77                 dataset_selection)
78             st.write("### Results:")
79             cols = cycle(st.columns(img_columns))
80             for idx, filtered_image in enumerate(images_in_ds
81 ):
82                 ds_image = Image.open(filtered_image)
83                 ds_np_image = np.array(ds_image.convert("RGB"
84 ))
85
86                 # Actual object detection
87                 ds_res_image, ds_result_faces =
88                 detect_objects(ds_np_image, classifier_id, scale_factor,
89                     min_neighbors, min_size)
90                 col = next(cols)
91                 col.image(ds_res_image, width=img_width,
92                     caption=f"{len(ds_result_faces)} objects")

```

Listing 5: dashboard/source dataset.py

```

1  """Implementation of a source-type to select a video and apply
2     the Haar Cascade classifier on it"""
3
4  # Core Pkgs
5  import av
6  import streamlit as st
7  from streamlit_webrtc import MediaPlayerFactory, WebRtcMode,
8     webrtc_streamer
9
10 from pki_a22_app.dashboard.sources import SourcesInterface
11 from pki_a22_app.haarcascades.haarcascades import detect_objects
12 from aiortc.contrib.media import MediaPlayer

```

```

12
13
14 class VideoSource(SourcesInterface):
15     def load_source(self, classifier_id: str, scale_factor: int,
16                     min_neighbors: int, min_size: int, show_results: bool = False
17                     ):
18         """
19         This function shows a video player where the haar cascade
20         detection is applied on single frames.
21         The player uses WebRTC to stream the video.
22
23         The parameter description was taken from:
24         https://hackaday.io/project/12384-autofan-automated-
25         control-of-air-flow/log/41956-face-detection-using-a-haar-
26         cascade-classifier
27
28         Parameters
29         -----
30         classifier_id : str
31             the classifier name. E.g. when you set it to "
32             frontalface_default", the configuration file
33             "haarcascade_frontalface_default.xml" will be loaded
34             scale_factor : float, optional, by default 1.1
35             determines the factor by which the detection window
36             of the classifier is scaled down per detection pass. A factor
37             of 1.1
38             corresponds to an increase of 10%. Hence, increasing
39             the scale factor increases performance, as the number of
40             detection
41             passes is reduced. However, as a consequence the
42             reliability by which a face is detected is reduced. See:
43             min_neighbors : int, optional, by default 4
44             determines the minimum number of neighboring facial
45             features that need to be present to indicate the detection of
46             a face by the
47             classifier. Decreasing the factor increases the
48             amount of false positive detections. Increasing the factor
49             might lead to missing
50             faces in the image. The argument seems to have no
51             influence on the performance of the algorithm. See:
52             min_size: int
53             determines the minimum size of the detection window
54             in pixels. Increasing the minimum detection window increases
55             performance.
56             However, smaller faces are going to be missed then.
57             In the scope of this project however a relatively big
58             detection window can
59             be used, as the user is sitting directly in front of
60             the camera.
61             show_results : bool, optional
62             If true we want to display the original images AND
63             the results, by default False
64         """
65         st.subheader("Video processing using WebRTC (peer2peer)")
66
67         # Callback function that takes an image frame that can

```

```

47     # be processed
48     def callback(frame):
49         # convert frame to ndarray
50         img = frame.to_ndarray(format="bgr24")
51
52         # actual object detection
53         result_img, result_faces = detect_objects(
54             img, classifier_id, scale_factor, min_neighbors,
min_size)
55
56         return av.VideoFrame.from_ndarray(result_img, format=
"bgr24")
57
58         rtc_configuration = { # Add this config
59             "iceServers": [{"urls": ["stun:stun.l.google.com
:19302"]}]}
60         }
61
62         def create_player():
63             return MediaPlayer("resources/videos/musk.mp4")
64
65         # This will show the video-UI elements in Streamlit
66         # Each frame gets sent to the callback method
67         webrtc_streamer(key="videoexample",
68                         video_frame_callback=callback,
69                         rtc_configuration=rtc_configuration,
70                         mode=WebRtcMode.RECVONLY,
71                         player_factory=create_player,
72                         media_stream_constraints={
73                             "video": True
74                         }
75                     )

```

Listing 6: dashboard/source video.py

```

1  """Implementation of a source-type to display a webcam stream and
2     apply the Haar Cascade classifier on it"""
3
4  # Core Pkgs
5  import av
6  import streamlit as st
7  from streamlit_webrtc import webrtc_streamer
8
9  from pki_a22_app.dashboard.sources import SourcesInterface
10 from pki_a22_app.haarcascades.haarcascades import detect_objects
11
12
13 class WebcamSource(SourcesInterface):
14     def load_source(self, classifier_id: str, scale_factor: int,
15                   min_neighbors: int,
16                   min_size: int, show_results: bool = False) ->
17         None:
18             """
19             This method shows an UI element to display the stream of
20             a webcam. The Haar Cascade classifier will then be applied to
21             the images
22             of the webcam.

```



```

19
20     The parameter description was taken from:
21     https://hackaday.io/project/12384-autofan-automated-
control-of-air-flow/log/41956-face-detection-using-a-haar-
cascade-classifier
22
23
24     Parameters
25     -----
26     classifier_id : str
27         the classifier name. E.g. when you set it to "
frontalface_default", the configuration file
28         "haarcascade_frontalface_default.xml" will be loaded
29     scale_factor : float, optional, by default 1.1
30         determines the factor by which the detection window
of the classifier is scaled down per detection pass. A factor
of 1.1
31         corresponds to an increase of 10%. Hence, increasing
the scale factor increases performance, as the number of
detection
32         passes is reduced. However, as a consequence the
reliability by which a face is detected is reduced. See:
33     min_neighbors : int, optional, by default 4
34         determines the minimum number of neighboring facial
features that need to be present to indicate the detection of
a face by the
35         classifier. Decreasing the factor increases the
amount of false positive detections. Increasing the factor
might lead to missing
36         faces in the image. The argument seems to have no
influence on the performance of the algorithm. See:
37     min_size: int
38         determines the minimum size of the detection window
in pixels. Increasing the minimum detection window increases
performance.
39         However, smaller faces are going to be missed then.
In the scope of this project however a relatively big
detection window can
40         be used, as the user is sitting directly in front of
the camera.
41     show_results : bool, optional
42         If true we want to display the original images AND
the results, by default False
43     """
44     st.subheader("Webcam processing using WebRTC (peer2peer)"
)
45
46     def callback(frame):
47         # convert frame to ndarray
48         img = frame.to_ndarray(format="bgr24")
49
50         # actual object detection
51         result_img, result_faces = detect_objects(
52             img, classifier_id, scale_factor, min_neighbors,
min_size)
53
54         return av.VideoFrame.from_ndarray(result_img, format=

```

```

55 "bgr24")
56
57     # This will show the webcam-UI elements in streamlit
58     # Each frame gets sent to the callback method
59     rtc_configuration = { # Add this config
60         "iceServers": [{ "urls": ["stun:stun.l.google.com
61         :19302"]}]}
62     }
63     webrtc_streamer(key="example", video_frame_callback=
64     callback,
65                     rtc_configuration=rtc_configuration)

```

Listing 7: dashboard/source webcam.py

```

1  """This modules contains helper methods to get information from
2     local file system"""
3  import os
4  import pathlib
5  from typing import List
6
7  def get_classifiers() -> List[str]:
8      """
9      Reads all files in /resources/haarcascades/ and returns only
10     the actual classifier
11     name without "haarcascade_" and ".xml". "E.g.
12     haarcascade_frontalface_default.xml"
13     gets to "haarcascade_frontalface_default".
14
15     Returns
16     -----
17     List[str]
18         List of found classifiers
19     """
20     classifier_files = os.listdir(os.getcwd() + "/resources/
21     haarcascades/")
22     classifiers = [item.replace("haarcascade_", "")
23     for item in classifier_files]
24     classifiers = [item.replace(".xml", "") for item in
25     classifiers]
26     classifiers.sort()
27     return classifiers
28
29  def get_datasets() -> List[str]:
30      """
31      Returns the folders found under /resources/datasets/
32
33      Returns
34      -----
35      List[str]
36         List of available dataset
37      """
38     datasets = os.listdir(os.getcwd() + "/resources/datasets/")
39     datasets = [item for item in datasets if not item.startswith(
40     ".")]
41     datasets.sort()
42     return datasets

```

```

39
40
41 def get_images_of_dataset(dataset_name: str) -> List[str]:
42     """
43     Returns the images contained in a dataset
44
45     Parameters
46     -----
47     dataset_name : str
48         Name of the dataset (folder)
49
50     Returns
51     -----
52     List[str]
53         List of found images
54     """
55     extensions = ['.jpg', '.png', '.jpeg']
56     img_list = [x for x in pathlib.Path(f"resources/datasets/{
dataset_name}").iterdir() if x.suffix.lower() in extensions]
57     img_list = [str(item) for item in img_list]
58     return list(img_list)

```

Listing 8: utils/file loader.py



## 5 Continuous Integration

[Johannes-Peter]

Um gemeinsam am Programmcode arbeiten zu können, haben wir uns ein Git-basiertes Codeverwaltungs-Tool entschieden. Hierfür haben wir auf GitHub ein eigenes **Git-Repository** (**Abbildung 7**) eingerichtet.

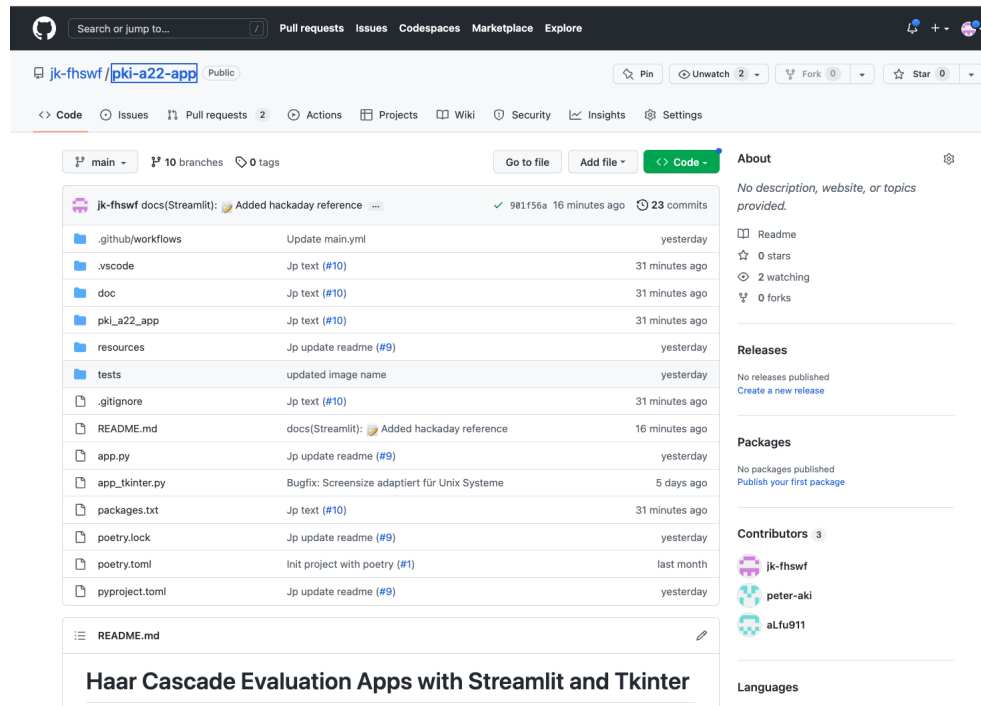


Abbildung 7: Git-Repository auf GitHub

Zur Qualitätssicherung haben wir uns dafür entschieden, für den main-Branch Pull-Requests zu verwenden und die Erfolgreiche Ausführung eines GitHub-Actions- Workflows für Unit-Tests vorauszusetzen (**Abbildung 8**). Eine Beispiel-  
lauf des Workflows ist in **Abbildung 9** zu sehen.

General

Access

Collaborators

Moderation options

Code and automation

Branches

Tags

Actions

Webhooks

Environments

Codespaces

Pages

Security

Code security and analysis

Deploy keys

Secrets and variables

Integrations

GitHub apps

Email notifications

Branch protection rule

Branch name pattern \*

main

Applies to 1 branch

main

Protect matching branches

☒ Require a pull request before merging

When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.

☒ Require approvals

When enabled, pull requests targeting a matching branch require a number of approvals and no changes requested before they can be merged.

Required number of approvals before merging: 1

☐ Dismiss stale pull request approvals when new commits are pushed

New reviewable commits pushed to a matching branch will dismiss pull request review approvals.

☐ Require review from Code Owners

Require an approved review in pull requests including files with a designated code owner.

☐ Require approval of the most recent push

The most recent push must be approved by someone other than the last pusher.

☒ Require status checks to pass before merging

Choose which status checks must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

☐ Require branches to be up to date before merging

This ensures pull requests targeting a matching branch have been tested with the latest code. This setting will not take effect unless at least one status check is enabled (see below).

Search for status checks in the last week for this repository

Status checks that are required

Abbildung 8: Konfiguration von QA-Optionen

GitHub repository view for `jk-fhswf/pki-a22-app` (Public). The interface shows the `Actions` tab selected, displaying a workflow run for `docs(Streamlit): :memo: Added hackaday reference #10`. The workflow file is `.github/workflows/main.yml`. The run status is **Success** (green checkmark).

On the left sidebar, the `Jobs` section lists the workflow jobs, with `pytest check` selected. Below it, the `Run details` section shows the workflow file.

The main content area displays the `pytest check` job details, which succeeded 12 minutes ago in 24s. The job log shows the following steps and their durations:

- Set up job: 3s
- Run actions/checkout@v2: 2s
- Run actions/setup-python@v2: 8s
- cache poetry install: 1s
- Run snok/install-poetry@v1: 3s
- cache deps: 4s
- Run poetry install --no-interaction --no-root: 8s
- Run poetry install --no-interaction: 5s
- Run poetry run pytest: 1s

The log for the `Run poetry run pytest` step shows the test session output:

```
1  ▶ Run poetry run pytest
2
3  ===== test session starts =====
4  platform linux -- Python 3.9.16, pytest-7.2.0, pluggy-1.0.0
5  rootdir: /home/runner/work/pki-a22-app/pki-a22-app
6  collected 5 items
7
8  tests/test_pki_a22_app.py . [ 20%]
9  tests/haarcascades/test_haarcascades.py . [ 40%]
10 tests/utlis/test_file_loader.py ... [100%]
11
12 ===== 5 passed in 0.65s =====
```

Abbildung 9: Beispiel-Ausführung von Tests mit pytest

## Literatur

- [Ada19] Darshan Adakane. *What are Haar Features used in Face Detection?* 2019. URL: <https://medium.com/analytics-vidhya/what-is-haar-features-used-in-face-detection-a7e531c8332b> (besucht am 11.01.2023).
- [han] hanno. *Face Detection using a Haar Cascade Classifier*. URL: <https://hackaday.io/project/12384-autofan-automated-control-of-air-flow/log/41956-face-detection-using-a-haar-cascade-classifier> (besucht am 11.01.2023).
- [Mar] Beni Keller Marco Schmid. *Einführung ins Programmieren mit Python*. URL: <https://pythonbuch.com/gui.html> (besucht am 11.01.2023).
- [Neu] Daniel Neumann. *Master Thesis - Fahrzeugetfassung mithilfe von Stereo-Vision sowie HOG-,LBP-, und Haar-Feature-basierter Bildklassifikatoren*. URL: [https://www.mi.fu-berlin.de/inf/groups/ag-ki/Theses/Completed-theses/Master\\_Diploma-theses/2016/Neumann/Master-Neumann.pdf](https://www.mi.fu-berlin.de/inf/groups/ag-ki/Theses/Completed-theses/Master_Diploma-theses/2016/Neumann/Master-Neumann.pdf) (besucht am 11.01.2023).
- [Phi] André Kaleja Philipp Viertel. *Haar Cascades*. URL: <https://fh-bielefeld-mif-sw-engineerin.gitbooks.io/script/content/embedded-computing/gesichtserkennung/haar-cascades.html> (besucht am 11.01.2023).