Programmierung für KI
Projektausarbeitung - Haar Cascades

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

15. Januar 2023

Inhaltsverzeichnis

[Ausarbeitung - Onur]

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 [1] 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 [**first**] und [**second**]).

1.1 Funktionsweise: Haar-like Features

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

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

[1] 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*

[2] Genauer handelt es sich hierbei um normierte Graustufenbilder

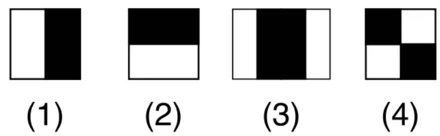

Abbildung 1: Haar-like Features

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 **??**.

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 [**drei**].

Um zu ermitteln, welche Haar-like Features mit Form und Größe nun am aussagekräftigsten sind, gibt es den sogenannten **Adaboost**-Algorithmus[3] 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 [**drei**].

[3] AdaBoost (adaptives 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]

```python
import tkinter as tk

def druecke_knopf():
  label.config(text="Button gedrueckt!")

root = tk.Tk()
root.geometry("1200x800")
root.title('Projekt Gruppe a2-2 - Thema: Bilderkennung Haar-Cascades')

button = tk.Button(root, text="Drueck mich", command=druecke_knopf)
button.pack()

label = tk.Label(root, text="Hallo Welt!")
label.pack()

root.mainloop()
```

Der obige Code erzeugt die recht simple Tkinter-Applikation mit einem Button und einem Label. Es sei hier auf [**vier**] verwiesen.

Abbildung 3: Einfache Tkinter App

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

Das Hauptfester wird mit dem Befehl

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

erzeugt und mit

```
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 [**vier**]. 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]

```python
import tkinter as tk
from tkinter import ttk
from tkinter import filedialog
import numpy as np
import cv2
from PIL import Image, ImageTk
import random
import os

from pki_a22_app.utils.file_loader import get_classifiers

path_haarcascade = "resources/haarcascades/haarcascade_"

```

```

classifier_list = get_classifiers()
```

2.2.2 Initialisierung und Geometrie

[Alle]

```
root = tk.Tk()
root.title('Projekt Gruppe a2-2 - Thema: Bilderkennung Haar-Cascades')


screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()


if screen_width/screen_height < 1.8:
  window_width = int(screen_width * 0.8)
else:
  window_width = int(screen_height * (screen_width/2/screen_height)*0.8)
  window_height = int(screen_height * 0.8)

img_max_width = int(window_width/2-75)
img_max_height = int(window_height-300)


center_x = int(screen_width/2 - window_width / 2)
center_y = int(screen_height/2 - window_height / 2)



root.geometry(f'{window_width}x{window_height}+{center_x}+{center_y}')
root.resizable(False, False)
```

2.2.3 Bild öffnen und Klassifizierer anwenden

[Peter]

```
def file_open():
  global input_image
  global output_image
  filename = filedialog.askopenfilename(filetypes=(("jpg files", "*.jpg"),("png files", "*.png")))
  input_image = Image.open(filename)
  width, height = input_image.size
  aspect_ratio = width / height
  if aspect_ratio > 1:
    new_width = img_max_width
    new_height = int(new_width / aspect_ratio)
  else:
  new_height = img_max_height
  new_width = int(new_height * aspect_ratio)
  input_image = input_image.resize((new_width,new_height), Image.ANTIALIAS)
```

```
  input_image_tk = ImageTk.PhotoImage(input_image)
  input_img_label.config(image=input_image_tk)
  input_img_label.image = input_image_tk

  output_image = Image.open(filename)
  output_image = output_image.resize((new_width,new_height), Image.ANTIALIAS)
  output_image_tk = ImageTk.PhotoImage(output_image)
  output_img_label.config(image=output_image_tk)
  output_img_label.image = output_image_tk
```

[Peter und Alexander]

```
def img_change(classifier):
  global input_image
  global output_image
  cascade = cv2.CascadeClassifier(path_haarcascade + classifier + ".xml")
  output_image_cv = np.array(output_image.convert('RGB'))
  output_image_cv_gray = cv2.cvtColor(output_image_cv, cv2.COLOR_BGR2GRAY)
  cascade_results = cascade.detectMultiScale(output_image_cv_gray, scaleFactor=s1.get_val(), minNeighbors = s2.get_val(), minSize=(s3.get_val(), s3.get_val()))
  iterations = 0

  if len(cascade_results) > 0:
    color = (random.randint(0,255),random.randint(0,255),random.randint(0,255))
    for (x,y,w,h) in cascade_results:
      cv2.rectangle(output_image_cv,(x,y),(x+w,y+h),(color),2)
      roi_gray = output_image_cv_gray[y:y+h, x:x+w]
      roi_color = output_image_cv[y:y+h, x:x+w]
    output_image = Image.fromarray(output_image_cv)
    output_img_label.image.paste(output_image)
  else:
    flag = False
    for i1 in np.arange (1.5, 0.9, -0.1):
      for i2 in range (6, 2, -1):
        for i3 in range (50, 9, -10):
          s1.s.set(i1)
          s2.s.set(i2)
          s3.s.set(i3)
          iterations = iterations + 1
          cascade_results = cascade.detectMultiScale(output_image_cv_gray, scaleFactor=s1.get_val(), minNeighbors = s2.get_val(), minSize=(s3.get_val(), s3.get_val()))
          if len(cascade_results) > 0:
            color = (random.randint(0,255),random.randint(0,255),random.randint(0,255))
            for (x,y,w,h) in cascade_results:
              cv2.rectangle(output_image_cv,(x,y),(x+w,y+h),(color),2)
              roi_gray = output_image_cv_gray[y:y+h, x:x+w]
              roi_color = output_image_cv[y:y+h, x:x+w]
              output_image = Image.fromarray(output_image_cv)
              output_img_label.image.paste(output_image)
```

```
              flag = True
              break
          if flag: break
        if flag: break
      if len(cascade_results) == 0:
        print(f"Kein Ergebnis nach {iterations} Iterationen")
```

[Peter]

```
def output_image_restart():
  global input_image
  global output_image
  output_image = input_image
  output_img_label.image.paste(output_image)
```

[Peter]

2.2.4 Slider und Button

```
class slider:
  def __init__(self,name,x_pos=0,y_pos=0,scale_from=0,scale_to=100,typ=int):
    self.name = name
    self.x_pos = x_pos
    self.y_pos = y_pos
    self.scale_from = scale_from
    self.scale_to = scale_to
    if typ==int:
      self.val = tk.IntVar()
    else:
      self.val = tk.DoubleVar()

    self.lbl = ttk.Label(root, text=name)
    self.lbl.place(x=self.x_pos,y=self.y_pos)

    self.s = ttk.Scale(root, from_=self.scale_from, to=self.scale_to,   orient="horizontal",command=self.slider_change,variable=self.val)
    self.s.place(x=self.x_pos+100,y=self.y_pos)

    self.v = ttk.Label(root,text=f"{self.get_val():.1f}")
    self.v.place(x=self.x_pos+210,y=self.y_pos)
  def get_val(self):
    return self.val.get()
  def slider_change(self, event):
    self.v.configure(text=f"{self.get_val():.1f}")
```

[Alexander]

```
def blur_rectangle(classifier):
  global input_image
  global output_image
  cascade = cv2.CascadeClassifier(path_haarcascade + classifier + ".xml")
  output_image_cv = np.array(output_image.convert('RGB'))
```

```
  output_image_cv_gray = cv2.cvtColor(output_image_cv, cv2.
    COLOR_BGR2GRAY)
  cascade_results = cascade.detectMultiScale(output_image_cv_gray
    , scaleFactor=s1.get_val(), minNeighbors = s2.get_val(),
    minSize=(s3.get_val(), s3.get_val()))
    if len(cascade_results) > 0:
      for (x,y,w,h) in cascade_results:
        face = output_image_cv[y:y+h, x:x+w]
        face = cv2.GaussianBlur(face, (23, 23), 30)
        output_image_cv[y:y+h, x:x+w] = face

      output_image = Image.fromarray(output_image_cv)
      output_img_label.image.paste(output_image)

def save_jpg():
  file_path = filedialog.asksaveasfilename(initialfile="
    output_image", filetypes=(("jpg files", "*.jpg"),("png files"
    , "*.png")), defaultextension=".jpeg")
  if file_path:
    output_image.save(file_path)
```

[Peter]

```
tk.Button(root, text="Bild aus Datei oeffnen", command=file_open)
    .place(x=window_width/2-window_width/4,y=150)
tk.Button(root, text="Classifier anwenden", command=lambda:
    img_change(dropdown.get())).place(x=window_width/2+
    window_width/4,y=150)
tk.Button(root, text="==>", command=output_image_restart).place(x
    =window_width/2-12.5,y=window_height/2)
tk.Button(root, text="Weichzeichnen",command=lambda:
    blur_rectangle(dropdown.get())).place(x=window_width/2+
    window_width/4+122,y=150)
tk.Button(root, text="Bild speichern",command=save_jpg).place(x=
    window_width/2+window_width/4+220,y=150)

dropdown = tk.StringVar(root)
dropdown.set(classifier_list[2])
dropdown_label = tk.OptionMenu(root, dropdown, *classifier_list)
dropdown_label.place(x=window_width/2-75,y=20)


xpos_slider_window = window_width/2 -75
ypos_slider_window = 60
s1 = slider("ScaleFactor",xpos_slider_window,ypos_slider_window
    ,1.01,1.5,float)
s1.s.set(1.1)
s2 = slider("MinNeighbors",xpos_slider_window,ypos_slider_window
    +30,3,6)
s2.s.set(4)
s3 = slider("minSize",xpos_slider_window,ypos_slider_window
    +60,10,50)
s3.s.set(30)


input_img_label = ttk.Label(root)
input_img_label.place(x=25,y=200)
```

```

output_img_label = ttk.Label(root)
output_img_label.place(x=window_width/2+50,y=200)

fh_logo = Image.open("resources/images/Logo.jpg")
fh_logo = fh_logo.resize((300,100), Image.ANTIALIAS)
fh_logo_tk = ImageTk.PhotoImage(fh_logo)
ttk.Label(root, image=fh_logo_tk).place(x=0,y=0)

root.mainloop()
```

3 Implementierung mit Streamlit

[Johannes]

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 **??**) oder das eigene Webcam-Bild zu verwenden. Für die Übertragung von Video und Webcam Bildern wurde die WebRTC-Erweiterung für Streamlit verwendet .

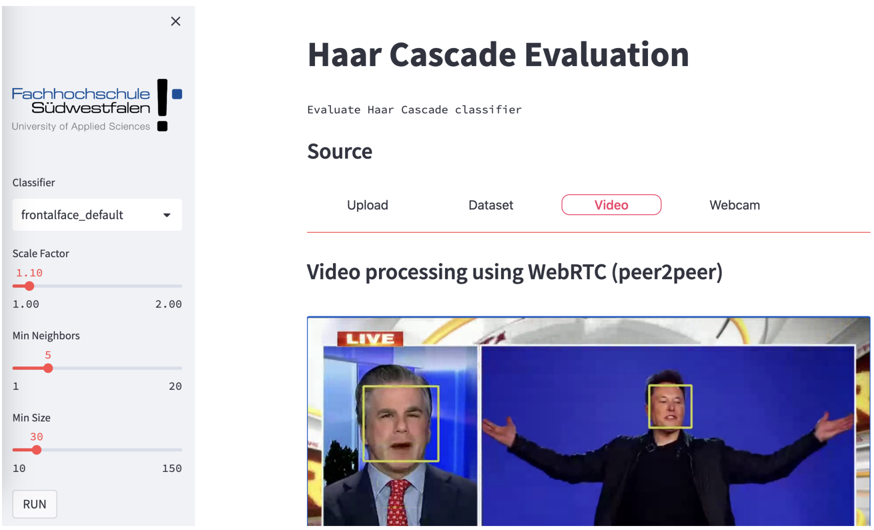

Abbildung 4: Streamlit Dashboard mit Haar Cascade Klassifizierer

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: [**hackaday**]

3.3 Code

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

```
# Hauptanwendung und Einstiegspunkt
app.py
# Funktion zur Ausfuehrung des eigentlichen Klassifizierers
pki_a22_app/haarcascades/haarcascades.py
# Interface fuer die unterschiedlichen Eingabequellen
pki_a22_app/dashboard/sources.py
# Eingabequelle mit einem einzelnen hochladbarem Bild
pki_a22_app/dashboard/source_upload.py
# Eingabequelle mit Auswahl eines Datensatzes mit mehreren
    Bildern
pki_a22_app/dashboard/source_dataset.py
# Eingabequelle zum Abspielen eines voreingestellten Videos
pki_a22_app/dashboard/source_video.py
# Eingabequelle bei der die eigene Webcam verwendet wird
pki_a22_app/dashboard/source_webcam.py
# Utility Modul, das vor allem Funktionen fuer Datei-Operationen
    enthaelt
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:

```
"""Start a streamlit dashboard showing some haarcascade
    operations with Opencv"""

# Core Pkgs

import extra_streamlit_components as stx
import streamlit as st
from pki_a22_app.dashboard.source_dataset import DatasetSource
from pki_a22_app.dashboard.source_upload import UploadSource
from pki_a22_app.dashboard.source_video import VideoSource
```

```python
from pki_a22_app.dashboard.source_webcam import WebcamSource

from pki_a22_app.utils.file_loader import (get_classifiers)


def main():
    """
    Main function that initiates the Streamlit dashboard
    """

    st.title("Haar Cascade Evaluation")
    st.text("Evaluate Haar Cascade classifier")

    # Configure the sources section
    st.subheader("Source")
    chosen_id = stx.tab_bar(data=[
        stx.TabBarItemData(id=1, title="Upload", description=""),
        stx.TabBarItemData(id=2, title="Dataset", description=""),
        stx.TabBarItemData(id=3, title="Video", description=""),
        stx.TabBarItemData(id=4, title="Webcam", description=""),
    ], default=1)

    st.sidebar.image("resources/images/logo-fh-swf-300x93.png")
    st.sidebar.markdown('#')

    # Configure the algorithm controls
    classifiers: list = get_classifiers()
    classifier_id = st.sidebar.selectbox("Classifier", classifiers, index=5)
    scale_factor = st.sidebar.slider('Scale Factor', 1.01, 2.0, 1.1, 0.01)
    min_neighbors = st.sidebar.slider('Min Neighbors', 1, 20, 5, 1)
    min_size = st.sidebar.slider('Min Size', 10, 150, 30, 5)

    show_results = st.sidebar.button("RUN")

    # Define different sources that render the input and output accordingly
    sources = {
        "1": UploadSource(),
        "2": DatasetSource(),
        "3": VideoSource(),
        "4": WebcamSource()
    }

    # Execute selected source
    sources[chosen_id].load_source(
        classifier_id, scale_factor,  min_neighbors, min_size, show_results)


if __name__ == '__main__':
    main()
```

Listing 1: app.py

```python
"""This module contains haar cascade core functionality"""

import cv2
from cv2 import Mat
import numpy.typing as npt


def detect_objects(np_input_image: npt.ArrayLike, classifier_id: str, scale_factor: float = 1.1, min_neighbors: int = 4, min_size: int = 10) -> Mat:
    """
    Detects objects in an image using Haar Cascades classifiers. It can usa a preset of trained classifiers in resources/haarcascades/*.
    The found objects get highlighted by a rectangle around it.

    The parameter description was taken from:
    https://hackaday.io/project/12384-autofan-automated-control-of-air-flow/log/41956-face-detection-using-a-haar-cascade-classifier


    Parameters
    ----------
    np_input_image : npt.ArrayLike
        The input image
    classifier_id : str
        the classifier name. E.g. when you set it to "frontalface_default", the configuration file
        "haarcascade_frontalface_default.xml" will be loaded
    scale_factor : float, optional, by default 1.1
        determines the factor by which the detection window of the classifier is scaled down per detection pass. A factor of 1.1
        corresponds to an increase of 10%. Hence, increasing the scale factor increases performance, as the number of detection
        passes is reduced. However, as a consequence the reliability by which a face is detected is reduced. See:
    min_neighbors : int, optional, by default 4
        determines the minimum number of neighboring facial features that need to be present to indicate the detection of a face by the
        classifier. Decreasing the factor increases the amount of false positive detections. Increasing the factor might lead to missing
        faces in the image. The argument seems to have no influence on the performance of the algorithm. See:
    min_size: int
        determines the minimum size of the detection window in pixels. Increasing the minimum detection window increases performance.
        However, smaller faces are going to be missed then. In the scope of this project however a relatively big detection window can
        be used, as the user is sitting directly in front of the camera.
```

```python

    Returns
    -------
    cv2.Mat
        The image in Mat representation
    list
        A list of found objects in the image
    """

    classifier = cv2.CascadeClassifier(
        f"resources/haarcascades/haarcascade_{classifier_id}.xml"
    )

    img = cv2.cvtColor(np_input_image, 1)
    gray = cv2.cvtColor(np_input_image, cv2.COLOR_BGR2GRAY)
    objects = classifier.detectMultiScale(
        gray, scaleFactor=scale_factor, minNeighbors=
    min_neighbors, minSize=(min_size, min_size))
    # pylint: disable=invalid-name
    for (x, y, w, h) in objects:
        cv2.rectangle(img, (x, y), (x + w, y + h), (72, 209, 204)
    , 4)
    return img, objects
```

Listing 2: haarcascades/haarcascades.py

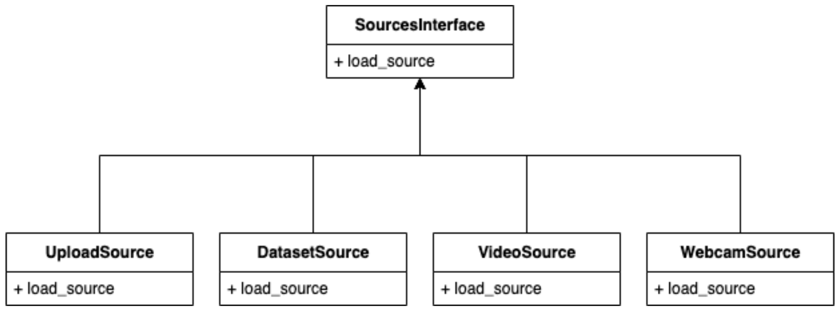

Abbildung 5: UML Diagramm für unterschiedliche Source-Typen

```python
"""
Contains an interface definition to use different source-types
in the Streamlit dashboard
"""
import abc


class SourcesInterface(metaclass=abc.ABCMeta):
    """
    Interface for handling different input source types

    Methods
    -------
    load_source(classifier_id: str, scale_factor: int,
    min_neighbors: int, show_results: bool = False)
        Load the sources and outputs with respect to the given
    parameters
    """

    @classmethod
    def __subclasshook__(cls, subclass):
        """
        Ensures that needed methods are implemented in your
    derivatives

        Parameters
        ----------
        subclass : Any
            subclass that was created

        Returns
        -------
        Bool
            True if it is a valid subclass
        """
        return (hasattr(subclass, 'load_source') and
                callable(subclass.load_source) or
                NotImplemented)


    @abc.abstractmethod
```

```python
    def load_source(self, classifier_id: str, scale_factor: int, min_neighbors: int, min_size: int, show_results: bool = False):
        """
        This method handles the display of different Streamlit-UI elements. Implement these in your
        concrete derivatives

        The parameter description was taken from:
        https://hackaday.io/project/12384-autofan-automated-control-of-air-flow/log/41956-face-detection-using-a-haar-cascade-classifier

        Parameters
        ----------
        classifier_id : str
            the classifier name. E.g. when you set it to "frontalface_default", the configuration file
            "haarcascade_frontalface_default.xml" will be loaded
        scale_factor : float, optional, by default 1.1
            determines the factor by which the detection window of the classifier is scaled down per detection pass. A factor of 1.1
            corresponds to an increase of 10%. Hence, increasing the scale factor increases performance, as the number of detection
            passes is reduced. However, as a consequence the reliability by which a face is detected is reduced. See:
        min_neighbors : int, optional, by default 4
            determines the minimum number of neighboring facial features that need to be present to indicate the detection of a face by the
            classifier. Decreasing the factor increases the amount of false positive detections. Increasing the factor might lead to missing
            faces in the image. The argument seems to have no influence on the performance of the algorithm. See:
        min_size: int
            determines the minimum size of the detection window in pixels. Increasing the minimum detection window increases performance.
            However, smaller faces are going to be missed then. In the scope of this project however a relatively big detection window can
            be used, as the user is sitting directly in front of the camera.
        show_results : bool, optional
            If true we want to display the original images AND the results, by default False

        Raises
        ------
        NotImplementedError
            Error that gets thrown if you do not implement this function
        """
```

```
        raise NotImplementedError
```

Listing 3: dashboard/sources.py

```
"""Implementation of a source-type to upload an image and apply the Haar Cascade classifier on it"""
# Core Pkgs

import numpy as np
import streamlit as st
from PIL import Image

from pki_a22_app.dashboard.sources import SourcesInterface
from pki_a22_app.haarcascades.haarcascades import detect_objects


class UploadSource(SourcesInterface):
    def load_source(self, classifier_id: str, scale_factor: int, min_neighbors: int, min_size: int, show_results: bool = False):
        """
        This method creates a form to upload an image that can be processed by the haar cascade classifier afterwards.

        The parameter description was taken from:
        https://hackaday.io/project/12384-autofan-automated-control-of-air-flow/log/41956-face-detection-using-a-haar-cascade-classifier

        Parameters
        ----------
        classifier_id : str
            the classifier name. E.g. when you set it to "frontalface_default", the configuration file
            "haarcascade_frontalface_default.xml" will be loaded
        scale_factor : float, optional, by default 1.1
            determines the factor by which the detection window of the classifier is scaled down per detection pass. A factor of 1.1
            corresponds to an increase of 10%. Hence, increasing the scale factor increases performance, as the number of detection
            passes is reduced. However, as a consequence the reliability by which a face is detected is reduced. See:
        min_neighbors : int, optional, by default 4
            determines the minimum number of neighboring facial features that need to be present to indicate the detection of a face by the
            classifier. Decreasing the factor increases the amount of false positive detections. Increasing the factor might lead to missing
            faces in the image. The argument seems to have no influence on the performance of the algorithm. See:
        min_size: int
            determines the minimum size of the detection window in pixels. Increasing the minimum detection window increases performance.
```

```
            However, smaller faces are going to be missed then. In the scope of this project however a relatively big detection window can
            be used, as the user is sitting directly in front of the camera.
        show_results : bool, optional
            If true we want to display the original images AND the results, by default False
        """
        st.subheader("Upload an image")

        # Show the upload form
        image_file = st.file_uploader(
            "Upload Image", type=["jpg", "png", "jpeg"])
        our_image = None

        # If a image was uploaded, display it
        if image_file is not None:
            our_image = Image.open(image_file)
            our_np_image = np.array(our_image.convert("RGB"))
            st.text("Original Image")
            st.image(our_image)
            image_location = st.empty()

        # Show processed image
        if show_results and classifier_id:
            # Actual object detection
            result_img, result_faces = detect_objects(
                our_np_image, classifier_id, scale_factor, min_neighbors, min_size)
            image_location.image(result_img)
            st.text("Resulting Image")
            st.success(f"Found {len(result_faces)} Objects")
```

Listing 4: dashboard/source upload.py

```
"""Implementation of a source-type that show multiple images at once"""

# Core Pkgs
from itertools import cycle

import numpy as np
import streamlit as st
from PIL import Image

from pki_a22_app.dashboard.sources import SourcesInterface
from pki_a22_app.haarcascades.haarcascades import detect_objects
from pki_a22_app.utils.file_loader import get_datasets, get_images_of_dataset


class DatasetSource(SourcesInterface):
    """
    Implementation for handling multiple images, also called dataset here

    Methods
```

```
    -------
    load_source(classifier_id: str, scale_factor: int, min_neighbors: int, show_results: bool = False)
        Load multiple images and display results
    """

    def load_source(self, classifier_id: str, scale_factor: int, min_neighbors: int, min_size: int, show_results: bool = False):
        """
        This method shows a selectbox to select an existing dataset. A dataset contains multiple images. The datasets are located
        at resource/datasets/. The Haar Cascade classifier will then be applied to all images of the dataset.

        The parameter description was taken from:
        https://hackaday.io/project/12384-autofan-automated-control-of-air-flow/log/41956-face-detection-using-a-haar-cascade-classifier

        Parameters
        ----------
        classifier_id : str
            the classifier name. E.g. when you set it to "frontalface_default", the configuration file
            "haarcascade_frontalface_default.xml" will be loaded
        scale_factor : float, optional, by default 1.1
            determines the factor by which the detection window of the classifier is scaled down per detection pass. A factor of 1.1
            corresponds to an increase of 10%. Hence, increasing the scale factor increases performance, as the number of detection
            passes is reduced. However, as a consequence the reliability by which a face is detected is reduced. See:
        min_neighbors : int, optional, by default 4
            determines the minimum number of neighboring facial features that need to be present to indicate the detection of a face by the
            classifier. Decreasing the factor increases the amount of false positive detections. Increasing the factor might lead to missing
            faces in the image. The argument seems to have no influence on the performance of the algorithm. See:
        min_size: int
            determines the minimum size of the detection window in pixels. Increasing the minimum detection window increases performance.
            However, smaller faces are going to be missed then. In the scope of this project however a relatively big detection window can
            be used, as the user is sitting directly in front of the camera.
        show_results : bool, optional
            If true we want to display the original images AND the results, by default False
```

```
        """
        st.subheader("Load an existing dataset")

        # Setup dataset selector
        datasets = get_datasets()
        datasets.insert(0, "None")
        dataset_selection = st.selectbox("Dataset", datasets)

        img_width = 300
        img_columns = 2

        # When a dataset was selected start presenting it
        if dataset_selection and dataset_selection != "None":
            images_in_ds = get_images_of_dataset(dataset_selection)

            # Show images in max n columns
            cols = cycle(st.columns(img_columns))
            for idx, filtered_image in enumerate(images_in_ds):
                next(cols).image(str(filtered_image), width=img_width)

            # We were requested to show the results
            if show_results and dataset_selection:
                images_in_ds = get_images_of_dataset(dataset_selection)
                st.write("### Results:")
                cols = cycle(st.columns(img_columns))

                for idx, filtered_image in enumerate(images_in_ds):
                    ds_image = Image.open(filtered_image)
                    ds_np_image = np.array(ds_image.convert("RGB"))
                    # Actual object detection
                    ds_res_image, ds_result_faces = detect_objects(ds_np_image, classifier_id, scale_factor,
                                                                    min_neighbors, min_size)
                    col = next(cols)
                    col.image(ds_res_image, width=img_width, caption=f"{len(ds_result_faces)} objects")
```

Listing 5: dashboard/source dataset.py

```
"""Implementation of a source-type to select a video and apply the Haar Cascade classifier on it"""

# Core Pkgs

import av
import streamlit as st
from streamlit_webrtc import MediaPlayerFactory, WebRtcMode, webrtc_streamer

from pki_a22_app.dashboard.sources import SourcesInterface
from pki_a22_app.haarcascades.haarcascades import detect_objects
from aiortc.contrib.media import MediaPlayer
```

```python


class VideoSource(SourcesInterface):
    def load_source(self, classifier_id: str, scale_factor: int, min_neighbors: int, min_size: int, show_results: bool = False):
        """
        This function shows a video player where the haar cascade detection is applied on  single frames.
        The player uses WebRTC to stream the video.

        The parameter description was taken from:
        https://hackaday.io/project/12384-autofan-automated-control-of-air-flow/log/41956-face-detection-using-a-haar-cascade-classifier


        Parameters
        ----------
        classifier_id : str
            the classifier name. E.g. when you set it to "frontalface_default", the configuration file
            "haarcascade_frontalface_default.xml" will be loaded
        scale_factor : float, optional, by default 1.1
            determines the factor by which the detection window of the classifier is scaled down per detection pass. A factor of 1.1
            corresponds to an increase of 10%. Hence, increasing the scale factor increases performance, as the number of detection
            passes is reduced. However, as a consequence the reliability by which a face is detected is reduced. See:
        min_neighbors : int, optional, by default 4
            determines the minimum number of neighboring facial features that need to be present to indicate the detection of a face by the
            classifier. Decreasing the factor increases the amount of false positive detections. Increasing the factor might lead to missing
            faces in the image. The argument seems to have no influence on the performance of the algorithm. See:
        min_size: int
            determines the minimum size of the detection window in pixels. Increasing the minimum detection window increases performance.
            However, smaller faces are going to be missed then. In the scope of this project however a relatively big detection window can
            be used, as the user is sitting directly in front of the camera.
        show_results : bool, optional
            If true we want to display the original images AND the results, by default False
        """
        st.subheader("Video processing using WebRTC (peer2peer)")

        # Callback function that takes an image frame that can
```

```
        # be processed
        def callback(frame):
            # convert frame to ndarray
            img = frame.to_ndarray(format="bgr24")

            # actual object detection
            result_img, result_faces = detect_objects(
                img, classifier_id, scale_factor, min_neighbors, min_size)

            return av.VideoFrame.from_ndarray(result_img, format="bgr24")

        rtc_configuration = {  # Add this config
            "iceServers": [{"urls": ["stun:stun.l.google.com:19302"]}]
        }

        def create_player():
            return MediaPlayer("resources/videos/musk.mp4")

        # This will show the video-UI elements in Streamlit
        # Each frame gets sent to the callback method
        webrtc_streamer(key="videoexample",
                        video_frame_callback=callback,
                        rtc_configuration=rtc_configuration,
                        mode=WebRtcMode.RECVONLY,
                        player_factory=create_player,
                        media_stream_constraints={
                            "video": True
                        }
                        )
```

Listing 6: dashboard/source video.py

```
"""Implementation of a source-type to display a webcam stream and apply the Haar Cascade classifier on it"""

# Core Pkgs

import av
import streamlit as st
from streamlit_webrtc import webrtc_streamer

from pki_a22_app.dashboard.sources import SourcesInterface
from pki_a22_app.haarcascades.haarcascades import detect_objects


class WebcamSource(SourcesInterface):
    def load_source(self, classifier_id: str, scale_factor: int, min_neighbors: int,
                    min_size: int, show_results: bool = False) -> None:
        """
        This method shows an UI element to display the stream of a webcam. The Haar Cascade classifier will then be applied to the images
        of the webcam.
```

```

    The parameter description was taken from:
    https://hackaday.io/project/12384-autofan-automated-control-of-air-flow/log/41956-face-detection-using-a-haar-cascade-classifier


    Parameters
    ----------
    classifier_id : str
        the classifier name. E.g. when you set it to "frontalface_default", the configuration file
        "haarcascade_frontalface_default.xml" will be loaded
    scale_factor : float, optional, by default 1.1
        determines the factor by which the detection window of the classifier is scaled down per detection pass. A factor of 1.1
        corresponds to an increase of 10%. Hence, increasing the scale factor increases performance, as the number of detection
        passes is reduced. However, as a consequence the reliability by which a face is detected is reduced. See:
    min_neighbors : int, optional, by default 4
        determines the minimum number of neighboring facial features that need to be present to indicate the detection of a face by the
        classifier. Decreasing the factor increases the amount of false positive detections. Increasing the factor might lead to missing
        faces in the image. The argument seems to have no influence on the performance of the algorithm. See:
    min_size: int
        determines the minimum size of the detection window in pixels. Increasing the minimum detection window increases performance.
        However, smaller faces are going to be missed then. In the scope of this project however a relatively big detection window can
        be used, as the user is sitting directly in front of the camera.
    show_results : bool, optional
        If true we want to display the original images AND the results, by default False
    """
    st.subheader("Webcam processing using WebRTC (peer2peer)")

    def callback(frame):
        # convert frame to ndarray
        img = frame.to_ndarray(format="bgr24")

        # actual object detection
        result_img, result_faces = detect_objects(
            img, classifier_id, scale_factor, min_neighbors, min_size)

        return av.VideoFrame.from_ndarray(result_img, format=
```

```
    "bgr24")

        # This will show the webcam-UI elements in streamlit
        # Each frame gets sent to the callback method
        rtc_configuration = {  # Add this config
            "iceServers": [{"urls": ["stun:stun.l.google.com
    :19302"]}]
        }
        webrtc_streamer(key="example", video_frame_callback=
    callback,
                        rtc_configuration=rtc_configuration)
```

Listing 7: dashboard/source webcam.py

```
"""This modules contains helper methods to get information from
    local file system"""
import os
import pathlib
from typing import List


def get_classifiers() -> List[str]:
    """
    Reads all files in /resources/haarcascades/ and returns only
    the actual classifier
    name without "haarcascade_" and ".xml". "E.g.
    haarcascade_frontalface_default.xml"
    gets to "haarcascade_frontalface_default".

    Returns
    -------
    List[str]
        List of found classifiers
    """
    classifier_files = os.listdir(os.getcwd() + "/resources/
    haarcascades/")
    classifiers = [item.replace("haarcascade_", "")
                   for item in classifier_files]
    classifiers = [item.replace(".xml", "") for item in
    classifiers]
    classifiers.sort()
    return classifiers


def get_datasets() -> List[str]:
    """
    Returns the folders found under /resources/datasets/

    Returns
    -------
    List[str]
        List of available dataset
    """
    datasets = os.listdir(os.getcwd() + "/resources/datasets/")
    datasets = [item for item in datasets if not item.startswith(
    ".")]
    datasets.sort()
    return datasets
```

```python


def get_images_of_dataset(dataset_name: str) -> List[str]:
    """
    Returns the images contined in a dataset

    Parameters
    ----------
    dataset_name : str
        Name of the dataset (folder)

    Returns
    -------
    List[str]
        List of found images
    """
    extensions = ['.jpg', '.png', '.jpeg']
    img_list = [x for x in pathlib.Path(f"resources/datasets/{dataset_name}").iterdir() if x.suffix.lower() in extensions]
    img_list = [str(item) for item in img_list]
    return list(img_list)
```

Listing 8: utils/file loader.py

4 Organisation

[Alle]

Besonders zu Beginn war es sehr hilfreich, eine Kollaborations-Plattform für das Projekt einzurichten, über die die unterschiedlichsten Informationen organisatorischer Natur ausgetauscht werden konnten. Wir haben uns für den Einsatz eines Miro-Boardes (Abbildung **??**) entschieden. das wir unter anderem nutzten für:

- Terminfindung
- Pinnwand
- Brainstorming
- Zeitplanung
- Wireframes

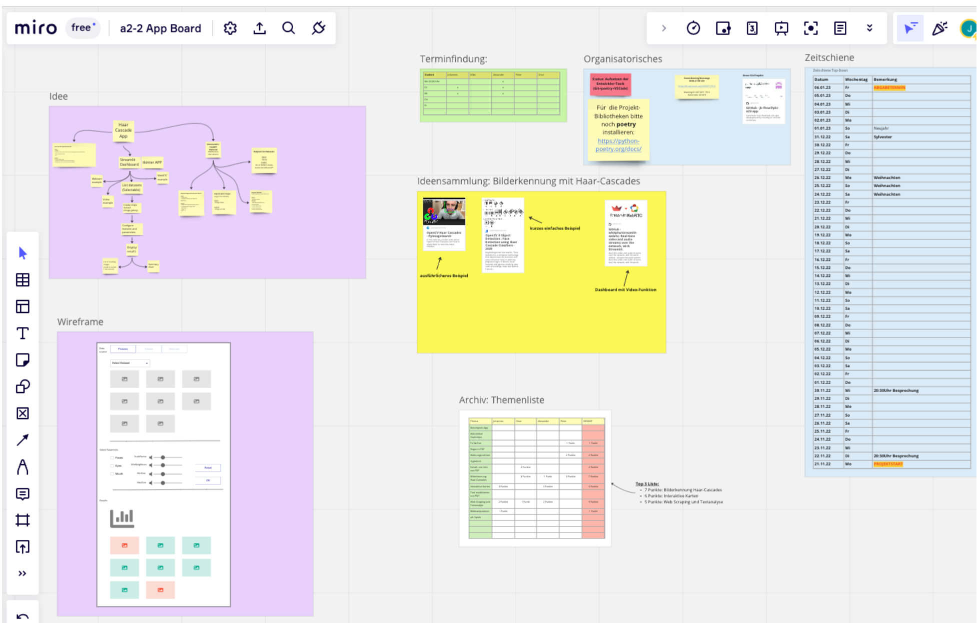

Abbildung 6: Miro-Board für organisatorische Aufgaben

5 Continuous Integration

[Johannes]

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 **??**) eingerichtet.

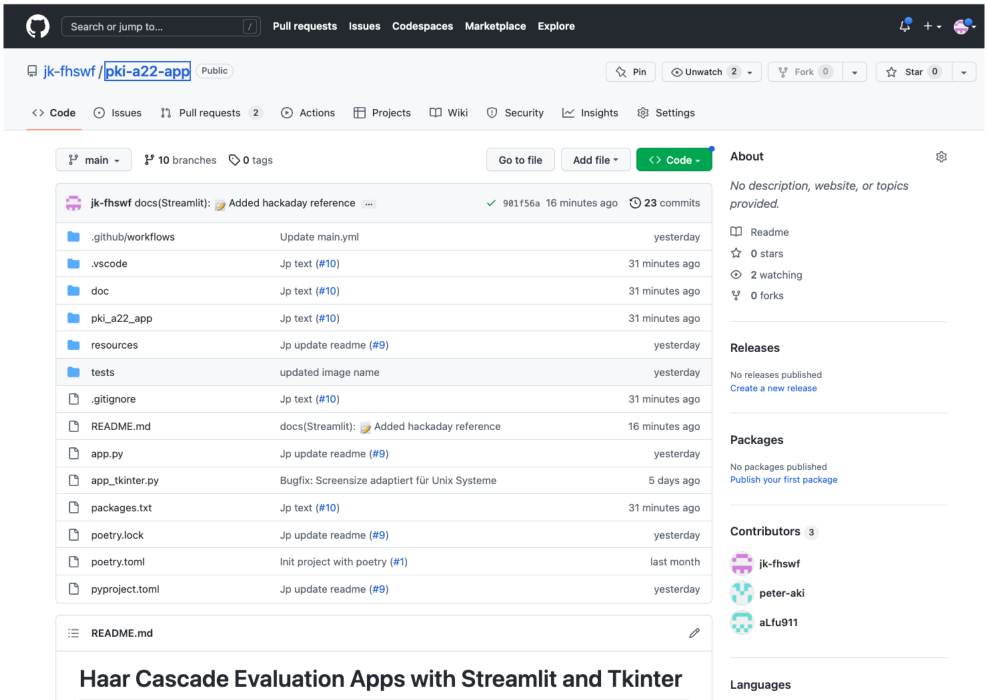

Abbildung 7: Git-Repository auf GitHub

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 **??**). Eine Beispiellauf des Workflows ist in Abbildung **??** zu sehen.

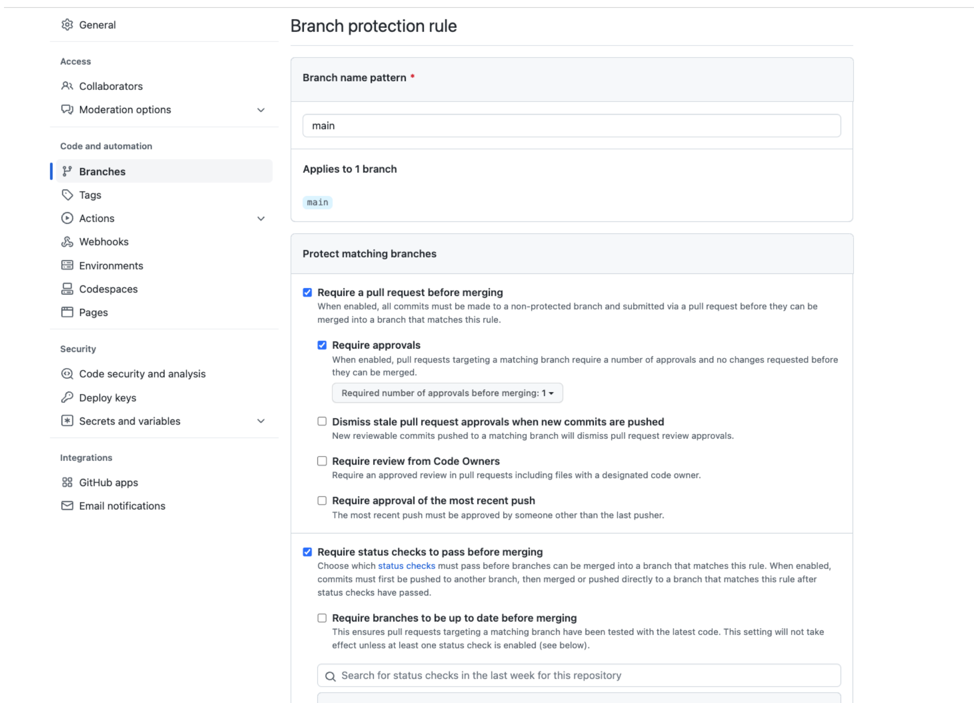

Abbildung 8: Konfiguration von QA-Optionen

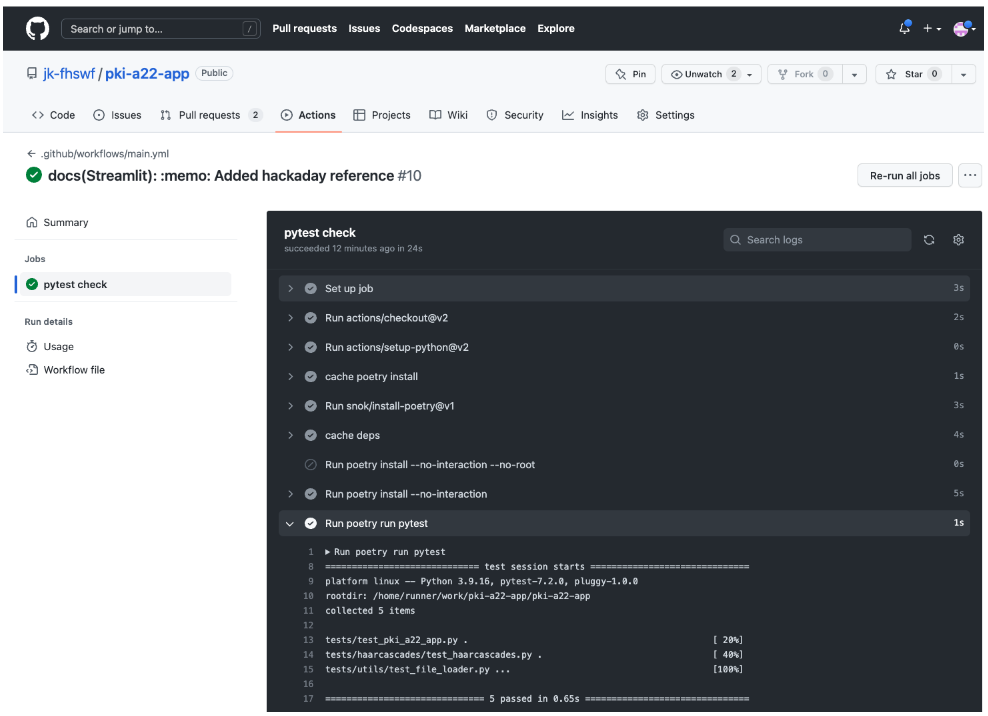

Abbildung 9: Beispiel-Ausführung von Tests mit pytest