**DIPLOMARBEIT**

DermaAI: Intelligente Hautanalyse

Gesamtprojekt

Entwicklung von KI-Modellen zur Detektion und Klassifikation pigmentierter Hautläsionen und deren Integration in eine Mobile App

Diplomarbeitsnummer  
5AHINF-24/25-DA11

**Datenbank, Verwaltung der medizinischen Daten**

Jonas Maier 5AHINF Betreuer: Dipl.-Ing. Dr.

Gerhard Gaube

**Mobile Integration, Frontend**

Jonas Bogensberger 5AHINF Betreuer: MSC

Michael Prader

**KI-Modelle in Python, API und Appanbindung**

Daniel Jessner 5AHINF Betreuer: Dipl.-Ing. Dr.

Gerhard Gaube

Schuljahr 2024/25

Abgabevermerk:

Datum: TT.MM.JJJJ übernommen von:

# Eidesstattliche Erklärung

Wir erklären an Eides statt, dass wir die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht haben.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ort, am TT.MM.JJJJ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Daniel Jessner

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Jonas Maier

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Jonas Bogensberger

# Allgemeines & Projektübersicht

*Kapitel verfasst von Daniel Jessner*

## Projektbeschreibung

Das Projekt „**DermaAI**“ stellt ein Abschlussprojekt für ein Projektteam der 5. AHINF an der HTL Saalfelden dar und hat als Ziel, ein System zu entwickeln, welches es einem Benutzer ermöglicht über sein Mobilgerät Bilder von Hautläsionen aufzunehmen und diese mithilfe von künstlicher Intelligenz auswerten zu lassen. Die Anwendung soll also dabei helfen eine erste Entscheidungsgrundlage zu bieten und eventuell bei Diagnosen helfen. Dazu eignet sich eine Dreiteilung in die Bereiche **Frontend** (Mobile Anwendung), **Backend** (Datenbank) und **Gateway** (KI-Komponente, Vermittler).

Wichtige Eckdaten:

* **Ausschreibung**

Das Projekt an sich wurde am **09.07.2024** von der Fachhochschule Joanneum ausgeschrieben und über die Lehrperson Gerhard Gaube dem Projektteam vorgestellt.

* **Projektstart**

Offizieller Projektstart war der **08.11.2024**. Dabei wurden sowohl eine erste Grundstruktur der Anwendung erstellt als auch das der Version Control zugrundeliegende Github-Repository eingerichtet. Projektplanung und Absprachen über Arbeitsteilung waren zu diesem Zeitpunkt bereits vollendet.

* **Projektende**

Das Projektende setzt sich grundlegend aus mehreren Zeitpunkten zusammen:

* + **.05.2025** Öffentliche Präsentation der Diplomarbeit
  + **21.05.2025** Schulinterne Verteidigung der Diplomarbeit
  + **.05.2025** Abgabe der gebundenen Diplomarbeit
  + **.05.2025** Abgabe der Diplomarbeitsvideos

Zum Zeitpunkt der Abgabe der Diplomarbeit besteht eine vollfunktionsfähige und dokumentierte erste Version der Anwendung, welche, so wie sie ist, vom Endnutzer genutzt werden kann.

## Projektteam und Schnittstellen

|  |  |  |  |
| --- | --- | --- | --- |
| **Rolle(n)** | **Name** | **Telefon** | **E-Mail** |
| Entwickler –  Frontend | Jonas Bogensberger | +43 676 3617230 | [jonas.bogensberger@htl-saalfelden.at](mailto:jonas.bogensberger@htl-saalfelden.at) |
| Entwickler –  Backend | Jonas Maier | +43 664 75087002 | [jonas.maier@htl-saalfelden.at](mailto:jonas.maier@htl-saalfelden.at) |
| Entwickler –  KI-Komponente | Daniel Jessner | +43 650 6349636 | [daniel.jessner@htl-saalfelden.at](mailto:daniel.jessner@htl-saalfelden.at) |

# Anforderungsanalyse

*Kapitel verfasst von Daniel Jessner*

In diesem Kapitel werden die funktionalen und nicht-funktionalen Anforderungen an das System detailliert analysiert. Ziel der Anforderungsanalyse ist es, die grundlegenden Erwartungen und Rahmenbedingungen für die Entwicklung klar zu definieren. Dabei werden sowohl technische als auch benutzerbezogene Anforderungen betrachtet, um eine optimale Umsetzung der Softwarelösung sicherzustellen. Die Analyse basiert auf den erhobenen Nutzerbedürfnissen, bestehenden Systemvorgaben und relevanten technischen Spezifikationen.

## Personas

Die Entwicklung eines nutzerzentrierten Systems erfordert ein tiefgehendes Verständnis der Zielgruppen und ihrer spezifischen Bedürfnisse. Im Rahmen der Anforderungsanalyse für das Diplomprojekt „DermaAI“ wurden daher verschiedene **Personas** definiert, die typische Nutzergruppen der Anwendung repräsentieren.

Diese Personas helfen dabei, die funktionalen und nicht-funktionalen Anforderungen gezielt zu formulieren und die Benutzerfreundlichkeit der Lösung zu optimieren. Sie berücksichtigen unterschiedliche Rollen wie **Ärzte, medizinisches Fachpersonal, Patienten und technische Experten**, die jeweils spezifische Herausforderungen und Erwartungen an das System haben.

Durch die Analyse dieser Personas können realistische Nutzungsszenarien entwickelt und Designentscheidungen besser begründet werden, um eine intuitive und effektive Anwendung für alle Beteiligten zu schaffen.

### Persona 1

**Dr. Anna Weber (Dermatologin)**

* **Alter:** 45 Jahre
* **Wohnort:** München, Deutschland
* **Einkommen:** > 80.000€
* **Ausbildung:** Studium der Medizin, Fachärztin für Dermatologie
* **Interessen:** Forschung, neue Technologien, Telemedizin
* **Eigenschaften:** Genau, analytisch, technologieaffin
* **Besonderheit:** Arbeitet in einer Klinik und bietet Telemedizin-Diagnosen an

**Ziele & Bedürfnisse:**  
✔ Schnelle Voranalyse von Hautläsionen zur besseren Patientensteuerung  
✔ Unterstützung bei Ferndiagnosen durch KI-gestützte Wahrscheinlichkeiten  
✔ Integration in bestehende Patientenverwaltungssoftware

**Herausforderungen:**  
⚠ Datenschutzbedenken bei der Nutzung einer mobilen App  
⚠ Akzeptanz neuer Technologien durch ältere Kollegen

### Persona 2

**Max Schröder (Endnutzer mit Hautproblemen)**

* **Alter:** 32 Jahre
* **Wohnort:** Berlin, Deutschland
* **Einkommen:** 45.000€
* **Beruf:** IT-Consultant
* **Interessen:** Fitness, Outdoor-Sport, Technik, Selbstoptimierung
* **Eigenschaften:** sicherheitsbewusst, gesundheitsbewusst
* **Besonderheit:** Hat oft mit Hautproblemen (Muttermale, Rötungen) zu tun, will Risiken frühzeitig erkennen

**Ziele & Bedürfnisse:**  
✔ Einfache Möglichkeit, auffällige Hautveränderungen selbst zu überprüfen  
✔ Vertrauenswürdige KI-Ergebnisse, die als erste Einschätzung dienen  
✔ Klare Handlungsempfehlungen, ob ein Arztbesuch nötig ist

**Herausforderungen:**  
⚠ Skepsis gegenüber KI-Diagnosen und deren Genauigkeit  
⚠ Datenschutz und sichere Speicherung seiner Bilder

### Persona 3

**Lisa König (Medizinische Assistentin in einer Hautarztpraxis)**

* **Alter:** 28 Jahre
* **Wohnort:** Hamburg, Deutschland
* **Einkommen:** 35.000€
* **Beruf:** Medizinische Fachangestellte
* **Interessen:** Medizinische Weiterbildung, Patientenkontakt, Effizienzsteigerung in der Praxis
* **Eigenschaften:** Freundlich, hilfsbereit, organisiert
* **Besonderheit:** Oft die erste Ansprechperson für Patienten mit Hautproblemen

**Ziele & Bedürfnisse:**  
✔ Unterstützung bei der Voruntersuchung durch KI-basierte Einschätzungen  
✔ Vereinfachung der Dokumentation und Bilderfassung für Ärzte  
✔ Schnelle Einschätzung, ob ein Patient einen dringenden Termin benötigt

**Herausforderungen:**  
⚠ Unsicherheit, ob KI-Ergebnisse zuverlässig genug sind  
⚠ Datenschutz und rechtliche Vorgaben zur Patientendatenverwaltung

### Persona 4

**Peter Maier (Patient 60+, gesundheitsbewusst)**

* **Alter:** 65 Jahre
* **Wohnort:** Stuttgart, Deutschland
* **Einkommen:** 50.000€ (Rente) **Beruf:** Pensionierter Lehrer
* **Interessen:** Wandern, Gartenarbeit, Gesundheitsvorsorge
* **Eigenschaften:** Vorsichtig, gesundheitsbewusst, technikinteressiert (aber nicht technikaffin)
* **Besonderheit:** Hatte in der Vergangenheit Hautkrebs-Vorstufen und möchte sich regelmäßig selbst kontrollieren

**Ziele & Bedürfnisse:**  
✔ Einfache, barrierefreie App-Nutzung für regelmäßige Hautkontrollen  
✔ Klare, leicht verständliche Erklärungen zu KI-Ergebnissen  
✔ Unterstützung bei der Entscheidung, wann er zum Arzt gehen sollte

**Herausforderungen:**  
⚠ Angst vor Fehldiagnosen durch KI  
⚠ Bedienbarkeit der App für ältere Menschen

### Persona 5

**Dr. Felix Wagner (KI-Entwickler im MedTech-Bereich)**

* **Alter:** 38 Jahre
* **Wohnort:** Berlin, Deutschland
* **Einkommen:** 90.000€
* **Beruf:** Data Scientist, spezialisiert auf medizinische Bildverarbeitung
* **Interessen:** Maschinelles Lernen, KI-Ethik, Open-Source-Projekte
* **Eigenschaften:** Innovativ, kritisch, lösungsorientiert
* **Besonderheit:** Arbeitet an der Verbesserung von KI-Modellen zur Hautbildanalyse

**Ziele & Bedürfnisse:**  
✔ Verbesserung der KI-Modelle durch qualitativ hochwertige Datensätze  
✔ Optimierung der Benutzerfreundlichkeit durch bessere Algorithmen  
✔ Sicherstellung der ethischen und rechtlichen Standards für KI im Gesundheitswesen

**Herausforderungen:**  
⚠ Zugang zu ausreichend diversifizierten Trainingsdaten  
⚠ Akzeptanz von KI in der Medizin und Vertrauen der Nutzer

### Persona 6

**Markus Schmidt (Systemadministrator und Modelltrainer)**

• **Alter:** 40 Jahre  
• **Wohnort:** Frankfurt, Deutschland   
• **Einkommen:** 70.000€  
• **Beruf:** Systemadministrator, Spezialist für Modelltraining und -überwachung im MedTech-Bereich  
• **Interessen:** IT-Infrastruktur, maschinelles Lernen, Big Data, Cybersicherheit  
• **Eigenschaften:** Detailorientiert, lösungsorientiert, analytisch  
• **Besonderheit:** Verantwortlich für die Verwaltung der IT-Infrastruktur, das Training und die Optimierung von KI-Modellen sowie für die technische Auswertung der Ergebnisse

**Ziele & Bedürfnisse:**  
✔ Effizientes Training von KI-Modellen mit kontinuierlicher Verbesserung der Genauigkeit  
✔ Gewährleistung der Skalierbarkeit und Performance der Infrastruktur für KI-Anwendungen  
✔ Bereitstellung klarer und benutzerfreundlicher Anweisungen und Anleitungen für die Endbenutzer  
✔ Implementierung und Überwachung der Datenschutzstandards gemäß gesetzlichen Vorgaben

**Herausforderungen:**  
⚠ Sicherstellung der Datenqualität und -sicherheit während des Trainingsprozesses  
⚠ Komplexität der Modellintegration und der Schnittstellen für Endanwender  
⚠ Anpassung der technischen Anleitungen für unterschiedliche Nutzergruppen (Ärzte, Patienten, medizinisches Personal)  
⚠ Mangel an ausreichend qualitativ hochwertigen und vielfältigen Trainingsdaten

## Funktionale Anforderungen

In diesem Kapitel werden die funktionalen Anforderungen des Projekts detailliert beschrieben. Dazu gehören die verschiedenen Use Cases sowie die spezifischen Anwendungsmöglichkeiten, die sowohl den Endnutzern als auch den Administratoren zur Verfügung stehen. Die dargestellten Anforderungen definieren die wesentlichen Funktionen des Systems und legen fest, welche Interaktionen zwischen den Nutzern und der Anwendung möglich sind.

Ein besonderer Fokus liegt dabei auf der Nutzerfreundlichkeit, der Effizienz der Systemprozesse sowie der klaren Abgrenzung der Berechtigungen zwischen den verschiedenen Nutzergruppen. Während Endnutzer beispielsweise bestimmte Kernfunktionen zur Nutzung des Systems erhalten, verfügen Administratoren über erweiterte Rechte zur Konfiguration, Verwaltung und Überwachung der Anwendung.

Die in diesem Kapitel beschriebenen funktionalen Anforderungen dienen als Grundlage für die Entwicklung und Implementierung des Systems und gewährleisten, dass alle notwendigen Features und Anwendungsfälle in der finalen Lösung berücksichtigt werden.

### Use Cases

Die folgenden Use Cases, also Interaktionsmöglichkeiten mit dem System, werden durch die Implementierung der Software abgedeckt. Sie beschreiben die verschiedenen Szenarien, in denen sowohl Anwender als auch Administratoren mit dem System arbeiten und bestimmte Funktionen ausführen können.

#### Anwender

Endnutzer des Systems haben Zugriff auf eine Reihe von Funktionen, die ihnen die Nutzung der Software ermöglichen. Diese Use Cases beschreiben typische Interaktionen, die Anwender im Rahmen ihrer Nutzung durchführen können.

##### Registrierung eines neuen Benutzers

**Akteure:**  
Benutzer (Neukunde), Anwendung (Registrierungssystem), Server (Datenbank für Nutzerdaten)

**Auslöser/Trigger-Event:**  
Der Benutzer startet die Anwendung und möchte sich mit Nutzernamen registrieren.

**Kurzbeschreibung:**  
Ein neuer Benutzer legt ein Konto an, um Zugriff auf die Funktionen der Anwendung zu erhalten.

**Beschreibung der einzelnen Schritte:**

1. Der Benutzer öffnet die Anwendung und wählt „Registrieren“.
2. Der Benutzer gibt persönliche Daten (E-Mail, Passwort) ein.
3. Die Anwendung überprüft die Eingaben auf Korrektheit und Vollständigkeit.
4. Die Registrierungsdaten werden an den Server gesendet und dort gespeichert.
5. Der Benutzer erhält eine Bestätigung über die erfolgreiche Registrierung.

**Beschreibung alternativer Schritte:**

* Falls die E-Mail bereits existiert, erhält der Benutzer eine Fehlermeldung und muss eine andere wählen.
* Falls das Passwort nicht den Anforderungen entspricht, wird eine entsprechende Meldung angezeigt.

**Vor- und Nachbedingungen:**  
Vorbedingung: Der Benutzer hat keinen bestehenden Account.  
Nachbedingung: Der Benutzer kann sich mit seinen Zugangsdaten anmelden.

**Systemgrenzen:**  
Anwendung, Server, Gateway, Internetverbindung

##### Anmeldung eines bestehenden Benutzers

**Akteure:**  
Benutzer (bestehender Kunde), Anwendung (Login-System), Server (Datenbank für Nutzerdaten)

**Auslöser/Trigger-Event:**  
Der Benutzer startet die Anwendung und möchte sich anmelden.

**Kurzbeschreibung:**  
Ein registrierter Benutzer gibt seine Zugangsdaten ein, um Zugriff auf sein Konto und die Analysefunktionen zu erhalten.

**Beschreibung der einzelnen Schritte:**

1. Der Benutzer öffnet die Anwendung und wählt „Anmelden“.
2. Der Benutzer gibt seine E-Mail-Adresse und sein Passwort ein.
3. Die Anwendung sendet die Daten an den Server zur Überprüfung.
4. Falls die Anmeldedaten korrekt sind, wird der Benutzer eingeloggt und auf das Dashboard weitergeleitet.

**Beschreibung alternativer Schritte:**

* Falls die E-Mail oder das Passwort falsch ist, wird eine Fehlermeldung ausgegeben.
* Falls der Benutzer sein Passwort vergessen hat, kann er eine „Passwort zurücksetzen“-Funktion nutzen.

**Vor- und Nachbedingungen:**  
Vorbedingung: Der Benutzer muss ein bestehendes Konto besitzen.  
Nachbedingung: Der Benutzer ist erfolgreich eingeloggt und kann auf seine Daten zugreifen.

**Systemgrenzen:**  
Anwendung, Server, Gateway, Internetverbindung

##### Einblick in vergangene Analysen

**Akteure:**  
Benutzer, Anwendung, Server (Datenbank mit analysierten Bildern)

**Auslöser/Trigger-Event:**  
Der Benutzer möchte eine Übersicht über seine bisher analysierten Hautläsionen einsehen.

**Kurzbeschreibung:**  
Der Benutzer kann eine Liste aller von der KI analysierten Bilder abrufen und Details zu den Analysen einsehen.

**Beschreibung der einzelnen Schritte:**

1. Der Benutzer öffnet die Anwendung und navigiert zum Bereich „Analysehistorie“.
2. Die Anwendung sendet eine Anfrage an den Server, um die analysierten Bilder abzurufen.
3. Die Bilder und zugehörigen Diagnosedaten werden angezeigt.
4. Der Benutzer kann einzelne Bilder auswählen, um detaillierte Informationen zu erhalten.

**Beschreibung alternativer Schritte:**

* Falls keine Bilder gespeichert sind, wird eine Meldung angezeigt, dass noch keine Analysen durchgeführt wurden.

**Vor- und Nachbedingungen:**  
Vorbedingung: Der Benutzer hat bereits mindestens ein Bild analysiert.  
Nachbedingung: Der Benutzer kann frühere Analysen einsehen.

**Systemgrenzen:**  
Anwendung, Server, Gateway, Internetverbindung

##### Bild analysieren lassen

**Akteure:**  
Benutzer, Anwendung (Kamera & Analysemodul), KI-Komponente

**Auslöser/Trigger-Event:**  
Der Benutzer möchte ein Bild seiner Hautläsion aufnehmen und analysieren lassen.

**Kurzbeschreibung:**  
Der Benutzer kann mit der Kamera seines Geräts ein Bild aufnehmen und ein KI-Modell seiner Wahl zur Analyse nutzen. Falls das gewünschte Modell nicht verfügbar ist, wird eine entsprechende Mitteilung ausgegeben. Außerdem kann er die Ergebnisse speichern lassen.

**Beschreibung der einzelnen Schritte:**

1. Der Benutzer öffnet die Anwendung und wählt „Neue Analyse starten“.
2. Die Anwendung fragt die verfügbaren Modelle von der KI-Komponente ab.
3. Die Anwendung aktiviert die Kamera, und der Benutzer nimmt ein Foto auf.
4. Die Anwendung zeigt eine Vorschau des Bildes an.
5. Der Benutzer bestätigt das Bild oder nimmt ein neues auf.
6. Der Benutzer wählt ein KI-Modell für die Analyse aus.
7. Falls das gewählte Modell nicht verfügbar ist (z. B. nicht trainiert), erhält der Benutzer eine Mitteilung und kann ein alternatives Modell auswählen.
8. Die Anwendung sendet das Bild an den Server zur Analyse mit dem gewählten Modell.
9. Die KI analysiert das Bild und sendet das Ergebnis an die Anwendung.
10. Der Benutzer sieht die Analyseergebnisse auf seinem Bildschirm.
11. Der Benutzer wird gefragt, ob die Analyse in seiner Historie gespeichert werden soll.

**Beschreibung alternativer Schritte:**

* Falls das ausgewählte KI-Modell nicht verfügbar ist, wird eine Benachrichtigung ausgegeben, und der Benutzer kann ein anderes Modell auswählen.

**Vor- und Nachbedingungen:**

Vorbedingung: Der Benutzer hat Zugriff auf eine Kamera und die Anwendung verfügt mindestens ein trainiertes KI-Modell.

Nachbedingung: Die Analyseergebnisse werden auf dem Bildschirm angezeigt.

**Systemgrenzen:**  
Anwendung, Kamera, KI-Komponente, Internetverbindung

##### Automatische Bildzuschnitt-Funktion

**Akteure:**  
Benutzer, Anwendung (Bildverarbeitungssystem), Gateway, Server (Zuschnitt-Funktion)

**Auslöser/Trigger-Event:**  
Der Benutzer hat ein Bild aufgenommen und die automatische Zuschnitt-Funktion wird angeboten.

**Kurzbeschreibung:**  
Nach der Aufnahme kann die Anwendung das Bild automatisch zuschneiden, um die relevante Hautstelle optimal für die Analyse vorzubereiten.

**Beschreibung der einzelnen Schritte:**

1. Der Benutzer nimmt ein Bild auf.
2. Die Anwendung bietet an, das Bild automatisch zuzuschneiden.
3. Der Benutzer kann die automatische Anpassung akzeptieren oder das Bild manuell zuschneiden.
4. Falls akzeptiert, verarbeitet der Server das Bild und schneidet es entsprechend zu.
5. Der Benutzer kann nun das Ergebnis betrachten und es akzeptieren oder selbst zuschneiden.
6. Das zugeschnittene Bild wird zur Analyse an die KI gesendet.

**Beschreibung alternativer Schritte:**

* Falls der Benutzer das automatische Zuschneiden ablehnt, kann er das Bild manuell anpassen.
* Falls dem Benutzer das Ergebnis der Zuschneide-Funktion nicht gefällt, kann er das Bild manuell anpassen.

**Vor- und Nachbedingungen:**  
Vorbedingung: Der Benutzer hat ein Bild aufgenommen.  
Nachbedingung: Das Bild ist zugeschnitten und bereit zur Analyse.

**Systemgrenzen:**  
Anwendung, Kamera, Server, Gateway, Internetverbindung

##### Informationen zu verfügbaren KI-Modellen abrufen

**Akteure:**  
Benutzer, Anwendung (Modul zur Modellinformation)

**Auslöser/Trigger-Event:**  
Der Benutzer möchte sich über die verfügbaren KI-Modelle und deren Funktionalitäten informieren.

**Kurzbeschreibung:**  
Der Benutzer kann eine Übersicht aller KI-Modelle aufrufen und sich Details zu deren Funktionsweise, Training und Anwendungsgebieten anzeigen lassen.

**Beschreibung der einzelnen Schritte:**

1. Der Benutzer öffnet die Anwendung und navigiert zum Bereich „KI-Modelle“.
2. Die Anwendung ruft die Liste der verfügbaren Modelle vom Server ab.
3. Die Anwendung zeigt die verfügbaren Modelle mit einer kurzen Beschreibung an.

**Vor- und Nachbedingungen:**

Vorbedingung: Der Benutzer hat Zugriff auf die Anwendung und der Server enthält Informationen über die KI-Modelle.

Nachbedingung: Der Benutzer hat sich über die verfügbaren KI-Modelle informiert und kann fundierte Entscheidungen über deren Nutzung treffen.

**Systemgrenzen:**  
Anwendung, KI-Komponente, Internetverbindung

#### Administrator

Administratoren verfügen über erweiterte Rechte innerhalb des Systems. Sie haben die Möglichkeit, Konfigurationen vorzunehmen, Nutzer zu verwalten und Systemprozesse zu überwachen. Die folgenden Use Cases beschreiben spezifische Interaktionen, die für Administratoren relevant sind.

##### KI-Modelle trainieren

**Akteure:**  
Administrator, Anwendung (Trainingsmodul), KI-Komponente, Datenbank

**Auslöser/Trigger-Event:**  
Der Administrator startet das Training eines spezifischen Modells oder aller Modelle.

**Kurzbeschreibung:**  
Der Administrator kann ein oder mehrere KI-Modelle trainieren. Die neuesten Datensätze werden aus der Datenbank abgerufen und zum Training verwendet. Das Training läuft auch bei Fehlern weiter, wobei fehlerhafte Modelle in einer Liste gespeichert und dem Administrator gemeldet werden.

**Beschreibung der einzelnen Schritte:**

1. Der Administrator öffnet die Anwendung und navigiert zur Modellverwaltung.
2. Er wählt aus, ob ein bestimmtes Modell oder alle Modelle trainiert werden sollen.
3. Der Administrator gibt die gewünschten Trainingsparameter ein:
   * **Resize Shape:** Zielgröße der Bilder für das Training
   * **Epochen:** Anzahl der Trainingsdurchläufe
4. Die Anwendung prüft, ob Benutzer berichtigt ist, Modelle zu trainieren.
5. Die Anwendung ruft die neuesten Datensätze aus der Datenbank ab.
6. Das Training startet für das ausgewählte Modell bzw. für alle Modelle.
7. Falls ein Modell nicht trainiert werden kann (z. B. aufgrund fehlerhafter Daten oder falscher Konfigurationen), wird dies in einer Fehlerliste gespeichert.
8. Das Training setzt mit dem nächsten Modell fort, ohne dass der Prozess abgebrochen wird.
9. Nach Abschluss des Trainings erhält der Administrator eine Übersicht über den Status der fehlerhaften Modelle (andere wurden erfolgreich trainiert).

**Beschreibung alternativer Schritte:**

* Falls der Administrator falsche Parameter eingibt (z. B. ungültiges Resize Shape oder Epochenzahl), wird eine Fehlermeldung ausgegeben, und er kann die Eingaben korrigieren.
* Falls der Benutzer nicht berechtigt ist, Modelle zu trainieren, wird eine Fehlermeldung angezeigt.

**Vor- und Nachbedingungen:**

Vorbedingung: Die Datenbank enthält Trainingsdaten.

Nachbedingung: Die gewählten Modelle wurden trainiert sowie gespeichert, und eine Fehlerliste wurde für nicht trainierbare Modelle erstellt.

**Systemgrenzen:**  
Anwendung, Server, Datenbank, Gateway, KI-Komponente, Internetverbindung

##### Classifier-Reports abrufen

**Akteure:**  
Administrator, Anwendung (Evaluierungsmodul), Server, KI-Komponente, Datenbank

**Auslöser/Trigger-Event:**  
Der Administrator möchte die Klassifikationsleistung eines spezifischen Modells oder aller Modelle anhand des aktuellen Datensatzes überprüfen.

**Kurzbeschreibung:**  
Der Administrator kann einen Klassifikationsbericht (Classifier Report) abrufen, um die Genauigkeit und Leistung eines oder aller Modelle zu analysieren. Dabei wird das Training ähnlich wie beim normalen Training durchgeführt, jedoch ohne Speicherung der trainierten Modelle. Stattdessen wird ein Report basierend auf 20 % Testdaten des gesamten Datenausmaßes ausgegeben.

**Beschreibung der einzelnen Schritte:**

1. Der Administrator öffnet die Anwendung und navigiert zur Modellbewertung.
2. Er wählt aus, ob ein spezifisches Modell oder alle Modelle analysiert werden sollen.
3. Die Anwendung prüft, ob der Benutzer berechtigt ist, Klassifikationsberichte erstellen zu lassen
4. Die Anwendung ruft die neuesten Datensätze aus der Datenbank ab.
5. Das Modell/die Modelle werden mit den aktuellen Daten getestet, ohne die trainierten Modelle zu speichern.
6. Die Anwendung verwendet 80 % der Daten für das Training und 20 % für die Evaluierung.
7. Nach der Evaluierung wird ein Klassifikationsbericht erstellt, der Informationen zu folgenden Metriken enthält:
   * **Präzision** (Precision)
   * **Trefferquote** (Recall)
   * **F1-Score**
   * **Genauigkeit** (Accuracy)
8. Der Report wird dem Administrator angezeigt und kann exportiert werden.

**Beschreibung alternativer Schritte:**

* Falls keine ausreichenden Daten für die Evaluierung vorhanden sind, wird eine Warnmeldung ausgegeben.
* Falls ein Modell fehlschlägt, wird dies in einer Fehlerliste gespeichert, und die Evaluierung fährt mit dem nächsten Modell fort.

**Vor- und Nachbedingungen:**

Vorbedingung: Die Anwendung hat Zugriff auf die neuesten Datensätze.

Nachbedingung: Der Administrator erhält einen detaillierten Report zur Modellleistung.

**Systemgrenzen:**  
Anwendung, Server, Datenbank, Gateway, KI-Komponente, Internetverbindung

## Nicht Funktionale Anforderungen

Dieses Kapitel enthält die Anforderungen, die an die Anwendung gestellt werden, welche nicht die Funktionalität des Systems betreffen. Diese definieren, abgekapselt von jeglicher Implementierungslogik und verwendeten Methoden, die Rahmenbedingungen nicht nur für den Endnutzer und seine Erfahrungen mit der entwickelten Software sondern auch jene für Entwickler/Administratoren.

### Benutzerfreundlichkeit

Die mobile Anwendung sollte intuitiv gestaltet sein, um eine einfache Bedienung für Benutzer aller Erfahrungsstufen zu gewährleisten. Außerdem sollte die Reaktionszeit des Servers sowie die Latenz zwischen Frontend und Backend so gering wie möglich sein, um ein reibungsloses und ansprechendes Benutzererlebnis zu ermöglichen.

### Zuverlässigkeit

Es ist ein äußert wichtiger Punkt, dass die Verbindung zwischen Frontend und Backend zu allen Zeiten voll funktional ist und verlässlich Benutzeranfragen entgegennimmt. Weiters soll das System dahingehend robust sein, dass es angemessen auf Ausnahmesituationen reagiert, um einen kontinuierlichen Betrieb auch bei unvorhergesehenen Ereignissen sicherzustellen.

### Skalierbarkeit

Es ist gewünscht, das System so zu gestalten, dass es auch in der Zukunft liegende Erweiterungen und zusätzliche Änderung ohne großen Aufwand unterstützt. Die Software sollte die Funktionalität bereitstellen, dem Benutzer mehrere Algorithmen zur Erkennung der Hautläsionen zur Verfügung zu stellen, ohne dabei die Perfomance oder Stabilität des Systems zu beeinträchtigen.

### Sicherheit

Sowohl die Speicherung als auch die Übertragung der benutzerbezogenen Daten (Email, Passwort, Bilder) sollte verschlüsselt vonstattengehen. Dafür werden Passwörter nur unter Benutzung eines Hashs persistiert und Übertragungen mittels eines geeigneten Algorithmus verschlüsselt.

### Plattformkompatibilität

Wie auch die Grundanforderungen des Projektes vorgeben, sollte die Software derartig unabhängig sein, dass der Benutzer sowohl in einer Windows-Umgebung als auch auf mobilen Endgeräten (Android) einsteigen kann. Dabei ist die Responsivität der Software von großer Wichtigkeit, um eine konsistente Darstellung zu jeder Zeit zu gewährleisten.

### Dokumentation und Wartbarkeit

Das Projekt sollte ausführlich dokumentiert sein, einschließlich Anleitungen zur Installation, Konfiguration und Verwendung der Steuerungssoftware. Dazu gehören auch eine gute Strukturierung sowie Kommentierung des Quellcodes, um eine einfache Wartung und Weiterentwicklung des Systems zu ermöglichen.

# Projektplanung

*Kapitel verfasst von Daniel Jessner*

Die erfolgreiche Umsetzung eines Projekts erfordert eine strukturierte und detaillierte Planung, die sowohl technische als auch organisatorische Aspekte berücksichtigt. In diesem Kapitel werden die verschiedenen Planungsschritte erläutert, die für die Entwicklung des Projekts durchgeführt wurden.

## Recherche / Vorarbeit

Bevor mit der eigentlichen Implementierung des Projekts begonnen werden konnte, war eine umfassende Recherche und theoretische Vorarbeit erforderlich. Jeder Projektteilnehmer hat sich mit spezifischen Fachthemen auseinandergesetzt, um eine solide Grundlage für die Entwicklung zu schaffen. Diese Vorbereitungen umfassten sowohl technische als auch konzeptionelle Aspekte, darunter die Analyse relevanter Technologien, wissenschaftlicher Grundlagen sowie bestehender Lösungen und Frameworks.

Die Recherche diente nicht nur der Wissensaneignung, sondern auch der Identifikation möglicher Herausforderungen und der Evaluierung geeigneter Methoden und Werkzeuge. Dadurch konnte sichergestellt werden, dass fundierte Entscheidungen für die Umsetzung des Projekts getroffen wurden. In den folgenden Abschnitten werden die einzelnen Beiträge der Projektteilnehmer detailliert erläutert.

### Daniel Jessner (KI-Komponente, Gateway)

*Kapitel verfasst von Daniel Jessner*

#### Einführung

Bevor mit dem eigentlichen Training von KI-Modellen begonnen werden kann, ist eine gründliche Recherche und Vorarbeit essenziell. Eine fundierte theoretische Grundlage ermöglicht es, die geeigneten Algorithmen und Frameworks auszuwählen sowie potenzielle Herausforderungen frühzeitig zu erkennen. Zudem hilft die Recherche dabei, bewährte Methoden zur Datenaufbereitung, Modelloptimierung und Fehlervermeidung zu identifizieren. Da sich das Projektteam mit dieser Arbeit in ein ihnen unbekanntes Terrain wagt, ist dieser Schritt unerlässlich für den Erfolg des Projektes.

Die Vorarbeit umfasst mehrere wichtige Schritte, die die Qualität und Effizienz des Modelltrainings maßgeblich beeinflussen. Dazu gehören die Auswahl geeigneter Technologien, die Beschaffung und Bereinigung von Trainingsdaten (eigene Projekt-Komponente) sowie erste Tests mit Basisalgorithmen. Durch eine systematische Vorgehensweise wird sichergestellt, dass das spätere Modelltraining auf einer stabilen und gut vorbereiteten Grundlage aufbaut.

#### Theoretische Grundlagen – KI

Diese Projekt-Komponente beschäftigt sich zum größten Teil mit der Verwaltung und auch Entwicklung von KI-Modellen, um Eingabedaten anhand zuvor definierter Trainingsdaten zu klassifizieren. Dabei bedient sie sich sogenannten „**Classifiers**“, also KI-Modellen, welche ein solches Konzept realisieren. Diese Classifier sind definierte, oftmals komplexe Algorithmen und basieren auf zwei Ansätzen:

1. **Maschinelles Lernen** ist ein Bereich der Künstlichen Intelligenz, der es Computern ermöglicht, Muster in Daten zu erkennen und Vorhersagen zu treffen, ohne explizit programmiert zu sein. Dabei unterscheidet man zwischen verschiedenen Lernarten:

* **Überwachtes Lernen**: Das Modell wird mit gekennzeichneten Daten (Labels) trainiert, um eine bestimmte Ausgabe vorherzusagen (relevant für dieses Projekt).
* **Unüberwachtes Lernen**: Das Modell erkennt Muster in unmarkierten Daten, z. B. durch Clustering.
* **Bestärkendes Lernen**: Das Modell lernt durch Belohnungen aus Interaktionen mit seiner Umgebung.

2. **Deep Learning** ist eine spezielle Form des maschinellen Lernens, die auf künstlichen neuronalen Netzen basiert. Diese Netzwerke bestehen aus mehreren Schichten (Deep Neural Networks) und sind besonders gut für komplexe Mustererkennung in großen Datenmengen geeignet. Im Vergleich zu klassischen ML-Algorithmen können DL-Modelle automatisch relevante Merkmale aus den Daten extrahieren, was sie für Aufgaben wie Bild- und Spracherkennung besonders leistungsfähig macht. Dieser Ansatz des maschinellen Lernens wird von CNNs, also **Convolutional Neural Networks**, implementiert, worauf später noch genauer eingegangen wird.

#### Auswahl geeigneter Technologien

Grundsätzlich war von Anfang an klar, dass sich im Bezug auf verwendete Technologien sowie das Basiskonstrukt des Projektes an die Vorgaben des Tech-Stacks des Projektgebers (siehe Kapitel XY) gehalten wird. Dadurch fiel die Entscheidung auf relativ einfach auf folgende Punkte:

* **KI-Modelle**
  + Pytorch
  + Tensorflow Keras
  + Scikit-Learn
* **API**
  + FastAPI

Abgesehen von den Vorgaben sprechen noch einige weitere Aspekte für die Verwendung oben genannter Technologien:

**1. Python**

Python ist die bevorzugte Programmiersprache für Künstliche Intelligenz und maschinelles Lernen aufgrund seiner einfachen Syntax, umfangreichen Bibliotheken und starken Community-Unterstützung.

Die wichtigsten Vorteile sind:

* **Lesbarkeit und Benutzerfreundlichkeit**: Ermöglicht eine schnelle Entwicklung und erleichtert die Zusammenarbeit.
* **Breites Ökosystem an ML- und DL-Bibliotheken**: Bibliotheken wie **Scikit-Learn, TensorFlow und PyTorch** bieten leistungsfähige Tools für maschinelles Lernen und Deep Learning.
* **Effiziente Datenverarbeitung**: Python unterstützt leistungsfähige Bibliotheken wie **NumPy, Pandas und Matplotlib**, die für Datenanalyse und Visualisierung essenziell sind.
* **Gute Integration mit anderen Technologien**: Python kann leicht mit **C++, Java und cloudbasierten ML-Plattformen** integriert werden, was die Skalierbarkeit von KI-Modellen verbessert.

**2. KI-Bibliotheken**

Je nach Anwendungsfall eignen sich Frameworks unterschiedlich gut für zu bewältigende Aufgaben. Die KI-Komponente des DermaAI-Systems implementiert einen vielfältigen Mix von Modellen aus den drei verwendeten Bibliotheken.

|  |  |  |  |
| --- | --- | --- | --- |
| **Framework** | **Einsatzgebiet** | **Vorteile** | **Nachteile** |
| |  | | --- | | **Scikit-Learn** | | |  | | --- | | Klassische ML-Modelle (z. B. Entscheidungsbäume, SVMs, logistische Regression) | | Einfache Implementierung, gute Dokumentation, ideal für kleine bis mittelgroße Datensätze | Nicht für neuronale Netze oder große Datenmengen optimiert |
| |  | | --- | | **PyTorch** | | |  | | --- | | Deep Learning, flexible neuronale Netze (z. B. CNNs, RNNs, Transformer) | | |  | | --- | | Dynamische Berechnungsgrafen, einfaches Debugging, intuitive API | | Etwas weniger für großskalige Produktlösungen optimiert als TensorFlow |
| |  | | --- | | **TensorFlow** | | Skalierbares Deep Learning für Produktion, Cloud und Mobilgeräte | Hohe Effizienz, TensorFlow Serving für Deployment, GPU-Unterstützung | Komplexere API als PyTorch, steilere Lernkurve |

**-> Scikit-Learn** eignet sich besonders für klassische ML-Modelle wie Klassifikation, Regression oder Clustering.  
**-> PyTorch** ist ideal für forschungsorientierte Deep-Learning-Experimente, da es eine flexible und intuitive API bietet.  
**-> TensorFlow** ist besser für großskalige neuronale Netze in Produktionsumgebungen, insbesondere für Cloud- und Mobile-Deployments.

**3. FastAPI**

**FastAPI** ist ein modernes und leistungsstarkes Web-Framework für den Bau von APIs mit Python. Es hat einige herausragende Merkmale, die es besonders attraktiv für Entwickler machen, die APIs bauen möchten:

**Hohe Leistung**

* **FastAPI** ist eines der schnellsten Web-Frameworks in Python und bietet eine außergewöhnliche Leistung, die mit **Node.js** und **Go** konkurrieren kann. Die hohe Geschwindigkeit ist ein wichtiger Faktor, wenn es um die Entwicklung von APIs geht, da sie eine geringe Latenz und schnelle Antworten gewährleisten kann.
* Der Grund für die Geschwindigkeit ist, dass **FastAPI** auf **Starlette** basiert, einem schnellen und asynchronen Web-Framework. FastAPI verwendet die **ASGI**-Spezifikation, die eine hohe Performance bei gleichzeitigen Verbindungen ermöglicht.

**Automatische Dokumentation**

* FastAPI erstellt automatisch interaktive API-Dokumentationen für deine Endpunkte, und zwar direkt während der Entwicklung.
* Es verwendet **Swagger UI** und **ReDoc**, um eine benutzerfreundliche, visuelle API-Dokumentation bereitzustellen. Diese wird unter der URL /docs (Swagger UI) und /redoc (ReDoc) bereitgestellt.

**Vorteile**:

* + Entwicklern und API-Nutzern wird eine leicht verständliche, sofort zugängliche Dokumentation zur Verfügung gestellt.
  + Die Dokumentation ist immer aktuell, da sie auf der tatsächlichen Implementierung der API basiert.

**Einfach zu bedienen und flexibel**

* FastAPI nutzt Python-Typannotationen (zum Beispiel int, str, float, etc.), um Eingaben zu validieren, was die Entwicklung vereinfacht und weniger fehleranfällig macht. Das bedeutet, dass du keine separate Validierungslogik schreiben musst, sondern die Typen automatisch validiert werden.
* Durch die Verwendung von **Pydantic** für die Datenvalidierung können Modelle sehr einfach und effizient erstellt werden. Pydantic sorgt für automatische Datentypprüfung und Fehlerbehandlung.

**Asynchronität (Async / Await)**

* FastAPI ermöglicht es, asynchrone Endpunkte zu erstellen, die bei Bedarf sehr viele gleichzeitige Anfragen effizient verarbeiten können. Das bedeutet, dass du **async**/**await** verwenden kannst, um **asynchrone Programmierung** zu integrieren und z. B. auf **Datenbanken** oder **Web-APIs** effizient zuzugreifen.
* Diese asynchronen Endpunkte bieten eine deutlich bessere Leistung bei hoher Last und stellen sicher, dass deine API schnell bleibt, selbst wenn viele gleichzeitige Anfragen verarbeitet werden müssen.

**Sicherheit und Authentifizierung**

* FastAPI bietet Unterstützung für gängige **Authentifizierungsmethoden** wie **OAuth2**, **JWT (JSON Web Tokens)**, und **API-Schlüssel**.
* Es gibt eingebaute Mechanismen, um sicherheitsrelevante Operationen wie **CORS (Cross-Origin Resource Sharing)**, **CSRF (Cross-Site Request Forgery)** und **Request Validation** effizient zu implementieren.

**Einfacher Einstieg und gute Dokumentation**

* FastAPI hat eine **exzellente Dokumentation**, die für Einsteiger gut verständlich ist. Die Dokumentation führt Schritt für Schritt durch den Prozess der API-Entwicklung und hilft dabei, Best Practices zu etablieren.
* Die Nutzung von Python-Typen für **Datenvalidierung** und **Autovervollständigung** in modernen IDEs wie **VS Code** erleichtert das Schreiben von Code.

**Unterstützung für moderne Python-Funktionen**

* FastAPI nutzt moderne Python-Funktionen wie **Type Hints** und **Pydantic-Modelle**. Dadurch kannst du sicherstellen, dass deine API effizient und fehlerfrei bleibt.
* Es unterstützt **Python 3.6+**, einschließlich der neuesten Versionen, die die neuesten Funktionen und Verbesserungen bieten.

**Leichtgewichtig und skalierbar**

* FastAPI ist leichtgewichtig, sodass es keine unnötige Komplexität oder Overhead mit sich bringt, aber gleichzeitig auch in der Lage ist, komplexe Anwendungen und Skalierung zu bewältigen.
* FastAPI kann man für kleine Projekte bis hin zu großen, skalierbaren APIs verwenden.

**Open-Source und Community**

* FastAPI ist **Open-Source**, was bedeutet, dass du jederzeit den Quellcode einsehen und zur Weiterentwicklung beitragen kannst.
* Es gibt eine aktive Community, die regelmäßig neue Funktionen, Plugins und Erweiterungen entwickelt.

**Sehr gute Integration mit anderen Technologien**

* FastAPI lässt sich sehr gut mit anderen Tools und Frameworks kombinieren, wie zum Beispiel **SQLAlchemy** (für die Datenbankinteraktion), **Celery** (für Hintergrundaufgaben), **Redis** (für Caching und Messaging) und vielen weiteren.
* FastAPI kann man problemlos mit anderen modernen Python-Bibliotheken und Frameworks verwenden.

#### Installation KI-Modelle

Um im Anschluss die KI-Bibliotheken verwenden zu können, müssen diese zuvor installiert werden. Dazu wird ganz einfach der für Python entwickelte Package Manager „**PIP, packager installer python**“ verwendet, die Bibliotheken werden automatisch aus dem Internet geladen:

* **Scikit-Learn**

****

* **Pytorch**
  + **Ohne GPU-Untersützung (CPU-Only)**

****

* + **Mit GPU-Unterstützung**

****

* **TensorFlow**
  + **Ohne GPU-Untersützung (CPU-Only)**

****

* + **Mit GPU-Unterstützung**

****

**1. Scikit-Learn (scikit-learn)**

* **Was wird installiert?**
  + **Scikit-Learn** ist eine weit verbreitete Bibliothek für maschinelles Lernen in Python. Sie stellt eine Sammlung von Werkzeugen für viele gängige maschinelle Lernverfahren zur Verfügung, wie Klassifikation, Regression, Clustering und Dimensionalitätsreduktion.
* **Was wird durch die Installation bereitgestellt?**
  + Funktionen zur Modellbildung, wie **lineare Regression, Support Vector Machines (SVM), K-nearest Neighbors (KNN), Random Forest** und viele andere klassische ML-Algorithmen.
  + Funktionen zur **Datenvorverarbeitung** (z. B. Skalierung von Merkmalen, Umgang mit fehlenden Werten).
  + **Kreuzvalidierung** und **Hyperparameter-Tuning**.

**2. PyTorch (torch)**

* **Was wird installiert?**
  + **PyTorch** ist ein Deep-Learning-Framework, das es Entwicklern ermöglicht, neuronale Netze zu bauen und zu trainieren. Die Bibliothek enthält Funktionen zum Arbeiten mit **Tensors** (mehrdimensionalen Arrays) und zur **Automatischen Differenzierung** (für Backpropagation und Gradientenberechnung).
* **Was wird durch die Installation bereitgestellt?**
  + Grundlegende Funktionen für Deep Learning, einschließlich **Tensor-Operationen** (z. B. Addition, Multiplikation).
  + **Autograd**: Automatische Differenzierung, um Gradienten für das Training von neuronalen Netzen zu berechnen.
  + Unterstützung für **GPU-Computing** (durch CUDA) und **Datenparallele Verarbeitung**.
  + Flexibilität, um benutzerdefinierte **neuronale Netzwerke** zu erstellen und zu trainieren.

**3. torchvision**

* **Was wird installiert?**
  + **torchvision** ist eine Erweiterung von **PyTorch**, die speziell auf **Bildverarbeitung** fokussiert ist. Es enthält Tools und vortrainierte Modelle für Computer Vision Aufgaben wie Bildklassifikation, Objekterkennung und Segmentierung.
* **Was wird durch die Installation bereitgestellt?**
  + Vorverarbeitungsfunktionen für Bilder wie **Zuschneiden**, **Normalisierung** und **Skalierung**.
  + Vortrainierte **CNN-Modelle** (Convolutional Neural Networks), wie **ResNet**, **VGG** und **AlexNet**.
  + **Daten-Datasets** wie **CIFAR-10**, **ImageNet**, **COCO**.
  + Methoden zur **Datenaugmentation** (z. B. Bilddrehen, Spiegeln), um die Trainingsdaten zu erweitern.

**4. cudatoolkit**

* **Was wird installiert?**
  + **CUDA-Toolkit** ist eine Sammlung von Tools und Bibliotheken, die von **NVIDIA** entwickelt wurden, um die Leistung von GPU-Computing zu nutzen. Es stellt die Schnittstellen und Bibliotheken zur Verfügung, die erforderlich sind, damit Software wie **TensorFlow** oder **PyTorch** die **GPU** zur Berechnung von rechenintensiven Aufgaben (wie das Training von neuronalen Netzen) verwenden kann.
* **Was wird durch die Installation bereitgestellt?**
  + **CUDA-Bibliotheken** und Tools, die eine schnelle Verarbeitung durch die **NVIDIA-GPU** ermöglichen.
  + Optimierung und Beschleunigung von Deep-Learning-Modellen.
  + Verbindung zwischen der Software und der GPU für **parallelisierte Berechnungen**.
* **Hinweis**: Um GPU-Computing zu nutzen, muss man sicherstellen, dass die verwendete Grafikkarte und Treiber mit CUDA kompatibel sind.

**5. TensorFlow**

* **Was wird installiert?**
  + **TensorFlow** ist ein Open-Source-Deep-Learning-Framework von **Google**, das für maschinelles Lernen und neuronale Netzwerke entwickelt wurde. Es unterstützt sowohl **CPU** als auch **GPU** und eignet sich für **Forschung** und **Produktion**.
* **Was wird durch die Installation bereitgestellt?**
  + **Modelle und Tools** für maschinelles Lernen und Deep Learning, wie **Klassifikation**, **Regressionsmodelle**, **Textanalyse**, **Bildverarbeitung** und **Sprachverarbeitung**.
  + **TensorFlow Serving** und **TensorFlow Lite** für den **Produktionsbetrieb** (z. B. Bereitstellung auf Servern oder mobilen Geräten).
  + Optimierung der **Modell-Performance** (einschließlich GPU-Beschleunigung, **TensorFlow Extended (TFX)** für Produktionspipelines).

**6. tensorflow-gpu**

* **Was wird installiert?**
  + **tensorflow-gpu** ist die Version von **TensorFlow**, die **GPU-Unterstützung** integriert hat. Dies ermöglicht es, die **GPU** für das Training und die Berechnung von Deep-Learning-Modellen zu verwenden, was die Berechnungen deutlich beschleunigt.
* **Was wird durch die Installation bereitgestellt?**
  + **GPU-Unterstützung** für TensorFlow, was besonders bei großen Modellen und großen Datensätzen nützlich ist.
  + Die **CUDA**-Bibliotheken und Treiber werden mitinstalliert, sodass TensorFlow automatisch auf **NVIDIA-GPUs** zugreifen kann.
  + Schnelleres Training von Deep-Learning-Modellen im Vergleich zur **CPU-Version**, da die **GPU** für parallele Berechnungen verwendet wird.

#### Installation FastAPI

Um eine anständige Verbindung zum Frontend zu gewährleisten, wird FastAPI verwendet. Ebenso wie bei den Modellen wird dieses Modul über den Package Manager installiert:



Wichtig hierbei ist, dass FastAPI als Web-Framework auch einen ASGI-Server benötigt, um ausgeführt werden zu können. Uvicorn ist ein leichtgewichtiger ASGI-Server, der für FastAPI empfohlen wird:



**Erklärung ASGI:**

**ASGI (Asynchronous Server Gateway Interface)** ist eine Spezifikation für Web-Server und Python-Frameworks, die asynchrone Kommunikation unterstützt. Es ist die Weiterentwicklung des älteren WSGI (Web Server Gateway Interface), das die Grundlage für die meisten Python-Web-Frameworks bildet.

ASGI wurde entwickelt, um die Einschränkungen von WSGI zu überwinden und asynchrone Programmierung und parallele Verarbeitung von Anfragen zu ermöglichen. Dies ist besonders nützlich für Web-APIs, echte Echtzeit-Anwendungen wie Chat-Apps oder Spiele, sowie Anwendungen, die viele gleichzeitige Verbindungen benötigen.

#### Testexperiment

Am Anfang ist es wichtig, die Arbeit in diesem Bereich etwas kennenzulernen, ganz besonders wenn jemand noch nie zuvor damit in Berührung gekommen ist. Im Falle des DermaAI-Projektteams stellt die KI-Welt etwas Fremdes dar, daher wurde zu Beginn ein kleines Testexperiment mit Scikit-Learn implementiert, um den Umgang mit den Python-Bibliotheken zu verstehen.

Scikit-Learn stellt eine große Palette an unterschiedlichen KI-Algorithmen zur Verfügung:

Ein Bild, das Text, Screenshot, Grafikdesign, Design enthält.

Automatisch generierte Beschreibung

* + Es wird sich hier, wie schon erwähnt, auf die Classification-Algorithmen beschränkt

Anschließend wird ein Testskript in Python erstellt, um eine kleine Selektion der Scikit-Learn Algorithmen zu testen und deren Ergebnisse zu vergleichen (jegliche nicht definierte Funktionen sind aus dem Scikit-Learn-Framework oder anderen importiert):

**1. Classifier deklarieren**

Hier wird eine Liste von Klassifikatoren und ihre entsprechenden Namen erstellt. Diese Klassifikatoren werden später verwendet, um die trainierten Modelle zu evaluieren und die Entscheidungsgrenzen zu visualisieren.

Ein Bild, das Text, Screenshot, Schrift, Dokument enthält.

KI-generierte Inhalte können fehlerhaft sein.

* + **names** enthält die Namen der verschiedenen Klassifizierer, die im Code verwendet werden.
  + **classifiers** enthält die Instanzen der verschiedenen Klassifizierer aus sklearn

Diese Algorithmen sind auch Teil der fertigen DermaAI-Anwendung und werden im Kapitel XY mit ihren Parametern genauer erklärt.

**2. Datensätze generieren**

In diesem Schritt werden die Datensätze erstellt, die für das Training und Testen der Classifier verwendet werden. Die **make\_classification**, **make\_moons** und **make\_circles** Funktionen erzeugen synthetische Datensätze für Tests.

1. X, y = make\_classification(

2. n\_features=2, n\_redundant=0, n\_informative=2, random\_state=1, n\_clusters\_per\_class=1)

3. rng = np.random.RandomState(2)

4. X += 2 \* rng.uniform(size=X.shape)

5. linearly\_separable = (X, y)

6.

7. datasets = [

8. make\_moons(noise=0.3, random\_state=0),

9. make\_circles(noise=0.2, factor=0.5, random\_state=1),

10. linearly\_separable,

11. ]

12.

* + **make\_classification** erzeugt einen Datensatz für eine Klassifikationsaufgabe mit 2 informativen Features.
  + **make\_moons** und **make\_circles** erzeugen jeweils Datensätze, die für Klassifikationen von halbmond- oder kreisförmigen Daten verwendet werden.
  + **linearly\_separable** ist ein benutzerdefinierter Datensatz, der eine lineare Trennbarkeit aufweist.

**3. Erstellen einer Plot-Figur**

Hier wird die Plot-Figur mit einer bestimmten Größe erstellt, die später alle Subplots für die Datensätze und die Ergebnisse der Klassifikatoren enthalten wird.

1. figure = plt.figure(figsize=(27, 9))

2. i = 1

* + **plt.figure(figsize=(27, 9))** legt eine Abbildung fest, die Platz für Subplots bietet.
  + **i** ist der Index für die Subplots, der später bei der Zuordnung der Plot-Positionen verwendet wird:

**4. Plotten der Datensätze**

In diesem Schritt wird jeder der drei Datensätze geplottet, wobei die Trainings- und Testdaten durch unterschiedliche Farben unterschieden werden:

1. for ds\_cnt, ds in enumerate(datasets):

2. # preprocess dataset, split into training and test part

3. X, y = ds

4. X\_train, X\_test, y\_train, y\_test = train\_test\_split(

5. X, y, test\_size=0.4, random\_state=42

6. )

7.

8. x\_min, x\_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5

9. y\_min, y\_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

10.

11. cm = plt.cm.RdBu

12. cm\_bright = ListedColormap(["#FF0000", "#0000FF"])

13. ax = plt.subplot(len(datasets), len(classifiers) + 1, i)

14. if ds\_cnt == 0:

15. ax.set\_title("Input data")

16. ax.scatter(X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap=cm\_bright, edgecolors="k")

17. ax.scatter(

18. X\_test[:, 0], X\_test[:, 1], c=y\_test, cmap=cm\_bright, alpha=0.6, edgecolors="k"

19. )

20. ax.set\_xlim(x\_min, x\_max)

21. ax.set\_ylim(y\_min, y\_max)

22. ax.set\_xticks(())

23. ax.set\_yticks(())

24. i += 1

* + **train\_test\_split** teilt die Daten in Trainings- und Testdaten (60% Training und 40% Test).
  + Die **scatter** Methode wird verwendet, um die Datenpunkte zu visualisieren.
  + **x\_min, x\_max, y\_min, y\_max** definieren den Bereich der x- und y-Achse, um sicherzustellen, dass alle Daten sichtbar sind.

**5. Trainieren und Auswerten der Klassifikatoren**

Für jeden Klassifikator wird ein Pipeline-Objekt erstellt, das den StandardScaler für die Datenvorverarbeitung und den jeweiligen Klassifikator beinhaltet. Der Klassifikator wird trainiert und auf den Testdaten evaluiert:

1. for name, clf in zip(names, classifiers):

2. ax = plt.subplot(len(datasets), len(classifiers) + 1, i)

3.

4. clf = make\_pipeline(StandardScaler(), clf)

5. clf.fit(X\_train, y\_train)

6. score = clf.score(X\_test, y\_test)

7. DecisionBoundaryDisplay.from\_estimator(

8. clf, X, cmap=cm, alpha=0.8, ax=ax, eps=0.5

9. )

10.

11. ax.scatter(

12. X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap=cm\_bright, edgecolors="k"

13. )

14. ax.scatter(

15. X\_test[:, 0],

16. X\_test[:, 1],

17. c=y\_test,

18. cmap=cm\_bright,

19. edgecolors="k",

20. alpha=0.6,

21. )

22.

23. ax.set\_xlim(x\_min, x\_max)

24. ax.set\_ylim(y\_min, y\_max)

25. ax.set\_xticks(())

26. ax.set\_yticks(())

27. if ds\_cnt == 0:

28. ax.set\_title(name)

29. ax.text(

30. x\_max - 0.3,

31. y\_min + 0.3,

32. ("%.2f" % score).lstrip("0"),

33. size=15,

34. horizontalalignment="right",

35. )

36. i += 1

* + **make\_pipeline** erstellt eine Pipeline, die den **StandardScaler** zur Normalisierung der Eingabedaten und den Klassifikator selbst enthält.
  + **clf.fit(X\_train, y\_train)** trainiert den Klassifikator auf den Trainingsdaten.
  + **DecisionBoundaryDisplay.from\_estimator** zeigt die Entscheidungsgrenze des Klassifikators.
  + Die **ax.text** Methode zeigt die Genauigkeit des Klassifikators auf dem Plot an.

**6. Layout anpassen und Plot anzeigen**

1. plt.tight\_layout()

2. plt.show()

* + **plt.tight\_layout()** stellt sicher, dass alle Subplots ordentlich und ohne Überlappung angezeigt werden.
  + **plt.show()** zeigt den Plot auf dem Bildschirm an.

**Ergebnis:**

**Teil 1:**

**Ein Bild, das Screenshot, Kunst, Mosaik enthält.

KI-generierte Inhalte können fehlerhaft sein.**

**Teil 2:**

**Ein Bild, das Screenshot, Mosaik, Kunst, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.**

Das gezeigte Bild stellt eine Visualisierung verschiedener Klassifikationsalgorithmen dar, die auf drei verschiedene Datensätze angewendet wurden. Jeder Algorithmus versucht, eine Entscheidungsgrenze zu lernen, um die beiden Klassen (rot und blau) korrekt zu trennen.

**1. Aufbau der Visualisierung**

Die Darstellung besteht aus einer **Matrix von Plots**, wobei:

* **Jede Zeile** einen anderen Datensatz repräsentiert (von oben nach unten: Mondförmige Daten (make\_moons), Kreisförmige Daten (make\_circles), Linear trennbare Daten (make\_classification)).
* **Jede Spalte** einen anderen Klassifikationsalgorithmus repräsentiert (von links nach rechts: Unterschiedliche Methoden wie k-Nächste Nachbarn, SVMs, Entscheidungsbäume, neuronale Netze etc.).
* Das erste Bild in jeder Zeile zeigt den **rohen Datensatz**, also die ursprünglichen Punkte ohne eine Klassifikation.
* Die übrigen Bilder zeigen die **Entscheidungsgrenzen**, die durch die jeweiligen Algorithmen gelernt wurden. Die eingefärbten Regionen repräsentieren die Vorhersagen des jeweiligen Klassifikators:
  + **Rote Regionen** → Klassifikation als Klasse 1 (rot).
  + **Blaue Regionen** → Klassifikation als Klasse 2 (blau).
  + Die Zahlen in den Diagrammen repräsentieren die Genauigkeit (Accuracy) des jeweiligen Klassifikators auf dem Testdatensatz.

**2. Analyse der Algorithmen und Entscheidungsgrenzen**

**Nearest Neighbors (K-Nearest Neighbors, KNN)**

* **Charakteristik:** KNN basiert darauf, dass ein Punkt die gleiche Klasse wie seine **nächsten Nachbarn** annimmt.
* **Entscheidungsgrenze:** Sehr flexibel und passt sich stark an die Struktur der Daten an, daher eher **unregelmäßige Grenzen**.
* **Stärken:** Gute Leistung bei komplexen, nicht-linearen Datensätzen.
* **Schwächen:** Kann übermäßig an die Trainingsdaten angepasst sein (Overfitting), besonders bei verrauschten Daten.

**Linear SVM (Support Vector Machine mit linearem Kernel)**

* **Charakteristik:** Versucht, eine **lineare Trennlinie** zu finden, die die beiden Klassen optimal trennt.
* **Entscheidungsgrenze:** Gerade Linie – sehr gut für den linearen Datensatz (unterste Zeile), aber **ungeeignet für nicht-lineare Daten** (Mond- und Kreisförmige Daten).
* **Stärken:** Sehr gut für **linear separierbare Daten**.
* **Schwächen:** Funktioniert nicht für komplexe, nicht-lineare Strukturen.

**RBF SVM (Support Vector Machine mit radialem Basisfunktions-Kernel)**

* **Charakteristik:** Erweitert die lineare SVM, indem sie einen nicht-linearen Kernel verwendet, der die Daten in eine höhere Dimension transformiert.
* **Entscheidungsgrenze:** Sehr geschwungene, **komplexe Entscheidungsmuster**, die sich gut an nicht-lineare Strukturen anpassen.
* **Stärken:** Sehr leistungsfähig für nicht-lineare Probleme (wie hier für make\_moons und make\_circles).
* **Schwächen:** Erfordert sorgfältige Wahl der Hyperparameter (gamma, C).

**Gaussian Process Classifier (GPC)**

* **Charakteristik:** Probabilistischer Ansatz, der Wahrscheinlichkeitsverteilungen über Funktionen modelliert.
* **Entscheidungsgrenze:** Sehr **glatte, weiche Trennungen**, besonders sichtbar bei den kreisförmigen Daten.
* **Stärken:** Liefert nicht nur eine Klassifikation, sondern auch eine **Unsicherheitsabschätzung**.
* **Schwächen:** Hoher Rechenaufwand für große Datensätze.

**Decision Tree (Entscheidungsbaum)**

* **Charakteristik:** Hierarchische Struktur, die schrittweise die Daten in Bereiche unterteilt.
* **Entscheidungsgrenze:** **Stufenförmige, eckige Trennungen**, da Entscheidungsbäume durch einfache Regeln arbeiten.
* **Stärken:** Schnell zu trainieren, interpretiert leicht.
* **Schwächen:** Neigt zum Overfitting, besonders bei kleinen Datenmengen.

**Random Forest (Ensemble von Entscheidungsbäumen)**

* **Charakteristik:** Kombiniert mehrere Entscheidungsbäume, um eine robustere Klassifikation zu ermöglichen.
* **Entscheidungsgrenze:** Ähnlich wie Decision Trees, aber **glatter und stabiler**, weil mehrere Bäume kombiniert werden.
* **Stärken:** Reduziert Overfitting, robuste Performance.
* **Schwächen:** Weniger interpretierbar als ein einzelner Entscheidungsbaum.

**Neural Network (Mehrschichtige neuronale Netze, MLPClassifier)**

* **Charakteristik:** Mehrere Schichten von Neuronen lernen **hochgradig nicht-lineare Entscheidungsgrenzen**.
* **Entscheidungsgrenze:** Sehr **geschwungene, flexible Grenzen**, besonders sichtbar bei make\_moons und make\_circles.
* **Stärken:** Kann sehr komplexe Muster lernen.
* **Schwächen:** Erfordert viel Rechenleistung und Daten für effektives Training.

**AdaBoost (Adaptive Boosting)**

* **Charakteristik:** Ensemble-Methode, die schwache Klassifikatoren kombiniert, um eine starke Gesamtentscheidung zu treffen.
* **Entscheidungsgrenze:** Etwas eckige Trennung, aber deutlich robuster als ein einzelner Entscheidungsbaum.
* **Stärken:** Verbessert schwache Klassifikatoren und funktioniert gut auf moderaten Datensätzen.
* **Schwächen:** Sensitiv gegenüber Ausreißern.

**Naive Bayes (Gaussian Naive Bayes)**

* **Charakteristik:** Basierend auf **Bayes‘ Theorem**, nimmt an, dass Merkmale unabhängig sind.
* **Entscheidungsgrenze:** Glatte, oft ovale oder kreisförmige Trennung.
* **Stärken:** Schnell zu trainieren, funktioniert oft überraschend gut.
* **Schwächen:** Die Annahme der **Merkmalsunabhängigkeit** ist oft nicht realistisch.

**QDA (Quadratic Discriminant Analysis)**

* **Charakteristik:** Erweiterung von Naive Bayes, erlaubt **quadratische** Trennungen.
* **Entscheidungsgrenze:** Starke, geschwungene Muster (besonders sichtbar in der letzten Spalte).
* **Stärken:** Erlaubt nicht-lineare Trennungen.
* **Schwächen:** Funktioniert nicht gut, wenn die Daten nicht den zugrundeliegenden Modellannahmen entsprechen.

**3. Fazit und Interpretation der Ergebnisse**

* **Die besten Algorithmen für nicht-lineare Daten (make\_moons, make\_circles)** sind **RBF SVM, Gaussian Process, Random Forest und Neural Networks**, da sie geschwungene Entscheidungsgrenzen erzeugen.
* **Lineare SVM funktioniert nur für lineare Datensätze** und ist für make\_moons und make\_circles nicht geeignet.
* **Decision Trees und Random Forests neigen zu kantigen Entscheidungsgrenzen**, funktionieren aber trotzdem gut.
* **Probabilistische Methoden wie Naive Bayes und QDA** zeigen eher weiche, geschwungene Entscheidungsgrenzen, sind aber nicht immer die besten.

Die Wahl des besten Klassifikators hängt also stark von der Datenstruktur ab. Während lineare Modelle für einfache Trennungen gut sind, sind komplexe, nicht-lineare Modelle für anspruchsvollere Muster nötig.

### Jonas Maier (Datenbank, Backend)

*Kapitel verfasst von Jonas Maier*

### Jonas Bogensberger (Frontend)

*Kapitel verfasst von Jonas Bogensberger*

## Variantenbildung

Bei der Planung und Umsetzung des Projekts wurden verschiedene Architektur- und Technologievarianten in Betracht gezogen, um eine möglichst effiziente und benutzerfreundliche Lösung zu entwickeln. Dabei spielten Aspekte wie Skalierbarkeit, Performance, Wartbarkeit und Benutzerfreundlichkeit eine zentrale Rolle. Die Auswahl der finalen Variante erfolgte nach einer sorgfältigen Abwägung der Vor- und Nachteile jeder Option.

**Mögliche Varianten:**

* Verwendung einer zentralisierten Server-Architektur oder einer dezentralen Lösung
* Speicherung der Analyseergebnisse in einer relationalen Datenbank (MySQL, PostgreSQL) oder einer NoSQL-Datenbank (MongoDB)
* Einsatz einer lokalen KI-Verarbeitung auf dem Endgerät oder einer serverbasierten KI-Analyse
* Auswahl von vortrainierten KI-Modellen oder Implementierung eines eigenen, speziellen Trainingssystems
* Nutzung einer Webanwendung oder einer mobilen Applikation zur Interaktion mit dem System

Nach eingehender Analyse entschied sich das Projektteam für eine **zentralisierte Server-Architektur mit einer serverbasierten KI-Analyse**. Diese Wahl gewährleistet eine hohe Skalierbarkeit, da die KI-Modelle zentral verwaltet und aktualisiert werden können, ohne dass die Endgeräte der Nutzer über hohe Rechenkapazitäten verfügen müssen.

Für die Speicherung der Analyseergebnisse wurde eine **NoSQL-Datenbank (MongoDB)** bevorzugt. Diese bietet eine flexible Struktur, die sich gut für unstrukturierte oder sich dynamisch ändernde Daten eignet und ermöglicht eine effiziente Verarbeitung der Analyseergebnisse.

Zudem fiel die Entscheidung auf die Entwicklung einer **mobilen Applikation in Kotlin für Android**. Diese Wahl bietet eine optimierte Benutzererfahrung auf mobilen Geräten, ermöglicht den direkten Zugriff auf die Kamera für die Bilderfassung und erlaubt eine nahtlose Interaktion mit der KI-Analyse. Außerdem wurde dieser Punkt vom Projektgeber vorgegeben.

Diese Architektur gewährleistet eine effiziente, benutzerfreundliche und leistungsstarke Lösung, die sowohl die Vorteile einer mobilen App als auch einer serverbasierten KI-Verarbeitung vereint.

## Machbarkeitsstudie

Die Machbarkeitsstudie untersucht, ob das geplante Diplomprojekt innerhalb der gegebenen Rahmenbedingungen technisch und zeitlich realisierbar ist. Dabei wurden sowohl die technischen Voraussetzungen als auch die zur Verfügung stehende Arbeitszeit analysiert. Durch eine frühzeitige Einarbeitung in relevante Technologien konnten mögliche Herausforderungen identifiziert und Lösungsansätze erarbeitet werden.

**Technische Machbarkeit:**

* Die Entwicklung einer mobilen Anwendung in **Kotlin für Android** ist technisch realisierbar und wird durch eine Vielzahl von Entwicklungsressourcen und Bibliotheken unterstützt.
* Die **serverseitige KI-Analyse** kann mithilfe moderner Machine-Learning-Frameworks wie TensorFlow oder PyTorch umgesetzt werden.
* Die Speicherung der Analyseergebnisse in einer **MongoDB-Datenbank** ermöglicht eine flexible und effiziente Verwaltung der Daten.
* Die Anbindung der mobilen Anwendung an den Server erfolgt über standardisierte **REST- oder WebSocket-Schnittstellen**, was eine stabile Kommunikation sicherstellt.
* Die Implementierung verschiedener **KI-Modelle zur Analyse von Hautläsionen** ist möglich, wobei vortrainierte Modelle genutzt oder eigene Modelle trainiert werden können.
* Die Anwendung erfordert eine **stabile Internetverbindung**, um Bilder an den Server zu senden und Ergebnisse zurückzuerhalten.

**Zeitliche Machbarkeit:**

* Jeder Projektteilnehmer hat ca. **180 Stunden** für das Diplomprojekt zur Verfügung.
* Die Arbeit am Projekt erfolgt sowohl **innerhalb der Schulzeit** (Diplomarbeitsstunden) als auch **außerhalb der Schulzeiten** (individuelle und gemeinsame Arbeitstreffen).
* Die Entwicklung gliedert sich in mehrere Phasen:
  + **Vorbereitungsphase**: Einarbeitung in relevante Technologien, Recherche und Planung
  + **Entwicklungsphase**: Umsetzung der mobilen Anwendung, Server-Architektur und KI-Integration
  + **Testphase**: Überprüfung der Funktionalität, Optimierung und Fehlerbehebung
  + **Dokumentation**: Erstellung der Abschlussdokumentation und Vorbereitung auf die Präsentation
* Trotz potenzieller Herausforderungen bei der Entwicklung von KI-Modellen und der Server-Integration wird das Projektteam durch eine klare Aufgabenverteilung und regelmäßige Abstimmung die gesetzten Meilensteine erreichen.

Insgesamt ist das Projekt aus technischer und zeitlicher Sicht gut realisierbar. Die gewählten Technologien sind bewährt und ermöglichen eine effiziente Umsetzung. Durch die geplante Arbeitsverteilung und die Kombination aus Schul- und Freizeitstunden steht ausreichend Zeit zur Verfügung, um das Projekt erfolgreich abzuschließen.

## Allgemeine Planungsinformationen

Neben den bereits behandelten Aspekten der Projektplanung gibt es weitere zentrale Überlegungen, die für den erfolgreichen Ablauf des Projekts von Bedeutung sind. Dieses Kapitel befasst sich mit grundlegenden Planungsinformationen, die nicht in anderen Abschnitten detailliert behandelt wurden, jedoch maßgeblich zur Organisation und Umsetzung beitragen.

### Projektziele

*(Die Grundlegenden Projektziele sind aus den Vorgaben des Projektgebers übernommen.)*

Die Zielsetzung dieses Projekts besteht in der Entwicklung einer **leistungsfähigen Machine-Learning-Anwendung** zur **automatischen Klassifikation von Hautläsionen** anhand von Bildern. Dabei wird der **HAM10000-Datensatz** als Grundlage genutzt. Das Projekt umfasst nicht nur die Entwicklung eines geeigneten KI-Modells, sondern auch die vollständige Umsetzung des **Machine-Learning-Life-Cycles** – von der Datenvorbereitung über das Modelltraining bis hin zur Integration in eine mobile Anwendung.

Ein weiteres zentrales Ziel ist die **didaktische Aufbereitung** des Entwicklungsprozesses. Entlang der verschiedenen Phasen sollen **Jupyter Notebooks** erstellt werden, die eine verständliche Einführung in die einzelnen Schritte der Modellentwicklung ermöglichen. Dadurch wird das Projekt nicht nur zur praktischen Lösung eines medizinischen Problems, sondern auch zur wertvollen Lernressource für die Arbeit mit Machine Learning.

Die konkreten Projektziele lassen sich in fünf Kernbereiche unterteilen:

**1. Datenvorbereitung**

* Bereitstellung und Verwaltung der Bilddaten über eine strukturierte oder unstrukturierte **Datenbank**.
* **Datenbereinigung und -augmentation**, um die Qualität und Diversität des Trainingsdatensatzes zu optimieren.
* Optionale Ergänzung des **HAM10000-Datensatzes** durch weitere medizinische Bilddatensätze (z. B. ISIC Challenge).
* Aufteilung der Daten in **Trainings-, Validierungs- und Testdatensätze**, um eine effiziente Modellbewertung zu ermöglichen.

**2. Modellentwicklung**

* Auswahl und Implementierung geeigneter **Machine-Learning-Algorithmen**, insbesondere **Convolutional Neural Networks (CNNs)**.
* Untersuchung verschiedener Ansätze, darunter direkte Klassifikation oder ein **zweistufiger Prozess**, bei dem zunächst die Läsion erkannt und anschließend klassifiziert wird (Bounding Box, Semantic Segmentation etc.).
* **Training und Optimierung der Modelle** auf Basis des HAM10000-Datensatzes durch **Hyperparameter-Tuning**.

**3. Modellbewertung**

* Evaluierung der **Modellleistung** anhand relevanter Metriken wie **Accuracy, Precision, Recall, AUROC und F1-Score**.

**4. Deployment**

* Bereitstellung des trainierten Modells in einer **serverseitigen Umgebung**.
* Entwicklung einer **RESTful API**, um das Modell als Webservice zur Verfügung zu stellen.
* Optional: Implementierung des Modells direkt auf dem mobilen Gerät mit **TensorFlow Lite oder PyTorch Edge**, um die Abhängigkeit vom Server zu reduzieren.

**5. Integration in eine mobile Anwendung**

* Entwicklung einer **mobilen App in Kotlin**, die es Nutzern ermöglicht, **Bilder von Hautläsionen aufzunehmen und klassifizieren zu lassen**.
* Integration der **RESTful API** zur Bereitstellung der Klassifikationsergebnisse in Echtzeit.

Dieses Projekt kombiniert moderne **Machine-Learning-Technologien** mit einer benutzerfreundlichen **mobilen Anwendung**, um einen praktischen Beitrag zur medizinischen Diagnostik zu leisten. Gleichzeitig stellt die durchdachte Planung und Aufbereitung sicher, dass der gesamte Entwicklungsprozess **nachvollziehbar und wiederverwendbar** bleibt.

#### Redisign

### Benötigte Ressourcen

Für die erfolgreiche Umsetzung des Projekts sind verschiedene Ressourcen erforderlich, die sowohl die technische als auch die organisatorische Realisierung unterstützen. Dieses Kapitel gibt einen Überblick über die benötigten Ressourcen, die in verschiedene Kategorien unterteilt sind: Hardware, Software, personelle Ressourcen und weitere unterstützende Materialien.

#### Hardware

Diese Diplomarbeit wird sowohl an den privaten Schullaptops diverser Hersteller des Projektteams als auch an Standrechnern zu Hause entwickelt und dokumentiert. Es sind keinerlei externe Hardware oder Mietserver von Nöten.

#### Software

Dadurch, dass im Projekt verschiedenste Technologien und Programmiersprachen verwendet werden, wird auch eine geeignete Softwareunterstützung zur Entwicklung in den jeweiligen Bereichen gefordert. Dazu gehören integrierte Entwicklungsumgebungen, Interpreter, Kompilierer oder auch Textverarbeitungsprogramme. Eine ausführliche Auflistung der verwendeten Programme/Tools ist im Kapitel **6.7 Softwareprogramme / Komponenten** zu finden.

#### Personelle Ressourcen

Das Projektteam arbeitet eigenständig und ohne jegliche Hilfe von außen an dem Projekt. Dafür stellt jedes Teammitglied um die 180 Stunden reine Arbeitszeit dem Projekt zur Verfügung.

#### Weitere Materialien

##### ISIC-Archive

Das **ISIC-Archive** (International Skin Imaging Collaboration) ist eine öffentlich zugängliche Datenbank mit dermatologischen Bildern, die zur Unterstützung der Forschung und Entwicklung im Bereich der Hautkrebs- und Hautläsionserkennung dient. Es enthält eine große Sammlung hochauflösender Bilder verschiedener Hauterkrankungen, darunter Melanome und andere Hautveränderungen. Die Datenbank wird häufig für das Training und die Validierung von KI-gestützten Diagnosemodellen genutzt und dient als Benchmark für medizinische Bildanalyse-Algorithmen.

Diese Datenbank wird als Datenquelle für Trainings- und Validierungsdaten der KI-Komponente verwendet.

### Entwicklungsmethodik

Für die Durchführung dieses Projekts wurde die **agile Entwicklungsmethodik** gewählt, um sicherzustellen, dass flexibel und effizient auf Anforderungen reagiert werden kann und der Projektfortschritt kontinuierlich überprüft wird. Der Fokus liegt auf iterativen Entwicklungszyklen, die eine schnelle Anpassung und Verbesserung des Produkts ermöglichen.

Im Speziellen wurde der Ansatz des **Scrum**-Frameworks verwendet, um das Projekt zu planen, zu steuern und zu überwachen. Scrum ermöglicht es, die Arbeit in überschaubare Iterationen, sogenannte **Sprints**, zu unterteilen. Jeder Sprint dauert in der Regel zwei Wochen und endet mit einer Überprüfung der erreichten Ziele. In der Planung des Projekts wurden alle Anforderungen und Aufgaben in **User Stories** umgewandelt, die dann geschätzt wurden. Die Schätzungen erfolgten sowohl in **Value Points** (für den Nutzen, den jede Story bringt) als auch in **Story Points** (für den Aufwand, der für die Umsetzung erforderlich ist).

Die Aufgaben und Anforderungen wurden im **Product Backlog** gesammelt, wobei für jeden Sprint eine Auswahl an Aufgaben ins **Sprint Backlog** übernommen wurde. Während der Sprintplanung wurde der Schwerpunkt auf die Umsetzung von Features gelegt, die den größten Mehrwert für das Projekt bringen.

Ein wesentlicher Bestandteil der Scrum-Methode ist die kontinuierliche Überprüfung des Fortschritts und das Einholen von Feedback. In diesem Projekt wurde der Fortschritt jedoch nicht nach jedem Sprint in einer Präsentation vor den Lehrpersonen präsentiert. Stattdessen wurden regelmäßige interne Besprechungen abgehalten, um den Fortschritt zu diskutieren und eventuelle Probleme zu identifizieren. Den Lehrpersonen wurde der Fortschritt zu festgelegten Zeitpunkten des Projekts präsentiert, um sicherzustellen, dass das Projekt in die richtige Richtung geht und alle Anforderungen eingehalten werden.

Die agile Methodik stellt sicher, dass das Projektteam jederzeit flexibel auf Änderungen oder neue Anforderungen reagieren kann, wodurch ein transparentes und anpassungsfähiges Projektumfeld geschaffen wird. Durch diesen iterativen Ansatz konnte die Qualität der Ergebnisse kontinuierlich verbessert und das Projekt effizient zum Erfolg geführt werden.

### Kommunikations- und Berichterstattungsstrategie

Im Rahmen dieses Projekts wird der Kontakt zu den Stakeholdern über die Lehrpersonen als zentrale Kommunikationsstelle gepflegt. Die Fortschritte des Projekts werden regelmäßig intern besprochen, und an festgelegten Meilensteinen werden die Lehrpersonen über den aktuellen Stand informiert. Die Berichterstattung erfolgt nicht in Form von regelmäßigen Präsentationen alle zwei Wochen, sondern fokussiert zu wichtigen Projektpunkten, bei denen eine detaillierte Rückmeldung erforderlich ist.

Die Erreichung der Sprintziele wird regelmäßig intern besprochen, wobei insbesondere darauf geachtet wird, welche User Stories im Sprint Backlog abgeschlossen wurden, welche noch offen sind und welche an den nächsten Sprint übergeben werden. Dabei wird auch die Sprint Velocity und der Projektverlauf mithilfe von Burndown-Charts und Säulendiagrammen visualisiert, um eine klare und nachvollziehbare Darstellung des Fortschritts zu gewährleisten.

**Live-Demo:**  
Ein weiteres Element der Kommunikationsstrategie ist die Bereitstellung einer Live-Demo. Diese ermöglicht den Stakeholdern, die neuesten Fortschritte des Projekts direkt zu erleben und gibt ihnen die Möglichkeit, Feedback zu geben. Die Live-Demo wird durch die Lehrperson organisiert, die auch den Kontakt zu den Stakeholdern koordiniert.

**Dokumentation:**  
Die gesamte Projektarbeit wird kontinuierlich in einer Dokumentation festgehalten, die laufend aktualisiert wird. Diese Dokumentation dient nicht nur als interner Bericht, sondern auch als Referenz für die Lehrpersonen und Stakeholder, um jederzeit einen detaillierten Überblick über den Projektverlauf zu erhalten. Sie gewährleistet, dass alle relevanten Informationen zugänglich sind und beantwortet mögliche Fragen.

### Projektrisiko(-bewertung)

Da es sich bei diesem Projekt um eine **Diplomarbeit** handelt, die im schulischen Kontext durchgeführt wird, ist das Risiko im Vergleich zu kommerziellen Projekten in gewisser Weise weniger gravierend. Dennoch gibt es auch hier Risiken, die berücksichtigt und im besten Fall frühzeitig gemanagt werden sollten. Die Risikobewertung erfolgt anhand potenzieller Faktoren, die den Verlauf und die erfolgreiche Umsetzung des Projekts beeinflussen könnten.

#### Mangelnde Verfügbarkeit von Ressourcen

Ein potenzielles Risiko besteht darin, dass möglicherweise nicht ausreichend Ressourcen wie Zeit, benötigte Software, Hardware oder Fachwissen zur Verfügung stehen. Besonders die **Zeit** stellt ein Risiko dar, da die Arbeit parallel zu schulischen Anforderungen durchgeführt wird. Die fehlende Verfügbarkeit von Experten oder spezifischen technischen Ressourcen könnte ebenfalls den Fortschritt verzögern.

**Ursachen:**  
Dieses Risiko kann verschiedene Ursachen haben, beispielsweise durch unvorhergesehene Verzögerungen, wie zusätzliche Zeitaufwände bei der Datensammlung, Modellierung oder dem Training der KI-Algorithmen. Auch technische Schwierigkeiten bei der Implementierung oder der Integration der Lösung in die mobile Anwendung können zusätzliche Ressourcen beanspruchen. Ein weiteres Risiko kann durch unerwartete Änderungen in den Projektanforderungen oder -spezifikationen entstehen.

**Auswirkungen:**  
Ein Mangel an Ressourcen könnte zu verschiedenen negativen Auswirkungen führen, darunter **Verzögerungen** im Zeitplan, eine **verminderte Qualität des Endprodukts** oder sogar eine **unvollständige Umsetzung** des Projekts. Dies könnte insbesondere in einer Diplomarbeit problematisch sein, da das Projekt rechtzeitig abgeschlossen und die Qualität des Endprodukts den Anforderungen entsprechen muss, um den akademischen Abschluss zu sichern.

**Risikobewertung:**  
Die Wahrscheinlichkeit dieses Risikos ist moderat, da es von mehreren Faktoren abhängt, darunter der Komplexität der Modellierung und der Implementierung der mobilen Anwendung, der Verfügbarkeit von benötigten Ressourcen und der Erfahrung des Projektteams. Die Projektarbeit könnte sich aufgrund von externen Faktoren wie Zeitmanagement oder unvorhergesehenen Schwierigkeiten bei der Implementierung verzögern, allerdings wurde durch die Festlegung von **realistischen Zeitrahmen** und **Kontrollen** das Risiko eingedämmt.

**Risikominderungsstrategien:**  
Zur Minimierung dieses Risikos können verschiedene Maßnahmen ergriffen werden:

* **Frühzeitige Planung** der benötigten Ressourcen, sowohl in Bezug auf Zeit als auch Materialien oder externe Unterstützung.
* **Realistische Zeitpläne** erstellen und regelmäßig überprüfen, um sicherzustellen, dass alle Aufgaben rechtzeitig abgeschlossen werden können.
* **Pufferzeiten** für unvorhergesehene technische oder logistische Probleme einplanen.
* Bei Bedarf auf **Zusatzressourcen** zurückgreifen, etwa zusätzliche Literaturquellen, technische Foren oder externe Unterstützung von Experten, etwa durch Online-Communities oder die Lehrpersonen als Ansprechpartner.

**Verantwortlichkeiten:**  
Die Projektverantwortlichen, also die **Projektteilnehmer** und die **Lehrpersonen**, sind dafür verantwortlich, das Risiko kontinuierlich zu überwachen und frühzeitig auf Anzeichen von Ressourcenmangel zu reagieren. Im Falle unerwarteter Herausforderungen oder Verzögerungen müssen geeignete Maßnahmen zur Risikominderung ergriffen und die Ressourcen gegebenenfalls angepasst werden.

## Projektumfeldanalyse

Die Projektumfeldanalyse bietet einen umfassenden Überblick über die verschiedenen Faktoren und Bedingungen, die den Erfolg dieses Projekts beeinflussen können. Sie berücksichtigt sowohl interne als auch externe Aspekte, die das Projekt vor, während und nach der Umsetzung beeinflussen könnten. Ziel dieser Analyse ist es, mögliche Chancen und Risiken zu identifizieren und die notwendigen Voraussetzungen für eine erfolgreiche Durchführung des Projekts zu schaffen.

**Zielsetzung des Projekts:**

Das Hauptziel des Projekts ist die Entwicklung einer mobilen Anwendung, die es den Nutzern ermöglicht, Bilder von Hautläsionen aufzunehmen und diese mithilfe eines Machine Learning Modells zu klassifizieren. Die Aufgabe ist in mehrere Hauptbereiche unterteilt:

* **Datenvorbereitung**
* **Modellentwicklung und -bewertung**
* **Integration in eine mobile Anwendung**

**Anforderungen an das Projekt:**

* **Funktionalität:** Die mobile Anwendung muss auf gängigen mobilen Endgeräten (insbesondere mit Android-Betriebssystem) lauffähig sein und eine benutzerfreundliche Oberfläche bieten. Sie muss in der Lage sein, Bilder zu erfassen und die Bilddaten an den Server zur Klassifikation zu senden.
* **Datenverarbeitung:** Die Anwendung benötigt eine zuverlässige Datenbank zur Speicherung der Bilddaten sowie der Klassifikationsergebnisse.
* **Zuverlässigkeit und Skalierbarkeit:** Das System muss skalierbar sein, um mit einer zunehmenden Menge von Nutzerdaten umgehen zu können. Es sollte auch in der Lage sein, Anfragen in Echtzeit zu verarbeiten.
* **Dokumentation:** Alle durchgeführten Schritte und Modellentwicklungen müssen dokumentiert werden, um eine nachvollziehbare Entwicklungsgeschichte sicherzustellen.

**Technologische Anforderungen:**

* **Mobile Entwicklung:** Die Anwendung wird mit Kotlin für Android entwickelt, um eine hohe Performance und nahtlose Integration auf mobilen Geräten zu gewährleisten.
* **Machine Learning Modell:** Ein tiefes neuronales Netzwerk (CNN) wird für die Klassifikation der Hautläsionen eingesetzt, trainiert auf einem umfangreichen Datensatz (HAM10000).
* **Backend und API:** Eine RESTful API wird benötigt, um das Machine Learning Modell in einer serverseitigen Umgebung bereitzustellen und die Kommunikation zwischen der mobilen Anwendung und dem Server zu ermöglichen.
* **Datenbank:** Eine NoSQL-Datenbank (wie MongoDB) wird zur Speicherung der Bilddaten und Klassifikationsergebnisse verwendet.

**Rahmenbedingungen:**

* **Projektzeitrahmen:** Das Projekt wird als Teil einer Diplomarbeit durchgeführt, und der Zeitrahmen ist durch das Ende des Schuljahres und die Abgabefrist der Diplomarbeit begrenzt.
* **Ressourcen:** Die notwendigen Ressourcen wie mobile Geräte (Android Smartphones), Computer für Modelltraining und Entwicklungsumgebungen werden durch die Schule bereitgestellt. Auch die Datensätze und Modellressourcen sind durch öffentliche Quellen zugänglich.
* **Expertise:** Das Projektteam besteht aus Schülern mit unterschiedlichen Kenntnissen in den Bereichen Machine Learning, Mobile Entwicklung und Datenbankmanagement. Eine kontinuierliche Unterstützung durch die Lehrperson ist für das Gelingen des Projekts von entscheidender Bedeutung.

**Externe Faktoren:**

* **Verfügbarkeit von Hardware und Software:** Die Nutzung von Open-Source-Bibliotheken für Machine Learning sowie die Verfügbarkeit der Datensätze über öffentliche Repositorien sind wesentliche Faktoren, die den Projekterfolg positiv beeinflussen. Zudem muss sichergestellt werden, dass die verwendete Software und die Entwicklungstools aktuell und kompatibel sind.
* **Technische Unterstützung:** Bei technischen Herausforderungen und der Implementierung können Online-Communities wie Stack Overflow oder spezialisierte Foren eine wertvolle Unterstützung bieten.
* **Risiken:** Unvorhergesehene technische Schwierigkeiten, wie z.B. Probleme bei der Modellintegration in die mobile Anwendung, können den Fortschritt verzögern. Ebenso können Engpässe bei der Ressourcenverfügbarkeit, wie z.B. mangelnde Zeit oder fehlende technische Geräte, das Projekt beeinflussen.

**Wettbewerbsfaktoren:**

* **Innovationspotenzial:** Es gibt verschiedene bestehende Projekte, die sich mit der Klassifikation von Hautläsionen befassen. Um sich von anderen Projekten abzuheben, wird in diesem Projekt die Integration des Machine Learning Modells in eine mobile App als innovativer Ansatz verfolgt. Diese Integration ermöglicht eine benutzerfreundliche und leicht zugängliche Lösung für die Hautläsionsklassifikation.
* **Benchmarking:** Ähnliche Projekte, die Machine Learning zur medizinischen Bildklassifikation einsetzen, können als Benchmarks für die Bewertung des eigenen Fortschritts und der angewandten Methoden dienen.

**Projektmanagement:**

* **Teamarbeit:** Eine klare Aufgabenverteilung und effektive Kommunikation sind entscheidend, um das Projekt termingerecht abzuschließen. Regelmäßige Teammeetings und der Austausch von Ideen und Fortschritten stellen sicher, dass alle Teammitglieder auf dem gleichen Stand sind.
* **Zeitplanung:** Die Meilensteine und Sprints des Projekts müssen klar definiert werden, um den Fortschritt zu überwachen und das Projekt innerhalb des vorgesehenen Zeitrahmens zu beenden.
* **Koordination mit der Lehrperson:** Der regelmäßige Austausch mit der Lehrperson stellt sicher, dass das Projekt auf dem richtigen Weg bleibt und wertvolles Feedback während der Entwicklung einfließt.

**Rechtliche Aspekte:**

* **Lizenzen:** Alle verwendeten Softwarebibliotheken und -tools müssen unter einer geeigneten Lizenz stehen, die den rechtlichen Anforderungen entspricht.
* **Datenschutz:** Da die mobile Anwendung Nutzerdaten (insbesondere Bilddaten) verarbeiten wird, müssen Datenschutzbestimmungen beachtet werden. Eventuell sind zusätzliche Maßnahmen wie eine Anonymisierung der Daten oder die Implementierung von Login-Systemen erforderlich, um die Privatsphäre der Nutzer zu gewährleisten.
* **Sicherheit:** Besonders bei der Übertragung von Bilddaten über Netzwerke müssen Sicherheitsaspekte wie Verschlüsselung berücksichtigt werden, um Missbrauch zu vermeiden.

Diese Projektumfeldanalyse stellt sicher, dass alle relevanten Faktoren für die erfolgreiche Durchführung des Projekts berücksichtigt werden. Sie dient als Grundlage für die Planung und Umsetzung und hilft dabei, potenzielle Risiken zu identifizieren und geeignete Maßnahmen zu ergreifen.

# Softwarearchitektur

*Kapitel verfasst von Daniel Jessner*

Ein fundiertes Verständnis der Softwarearchitektur ist unerlässlich, um die Funktionsweise eines Systems vollständig zu begreifen und eine effiziente Entwicklung sowie Wartung zu gewährleisten. In diesem Kapitel wird die Architektur der entwickelten Software detailliert vorgestellt. Es werden die verschiedenen Systemkomponenten und deren Interaktionen erläutert, um aufzuzeigen, wie sie zusammenarbeiten, um die angestrebte Funktionalität zu erreichen. Dabei wird besonders auf die Trennung der verschiedenen Softwareebenen und deren Kommunikation eingegangen, um ein ganzheitliches Bild der Systemstruktur zu vermitteln. Ein besonderer Fokus liegt auf der Strukturierung der KI-Komponente, der mobilen Anwendung sowie der dazugehörigen Backend-Architektur. Um diese komplexen Beziehungen klar und verständlich zu machen, werden die relevanten Architekturkomponenten durch Diagramme veranschaulicht, die die Zusammenhänge und Abläufe visuell verdeutlichen.

Das folgende Kapitel beleuchtet die grundlegenden Softwarekomponenten des Systems:

* **Mobile Anwendung (Android, Kotlin)**
* **Backend-Server (AdonisJS, Python)**
  + **Backend API**
* **KI-Komponente(Python, Tensorflow, Pytorch, Scikit-Learn)**
  + **Gateway-API (FastAPI)**
* **Datenbank (MongoDB)**

Durch diese detaillierte Darstellung wird ein klarer Überblick darüber gegeben, wie die einzelnen Teile des Systems zusammenspielen und miteinander kommunizieren, um die geplante Lösung zu realisieren.

## Aktivitätsdiagramme

Aktivitätsdiagramme sind ein wesentliches Werkzeug in der Softwaredokumentation, um Arbeitsabläufe und Prozesse innerhalb eines Systems grafisch darzustellen. Sie helfen dabei, die Sequenz von Aktivitäten und die logischen Abläufe in einer Anwendung zu visualisieren. Dieses Kapitel enthält die relevanten Aktivitätsdiagramme für das dokumentierte Projekt, die detailliert die verschiedenen Prozesse und deren Abläufe illustrieren.

### Neuen Benutzer registrieren

Die Registrierung eines neuen Benutzers ist ein essenzieller Prozess, um neuen Nutzern Zugang zum System zu ermöglichen. Dieses Aktivitätsdiagramm beschreibt die erforderlichen Schritte und Abläufe, um sicherzustellen, dass ein Benutzer erfolgreich registriert wird und in der Datenbank gespeichert werden kann.

Der Prozess beginnt mit der Registrierung im Mobile App-Modul, wo der Benutzer seine Daten eingibt. Diese Daten werden dann über das **AI-Gateway** an den **Datenbankserver** weitergeleitet, der die Validierung der Benutzerdaten durchführt.

* Falls die Validierung fehlschlägt (z. B. aufgrund ungültiger oder bereits bestehender Benutzerdaten), wird eine Fehlermeldung zurückgesendet und in der mobilen App angezeigt.
* Falls die Validierung erfolgreich ist, wird der neue Benutzer in der Datenbank gespeichert. Anschließend wird eine Bestätigung zurück an die App gesendet, die eine erfolgreiche Registrierung anzeigt.

Das Diagramm zeigt auch den alternativen Pfad für eine fehlgeschlagene Registrierung, um sicherzustellen, dass der Benutzer die entsprechenden Fehlermeldungen erhält und notwendige Anpassungen vornehmen kann.

Aktivitätsdiagramm:

Ein Bild, das Text, Diagramm, Reihe, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Bestehenden Benutzer einloggen

Die Anmeldung eines bestehenden Benutzers ist ein wichtiger Prozess, um Nutzern Zugang zum System zu gewähren. Dieses Aktivitätsdiagramm beschreibt die erforderlichen Schritte und Abläufe, um sicherzustellen, dass ein Benutzer erfolgreich authentifiziert wird und auf die Anwendung zugreifen kann.

Der Prozess beginnt in der **Mobile App**, wo der Benutzer seine Anmeldedaten eingibt. Diese Daten werden dann an das **AI-Gateway** gesendet, das die Anfrage verarbeitet und an den **Datenbankserver** weiterleitet. Der Datenbankserver überprüft die Benutzerinformationen und gibt eine entsprechende Antwort zurück.

* Falls die Validierung fehlschlägt (z. B. aufgrund falscher Anmeldedaten oder eines nicht existierenden Benutzers), wird eine Fehlermeldung an die App gesendet, die diese dem Benutzer anzeigt.
* Falls die Validierung erfolgreich ist, wird eine Bestätigung an die App gesendet, die den Benutzer über die erfolgreiche Anmeldung informiert.

Das Diagramm zeigt auch den alternativen Pfad für eine fehlgeschlagene Anmeldung, um sicherzustellen, dass der Benutzer die entsprechenden Fehlermeldungen erhält und notwendige Anpassungen vornehmen kann.

Aktivitätsdiagramm:

Ein Bild, das Text, Diagramm, Screenshot, Kreis enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Analyse-Historie einsehen

Die Anzeige der Analysehistorie ermöglicht es dem Benutzer, vergangene Analysen einzusehen. Dieses Aktivitätsdiagramm beschreibt die Schritte zur Abfrage und Anzeige der gespeicherten Analysedaten.

Der Prozess beginnt in der **Mobile App**, wenn der Benutzer die Historie aufruft. Die App sendet eine Anfrage mit den Benutzerdaten an das **AI-Gateway**, das diese weiter an den **Datenbankserver** leitet. Der Datenbankserver überprüft die Benutzerinformationen und ruft die gespeicherten Analysen ab.

* Falls Analysedaten vorhanden sind, wird die Historie an die App zurückgesendet und dem Benutzer angezeigt.
* Falls keine Daten vorhanden sind, erhält der Benutzer eine entsprechende Nachricht in der UI, die ihn darüber informiert, dass keine früheren Analysen existieren.

Das Diagramm zeigt deutlich die alternativen Abläufe je nach Vorhandensein oder Fehlen von gespeicherten Analysen, um eine benutzerfreundliche Interaktion zu gewährleisten.

Aktivitätsdiagramm:

Ein Bild, das Text, Diagramm, Screenshot, Kreis enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Bildvorhersage-Prozess

Der Prozess der Bildvorhersage ermöglicht es dem Benutzer, ein Bild aufzunehmen und eine Vorhersage basierend auf einem trainierten Modell zu erhalten. Das Aktivitätsdiagramm beschreibt die verschiedenen Schritte, die zur Durchführung einer Bildvorhersage erforderlich sind.

1. **Modellauswahl**: Der Benutzer fordert verfügbare Modelle an, wählt eines aus und sieht die Auswahl in der UI.
2. **Bilderfassung**: Der Benutzer nimmt ein Foto auf, das zusammen mit dem Modell an das **AI-Gateway** gesendet wird.
3. **Modellvalidierung**: Das **AI-Gateway** überprüft, ob das Modell existiert und trainiert wurde. Falls nicht, wird ein Fehler zurückgegeben.
4. **Vorhersageprozess**:
   * Labels werden gelesen und das Bild wird auf die passende Größe skaliert.
   * Falls erforderlich, wird das Modell angepasst.
   * Das Bild wird vorverarbeitet und die Vorhersage wird durchgeführt.
5. **Vorhersageanzeige**: Die ermittelte Vorhersage wird an die **Mobile App** zurückgesendet und angezeigt.
6. **Speicherung der Analyse**: Der Benutzer kann entscheiden, ob die Analyse gespeichert werden soll.
   * Falls ja, wird eine Anfrage mit den Analyse- und Benutzerdaten an das **AI-Gateway** gesendet, das die Anfrage weiterleitet.
   * Die Gültigkeit der Anfrage wird überprüft. Falls sie ungültig ist, wird ein Fehler angezeigt.
   * Falls die Anfrage gültig ist, wird die erfolgreiche Vorhersage bestätigt.

Dieses Diagramm verdeutlicht die Interaktion zwischen der **Mobile App**, dem **AI-Gateway** und dem **Datenbankserver**, um eine benutzerfreundliche Vorhersagepipeline zu gewährleisten.

Aktivitätsdiagramm:

Ein Bild, das Text, Diagramm, Reihe, parallel enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Automatisches Zuschneiden von Bildern

Der Prozess des automatischen Zuschneidens von Bildern ermöglicht es dem Benutzer, ein aufgenommenes Foto automatisch durch die App oder den Server anpassen zu lassen. Das Aktivitätsdiagramm beschreibt die verschiedenen Schritte dieses Prozesses.

1. **Fotoaufnahme**: Der Benutzer nimmt ein Foto zur Vorhersage auf.
2. **Angebot der automatischen Zuschneidefunktion**: Die App bietet dem Benutzer die Möglichkeit, das Bild automatisch zu zuschneiden.
3. **Entscheidung des Benutzers**:
   * Falls der Benutzer die automatische Funktion akzeptiert, wird das Bild an den **Datenbankserver** gesendet.
   * Falls der Benutzer die Funktion ablehnt, kann er das Bild manuell zuschneiden.
4. **Verarbeitung auf dem Server**:
   * Der **AI-Gateway** leitet die Anfrage an den **Datenbankserver** weiter.
   * Der **Datenbankserver** führt das automatische Zuschneiden durch und identifiziert den relevanten Bereich.
   * Das zugeschnittene Bild wird als Antwort zurückgesendet.
5. **Anzeige und Entscheidung des Benutzers**:
   * Falls der Benutzer mit dem zugeschnittenen Bild einverstanden ist, wird die Vorhersage durchgeführt.
   * Falls der Benutzer nicht einverstanden ist, kann er das Bild manuell zuschneiden, bevor die Vorhersage erfolgt.

Dieses Diagramm zeigt den Ablauf des Zuschneideprozesses und die Interaktion zwischen der **Mobile App**, dem **AI-Gateway** und dem **Datenbankserver** für eine optimierte Nutzererfahrung.

Aktivitätsdiagramm:

Ein Bild, das Text, Diagramm, Screenshot, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Modellinformationen anzeigen lassen

Der Prozess des Abrufs von Modellinformationen ermöglicht es dem Benutzer, Details über ein Modell direkt über die App zu erhalten. Das Aktivitätsdiagramm beschreibt die verschiedenen Schritte dieses Prozesses.

1. **Modellinformationen abrufen**
   * Der Benutzer wählt in der App die Option, Informationen zu einem Modell anzuzeigen.
2. **Anfrage senden**
   * Die App sendet eine Anfrage zur Modellinformation an das AI-Gateway.
3. **Verarbeitung durch das AI-Gateway**
   * Das AI-Gateway empfängt die Anfrage und leitet sie an den entsprechenden Dienst weiter.
   * Das AI-Gateway ruft das entsprechende Modelldokument ab und sendet es als Antwort zurück.
4. **Anzeige in der App**
   * Die empfangene Modellinformation wird in der App angezeigt.
   * Der Benutzer kann die Informationen einsehen und entsprechend nutzen.

Dieses Diagramm zeigt den Ablauf des Anzeigens von Modellinformationen und die Interaktion zwischen der Mobile App und dem AI-Gateway für eine optimierte Nutzererfahrung.

Aktivitätsdiagramm:

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Modelltraining

Der Prozess des Modelltrainings ermöglicht es dem Benutzer, spezifische oder alle Modelle für das Training auszuwählen. Das Aktivitätsdiagramm beschreibt die verschiedenen Schritte dieses Prozesses.

1. **Modellauswahl und Trainingseinstellungen**
   * Der Benutzer wählt ein spezifisches Modell oder alle Modelle zum Trainieren aus.
   * Die App fordert den Benutzer auf, Trainingsparameter anzugeben.
   * Eine Trainingsanfrage mit Benutzerdaten wird gesendet.
2. **Benutzerauthentifizierung**
   * Das AI-Gateway sendet eine Anfrage zur Validierung des Benutzers an den Datenbankserver.
   * Der Datenbankserver überprüft die Benutzerdaten:
     + Falls der Benutzer ungültig ist, wird eine Fehlermeldung gesendet.
     + Falls der Benutzer gültig ist, wird überprüft, ob er zum Training berechtigt ist.
3. **Trainingsdatenerfassung**
   * Falls der Benutzer Admin-Rechte hat, ruft das System die Trainingsdaten aus der Datenbank ab.
   * Falls die Daten umfangreich sind, erfolgt eine paginierte Datenübertragung.
4. **Trainingsprozess**
   * Das Modelltraining erfolgt mit folgenden Schritten:
     + Anpassung des Datensatzes an die Admin-Set-Größe.
     + Modifikation der Eingabe- und Ausgabeschichten des Modells falls erforderlich.
     + Umwandlung des Datensatzes in das für das Modell geeignete Format.
     + Mehrfaches Anpassen des Modells basierend auf den Admin-Einstellungen.
     + Speichern der trainierten Modellparameter.
5. **Ergebnisverarbeitung**
   * Falls nicht alle Modelle erfolgreich trainiert wurden, wird der Fehler in eine Fehlerliste aufgenommen.
   * Falls das Training eines Modells fehlschlägt, wird dies in der Liste für den Admin dokumentiert.
   * Falls alle Modelle erfolgreich trainiert wurden, wird dies bestätigt.
6. **Anzeige der Ergebnisse**
   * Die Trainingsantwort wird an die mobile App gesendet.
   * Die App zeigt das Ergebnis der Trainingsanfrage im UI an.

Dieses Diagramm beschreibt den Ablauf des Modelltrainings und die Interaktion zwischen der Mobile App, dem AI-Gateway und dem Datenbankserver für eine effiziente Verwaltung und Durchführung von Modelltrainingsprozessen.

Aktivitätsdiagramm:

Ein Bild, das Text, Diagramm, Plan, technische Zeichnung enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Klassifikationsberichte erstellen

Der Prozess des Abrufens von Klassifizierungsberichten ermöglicht es dem Benutzer, spezifische oder alle Modelle zu analysieren. Das Aktivitätsdiagramm beschreibt die verschiedenen Schritte dieses Prozesses.

1. **Modellauswahl und Parameterdefinition**
   * Der Benutzer wählt ein spezifisches Modell oder alle Modelle zur Klassifizierung aus.
   * Die App fordert den Benutzer auf, die notwendigen Parameter anzugeben.
   * Eine Klassifizierungsanfrage mit Benutzerdaten wird gesendet.
2. **Benutzerauthentifizierung**
   * Das AI-Gateway sendet eine Anfrage zur Validierung des Benutzers an den Datenbankserver.
   * Der Datenbankserver überprüft die Benutzerdaten:
     + Falls der Benutzer ungültig ist, wird eine Fehlermeldung gesendet.
     + Falls der Benutzer gültig ist, wird überprüft, ob er berechtigt ist, Berichte zu erstellen.
3. **Trainingsdatenerfassung**
   * Falls der Benutzer Admin-Rechte hat, ruft das System die Trainingsdaten aus der Datenbank ab.
   * Falls die Daten umfangreich sind, erfolgt eine paginierte Datenübertragung.
4. **Erstellung des Klassifizierungsberichts**
   * Der Klassifizierungsprozess beinhaltet folgende Schritte:
     + Aufteilung des Datensatzes in Trainings- und Testdaten.
     + Anpassung des Datensatzes an die Admin-Set-Größe.
     + Anpassung der Eingabe- und Ausgabeschichten des Modells falls erforderlich.
     + Umwandlung des Datensatzes in das für das Modell geeignete Format.
     + Mehrfaches Anpassen des Modells basierend auf den Admin-Einstellungen.
     + Testen des trainierten Modells mit den Testdaten.
5. **Ergebnisverarbeitung**
   * Falls nicht alle Modelle erfolgreich analysiert wurden, wird der Fehler in eine Fehlerliste aufgenommen.
   * Falls die Klassifizierung eines Modells fehlschlägt, wird dies dokumentiert.
   * Falls alle Modelle erfolgreich klassifiziert wurden, wird dies bestätigt.
6. **Anzeige der Ergebnisse**
   * Die Klassifizierungsberichte und eine Fehlerliste werden zurückgegeben.
   * Die App zeigt das Ergebnis der Klassifizierungsanfrage im UI an.

Dieses Diagramm beschreibt den Ablauf des Klassifizierungsprozesses und die Interaktion zwischen der Mobile App, dem AI-Gateway und dem Datenbankserver für eine effiziente Analyse und Berichterstellung.

Aktivitätsdiagramm:

Ein Bild, das Text, Diagramm, Plan, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Sequenzdiagramme

Sequenzdiagramme sind genauso wie die Aktivitätsdiagramme ein wesentliches Werkzeug in der Softwaredokumentation zur Darstellung von Interaktionen zwischen Systemkomponenten im zeitlichen Verlauf. Dieses Kapitel enthält die relevanten Sequenzdiagramme für unser Projekt, die detailliert die Kommunikation und Abläufe zwischen den verschiedenen Systemelementen veranschaulichen.

Folgende Sequenzdiagramme wurden mit dem Tool von **sequencediagram.org** erstellt.

### Neuen Benutzer registrieren

Das folgende Sequenzdiagramm beschreibt den Prozess der Benutzerregistrierung in einem System. Es zeigt die zeitliche Abfolge der Interaktionen zwischen dem Benutzer, der Anwendung, dem AI-Gateway und dem Server.

Die wichtigsten Schritte in dieser Sequenz umfassen:

1. **Start der Registrierung:**
   * Der Benutzer öffnet die Anwendung und wählt die Option „Registrieren“.
   * Anschließend gibt er seine persönlichen Daten (E-Mail und Passwort) ein.
2. **Eingabevalidierung:**
   * Die Anwendung überprüft die Eingaben auf Korrektheit und Vollständigkeit.
   * Falls das Passwort nicht den Anforderungen entspricht, wird eine Fehlermeldung angezeigt („Passwort erfüllt nicht die Anforderungen“).
3. **Registrierungsanfrage:**
   * Wenn die Eingabe gültig ist, sendet die Anwendung die Registrierungsdaten (E-Mail, Passwort) an das AI-Gateway.
   * Das AI-Gateway leitet die Anfrage zur Validierung an den Server weiter.
4. **Überprüfung der E-Mail-Adresse:**
   * Der Server prüft, ob die angegebene E-Mail-Adresse bereits existiert.
   * Falls die E-Mail-Adresse bereits registriert ist, wird eine Fehlermeldung zurückgegeben („E-Mail bereits vergeben“).
5. **Erfolgreiche Registrierung:**
   * Falls die E-Mail-Adresse noch nicht existiert, speichert der Server die Benutzerdaten in der Datenbank.
   * Eine Bestätigung („Registrierung erfolgreich“) wird zurückgesendet und dem Benutzer angezeigt.

Dieses Sequenzdiagramm veranschaulicht die verschiedenen Pfade des Registrierungsprozesses, einschließlich möglicher Fehlerfälle, um eine fehlerfreie und sichere Registrierung zu gewährleisten.

Sequenzdiagramm:

Ein Bild, das Text, Diagramm, parallel, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Bestehenden Benutzer einloggen

Das folgende Sequenzdiagramm beschreibt den Ablauf des Benutzer-Login-Prozesses. Es zeigt die zeitliche Reihenfolge der Interaktionen zwischen dem Benutzer, der Anwendung, dem AI-Gateway und dem Server.

Die wichtigsten Schritte in dieser Sequenz umfassen:

1. **Eingabe der Zugangsdaten:**
   * Der Benutzer öffnet die Anwendung und gibt seine Anmeldedaten (E-Mail, Passwort) ein.
2. **Login-Anfrage:**
   * Die Anwendung sendet die Login-Anfrage mit den Benutzerdaten an das AI-Gateway.
   * Das AI-Gateway leitet die Anfrage an den Server weiter.
3. **Validierung der Benutzerdaten:**
   * Der Server überprüft die angegebenen Zugangsdaten.
4. **Antwort des Servers:**
   * Falls die Zugangsdaten korrekt sind, sendet der Server eine Bestätigung („Login erfolgreich“).
   * Falls die Zugangsdaten ungültig sind, wird eine Fehlermeldung zurückgegeben („Ungültige Anmeldedaten“).
5. **Anzeige der Rückmeldung an den Benutzer:**
   * Die Anwendung zeigt dem Benutzer entweder eine erfolgreiche Anmeldung oder eine Fehlermeldung an.

Dieses Diagramm veranschaulicht sowohl den erfolgreichen als auch den fehlgeschlagenen Login-Vorgang, um eine klare Übersicht über die Systeminteraktionen zu geben.

Sequenzdiagramm:

Ein Bild, das Text, Diagramm, Screenshot, parallel enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Analyse-Historie einsehen

Das folgende Sequenzdiagramm beschreibt den Ablauf der Anzeige der Analysehistorie eines Benutzers. Es zeigt die zeitliche Reihenfolge der Interaktionen zwischen dem Benutzer, der Anwendung, dem AI-Gateway und dem Server.

Die wichtigsten Schritte in dieser Sequenz umfassen:

1. **Anfrage zur Historie:**
   * Der Benutzer öffnet die Anwendung und wählt die Option „Historie“.
2. **Anforderung der Analysehistorie:**
   * Die Anwendung sendet eine Anfrage mit den Benutzerdaten an das AI-Gateway.
   * Das AI-Gateway leitet die Anfrage an den Server weiter.
3. **Validierung und Antwort des Servers:**
   * Der Server überprüft die Benutzerinformationen und sendet eine Antwort mit den gespeicherten Analysen.
4. **Anzeige der Analysehistorie:**
   * Falls der Benutzer Analysen durchgeführt hat, werden diese in der Benutzeroberfläche angezeigt.
   * Falls keine Analysen vorhanden sind, erhält der Benutzer die Meldung „Keine Analysehistorie verfügbar“.

Dieses Diagramm veranschaulicht sowohl den Fall einer vorhandenen als auch einer leeren Historie, um eine klare Übersicht über die Systeminteraktionen zu geben.

Sequenzdiagramm:

Ein Bild, das Text, Diagramm, Screenshot, parallel enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Bildvorhersage-Prozess

Dieses Sequenzdiagramm beschreibt den Ablauf der Vorhersage eines Bildes mittels eines KI-Modells in der mobilen App. Es zeigt die Interaktionen zwischen dem Benutzer, der mobilen App, dem AI-Gateway und dem Datenbankserver.

**Ablauf der Vorhersage:**

1. **Modellauswahl:**
   * Der Benutzer fordert verfügbare Modelle an.
   * Die mobile App sendet eine Anfrage an das AI-Gateway.
   * Das AI-Gateway antwortet mit einer Liste der verfügbaren Modelle.
   * Der Benutzer wählt ein Modell aus, das in der UI angezeigt wird.
2. **Bildaufnahme und Verarbeitung:**
   * Der Benutzer macht ein Foto.
   * Die mobile App sendet das Bild und das ausgewählte Modell an das AI-Gateway.
   * Das AI-Gateway überprüft, ob das Modell existiert und trainiert ist.
3. **Vorhersageprozess:**
   * Falls das Modell gültig ist, wird die Vorhersage durchgeführt und an die mobile App gesendet.
   * Falls das Modell nicht gültig ist, wird eine Fehlermeldung zurückgegeben.
   * Die App zeigt das Ergebnis in der UI an.
4. **Speicherung der Analyse (optional):**
   * Der Benutzer wird gefragt, ob die Analyse gespeichert werden soll.
   * Falls der Benutzer zustimmt, sendet die App eine Anfrage mit den Analyse- und Benutzerdaten.
   * Der Server validiert die Benutzerdaten und speichert die Analyse, falls der Benutzer neu ist.
   * Falls der Benutzer nicht gültig ist, wird eine Fehlermeldung gesendet.
5. **Anzeige des Ergebnisses:**
   * Falls die Anfrage gültig ist, wird die erfolgreiche Vorhersage angezeigt.
   * Falls nicht, wird eine Fehlermeldung in der UI angezeigt.

Dieses Diagramm stellt eine detaillierte Übersicht über die Systeminteraktionen dar, um eine KI-basierte Vorhersage durchzuführen und optional zu speichern.

Sequenzdiagramm:

Ein Bild, das Text, Diagramm, parallel, Plan enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Automatisches Zuschneiden von Bildern

Dieses Sequenzdiagramm beschreibt den Prozess des automatischen Zuschneidens eines Bildes in der mobilen App. Es zeigt die Interaktionen zwischen dem Benutzer, der mobilen App, dem AI-Gateway und dem Datenbankserver.

**Ablauf des automatischen Zuschneidens:**

1. **Bildaufnahme:**
   * Der Benutzer macht ein Foto für die Vorhersage.
   * Die mobile App bietet die Option einer automatischen Zuschneidefunktion an.
2. **Automatisches Zuschneiden (falls akzeptiert):**
   * Falls der Benutzer zustimmt, wird das Bild an den Datenbankserver gesendet.
   * Der Datenbankserver verarbeitet die Anfrage und schneidet das Bild automatisch zu.
   * Das zugeschnittene Bild wird zurück an die App gesendet.
   * Die App zeigt das bearbeitete Bild an und setzt den Vorhersageprozess fort.
3. **Manuelles Zuschneiden (falls abgelehnt):**
   * Falls der Benutzer die automatische Zuschneidefunktion ablehnt, kann er das Bild manuell zuschneiden.
   * Danach setzt er den Vorhersageprozess fort.

Dieses Diagramm stellt eine effiziente Methode zur Bildbearbeitung dar, indem es Benutzern eine automatische Funktion anbietet, aber auch eine manuelle Bearbeitung ermöglicht.

Sequenzdiagramm:

Ein Bild, das Text, Screenshot, Diagramm, parallel enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Modellinformationen anzeigen lassen

Dieses Sequenzdiagramm beschreibt den Prozess der Anzeige von Modellinformationen in einer mobilen App.

**Ablauf:**

1. **Anforderung von Modellinformationen:**
   * Der Benutzer fordert über die mobile App Informationen zu einem bestimmten Modell an.
2. **Weiterleitung der Anfrage:**
   * Die mobile App sendet die Anfrage an das AI-Gateway.
3. **Abruf der Modellinformationen:**
   * Das AI-Gateway verarbeitet die Anfrage und gibt das Modellinformationsdokument zurück.
4. **Anzeige in der Benutzeroberfläche:**
   * Die mobile App empfängt die Informationen und zeigt sie dem Benutzer in der UI an.

Dieses einfache, aber effektive Diagramm zeigt eine klare Kette von Aktionen für den Abruf von Modellinformationen in einer KI-gestützten Anwendung.

Sequenzdiagramm:

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Modelltraining

Dieses Diagramm beschreibt den Ablauf des Trainingsprozesses für ein Modell in einer KI-Anwendung.

**Ablauf:**

1. **Modellauswahl:**
   * Der Benutzer wählt spezifische oder alle Modelle zum Training aus.
2. **Abfrage der Trainingsparameter:**
   * Die mobile App fragt den Benutzer nach Trainingsparametern.
   * Der Benutzer gibt die Parameter an.
3. **Überprüfung der Benutzerberechtigungen:**
   * Die mobile App sendet die Trainingsanfrage mit Benutzerdaten an das AI-Gateway.
   * Das AI-Gateway überprüft, ob der Benutzer existiert und berechtigt ist.
   * Falls der Benutzer nicht berechtigt ist, wird ein Fehler zurückgegeben und in der UI angezeigt.
4. **Prüfung der Adminrechte:**
   * Falls der Benutzer existiert, wird geprüft, ob er Adminrechte hat.
   * Falls nicht, wird ein Fehler zurückgegeben und angezeigt.
5. **Trainingsdaten abrufen & Training durchführen:**
   * Falls der Benutzer Administrator ist, werden die Trainingsdaten in Batches geladen.
   * Die Daten werden seitenweise verarbeitet.
   * Das Modelltraining wird durchgeführt.
6. **Rückmeldung des Ergebnisses:**
   * Nach Abschluss des Trainings sendet das AI-Gateway eine Antwort an die mobile App.
   * Das Ergebnis wird in der Benutzeroberfläche angezeigt.

Dieses Diagramm stellt sicher, dass nur berechtigte Benutzer Modelltraining durchführen können und dass das Training effizient durch Batches abgewickelt wird.

Sequenzdiagramm:

Ein Bild, das Text, Diagramm, parallel, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Klassifikationsberichte erstellen

Dieses Diagramm beschreibt den Prozess zum Abrufen eines Klassifikationsberichts für ein Modell.

**Ablauf:**

1. **Modellauswahl:**
   * Der Benutzer wählt spezifische oder alle Modelle zur Klassifikation aus.
2. **Abfrage der Klassifikationsparameter:**
   * Die mobile App fragt den Benutzer nach Klassifikationsparametern.
   * Der Benutzer gibt die Parameter an.
3. **Überprüfung der Benutzerberechtigungen:**
   * Die mobile App sendet die Klassifikationsanfrage mit Benutzerdaten an das AI-Gateway.
   * Das AI-Gateway überprüft, ob der Benutzer existiert und berechtigt ist.
   * Falls der Benutzer nicht berechtigt ist, wird ein Fehler zurückgegeben und in der UI angezeigt.
4. **Prüfung der Adminrechte:**
   * Falls der Benutzer existiert, wird geprüft, ob er Adminrechte hat.
   * Falls nicht, wird ein Fehler zurückgegeben und angezeigt.
5. **Abrufen der Trainingsdaten & Berichtserstellung:**
   * Falls der Benutzer Administrator ist, werden die Trainingsdaten in Batches geladen.
   * Die Daten werden seitenweise verarbeitet.
   * Der Klassifikationsbericht wird erstellt.
6. **Rückmeldung des Berichts:**
   * Nach der Erstellung sendet das AI-Gateway eine Antwort mit dem Klassifikationsbericht an die mobile App.
   * Der Bericht wird in der Benutzeroberfläche angezeigt.

Dieses Diagramm stellt sicher, dass nur berechtigte Benutzer Klassifikationsberichte abrufen können und dass die Verarbeitung effizient in Batches erfolgt.

Sequenzdiagramm:

Ein Bild, das Text, Diagramm, parallel, Plan enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Komponentendiagramm

Dieses Komponentendiagramm dient dazu, die strukturelle Organisation des Systems zu visualisieren und die Beziehungen zwischen seinen Komponenten aufzuzeigen. Es bietet einen ganzheitlichen Überblick über die einzelnen Bausteine des Systems und ihre Interaktionen.

**Beschreibung:**

Das Komponentendiagramm zeigt die verschiedenen technischen Komponenten des Systems sowie deren Kommunikationswege und Schnittstellen. Jede Komponente übernimmt eine spezifische Aufgabe und interagiert mit seinem Kommunikationspartner über definierte Schnittstellen. Das System besteht aus einem Client, einer Datenbank, einem Backend zur Datenverarbeitung und einem KI-Backend (Gateway) zur Modellierung und Vorhersage.

**Komponenten:**

**1. Client-Anwendung**

* **Technologie:** Kotlin (Android)
* **Beschreibung:** Die Client-Anwendung ist die Benutzeroberfläche des Systems. Sie ermöglicht es den Nutzern, mit dem System zu interagieren, Daten einzugeben und Ergebnisse abzurufen.
* **Schnittstelle:** Kommuniziert über **HTTP** mit dem Backend.

**2. Datenbank-Backend**

* **Technologie:** AdonisJS, Python
* **Beschreibung:** Das Database-Backend dient als Schnittstelle zur Speicherung und Bereitstellung von Benutzer- und Trainingsdaten für KI-Modelle. Es empfängt Daten vom Client und speichert diese in der MongoDB.
* **Schnittstellen:**
  + **HTTP:** Kommunikation mit dem Client über das Gateway.
  + **MongoDB:** Speicherung und Verwaltung von Benutzerdaten/Trainingsdaten.

**3. MongoDB (NoSQL-Datenbank)**

* **Technologie:** MongoDB
* **Beschreibung:** Diese NoSQL-Datenbank speichert alle relevanten Benutzerdaten sowie Trainingsdaten für das KI-Modell. Sie wird vom Datenbank-Backend verwaltet und dient als zentrale Datenquelle.
* **Schnittstellen:** Verknüpft mit dem Datenbank-Backend zur Speicherung und Bereitstellung von Daten.

**4. KI-Backend**

* **Technologie:** Python
* **Beschreibung:** Das KI-Backend ist für das Training von Modellen und das Erstellen von Vorhersagen verantwortlich. Es verarbeitet die Daten aus der Datenbank und stellt Analysen zur Verfügung. Außerdem dient es als Gateway zwischen Frontend und Datenbank-Backend.
* **Schnittstellen:**
  + **HTTP:** Kommunikation mit dem Client und Datenbank-Backend.
  + **PORT 8000:** API für Frontend
  + **PORT 6969:** Automatischer Bildzuschnitt (Resize Image).
  + **PORT 3333:** Datenbank-Backend

**Systemgrenzen:**

Das System umfasst interne Komponenten, die über definierte Schnittstellen miteinander kommunizieren. Die Client-Anwendung, das Datenbank-Backend und die Datenbank bilden zusammen das zentrale System, während das KI-Backend die Funktion zur Modellierung und Vorhersage darstellt und als Gateway dient.

Komponentendiagramm:

Ein Bild, das Text, Diagramm, Plan, technische Zeichnung enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Verteilungsdiagramm

Das Verteilungsdiagramm zeigt an, wie die einzelnen Teile der Software auf die Hardwarekomponenten verteilt sind und wie die Hardwarekomponenten miteinander verbunden sind. Sprich: Auf welchem Rechner läuft welche Software und wie sind diese übers Netzwerk miteinander verbunden. Es ist dem Komponentendiagramm sehr ähnlich.

**Komponenten:**

* **Device (Mobile) – Kotlin Android:**
  + Beschreibung: Die mobile Anwendung dient als Benutzer-Frontend, über das Nutzer mit dem System interagieren können. Sie sendet Anfragen über HTTP an das Backend.
* **AI-Backend – Python:**
  + Beschreibung: Diese Komponente verarbeitet KI-bezogene Anfragen, trainiert Modelle und führt Vorhersagen durch. Sie empfängt Daten über HTTP und gibt die verarbeiteten Ergebnisse zurück.
* **DB-Backend – AdonisJS, FastAPI:**
  + Beschreibung: Diese Schicht fungiert als Schnittstelle zur Datenbank und bietet APIs zur Speicherung und Bereitstellung von Daten. AdonisJS verwaltet Backend-Logik, während FastAPI für schnelle API-Anfragen genutzt wird.
* **DB-Server – MongoDB:**
  + Beschreibung: Die NoSQL-Datenbank speichert Benutzerdaten, Trainingsdaten und andere relevante Informationen, die von den Backend-Diensten benötigt werden.

**Interaktionen:**

* Die mobile Anwendung kommuniziert über HTTP (Port 8000) mit dem AI-Backend.
* Das AI-Backend kommuniziert über HTTP (Port 8000) mit dem DB-Backend, um notwendige Daten abzurufen oder zu speichern.
* Das DB-Backend verwaltet die Daten in der MongoDB-Datenbank und stellt sie dem AI-Backend zur Verfügung.

**Systemgrenzen:**  
Das Diagramm beschreibt die Verteilung der Software-Komponenten auf die jeweiligen Hardware-Ressourcen und veranschaulicht die Kommunikation zwischen diesen Komponenten im System.

Verteilungsdiagramm:

Ein Bild, das Text, Diagramm, Rechteck, Plan enthält.

KI-generierte Inhalte können fehlerhaft sein.

## C4-Modell

In diesem Kapitel wird das **C4-Modell** zur Architekturvisualisierung des Systems vorgestellt. Das C4-Modell bietet eine strukturierte Methode zur Darstellung von Softwarearchitekturen auf verschiedenen Abstraktionsebenen. Es hilft dabei, die Systemkomponenten sowie deren Beziehungen zu verstehen und ermöglicht eine klare Kommunikation zwischen Entwicklern, Architekten und anderen Stakeholdern.

Das Kapitel ist in mehrere Unterpunkte unterteilt, welche sich jeweils auf einen speziellen Bereich der Anwendung konzentrieren. Durch diese schrittweise Detaillierung bietet das Kapitel eine umfassende und verständliche Darstellung der Systemarchitektur, von der höchsten Abstraktionsebene bis hin zur konkreten Implementierung.

### Top-Layer

Der **Top-Layer** stellt die oberste Ebene des C4-Modells dar und dient dazu, einen Überblick über das gesamte System und seine Hauptbestandteile zu geben. Diese Ebene hilft, das System in seinem Kontext zu verstehen und die zentralen Komponenten sowie deren Interaktionen zu identifizieren.

#### Kontext-Ebene

Die Kontext-Ebene beschreibt das **DermAI-System** im Gesamtzusammenhang und zeigt, wie es mit externen Akteuren interagiert. In diesem Fall interagiert der **Benutzer** mit dem System, indem er eine **Analyse von Hautläsionen** anfordert. Das System ermöglicht es dem Benutzer, Bilder hochzuladen und von einer **KI-basierten Analyse** bewerten zu lassen.

##### Benutzer

Das zugehörige Diagramm visualisiert diese Beziehung, indem es das **DermAI-System als zentrales Software-System** darstellt, das von einem **Benutzer** genutzt wird. Die Verbindung zwischen beiden beschreibt die zentrale Funktion des Systems.

Ein Bild, das Text, Screenshot, Schrift, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### Admin

Die **Kontext-Ebene für Administratoren** beschreibt die Interaktion zwischen einem **Admin** und dem **DermAI-System**. Während normale Benutzer das System zur Analyse von Hautläsionen nutzen, haben Administratoren weitergehende Funktionen, insbesondere:

* **Training von Modellen:** Administratoren können neue KI-Modelle trainieren oder bestehende Modelle verbessern.
* **Erstellung von Klassifikationsberichten:** Sie können detaillierte Berichte über die Modellleistung generieren.

Das zugehörige Diagramm zeigt diese Beziehung, indem es den **Admin als Akteur** darstellt, der das **DermAI-System verwendet**.

Ein Bild, das Text, Screenshot, Schrift, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.

#### Container-Ebene

### AI-Gateway

#### Komponenten-Ebene

#### Code-Ebene

### Backend

#### Komponenten-Ebene

#### Code-Ebene

### Frontend

#### Komponenten-Ebene

#### Code-Ebene

## Ablaufdiagramm

In diesem Kapitel wird ein zentrales Ablaufdiagramm vorgestellt, das zu Beginn der Projektplanung als Orientierungshilfe diente. Es veranschaulicht den grundlegenden Ablauf für einen Benutzer innerhalb des Systems und bietet eine erste visuelle Darstellung der wichtigsten Prozessschritte.

Das Diagramm hilft dabei, die Hauptinteraktion des Nutzers mit dem System zu verstehen und dient als Grundlage für die spätere detaillierte Modellierung und Implementierung der Systemarchitektur.

Ein Bild, das Text, Diagramm, Reihe, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein. Ein Bild, das Text, Diagramm, Reihe, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Diagramm, Reihe, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Softwareprogramme / Komponenten

Ein erfolgreiches Softwareprojekt erfordert eine durchdachte Auswahl und den gezielten Einsatz geeigneter Programme, Softwarekomponenten und Frameworks. In diesem Kapitel werden die für die Entwicklung des Projekts verwendeten Tools und Technologien angegeben. Dazu gehören sowohl die Entwicklungsumgebungen und unterstützenden Programme als auch die spezifischen Frameworks und Bibliotheken, die für die Umsetzung der Softwarearchitektur essenziell sind.

### Softwareprogramme

Im Laufe der Entwicklung wurden einige Tools/Programme verwendet, um das Projekt beziehungsweise seine Dokumentation zu bearbeiten. Diese sind hier aufgelistet:

|  |  |  |  |
| --- | --- | --- | --- |
| **Programm** | **Version** | **Hersteller** | **Beschreibung** |
| PyCharm | 2023.3.2 | JetBrains | Entwicklungsumgebung für Python |
|  |  |  |  |
|  |  |  |  |
| Github Desktop + git | 3.3.13 (x64) | Github, Inc. | Versionskontroll-Software zur Code-Sicherung |
| Microsoft Word | / | Microsoft | Textvearbeitungsprogramm |

### Softwarekomponenten

Die unten gelisteten Softwarekomponenten sind die Sprachen beziehungsweise Frameworks welche zur Realisierung des Projektes verwendet wurden.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SW-Komponent** | **Version** | **Hersteller** | **Bezugsquelle** | **SW-Lizenz** | **Beschreibung** |
| Python | 3.11.0 | Python Software Foundation | https://www.python.org | / | Programmiersprache |
| Tensorflow Keras |  |  |  |  |  |
| Pytorch |  |  |  |  |  |
| Scikit-Learn |  |  |  |  |  |

# Projektdurchführung

*Kapitel verfasst von Daniel Jessner*

Die erfolgreiche Umsetzung eines Softwareprojekts erfordert eine klare Aufgabenverteilung und eine enge Zusammenarbeit der beteiligten Entwickler. In diesem Kapitel wird die konkrete Projektdurchführung dokumentiert, indem die einzelnen Teammitglieder ihre jeweiligen Beiträge detailliert vorstellen. Dabei wird nicht nur auf die technische Implementierung eingegangen, sondern auch auf die Herausforderungen, Lösungsansätze und spezifischen Entscheidungen, die während der Entwicklung getroffen wurden.

Jeder Abschnitt dieses Kapitels widmet sich einer zentralen Systemkomponente, für die jeweils ein Teammitglied verantwortlich war:

* **Daniel Jessner** beschreibt die Entwicklung der **KI-Komponente** und die Implementierung des **Gateways**, das als Schnittstelle zwischen verschiedenen Systemen fungiert. Dabei geht er insbesondere auf die Anbindung der KI-Modelle, die Datenverarbeitung und die Kommunikation mit anderen Systembestandteilen ein.
* **Jonas Maier** erläutert die Konzeption und Umsetzung der **Datenbankarchitektur** sowie die Entwicklung des **Backends**. Hierbei werden Themen wie Datenmanagement, API-Schnittstellen und Sicherheitsaspekte behandelt, die für eine effiziente und skalierbare Systemarchitektur essenziell sind.
* **Jonas Bogensberger** fokussiert sich auf die Entwicklung des **Frontends**, das die Benutzerinteraktion ermöglicht. In diesem Abschnitt wird erläutert, wie das User Interface gestaltet wurde, welche Frameworks und Technologien zum Einsatz kamen und wie die Verbindung zwischen Frontend und Backend realisiert wurde.

Dieses Kapitel liefert somit eine strukturierte Übersicht über die technische Umsetzung des Projekts. Es verdeutlicht nicht nur den individuellen Beitrag jedes Teammitglieds, sondern auch das Zusammenspiel der verschiedenen Systemkomponenten, das für die Gesamtfunktionalität der Anwendung entscheidend ist.

## Daniel Jessner (KI-Komponente, Gateway)

*Kapitel verfasst von Daniel Jessner*

Dieses Kapitel beschäftigt sich mit der Konzeptionierung und Implementierung der KI-Komponente des DermaAI-Systems. Diese stellt nicht nur die gesamte KI-Funktionalität mit allen bereits erwähnten Optionen bereit, sondern dient auch als Gateway zwischen Frontend und Backend, wenn dies erforderlich ist.

### Tech Stack

Die Entwicklung der KI-Komponente basiert auf einer Vielzahl moderner Technologien und Bibliotheken, die speziell für maschinelles Lernen, Deep Learning und die Bereitstellung von KI-gestützten Diensten optimiert sind. Die wichtigsten verwendeten Technologien werden folgend erklärt.

#### Programmiersprache: Python

Python ist die Hauptprogrammiersprache für die Implementierung der KI-Komponente. Aufgrund seiner umfangreichen Bibliotheken, einfachen Syntax und starken Community-Unterstützung ist Python die bevorzugte Wahl für maschinelles Lernen und Deep Learning.

Da unteranderem Tensorflow keine Unterstützung für die neuesten Python-Versionen bietet, wurde auf die ältere, aber dennoch stabile Version **3.11.0** zurückgegriffen.

#### Machine Learning & Deep Learning Bibliotheken

Um die KI-Komponente des DermaAI-Systems effizient zu entwickeln, werden verschiedene Bibliotheken für maschinelles Lernen und Deep Learning eingesetzt. Diese Bibliotheken ermöglichen es, sowohl klassische Algorithmen als auch tief neuronale Netze zu implementieren, zu trainieren und zu optimieren. Während Scikit-Learn vor allem für traditionelle ML-Methoden verwendet wird, kommen TensorFlow und PyTorch für komplexe neuronale Netzwerke zum Einsatz.

Durch den gezielten Einsatz dieser Technologien kann das System eine hohe Erkennungsgenauigkeit erzielen, effiziente Modelltrainings durchführen und flexibel auf neue Anforderungen reagieren. Die folgenden Abschnitte geben einen detaillierten Überblick über die verwendeten Bibliotheken und deren spezifische Anwendungsbereiche innerhalb des Projekts.

##### Scikit-Learn

Scikit-Learn ist eine der bekanntesten Bibliotheken für klassisches maschinelles Lernen. Sie wird in der KI-Komponente für die Implementierung verschiedener Klassifikations- und Regressionsmodelle eingesetzt, insbesondere für:

* Vorverarbeitung der Daten
* Feature Engineering
* Implementierung von Basismodellen
* Evaluierung der Modellleistung

##### TensorFlow

TensorFlow ist eine Open-Source-Plattform für maschinelles Lernen, die speziell für skalierbare neuronale Netzwerke entwickelt wurde. In der KI-Komponente wird TensorFlow für:

* Das Training und die Bereitstellung tiefer neuronaler Netze
* Die Optimierung von Modellen für mobile Anwendungen
* Die Nutzung von GPU-Beschleunigung zur Verbesserung der Trainingsleistung

##### PyTorch

PyTorch ist eine flexible Deep-Learning-Bibliothek, die aufgrund ihrer dynamischen Berechnungsgraphen und einfachen Handhabung für Forschungsprojekte beliebt ist. Sie wird in der KI-Komponente genutzt für:

* Das Training experimenteller neuronaler Netzwerke
* Die Implementierung von Transfer Learning
* Die Optimierung und Feinabstimmung bestehender Modelle

#### Web Framework: FastAPI

FastAPI ist ein modernes Webframework für den Aufbau performanter APIs mit Python. Es wird als Kern der Gateway-Komponente genutzt und bietet folgende Vorteile:

* **Asynchrone Verarbeitung**: Unterstützt effiziente API-Anfragen mit hoher Geschwindigkeit
* **Einfache Integration**: Lässt sich problemlos mit TensorFlow und PyTorch kombinieren
* **Automatische Dokumentation**: Erzeugt automatisch OpenAPI- und Swagger-Dokumentationen

#### Weitere Bibliotheken und Tools

Im Laufe der Entwicklung wurde eine Vielzahl weiterer, oben nicht speziell genannten Bibliotheken verwendet. Diese werden jeweilig dann erwähnt, wenn sie verwendet werden.

### Modules

#### Image Processor

#### Model Processor

### Modelle

In diesem Abschnitt werden die im DermaAI-System verwendeten Modelle detailliert beschrieben. Die Wahl der Modelle spielt eine entscheidende Rolle für die Qualität der Hautläsionsanalyse und beeinflusst sowohl die Genauigkeit als auch die Effizienz des Systems.

Es werden verschiedene Machine-Learning- und Deep-Learning-Modelle von verschiedenen Anbietern eingesetzt, um dem Benutzer ein breites Spektrum an Auswahlmöglichkeiten zu bieten. Die Auswahl der Architektur, die Hyperparameter-Optimierung und das Training der Modelle sind essenzielle Schritte, um eine zuverlässige und performante KI-Lösung zu gewährleisten. Die insgesamt **36** implementierten Modelle werden im DermaAI-Life-Cycle sowohl eigenständig trainiert und ausgewertet als auch dem Benutzer dynamisch zur Verfügung gestellt.

Im Folgenden werden die eingesetzten Modelle, ihre Struktur sowie deren spezifische Implementierung innerhalb des Systems vorgestellt.

#### Modell-Interface

Das Modell-Interface bildet die zentrale Schnittstelle für die Implementierung verschiedener Machine-Learning- und Deep-Learning-Modelle im DermaAI-System. Es stellt eine abstrakte Basisklasse bereit, die die grundlegenden Methoden definiert, die jedes Modell implementieren muss, um eine einheitliche Struktur und Wiederverwendbarkeit innerhalb des Systems zu gewährleisten.

##### Imports

Für die Implementierung des Modellinterfaces werden folgende Module benötigt:

* **json** – zur Verarbeitung und Speicherung von Metadaten
* **os** – für Dateisystemoperationen
* **ABC** – zur Definition einer abstrakten Basisklasse
* **abstractmethod** – zur Deklaration abstrakter Methoden innerhalb der Basisklasse

##### IModel.py

Die abstrakte Klasse **IModel** stellt sicher, dass jedes Modell grundlegende Funktionalitäten wie das Trainieren, Klassifizieren und Evaluieren von Daten bereitstellt. Darüber hinaus enthält das Interface Mechanismen zum Speichern und Laden von Metadaten, um sicherzustellen, dass wichtige Informationen über Modellkonfigurationen und Klassifikationslabels erhalten bleiben.

Durch diese Architektur wird die Entwicklung neuer Modelle vereinfacht, da diese lediglich die definierten Methoden implementieren müssen, um nahtlos in das bestehende System integriert zu werden.

Code-Implementierung:

class IModel(ABC):

Da IModel von **ABC** (Abstract Base Class) erbt, können keine direkten Instanzen dieser Klasse erstellt werden. Stattdessen müssen alle abgeleiteten Modellklassen die definierten abstrakten Methoden (**@abstractmethod**) implementieren.

###### Methoden

In diesem Abschnitt wird die Implementierung der oben beschriebenen Klasse detailliert analysiert. Dabei werden die einzelnen Methoden, ihre Funktionsweise sowie ihre spezifische Rolle innerhalb der KI-Komponente erläutert.

Anmerkung: Mit (**@abstractmethod** gekennzeichnete Methoden werden von der eigentlichen Modellimplementierung überschrieben, da gewisse Abweichungen in der Verwendung unterschiedlicher Modelle existieren. Die Methoden **load\_model\_metadata** und **save\_model\_metadata** werden, so wie sie sind, weitervererbt, weil der Prozess des Speichern und Ladens der Metadaten immer grundsätzlich der selbe bleibt.

Check(self)

Diese Methode dient dazu, zu überprüfen, ob das Modell ordnungsgemäß geladen wurde oder bereit für die Nutzung ist. Ihre Implementierung kann beispielsweise eine Überprüfung auf vorhandene gespeicherte Gewichte oder Modelldateien enthalten.

Code-Implementierung:

@abstractmethod

def check(self):  
 pass

train(self, dataset, epochs, reshape\_size)

Die train-Methode ist für das Training des Modells zuständig. Sie nimmt folgende Parameter entgegen:

* dataset: Die Trainingsdaten, mit denen das Modell trainiert wird (aufgeteilt in x=Daten, y=Labels).
* epochs: Die Anzahl der Durchläufe (Epochen), die das Modell über die Trainingsdaten macht.
* reshape\_size: Die Größe, auf die die Eingabebilder skaliert werden müssen.

Jede spezifische Modellimplementierung muss diese Methode mit der entsprechenden Trainingslogik versehen.

Code-Implementierung:

@abstractmethod

def train(self, dataset, epochs, reshape\_size):  
 pass

Optionale Parameter

Je nach Modellanbieter sowie Art und Weise, wie ein Modell zu verwalten ist, gibt es bestimmte Abweichungen in den Parametern beim Trainieren. TensorFlow und PyTorch bieten unteranderem die Möglichkeit, **Batch Size** und **Learning Rate** zu beinflussen und auch festzulegen, wie oft über den Trainingsdatensatz iteriert wird (**Epochen**). Scikit-Learn bietet diese Optionen nicht. Da von außen auf die Modelle jeweils mit der gleichen Methode zugegriffen und es dem Benutzer (Admin) ermöglicht werden soll, die Epochen-Anzahl einzustellen (längeres oder kürzeres Training), werden die Epochen immer mitgegeben und bei Bedarf verwendet.

Die Batch Size und die Learning Rate stellen hingegen Parameter dar, die durch Testen der Modelle entstehen und werden, wenn sie benötigt werden, direkt in der Methodendeklaration als sogenannte „Keyword Arguments“ angeführt (siehe Code-Implementierung). Für einige Modelle funktioniert eine Batch Size von **32**, für andere jedoch eine von **16**. Das hängt von der Modellstruktur und deren Forderungen an die Hardware (CPU, GPU) ab. Für die Learning Rate hat sich beim Testen der Modelle ein Standardwert von **0.001** als genau und stabilherauskristallisiert.

Batch Size

Die **Batch Size** bezieht sich auf die Anzahl von Trainingsbeispielen (Datenpunkten), die gleichzeitig durch das Modell verarbeitet werden, bevor die Modellparameter (z.B. die Gewichte eines neuronalen Netzwerks) aktualisiert werden. In einem Training durchläuft das Modell in mehreren **Epochs** (vollständige Durchläufe durch das gesamte Trainingsdatenset, sollte das Modell dies unterstützen) alle Trainingsdaten. Diese Daten werden jedoch in **Batches** unterteilt, um den Trainingsprozess effizienter und stabiler zu gestalten.

Legt man beispielsweise eine **Batch Size** von **32** fest, bedeutet dies, dass bei jedem Trainingsschritt das Modell mit 32 Eingabedatenpunkten gleichzeitig arbeitet.

Learning Rate

Die **Learning Rate (lr)** ist ein Hyperparameter, der bestimmt, wie groß die Anpassung der Modellgewichte bei jedem Schritt des Trainingsprozesses ist. Sie steuert die Geschwindigkeit, mit der das Modell lernt.

* **Hohe Lernrate:**  
  Eine hohe Lernrate kann dazu führen, dass das Modell sehr schnell lernt, aber die Gefahr besteht, dass es die optimalen Modellparameter überspringt oder instabil wird. Das Modell könnte dann in der Nähe des Minimums "herumhüpfen", ohne es zu erreichen.
* **Niedrige Lernrate:**  
  Eine niedrige Lernrate sorgt für langsameres, stabileres Lernen. Das Modell passt die Gewichte mit kleinen, präzisen Schritten an, was zu einer besseren Konvergenz führen kann. Allerdings kann es auch länger dauern, bis das Modell das optimale Minimum erreicht, und in einigen Fällen könnte es in lokalen Minima stecken bleiben.

make\_prediction(self, image)

Diese Methode führt eine Vorhersage auf Basis eines übergebenen Bildes durch. Die Implementierung muss sicherstellen, dass das Bild korrekt vorverarbeitet wird, bevor es an das Modell übergeben wird.

@abstractmethod

def make\_prediction(self, image):  
 pass

get\_classification\_report(self, dataset, reshape\_size)

Diese Methode erstellt eine Evaluationsübersicht für das Modell, indem es Klassifikationsmetriken wie Genauigkeit, Präzision und Recall berechnet.

Die Parameter:

* dataset: Der Datensatz zur Evaluierung.
* reshape\_size: Die Größe, auf die die Bilder skaliert werden müssen.

Diese Methode hilft dabei, die Leistung des Modells zu bewerten und zu verbessern.

Code-Implementierung:

@abstractmethod

def get\_classification\_report(self, dataset, reshape\_size):  
 pass

save\_model\_metadata(self, model\_save\_path, model\_name, reshape\_size, class\_labels)

Diese Methode speichert Metadaten des Modells in einer JSON-Datei (model\_metadata.json). Dabei werden folgende Informationen gespeichert:

* reshape\_size: Die Bildgröße, mit der das Modell trainiert wurde.
* class\_labels: Die Klassen, mit denen das Modell trainiert wurde.

Falls die Metadaten-Datei bereits existiert, wird sie geladen und aktualisiert. Ansonsten wird eine neue Datei erstellt.

Code-Implementierung:

def save\_model\_metadata(self, model\_save\_path, model\_name, reshape\_size, class\_labels):  
 metadata\_file = os.path.join(model\_save\_path, "model\_metadata.json")  
  
 if os.path.exists(metadata\_file):  
 with open(metadata\_file, "r") as f:  
 metadata = json.load(f)  
 else:  
 metadata = {}  
  
 metadata[model\_name] = {  
 "reshape\_size": reshape\_size,  
 "class\_labels": class\_labels }  
  
 with open(metadata\_file, "w") as f:  
 json.dump(metadata, f, indent=4)

load\_model\_metadata(self, model\_save\_path, model\_name)

Diese Methode lädt die gespeicherten Metadaten eines Modells. Falls die Datei existiert, werden reshape\_size und class\_labels zurückgegeben. Andernfalls werden Standardwerte (224 für reshape\_size und None für class\_labels) verwendet.

Code-Implementierung:

def load\_model\_metadata(self, model\_save\_path, model\_name):  
 metadata\_file = os.path.join(model\_save\_path, "model\_metadata.json")  
  
 if os.path.exists(metadata\_file):  
 with open(metadata\_file, "r") as f:  
 metadata = json.load(f)  
  
 if model\_name in metadata:  
 reshape\_size = metadata[model\_name].get("reshape\_size", 224)  
 class\_labels = metadata[model\_name].get("class\_labels", None)  
 return reshape\_size, class\_labels  
  
 return 224, None

#### Scikit-Learn

Zuallererst wird auf die Modelle aus dem Scikit-Learn-Angebot eingegangen. Diese waren auch innerhalb der Anwendung als erstes implementiert und bilden den Grundstein für die Struktur und Verwaltung aller anderen Vorgänge im System.

##### Basismodell

Das Basismodell für Scikit-Learn stellt eine Wrapper-Klasse für die instanziierten Modelle dar und definiert die Schnittstellen nach außen. Durch sie erhält das Modell die eigentlichen angebotenen Funktionen und kommuniziert mit Klassen, die eine Aggregation mit ihr eingehen.

###### Imports

Für die Implementierung der Basismodell-Klasse für die Scikit-Learn-Modelle werden folgende Module benötigt:

* **os**: Wird für Dateioperationen verwendet, um zu überprüfen, ob ein Modell bereits gespeichert wurde und um mit Dateipfaden zu arbeiten. Dies ist notwendig, um das Modell zu laden oder zu speichern.
* **joblib**: Wird zum Speichern und Laden des Modells genutzt. Es ermöglicht die persistente Speicherung des Modells auf der Festplatte, um es später wieder zu verwenden.
* **numpy**: Wird für numerische Berechnungen verwendet. Insbesondere wird es eingesetzt, um mit Arrays zu arbeiten, die die Daten und Labels des Modells enthalten.
* **classification\_report** aus **sklearn.metrics**: Wird verwendet, um Klassifikationsmetriken wie Präzision, Recall und F1-Score zu berechnen, um die Leistung eines Modells zu bewerten.
* **train\_test\_split** aus **sklearn.model\_selection**: Wird verwendet, um Datensätze in Trainings- und Testdaten aufzuteilen, was für eine objektive Modellbewertung erforderlich ist.
* **preprocess\_image** aus **modules.ImageProcessor**: Eine benutzerdefinierte Funktion, die zum Vorverarbeiten von Bildern dient, um sicherzustellen, dass die Bilder in einem für das Modell geeigneten Format vorliegen.
* **IModel** aus **..IModel**: Benutzerdefinierte Schnittstelle/Basisklasse jede Verwendung von Modellen innerhalb des Systems

###### Training\_basemodel\_sklearn.py

Folgende Klasse implementiert oben genanntes IModel-Interface und stellt die logische Funktionalität für die Scikit-Learn-Modelle bereit.

Code-Implementierung:

class BaseModel(IModel):  
 def \_\_init\_\_(self, model, model\_save\_path, model\_name\_extension=""):  
 self.model = model

self.model\_save\_path = model\_save\_path

self.model\_name = self.model.\_\_class\_\_.\_\_name\_\_ + model\_name\_extension

Konstruktor

Die \_\_init\_\_-Methode ist der **Konstruktor** der BaseModel-Klasse und dient zur Initialisierung eines Modell-Objekts. Sie speichert das übergebene Modell sowie den Speicherpfad für das Modell und generiert automatisch einen Modellnamen anhand der Klasse des Modells.

Parameter

* model (**object**):  
  Das initialisierte Modell, das von dieser Instanz verwaltet wird.
* model\_save\_path (**str**):  
  Der Pfad, unter dem das Modell gespeichert wird.
* model\_name\_extension (**str**, optional, Standard: ""):  
  Ein optionaler String, der an den Modellnamen angehängt wird, da es vorkommen kann, dass Modelle den gleichen Klassennamen besitzen und sich nur anhand ihrer Einstellungen unterscheiden.

Methoden

In diesem Abschnitt wird die Implementierung der oben beschriebenen Klasse detailliert analysiert. Dabei werden die einzelnen Methoden, ihre Funktionsweise sowie ihre spezifische Rolle innerhalb der KI-Komponente erläutert.

check(self)

In der folgenden Methode check() wird überprüft, ob ein trainiertes Modell vorhanden ist und erfolgreich geladen werden kann. Die Methode verfolgt dabei den folgenden Ablauf:

1. **Pfad des Modells festlegen:**  
   Zunächst wird der Pfad des gespeicherten Modells konstruiert. Der Pfad setzt sich zusammen aus dem Basisverzeichnis self.model\_save\_path und dem Modellnamen, der durch self.model\_name definiert ist. Es wird angenommen, dass das Modell im Format .joblib gespeichert ist, was durch den String f"{self.model\_save\_path}trained\_{self.model\_name}.joblib" realisiert wird.
2. **Überprüfung, ob die Datei existiert:**  
   Danach wird mittels os.path.isfile(model\_path) geprüft, ob an dem angegebenen Pfad eine Datei existiert. Sollte dies der Fall sein, wird die Ausführung fortgesetzt.
3. **Laden des Modells:**  
   Wenn die Datei existiert, wird versucht, das Modell mit der joblib.load(model\_path) Funktion zu laden. joblib ist eine weit verbreitete Bibliothek zum Speichern und Laden von Python-Objekten, insbesondere für maschinelles Lernen, da sie die effiziente Speicherung und das Laden von großen Modellen ermöglicht.
4. **Rückgabewert bei Erfolg:**  
   Wird das Modell erfolgreich geladen, gibt die Methode True zurück, um anzuzeigen, dass das Modell erfolgreich geladen wurde.
5. **Fehlerbehandlung:**  
   Tritt während des Ladeprozesses ein Fehler auf (zum Beispiel, wenn die Datei beschädigt ist oder nicht geladen werden kann), wird eine Ausnahme (Exception) ausgelöst. In diesem Fall gibt die Methode False zurück, was darauf hinweist, dass das Modell nicht erfolgreich geladen werden konnte.
6. **Rückgabewert, wenn keine Datei vorhanden ist:**  
   Falls keine Datei am angegebenen Pfad gefunden wird, wird ebenfalls False zurückgegeben, da das Modell nicht geladen werden konnte.

Zusammenfassend stellt diese Methode sicher, dass nur dann mit dem Modell gearbeitet wird, wenn es erfolgreich geladen werden konnte. Ansonsten wird der Prozess abgebrochen und False zurückgegeben, um den Benutzer oder das System über das Fehlschlagen des Ladeprozesses zu informieren.

Code-Implementierung:

def check(self):  
 try:  
 model\_path = f"{self.model\_save\_path}trained\_{self.model\_name}.joblib"

if os.path.isfile(model\_path):  
 self.model = joblib.load(model\_path)  
 return True

except Exception:  
 return False

return False

train(self, dataset, epochs, reshape\_size, batch\_size=None, lr=None):

In der Methode train() wird der Trainingsprozess eines Scikit-Learn-Modells durchgeführt und das trainierte Modell anschließend gespeichert. Folgende Schritte werden getätigt:

**1. Extraktion der Eingabedaten und Zielwerte:**

Die Methode entpackt das dataset in zwei separate Variablen: x für die Eingabedaten und y für die zugehörigen Zielwerte.

**2. Training des Modells:**

Das Modell wird mittels der fit()-Methode auf den Trainingsdaten (x) und den Zielwerten (y) trainiert. Die fit()-Methode des Modells ist die Standardmethode in vielen maschinellen Lernmodellen, um das Modell auf den Trainingsdaten zu trainieren, hier aus dem Scikit-Learn-Angebot.

**3. Speichern des trainierten Modells:**

Nach dem erfolgreichen Training wird das Modell mit joblib.dump() gespeichert. Der Speicherort für das Modell wird durch die Kombination des Basisverzeichnisses self.model\_save\_path, dem Modellnamen self.model\_name und einer Dateiendung (die durch self.model\_name\_extension festgelegt wird) bestimmt. Das Modell wird so mit einem .joblib-Format gespeichert, das eine effiziente Speicherung und ein einfaches Laden von Modellen ermöglicht.

* Die Methode gibt eine Bestätigungsmeldung aus, dass das Modell erfolgreich trainiert und gespeichert wurde: 'Trained and dumped model: ', self.model\_name

**4. Speichern der Metadaten:**

Nachdem das Modell gespeichert wurde, werden die Metadaten des Modells mit der Methode save\_model\_metadata() gespeichert. Diese Methode speichert die reshape\_size und die Labels (unique) die beim Training verwendet wurden.

**5. Fehlerbehandlung:**

Sollte während des Trainings oder beim Speichern des Modells ein Fehler auftreten, wird eine Ausnahme (Exception) abgefangen. Der Fehler wird in der Variablen error gespeichert, und es wird eine Fehlermeldung in der Form von '[Modellname] could not be trained. Error: [Fehlermeldung]' generiert. In diesem Fall gibt die Methode False zurück und gibt die Fehlermeldung als zweiten Wert zurück.

**6. Rückgabewerte:**

* **Erfolgreiches Training:** Wenn das Modell erfolgreich trainiert und gespeichert wurde, gibt die Methode True und None zurück.
* **Fehlgeschlagenes Training:** Sollte ein Fehler auftreten, gibt die Methode False und die Fehlernachricht zurück.

**Zusammenfassung:**

Die Methode train() führt den Trainingsprozess eines Modells durch und speichert das Modell anschließend. Sollte ein Fehler auftreten, wird dieser abgefangen und eine entsprechende Fehlermeldung zurückgegeben. Erfolgreiche Ausführungen werden durch den Rückgabewert True angezeigt, während Fehler durch False und die Fehlermeldung zurückgegeben werden.

Code-Implementierung:

def train(self, dataset, epochs, reshape\_size, batch\_size=None, lr=None):  
 try:  
 x, y = dataset self.model.fit(x, y)  
 joblib.dump(self.model, f"{self.model\_save\_path}trained\_" f"{self.model\_name}{self.model\_name\_extension}.joblib")  
 print('Trained and dumped model: ', self.model\_name)  
 self.save\_model\_metadata(self.model\_save\_path, self.model\_name, reshape\_size, list(np.unique(y)))  
 return True, None except Exception as e:  
 error = f'{self.model\_name} could not be trained. Error: {str(e)}' return False, error

make\_prediction(self, image)

Die Methode make\_prediction dient der Vorhersage einer Klasse für ein gegebenes Eingabebild. Dabei wird das trainierte Modell verwendet, um eine Vorhersage zu treffen und gegebenenfalls Wahrscheinlichkeiten für jede mögliche Klasse zu berechnen. Der Ablauf dieser Methode kann in die folgenden Schritte unterteilt werden:

1. **Überprüfung des Modellstatus:**

def make\_prediction(self, image):  
 if self.check() is False:  
 raise ValueError('Model is not trained')

Zu Beginn wird geprüft, ob das Modell bereits erfolgreich geladen und trainiert wurde. Hierzu wird die Methode self.check() aufgerufen, die prüft, ob ein gespeichertes Modell vorhanden ist. Sollte das Modell nicht verfügbar oder nicht trainiert sein, wird eine ValueError-Ausnahme mit der Nachricht 'Model is not trained' ausgelöst, um den Fehler an den Anwender zu kommunizieren.

2. **Laden der Modellmetadaten:**

reshape\_size, class\_labels = self.load\_model\_metadata(self.model\_save\_path, self.model\_name)

Nachdem das Modell als trainiert bestätigt wurde, lädt die Methode die Metadaten des Modells. Diese Metadaten beinhalten:

* **reshape\_size:** Die Größe, in die das Eingabebild umgeformt werden muss, bevor es in das Modell eingespeist wird. Dies stellt sicher, dass das Bild die gleiche Dimension hat wie die Trainingsdaten, auf denen das Modell trainiert wurde.
* **class\_labels:** Eine Liste der Klassenbezeichner (z.B. ["cat", "dog"]), die während des Trainings des Modells festgelegt wurden. Diese Bezeichner werden später dazu verwendet, die Vorhersagen des Modells den tatsächlichen Klassen zuzuordnen.

3. **Vorverarbeitung des Eingabebildes:**

image\_array = preprocess\_image(image, reshape\_size)

Das Eingabebild wird dann durch die Funktion preprocess\_image vorverarbeitet. Das Ergebnis dieser Vorverarbeitung ist das **image\_array**, das die umgeformten Bilddaten enthält und für das Modell bereit ist.

4. **Vorhersage mit Wahrscheinlichkeiten (falls verfügbar):**

if hasattr(self.model, "predict\_proba"):  
 probabilities = self.model.predict\_proba(image\_array)[0]

Es wird überprüft, ob das Modell die Methode predict\_proba unterstützt. Diese Methode liefert für jede Klasse eine Wahrscheinlichkeit, die angibt, wie sicher das Modell ist, dass das Eingabebild zu dieser Klasse gehört.

* Wenn **predict\_proba** verfügbar ist, wird diese Methode aufgerufen, um die Wahrscheinlichkeiten für jede mögliche Klasse zu berechnen. Das Ergebnis ist ein Array von Wahrscheinlichkeiten, wobei jede Wahrscheinlichkeit einer Klasse zugeordnet ist.

5. **Zuweisung von Klassenbezeichnern (falls nicht vorhanden):**

if class\_labels is None:  
 class\_labels = [f"class\_{i}" for i in range(len(probabilities))]

Falls keine **Klassenbezeichner** aus den Modellmetadaten geladen werden konnten (z.B. wenn das Modell diese Informationen nicht speichert), wird ein Standard-Label für jede Klasse generiert. Diese Labels werden als class\_0, class\_1, ..., class\_n erstellt, wobei n die Anzahl der Klassen ist.

6. **Erstellen des Vorhersage-Dictionaries:**

prediction\_dict = {label: float(prob) for label, prob in zip(class\_labels, probabilities)}

return prediction\_dict

Ein **Dictionary** wird erstellt, in dem die **Klassenbezeichner** mit den jeweiligen **Wahrscheinlichkeiten** aus den Modellvorhersagen verknüpft werden und zurückgegeben. Jede Klasse wird mit der entsprechenden Wahrscheinlichkeit als Fließkommazahl (float) versehen.

Beispiel:

{

"dog": 0.3,

"cat": 0.7

}

**7. Vorhersage ohne Wahrscheinlichkeiten (falls predict\_proba nicht verfügbar):**

else:

prediction = self.model.predict(image\_array)

return {"predicted\_label": prediction}

Falls das Modell die Methode predict\_proba nicht unterstützt (z.B. bei bestimmten Modellen, die keine Wahrscheinlichkeiten berechnen), wird stattdessen die Methode predict verwendet, um eine Einzelvorhersage für die Klasse des Bildes zu berechnen.

In diesem Fall gibt die Methode ein Dictionary mit der vorhergesagten Klasse zurück:

{

"predicted\_label": "class\_1"

}

get\_classification\_report(self, dataset, reshape\_size)

Die Methode führt eine Evaluierung des Modells durch, indem sie den **Klassifikationsbericht** (classification report) auf einem Testdatensatz berechnet. Sie teilt sich in mehrere Schritte:

1. **Aufteilen des Datensatzes:**

def get\_classification\_report(self, dataset, reshape\_size):  
 try:  
 x, y = dataset

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

Zunächst wird das übergebene dataset in **Eingabedaten (x)** und **Zielwerte (y)** unterteilt. Anschließend wird der Datensatz in Trainings- und Testdaten aufgeteilt. Dabei wird 20% des Datensatzes als Testdatensatz (x\_test, y\_test) verwendet, und 80% des Datensatzes gehen in das Training (x\_train, y\_train). train\_test\_split ist eine Funktion aus der Bibliothek **sklearn.model\_selection**, die dabei hilft, den Datensatz zufällig zu teilen. Der **random\_state=42** sorgt für Reproduzierbarkeit der Teilung.

2. **Training des Modells:**

self.model.fit(x\_train, y\_train)

Das Modell wird mit den Trainingsdaten (x\_train und y\_train) trainiert. Die Methode fit() passt die Modellparameter so an, dass das Modell die beste Vorhersage für die Zielwerte (y\_train) basierend auf den Eingabedaten (x\_train) liefert. Dieser Schritt trainiert das Modell, sodass es in der Lage ist, die Zielwerte des Testdatensatzes vorherzusagen.

3. **Vorhersage auf dem Testdatensatz:**

y\_pred = self.model.predict(x\_test)

Nachdem das Modell trainiert wurde, wird es verwendet, um Vorhersagen auf dem **Testdatensatz** (x\_test) zu machen. Die Methode predict() gibt die vom Modell vorhergesagten Zielwerte zurück, die in der Variablen y\_pred gespeichert werden.

4. **Berechnung des Klassifikationsberichts:**

report = classification\_report(y\_test, y\_pred, zero\_division=0)

Der Parameter zero\_division=0 sorgt dafür, dass im Fall von Divisionen durch Null (z.B. wenn es keine positiven Vorhersagen gibt) eine 0 zurückgegeben wird, anstatt einen Fehler zu werfen.

5. **Rückgabe des Klassifikationsberichts:**

return True, report

Wenn keine Fehler auftreten, wird der **Klassifikationsbericht** (report) als String zurückgegeben, zusammen mit einem Wert True, um den Erfolg der Methode anzuzeigen.

6. **Fehlerbehandlung:**

except Exception as e:

error = f"Error during model evaluation: {str(e)}"

return False, error

Sollte während des Prozesses ein Fehler auftreten (z.B. beim Training des Modells oder der Berechnung des Klassifikationsberichts), wird der Fehler abgefangen und eine Fehlermeldung zurückgegeben. Der Rückgabewert ist in diesem Fall False und die Fehlermeldung als String (error).

##### Modellinstanzen

In diesem Abschnitt wird erklärt, wie die Modellinstanzen in Verbindung mit der **Basisklasse** instanziiert werden, um spezifische Modelle aus der **Scikit-Learn**-Bibliothek zu erstellen und zu trainieren. Das Basismodell dient als Grundlage für die Instanziierung verschiedener Algorithmen, wobei jedes Modell mit den jeweiligen Parametern an das Basismodell übergeben wird. Da jedes „Modell“ einen eigenen Algorithmus implementiert und diese unterschiedliche Parameter erwarten, ist das eigentliche Erstellen der Modelle in unten angeführte Klassen ausgelagert, welche das erstellte Modell an die Basisklasse weitergeben.

###### AdaBoostClassifier

**AdaBoost** (Adaptive Boosting) ist ein Ensemble-Lernverfahren, das darauf abzielt, die Leistung von schwachen Klassifikatoren zu verbessern, indem mehrere schwache Modelle kombiniert werden, um ein starkes Modell zu bilden. Dabei werden aufeinanderfolgende Modelle trainiert, wobei jedes neue Modell sich auf die Fehler des vorherigen Modells konzentriert. Das bedeutet, dass falsch klassifizierte Datenpunkte aus vorherigen Iterationen mehr Gewicht erhalten, sodass das Modell sich darauf konzentrieren kann, schwierige Beispiele besser zu klassifizieren.

Theorie

Im folgenden Abschnitt wird der **AdaBoost-Classifier** erläutert. Es wird erklärt, wie dieser Ensemble-Algorithmus funktioniert, welche Konzepte dahinterstehen und welche Vorteile er bietet.

Anmerkung: Der Begriff **Ensemble** im Kontext des maschinellen Lernens bezieht sich auf eine Methode, bei der mehrere Modelle (auch **Klassifikatoren** oder **Lernalgorithmen** genannt) kombiniert werden, um ein stärkeres und stabileres Modell zu bilden. Die Idee dahinter ist, dass die Kombination von mehreren schwachen Modellen (die einzeln nicht sehr leistungsfähig sind) zu einem leistungsfähigeren Modell führen kann.

**Aufbau und Funktionsweise**

AdaBoost arbeitet durch eine iterative Anpassung der Gewichtung der Trainingsdaten und des Modells. In jeder Runde wird ein neuer Klassifikator trainiert, der die Fehler des vorherigen Klassifikators korrigiert. Die Vorhersage wird dann als gewichtete Summe der Vorhersagen der schwachen Klassifikatoren getroffen. AdaBoost arbeitet besonders gut mit einfachen Klassifikatoren, die als schwach gelten, z.B. Entscheidungsbäume mit geringer Tiefe.

**Schwacher Klassifikator:** In der Regel wird bei AdaBoost ein **Entscheidungsbaum** verwendet, jedoch können auch andere Klassifikatoren eingesetzt werden. AdaBoost ist sehr empfindlich gegenüber übermäßigen Fehlern in den schwachen Klassifikatoren, wodurch es zu einer stabilen und dennoch leistungsfähigen Methode wird.

Anwendungen

 **Textklassifikation:** AdaBoost wird häufig in Anwendungen wie der Sentimentanalyse und Spamfilterung eingesetzt.

 **Gesichtserkennung:** AdaBoost wird auch zur Objekterkennung, insbesondere in der Gesichtserkennung, verwendet.

 **Finanzmarktprognosen:** Das Modell wird aufgrund seiner Flexibilität auch in der Finanzwelt eingesetzt, um Marktentwicklungen vorherzusagen.

Verlässlichkeit und Performance

AdaBoost ist für seine Fähigkeit bekannt, Modelle aus schwachen Lernalgorithmen zu erstellen, die eine hohe Genauigkeit erreichen können. Allerdings ist es anfällig für Ausreißer und kann übermäßig empfindlich auf Rauschen in den Daten reagieren, was zu einer schlechten Performance bei unvorhergesehenen Daten führen kann. Der Schlüssel zu einer erfolgreichen Anwendung von AdaBoost liegt in der richtigen Wahl des schwachen Klassifikators und einer guten Handhabung von Rauschen und Ausreißern in den Trainingsdaten.

Imports

Die notwendigen Imports für dieses Modell sind:

from sklearn.ensemble import AdaBoostClassifier

from .training\_basemodel import BaseModel

* **AdaBoostClassifier**: Wird aus dem **sklearn.ensemble**-Modul importiert, um den AdaBoost-Klassifikator zu verwenden.
* **BaseModel**: Das für Scikit-Learn definierte Basismodell

training\_ab\_sklearn.py

In diesem Fall wird der **AdaBoostClassifier** mit der Anzahl von Basislernmodellen, die im Parameter n\_estimators=50 festgelegt ist, instanziiert. Dies bedeutet, dass AdaBoost insgesamt 50 schwache Klassifikatoren verwendet, um die endgültige Entscheidung zu treffen. Dieser Parameter ist natürlich kein Richtwert und grundsätzlich wird die Modellleistung besser, je mehr Basislernmodelle eingesetzt werden.

class AdaBoost(BaseModel):

def \_\_init\_\_(self, model\_save\_path):

# AdaBoostClassifier wird mit einer spezifischen Anzahl an Entscheidungsbäumen (n\_estimators) instanziiert

super().\_\_init\_\_(AdaBoostClassifier(n\_estimators=50), model\_save\_path)

* **AdaBoostClassifier(n\_estimators=50)**: Instanziiert einen AdaBoost-Klassifikator, bei dem n\_estimators=50 die Anzahl der Basislerner angibt. Dies bedeutet, dass 50 Iterationen durchgeführt werden, wobei jeder Iterationsklassifikator auf den Fehler des vorherigen Modells fokussiert.
* **super().\_\_init\_\_(AdaBoostClassifier(...), model\_save\_path)**: Der Aufruf an die super()-Methode ruft den Konstruktor der Basisklasse **BaseModel** auf, wodurch der AdaBoostClassifier zusammen mit dem Pfad zum Speichern des Modells übergeben wird. Die Basisklasse **BaseModel** übernimmt dann die Verantwortung für die Modellbereitstellung, das Training und das Speichern des Modells.

###### DecisionTreeClassifier

Der DecisionTreeClassifier ist ein überwacht lernender Algorithmus, der für Klassifikationsaufgaben verwendet wird. Er basiert auf einer baumartigen Struktur, bei der Daten in rekursiven Schritten anhand von Entscheidungsregeln aufgeteilt (Binärbaum) werden. Dies ermöglicht eine intuitive Darstellung und Interpretation der Entscheidungsfindung.

Theorie

Ein Entscheidungsbaum ist ein hierarchisches Modell, das Entscheidungen basierend auf Merkmalen der Eingabedaten trifft. Der Baum besteht aus Knoten, die Entscheidungen basierend auf bestimmten Merkmalen treffen, und Blättern, die die finale Klassenzuordnung darstellen.

Vorteile des DecisionTreeClassifier:

* Verständlichkeit und Interpretierbarkeit
* Geringe Vorverarbeitung der Daten erforderlich
* Fähigkeit, nicht-lineare Zusammenhänge zu modellieren

Nachteile:

* Neigung zum Overfitting bei tiefen Bäumen
* Sensitivität gegenüber kleinen Änderungen in den Daten

**Aufbau und Funktionsweise**

Der DecisionTreeClassifier teilt die Trainingsdaten anhand einer bestimmten Metrik (z. B. Gini-Index oder Entropie) in Teilmengen auf. Dies geschieht rekursiv, bis eine Stoppregel erfüllt ist, wie z. B. eine maximale Tiefe des Baums oder eine minimale Anzahl von Proben pro Blatt.

Wichtige Hyperparameter:

* criterion: Gibt an, welche Metrik zur Berechnung der besten Trennung verwendet wird (z. B. "gini" oder "entropy").
* max\_depth: Die maximale Tiefe des Baums, um Overfitting zu vermeiden.
* min\_samples\_split: Die Mindestanzahl an Samples, die erforderlich sind, um einen Knoten weiter aufzuteilen.

Anwendungen

 **Kundensegmentierung:** Entscheidungsbäume werden in Marketinganalysen eingesetzt, um Kunden in verschiedene Gruppen zu unterteilen.

 **Medizinische Diagnostik:** Sie helfen bei der Klassifikation von Krankheiten basierend auf Symptomen.

 **Betrugserkennung:** In der Finanzbranche werden sie zur Erkennung betrügerischer Transaktionen genutzt.

Verlässlichkeit und Performance

Entscheidungsbäume sind leistungsfähig und schnell, aber sie neigen dazu, sich zu stark an das Training anzupassen (Overfitting). Daher werden sie oft mit Ensemble-Methoden wie RandomForestClassifier oder AdaBoostClassifier kombiniert, um die Generalisierungsfähigkeit zu verbessern.

Imports

Die notwendigen Imports für das Modell sind:

from sklearn.tree import DecisionTreeClassifier

from .training\_basemodel import BaseModel

* **DecisionTreeClassifier:** Wird aus dem **sklearn.tree**-Modul importiert, um einen Decision Tree zu implementieren
* **BaseModel:** Die für Scikit-Learn definierte Basisklasse

training\_dtc\_sklearn.py

Der DecisionTreeClassifier wird mit Standardparametern instanziiert. Die Modellleistung kann durch Anpassung von Hyperparametern wie max\_depth oder min\_samples\_split optimiert werden.

class DecisionTree(BaseModel):

def \_\_init\_\_(self, model\_save\_path):

super().\_\_init\_\_(DecisionTreeClassifier(), model\_save\_path)

* DecisionTreeClassifier(): Erstellt einen Entscheidungsbaum mit Standardparametern.
* super().\_\_init\_\_(...): Ruft den Konstruktor der BaseModel-Klasse auf, um das Modell zu initialisieren und zu speichern.

###### GaussianProcessClassifier

Der GaussianProcessClassifier (GPC) ist ein probabilistischer Klassifikationsalgorithmus, der auf Gaußschen Prozessen basiert. Er modelliert die Wahrscheinlichkeitsverteilung der Klassenzugehörigkeit als einen Gaußschen Prozess und ermöglicht eine flexible, nichtlineare Trennung der Daten.

Theorie

GPC gehört zur Familie der Kernel-Methoden und verwendet eine Bayessche Herangehensweise zur Klassifikation. Im Gegensatz zu deterministischen Methoden liefert GPC Wahrscheinlichkeiten für Klassenzuordnungen, was ihn besonders nützlich für Unsicherheitsabschätzungen macht.

Vorteile des GaussianProcessClassifier:

* Liefert Wahrscheinlichkeitswerte für Vorhersagen
* Kann hochgradig nichtlineare Entscheidungsgrenzen modellieren
* Flexibel durch anpassbare Kernel-Funktionen

Nachteile:

* Hoher Rechenaufwand, insbesondere bei großen Datensätzen
* Sensibel gegenüber Wahl des Kernels und Hyperparameter-Tuning

**Aufbau und Funktionsweise**

GPC nutzt einen Kernel-Trick, um Daten in einen hochdimensionalen Raum zu projizieren, in dem sie besser trennbar sind. Der Algorithmus basiert auf einer Wahrscheinlichkeitsverteilung für jede mögliche Klassenzugehörigkeit eines Punktes und maximiert die posterior-Verteilung.

Wichtige Hyperparameter:

* kernel: Definiert die Art der nichtlinearen Transformation (z. B. RBF, Matern).
* optimizer: Bestimmt die Methode zur Optimierung der Hyperparameter.
* max\_iter\_predict: Die maximale Anzahl an Iterationen für die Vorhersage.

Anwendungen

 **Bildverarbeitung:** GPC wird für Bildklassifikation und Handschriftenerkennung eingesetzt.

 **Bioinformatik:** Verwendung in der Genexpressionsanalyse zur Klassifikation von Zelltypen.

 **Finanzwesen:** Modellierung von Markttrends und Vorhersage von Risiken.

Verlässlichkeit und Performance

GPC ist ein leistungsfähiges Modell für kleine bis mittlere Datensätze, bei denen Unsicherheiten berücksichtigt werden müssen. Allerdings skaliert er schlecht mit großen Datensätzen, da die Berechnung der Kovarianzmatrix eine quadratische bis kubische Komplexität aufweist.

Imports

Die notwendigen Imports für dieses Modell sind:

from sklearn.gaussian\_process import GaussianProcessClassifier

from .training\_basemodel import BaseModel

* **GaussianProcessClassifier**: Wird aus dem sklearn.gaussian\_process-Modul importiert, um einen Gaußschen Prozess zur Klassifikation zu implementieren.
* **BaseModel**: Die für Scikit-Learn definierte Basisklasse.

training\_gpc\_sklearn.py

Der GaussianProcessClassifier wird mit Standardparametern instanziiert. Die Leistung kann durch Auswahl eines geeigneten Kernels optimiert werden.

class GaussianProcess(BaseModel):

def \_\_init\_\_(self, model\_save\_path):

super().\_\_init\_\_(GaussianProcessClassifier(), model\_save\_path)

* **GaussianProcessClassifier()**: Erstellt ein Modell mit Standardparametern.
* **super().init(...)**: Ruft den Konstruktor der BaseModel-Klasse auf, um das Modell zu initialisieren und zu speichern.

###### KNeighborsClassifier

Der KNeighborsClassifier (KNN) ist ein nichtparametrischer Klassifikationsalgorithmus, der auf der Nähe zu den Trainingsdaten basiert. Er klassifiziert neue Datenpunkte, indem er die Mehrheitsklasse der k nächsten Nachbarn bestimmt.

Theorie

KNN gehört zu den instanzbasierten Lernverfahren, da er kein explizites Modell trainiert, sondern die gesamten Trainingsdaten speichert und zur Klassifikation heranzieht.

**Vorteile des KNeighborsClassifier:**

* Einfach zu verstehen und zu implementieren
* Keine Annahmen über die Datenverteilung erforderlich
* Kann beliebig komplexe Entscheidungsgrenzen modellieren

**Nachteile:**

* Hoher Speicherbedarf, da alle Trainingsdaten gespeichert werden
* Langsame Vorhersagen bei großen Datensätzen
* Empfindlich gegenüber irrelevanten oder skalierungsabhängigen Merkmalen

**Aufbau und Funktionsweise**

KNN berechnet die Distanz eines neuen Datenpunkts zu allen Trainingspunkten und wählt die k nächsten Nachbarn basierend auf einer Distanzmetrik (z. B. euklidische Distanz). Die Klasse mit der höchsten Häufigkeit unter den k Nachbarn wird als Vorhersage verwendet.

**Wichtige Hyperparameter:**

* n\_neighbors: Anzahl der nächsten Nachbarn für die Klassifikation.
* metric: Die verwendete Distanzmetrik (z. B. euclidean, manhattan).
* weights: Gibt an, ob alle Nachbarn gleich gewichtet werden oder eine Distanzgewichtung erfolgt (uniform oder distance).

Anwendungen

 **Mustererkennung:** KNN wird in der Handschriftenerkennung und Bilderkennung eingesetzt.

 **Medizinische Diagnostik:** Klassifikation von Krankheiten basierend auf Patientendaten.

 **Empfehlungssysteme:** Ähnlichkeitsbasierte Empfehlungen für Nutzerpräferenzen.

Verlässlichkeit und Performance

KNN kann sehr genaue Ergebnisse liefern, wenn die richtige Anzahl an Nachbarn (k) gewählt wird. Allerdings skaliert er schlecht mit großen Datensätzen, da die Berechnung der Distanzen zu allen Punkten sehr aufwendig ist. Eine effiziente Implementierung mit KD-Trees oder Ball-Trees kann die Performance verbessern.

Imports

Die notwendigen Imports für dieses Modell sind:

from sklearn.neighbors import KNeighborsClassifier

from .training\_basemodel import BaseModel

* **KNeighborsClassifier**: Wird aus dem sklearn.neighbors-Modul importiert, um den k-Nearest-Neighbors-Algorithmus zu implementieren.
* **BaseModel**: Die für Scikit-Learn definierte Basisklasse.

training\_dtc\_sklearn.py

Der KNeighborsClassifier wird mit n\_neighbors=1 instanziiert, sodass die Klassifikation anhand des nächsten Nachbarn erfolgt.

class KNN(BaseModel):

def \_\_init\_\_(self, model\_save\_path):

super().\_\_init\_\_(KNeighborsClassifier(n\_neighbors=1), model\_save\_path)

* **KNeighborsClassifier(n\_neighbors=1)**: Erstellt ein Modell, das nur den nächsten Nachbarn zur Klassifikation verwendet.
* **super().init(...)**: Ruft den Konstruktor der BaseModel-Klasse auf, um das Modell zu initialisieren und zu speichern.

###### MLPClassifier

Der MLPClassifier (Multi-Layer Perceptron) ist ein neuronales Netzwerk, das für Klassifikationsaufgaben verwendet wird. Es basiert auf mehreren Schichten von künstlichen Neuronen, die durch Aktivierungsfunktionen transformiert werden.

Theorie

Der MLPClassifier gehört zur Familie der überwachten neuronalen Netze und nutzt Backpropagation zur Gewichtsaktualisierung während des Trainings. Er besteht aus einer Eingabeschicht, einer oder mehreren versteckten Schichten und einer Ausgabeschicht.

**Vorteile des MLPClassifier:**

* Kann komplexe nichtlineare Zusammenhänge modellieren
* Skalierbar durch die Anzahl der Neuronen und Schichten
* Unterstützt verschiedene Optimierungsverfahren wie adam und sgd

**Nachteile:**

* Hohe Rechenanforderungen, insbesondere bei großen Netzen
* Erfordert sorgfältiges Hyperparameter-Tuning
* Kann zu Overfitting neigen, wenn das Netzwerk zu groß ist

**Aufbau und Funktionsweise**

Der MLPClassifier besteht aus mehreren Schichten von Neuronen, die durch Aktivierungsfunktionen (z. B. ReLU, Sigmoid) transformiert werden. Die Gewichte des Netzwerks werden durch einen Optimierungsalgorithmus aktualisiert, um den Fehler zu minimieren.

**Wichtige Hyperparameter:**

* hidden\_layer\_sizes: Die Anzahl der Neuronen in den versteckten Schichten.
* activation: Die Aktivierungsfunktion (z. B. relu, tanh).
* solver: Der Optimierungsalgorithmus (adam, sgd, lbfgs).
* max\_iter: Die maximale Anzahl der Iterationen für das Training.

Anwendungen

 **Bildverarbeitung:** MLP wird in der Handschriftenerkennung und Objekterkennung eingesetzt.

 **Sprachverarbeitung:** Anwendung in der Spracherkennung und Übersetzungsmodellen.

 **Finanzanalyse:** Vorhersagemodelle für Marktbewegungen und Risikomanagement.

Verlässlichkeit und Performance

MLP kann sehr leistungsfähig sein, erfordert jedoch eine sorgfältige Abstimmung der Hyperparameter und eine ausreichende Menge an Trainingsdaten, um Overfitting zu vermeiden. Die Wahl der Aktivierungsfunktion und des Optimierungsverfahrens beeinflusst die Konvergenzgeschwindigkeit und die Vorhersagegenauigkeit.

Imports

Die notwendigen Imports für dieses Modell sind:

from sklearn.neural\_network import MLPClassifier

from .training\_basemodel import BaseModel

* **MLPClassifier**: Wird aus dem sklearn.neural\_network-Modul importiert, um ein künstliches neuronales Netzwerk zu implementieren.
* **BaseModel**: Die für Scikit-Learn definierte Basisklasse.

training\_mlpc\_sklearn.py

Der MLPClassifier wird mit einer einzelnen versteckten Schicht von 128 Neuronen und einer maximalen Anzahl von 1000 Trainingsiterationen instanziiert.

class NeuralNet(BaseModel):

def \_\_init\_\_(self, model\_save\_path):

super().\_\_init\_\_(MLPClassifier(hidden\_layer\_sizes=(128,), max\_iter=1000), model\_save\_path)

* **MLPClassifier(hidden\_layer\_sizes=(128,), max\_iter=1000)**: Erstellt ein neuronales Netzwerk mit einer versteckten Schicht von 128 Neuronen und maximal 1000 Iterationen.
* **super().init(...)**: Ruft den Konstruktor der BaseModel-Klasse auf, um das Modell zu initialisieren und zu speichern.

###### GaussianNB

Der GaussianNB ist eine Implementierung des Naive-Bayes-Klassifikators für kontinuierliche Daten, die einer Gaußschen (normalen) Verteilung folgen. Er basiert auf der Bayesschen Wahrscheinlichkeitstheorie und nimmt an, dass die Merkmale unabhängig voneinander sind.

Theorie

Naive Bayes ist ein probabilistisches Modell, das die bedingte Wahrscheinlichkeit jeder Klasse anhand der Wahrscheinlichkeiten der einzelnen Merkmale berechnet. Trotz der starken Annahme der Unabhängigkeit der Merkmale zeigt der Algorithmus in vielen praktischen Anwendungen eine robuste Leistung.

**Vorteile des GaussianNB:**

* Sehr schnell in Training und Inferenz
* Funktioniert gut mit hochdimensionalen Daten
* Gut interpretierbar und einfach zu implementieren

**Nachteile:**

* Annahme der Unabhängigkeit der Merkmale ist oft nicht realistisch
* Funktioniert schlecht, wenn die Merkmale nicht normalverteilt sind
* Sensitiv gegenüber irrelevanten oder korrelierten Merkmalen

**Aufbau und Funktionsweise**

Der GaussianNB-Klassifikator berechnet für jede Klasse die Wahrscheinlichkeitsverteilung der Merkmale unter der Annahme einer Normalverteilung. Mithilfe dieser Wahrscheinlichkeiten wird anschließend bestimmt, zu welcher Klasse ein neues Beispiel mit der höchsten Wahrscheinlichkeit gehört.

Da der Algorithmus lediglich Mittelwert und Standardabweichung jedes Merkmals für jede Klasse speichert, ist das Modell besonders speichereffizient und schnell in der Inferenz.

Anwendungen

 **Textklassifikation:** Häufig eingesetzt für Spam-Filter und Sentiment-Analyse

 **Medizinische Diagnostik:** Identifikation von Krankheiten anhand von Patientendaten

 **Bildverarbeitung:** Klassifikation von einfachen Mustern oder Objekten

Verlässlichkeit und Performance

GaussianNB ist besonders leistungsfähig, wenn die Annahmen über die Datenverteilung zutreffen. Er kann mit wenigen Trainingsdaten gut generalisieren und ist weniger anfällig für Overfitting als komplexe Modelle. Allerdings kann die Klassifikationsleistung leiden, wenn die Merkmale stark voneinander abhängen oder nicht normalverteilt sind.

Imports

Die notwendigen Imports für dieses Modell sind:

from sklearn.naive\_bayes import GaussianNB

from .training\_basemodel import BaseModel

* **GaussianNB**: Wird aus dem sklearn.naive\_bayes-Modul importiert, um das Naive-Bayes-Klassifikationsmodell mit einer Gaußschen Verteilung zu implementieren.
* **BaseModel**: Die für Scikit-Learn definierte Basisklasse.

training\_nb\_sklearn.py

Der GaussianNB-Klassifikator wird ohne zusätzliche Parameter instanziiert, da er direkt mit den Standardwerten verwendet werden kann.

class NaiveBayes(BaseModel):

def \_\_init\_\_(self, model\_save\_path):

super().\_\_init\_\_(GaussianNB(), model\_save\_path)

* **GaussianNB()**: Erstellt ein Standard-Naive-Bayes-Modell mit einer Gaußschen Wahrscheinlichkeitsverteilung.
* **super().init(...)**: Ruft den Konstruktor der BaseModel-Klasse auf, um das Modell zu initialisieren und zu speichern.

###### QuadraticDiscriminantAnalysis

Die Quadratic Discriminant Analysis (QDA) ist ein Klassifikationsverfahren, das verwendet wird, um Daten in verschiedene Gruppen oder Klassen zu unterteilen. Es basiert auf der Annahme, dass die Merkmale der verschiedenen Klassen einer Normalverteilung (Gaußschen Verteilung) folgen. QDA geht jedoch einen Schritt weiter als ähnliche Verfahren wie die **Lineare Diskriminanzanalyse (LDA)**, da es nicht annimmt, dass alle Klassen die gleiche Kovarianzmatrix haben. Das bedeutet, dass QDA unterschiedliche Formen der Streuung (Kovarianz) für jede Klasse zulässt und damit flexibler wird.

Theorie

Im Gegensatz zur **Linearen Diskriminanzanalyse (LDA)**, bei der eine lineare Entscheidungsgrenze zwischen den Klassen gesucht wird, ermöglicht QDA eine **quadratische Entscheidungsgrenze**. Dies bedeutet, dass QDA in der Lage ist, Daten zu klassifizieren, die sich nicht durch eine gerade Linie trennen lassen – was bei LDA nicht möglich wäre.

**Warum quadratisch?** QDA berücksichtigt für jede Klasse eine eigene **Kovarianzmatrix** (also eine Matrix, die die Variabilität der Merkmale innerhalb einer Klasse beschreibt). Diese unterschiedliche Streuung der Merkmale innerhalb der Klassen führt zu einer **quadratischen** Entscheidungsgrenze, die für komplexere Datenstrukturen besser geeignet ist.

**Vorteile von QDA:**

* **Flexibilität:** Da QDA unterschiedliche Kovarianzmatrizen für jede Klasse verwendet, kann es auch nicht-lineare Trennungen modellieren und bietet daher mehr Flexibilität als LDA.
* **Bessere Modellierung:** Besonders bei Klassen mit unterschiedlichen Streuungen der Merkmale (z. B. Klassen mit stark unterschiedlichen Formen) kann QDA bessere Ergebnisse liefern.

**Nachteile von QDA:**

* **Überanpassung:** QDA neigt dazu, bei kleinen Datensätzen oder bei Datensätzen mit vielen Merkmalen zu **überanpassen** (Overfitting). Das bedeutet, dass das Modell die Trainingsdaten zu genau abbildet und dann auf neuen, unbekannten Daten schlechter funktioniert.
* **Annahme der Normalverteilung:** QDA setzt voraus, dass die Daten in jeder Klasse normalverteilt sind, was nicht immer zutrifft.

**Aufbau und Funktionsweise**

QDA berechnet für jede Klasse die Wahrscheinlichkeitsdichte der Merkmale basierend auf einer **Normalverteilung**, aber mit einer eigenen **Kovarianzmatrix** für jede Klasse. Anschließend wird das neue Beispiel der Klasse mit der höchsten **posterioren Wahrscheinlichkeit** zugeordnet. Die Entscheidungsgrenze zwischen den Klassen ist **quadratisch**, da die Kovarianzmatrizen für jede Klasse unterschiedlich sind. Dies ermöglicht es dem Modell, komplexe, nicht-lineare Beziehungen zwischen den Klassen zu erfassen.

**Funktionsweise im Detail:**

1. **Berechnung der Wahrscheinlichkeiten:** Für jede Klasse wird die Wahrscheinlichkeit berechnet, dass ein Datenpunkt zu dieser Klasse gehört, basierend auf der Normalverteilung mit der entsprechenden Mittelwert- und Kovarianzmatrix.
2. **Zuordnung der Klasse:** Der Datenpunkt wird der Klasse zugeordnet, bei der die Wahrscheinlichkeit am höchsten ist.
3. **Quadratische Entscheidungsgrenze:** Da jede Klasse eine eigene Kovarianzmatrix hat, entsteht eine quadratische Entscheidungsgrenze zwischen den Klassen.

Anwendungen

QDA wird in vielen Bereichen eingesetzt, in denen komplexe, nicht-lineare Trennungen zwischen den Klassen erforderlich sind. Typische Anwendungen sind:

* **Finanzmodellierung:** Identifikation von Kreditrisiken oder Erkennung von betrügerischen Aktivitäten.
* **Bioinformatik:** Klassifikation von Genexpressionsdaten zur Erkennung von Krankheiten oder biologischen Mustern.
* **Bildverarbeitung:** Objekterkennung und Mustererkennung in Bildern, insbesondere wenn die Klassen komplexe, nicht-lineare Grenzen haben.

Verlässlichkeit und Performance

QDA bietet eine hohe Flexibilität bei der Modellierung von komplexen, nicht-linearen Trennungen zwischen den Klassen. Die Leistung hängt jedoch stark davon ab, wie gut die Annahmen über die Normalverteilung der Merkmale zutreffen. Wenn die Merkmale tatsächlich normalverteilt sind und jede Klasse eine eigene Kovarianzmatrix benötigt, kann QDA sehr gute Ergebnisse liefern. Bei kleinen Datensätzen oder falschen Annahmen über die Verteilung kann das Modell jedoch **überanpassen** und zu schlechteren Ergebnissen führen.

Imports

Die notwendigen Imports für dieses Modell sind:

from sklearn.discriminant\_analysis import QuadraticDiscriminantAnalysis

from .training\_basemodel import BaseModel

* **QuadraticDiscriminantAnalysis:** Wird aus dem sklearn.discriminant\_analysis-Modul importiert, um das QDA-Modell zu implementieren.
* **BaseModel:** Die für Scikit.Learn definierte Basisklasse

training\_qda\_sklearn.py

Der QDA-Klassifikator wird hier mit einem **Regularisierungsparameter** instanziiert, um das Modell zu stabilisieren und zu verhindern, dass es bei kleinen Datensätzen überangepasst wird. Der **Regularisierungsparameter** (reg\_param=0.1) sorgt dafür, dass das Modell weniger empfindlich auf schwankende Daten reagiert.

class QDA(BaseModel):

def \_\_init\_\_(self, model\_save\_path):

super().\_\_init\_\_(QuadraticDiscriminantAnalysis(reg\_param=0.1), model\_save\_path)

* **QuadraticDiscriminantAnalysis(reg\_param=0.1):** Erstellt das QDA-Modell mit einem Regularisierungsparameter von 0.1. Dieser Parameter sorgt für eine numerische Stabilität und hilft, das Modell zu generalisieren.
* **super().init(...):** Ruft den Konstruktor der BaseModel-Klasse auf, um das QDA-Modell zu initialisieren und für das Speichern zu konfigurieren.

###### SVC

Die Support Vector Machine (SVM) ist ein überwacht lernendes Klassifikationsmodell, das eine **optimale Entscheidungsgrenze** zwischen den Klassen sucht. SVMs maximieren den Abstand (Margin) zwischen den verschiedenen Klassen und verwenden Support Vektoren, die die Klassengrenze definieren.

Theorie

SVMs sind besonders nützlich, wenn die Klassen **nicht-linear trennbar** sind, da sie verschiedene Kernel-Funktionen unterstützen. Der klassische Fall, der als **lineare SVM** bezeichnet wird, sucht eine lineare Trennlinie zwischen den Klassen, während die **nicht-lineare SVM** Kernel verwendet, um die Daten in höherdimensionale Räume zu projizieren, um dort eine lineare Trennung zu ermöglichen.

**Vorteile der SVM:**

* Sehr **effizient** bei der Handhabung von hochdimensionalen Daten.
* Robust gegenüber **Overfitting**, besonders in höheren Dimensionen.
* Kann mit verschiedenen **Kernelfunktionen** flexibel an nicht-lineare Trennungen angepasst werden.

**Nachteile der SVM:**

* SVMs können bei sehr **großen Datensätzen** rechenintensiv sein.
* Die Wahl des **Kernels** und der **Hyperparameter** (z.B. Regularisierungsparameter) kann einen erheblichen Einfluss auf die Leistung haben und erfordert oft **Parameterabstimmung**.

**Aufbau und Funktionsweise**

SVM sucht nach der optimalen Entscheidungsgrenze (Hyperplane), die die verschiedenen Klassen trennt. Der Algorithmus maximiert den Abstand (Margin) zwischen den Datenpunkten der verschiedenen Klassen. Der **Kernel** wird verwendet, um die Eingabedaten in einen höherdimensionalen Raum zu projizieren, in dem eine lineare Trennung der Daten möglich wird.

**Funktionsweise im Detail:**

1. **Berechnung des optimalen Hyperplanes:** Die SVM berechnet den besten Hyperplane, der die Datenpunkte der verschiedenen Klassen mit dem größten Abstand (Margin) trennt.
2. **Verwendung von Support Vektoren:** Die Support Vektoren sind die Datenpunkte, die der Trennlinie am nächsten liegen und die Entscheidung beeinflussen.
3. **Kernels:** Für nicht-lineare Trennungen kann die SVM verschiedene Kernel verwenden (z.B. polynomiale oder RBF-Kernel).

Anwendungen

SVMs sind in vielen Bereichen von Bedeutung, insbesondere in Szenarien, bei denen eine klare Trennung zwischen den Klassen erforderlich ist:

* **Textklassifikation:** Häufig verwendet in der **Spam-Filterung** oder **Sentiment-Analyse**.
* **Bildverarbeitung:** SVMs sind sehr effektiv für die **Gesichtserkennung** und **Mustererkennung**.
* **Medizinische Diagnostik:** Klassifikation von **Krankheiten** basierend auf medizinischen Daten.

Verlässlichkeit und Performance

SVMs bieten eine **robuste Leistung**, wenn es um Klassifikationen mit klaren Grenzen zwischen den Klassen geht. Sie sind besonders nützlich, wenn die Daten in einem hochdimensionalen Raum gut separierbar sind. Allerdings erfordert die Wahl des richtigen **Kernels** und der **Hyperparameter** sorgfältige **Abstimmung**. SVMs sind empfindlich gegenüber **Ausreißern** und können bei unbalancierten Datensätzen suboptimale Ergebnisse liefern.

Imports

Die notwendigen Imports für dieses Modell sind:

from sklearn.svm import SVC

from .training\_basemodel import BaseModel

* **SVC:** Wird aus dem sklearn.svm-Modul importiert, um das Support Vector Machine-Modell mit einem linearen Kernel zu implementieren.
* **BaseModel:** Die für Scikit-Learn definierte Basisklasse.

training\_svc\_sklearn.py

Der SVM-Klassifikator wird hier mit einem **linearen Kernel** instanziiert, um die Daten in einem linearen Raum zu trennen. Der Parameter "\_linear" wird als Zusatz im Modellpfad gespeichert.

class SVM(BaseModel):

def \_\_init\_\_(self, model\_save\_path):

super().\_\_init\_\_(SVC(kernel='linear'), model\_save\_path, "\_linear")

* **SVC(kernel='linear'):** Erstellt das SVM-Modell mit einem **linearen Kernel**, was bedeutet, dass das Modell eine lineare Trennlinie zwischen den Klassen sucht.
* **super().init(...):** Ruft den Konstruktor der BaseModel-Klasse auf, um das SVM-Modell zu initialisieren und für das Speichern zu konfigurieren. Der Zusatz "\_linear" wird als Identifier für das lineare Modell im Speicherpfad verwendet.

RBF-SVC

Das **RBF-Support Vector Machine** Modell ist eine Variante der Support Vector Machine (SVM), die den **Radial Basis Function (RBF)-Kernel** verwendet, um eine **nicht-lineare Trennung** der Klassen zu ermöglichen. Dieser Kernel ermöglicht es der SVM, komplexe Trennungen zu modellieren, indem er die Daten in einen höherdimensionalen Raum projiziert, in dem eine lineare Trennung möglich ist.

Theorie

Der RBF-Kernel ist besonders geeignet für Datensätze, bei denen eine **nicht-lineare** Trennung der Klassen erforderlich ist. Statt eine lineare Trennlinie zu suchen, wird der RBF-Kernel verwendet, um die Daten in einen hochdimensionalen Raum zu projizieren, wo eine lineare Trennung gefunden werden kann. Dies ermöglicht es der SVM, komplexe, nicht-lineare Grenzen zwischen den Klassen zu erkennen und zu modellieren.

**Vorteile des RBF-SVM:**

* **Flexible Modellierung** von nicht-linearen Entscheidungsgrenzen.
* Sehr **leistungsfähig** bei hochdimensionalen und komplexen Datensätzen.
* Geringere Notwendigkeit für manuelle Feature-Transformationen, da der RBF-Kernel diese automatisch durch die Projektion in höhere Dimensionen übernimmt.

**Nachteile des RBF-SVM:**

* **Empfindlich** gegenüber der Wahl des Parameters gamma, der die Form der Entscheidungsgrenze beeinflusst.
* **Hoher Rechenaufwand** bei großen Datensätzen oder einer hohen Anzahl von Merkmalen.
* Die **Wahl der Hyperparameter** (insbesondere C und gamma) erfordert oft eine sorgfältige **Abstimmung**, um Überanpassung oder unzureichende Generalisierung zu vermeiden.

**Aufbau und Funktionsweise**

Das RBF-SVM-Modell nutzt den **RBF-Kernel**, der die Datenpunkte durch eine nicht-lineare Transformation in einen höherdimensionalen Raum projiziert, sodass eine lineare Trennung möglich wird. In diesem Raum sucht die SVM nach dem besten Hyperplane, der die verschiedenen Klassen mit maximalem Abstand trennt. Der RBF-Kernel führt dabei eine **ähnlichkeitsbasierte Berechnung** durch, die auf den Abständen zwischen den Datenpunkten basiert.

**Funktionsweise im Detail:**

1. **Datenprojektion:** Der RBF-Kernel projiziert die Eingabedaten in einen hochdimensionalen Raum.
2. **Berechnung des optimalen Hyperplanes:** Die SVM berechnet den besten Hyperplane, der die Datenpunkte mit dem größten Abstand trennt.
3. **Support Vektoren:** Die Support Vektoren sind die Datenpunkte, die am nächsten zur Entscheidungsgrenze liegen und die Modellentscheidung bestimmen.

Anwendungen

Das RBF-SVM-Modell wird in verschiedenen Bereichen eingesetzt, in denen eine nicht-lineare Trennung der Klassen erforderlich ist. Typische Anwendungen umfassen:

* **Bildverarbeitung:** Klassifikation von komplexen Objekten oder Mustern in Bildern, wo lineare Trennungen nicht ausreichen.
* **Bioinformatik:** Klassifikation von Genexpressionsdaten oder Erkennung von Krankheiten, bei denen die Daten nicht-lineare Muster aufweisen.
* **Sprachverarbeitung:** Erkennung von Mustern in Audiodaten oder Sentiment-Analyse in Textdaten.

Verlässlichkeit und Performance

RBF-SVM ist besonders leistungsfähig, wenn die **Daten komplexe, nicht-lineare Muster** enthalten. Durch die Projektion in einen höherdimensionalen Raum kann es eine sehr gute Trennung der Klassen erreichen. Allerdings ist es empfindlich gegenüber der Wahl der **Hyperparameter** (insbesondere gamma), und eine unsachgemäße Wahl dieser Parameter kann zu **Überanpassung** oder **schlechter Generalisierung** führen. Bei sehr großen Datensätzen kann es zudem **rechenintensiv** werden.

Imports

Die notwendigen Imports für dieses Modell sind:

from sklearn.svm import SVC

from .training\_basemodel import BaseModel

* **SVC:** Wird aus dem sklearn.svm-Modul importiert, um das Support Vector Machine-Modell mit einem RBF-Kernel zu implementieren.
* **BaseModel:** Die für Scikit-Learn definierte Basisklasse, die grundlegende Trainings- und Speichermethoden bereitstellt.

training\_rbf\_svc\_sklearn.py

Der RBF-SVM-Klassifikator wird mit dem **RBF-Kernel** instanziiert, um eine nicht-lineare Trennung zwischen den Klassen zu ermöglichen. Der Zusatz "\_rbf" wird als Identifier im Modellpfad gespeichert.

class RBF\_SVM(BaseModel):

def \_\_init\_\_(self, model\_save\_path):

super().\_\_init\_\_(SVC(kernel='rbf'), model\_save\_path, "\_rbf")

* **SVC(kernel='rbf'):** Erstellt das SVM-Modell mit einem **RBF-Kernel**, der für die nicht-lineare Trennung zwischen den Klassen verwendet wird.
* **super().init(...):** Ruft den Konstruktor der BaseModel-Klasse auf, um das SVM-Modell zu initialisieren und für das Speichern zu konfigurieren. Der Zusatz "\_rbf" wird als Identifier für das RBF-Modell im Speicherpfad verwendet.

###### RandomForestClassifier

Das **Random Forest**-Modell ist ein Ensemble-Lernalgorithmus, der auf der Kombination mehrerer Entscheidungsbäume basiert. Jeder Baum wird auf einem zufällig ausgewählten Teil des Datensatzes trainiert, und die endgültige Klassifikation wird durch die Mehrheitsentscheidung aller Bäume getroffen. Dies ermöglicht eine robuste Modellierung und reduziert das Risiko des **Overfitting**.

Theorie

Random Forest kombiniert die Vorhersagen vieler **Entscheidungsbäume**, um eine **genauere und stabilere** Klassifikation zu erzielen. Dabei wird für jeden Baum ein **Random Sampling** von Datenpunkten und Merkmalen vorgenommen, wodurch jeder Baum im Ensemble unterschiedliche Trainingsdaten erhält. Das Aggregieren der Vorhersagen der Bäume hilft, **Fehler zu minimieren** und zu einer besseren Generalisierung zu führen.

**Vorteile des Random Forest:**

* **Robust gegenüber Overfitting**, da viele Bäume kombiniert werden.
* Kann **nicht-lineare Beziehungen** und komplexe Muster in den Daten modellieren.
* Funktioniert gut mit **hochdimensionalen Daten** und großen Datensätzen.
* **Geringere Empfindlichkeit gegenüber Ausreißern** als einzelne Entscheidungsbäume.

**Nachteile des Random Forest:**

* Kann bei sehr großen Datensätzen **rechenintensiv** sein.
* Die Modelle sind oft **weniger interpretierbar** als einzelne Entscheidungsbäume.
* **Speicherintensiv**, da viele Entscheidungsbäume im Modell gespeichert werden müssen.

Aufbau und Funktionsweise

Das Random Forest-Modell besteht aus einer Vielzahl von Entscheidungsbäumen. Für jedes Training werden zufällig **Teilmengen der Daten** und **Merkmalsuntergruppen** verwendet, um eine Vielzahl von Entscheidungsbäumen zu erstellen. Am Ende gibt das Modell die Klasse aus, die von den meisten Bäumen vorhergesagt wird (Mehrheitsvotum).

**Funktionsweise im Detail:**

1. **Random Sampling:** Bei jedem Baum wird eine zufällige Teilmenge der Trainingsdaten ausgewählt (Bootstrap Aggregation, oder Bagging), und der Baum wird auf dieser Teilmenge trainiert.
2. **Zufällige Merkmalsauswahl:** Für die Entscheidungsfindung bei jedem Split des Entscheidungsbaums wird eine zufällige Teilmenge der Merkmale ausgewählt.
3. **Ensemble-Vorhersage:** Die endgültige Vorhersage des Modells wird durch die Aggregation der Vorhersagen aller Bäume (Mehrheitsvotum bei Klassifikation) ermittelt.

Anwendungen

Random Forests sind sehr vielseitig und können in vielen Bereichen eingesetzt werden:

* **Kreditrisikoanalyse:** Identifikation von Kreditnehmern mit hohem Risiko.
* **Medizinische Diagnostik:** Klassifikation von Krankheiten anhand von Patientendaten.
* **Bildverarbeitung:** Erkennung von Objekten und Mustern in Bildern.
* **Marketing:** Vorhersage von Kundenverhalten und Segmentierung.

Verlässlichkeit und Performance

Random Forest ist in der Regel **sehr robust** und bietet eine hohe **Generalierungsfähigkeit**, auch bei komplexen und hochdimensionalen Daten. Die **Robustheit** gegenüber Overfitting kommt durch das Aggregieren vieler Entscheidungsbäume. Die Leistung des Modells kann durch die Wahl der **Anzahl der Bäume** (n\_estimators) und der **maximalen Tiefe der Bäume** beeinflusst werden. Das Modell benötigt jedoch relativ viel **Speicherplatz** und **Rechenleistung**, insbesondere bei sehr großen Datensätzen.

Imports

Die notwendigen Imports für dieses Modell sind:

from sklearn.ensemble import RandomForestClassifier

from .training\_basemodel import BaseModel

* **RandomForestClassifier:** Wird aus dem sklearn.ensemble-Modul importiert, um das Random Forest-Klassifikationsmodell zu implementieren.
* **BaseModel:** Die für Scikit-Learn definierte Basisklasse, die grundlegende Trainings- und Speichermethoden bereitstellt.

training\_\_sklearn.py

Der Random Forest-Klassifikator wird mit einer **festgelegten Anzahl von Bäumen** (n\_estimators=100) instanziiert, um eine robuste Klassifikation zu gewährleisten.

class RandomForest(BaseModel):

def \_\_init\_\_(self, model\_save\_path):

super().\_\_init\_\_(RandomForestClassifier(n\_estimators=100), model\_save\_path)

* **RandomForestClassifier(n\_estimators=100):** Erstellt das Random Forest-Modell mit 100 Entscheidungsbäumen. Dies ist die Standardanzahl von Bäumen, die verwendet wird, aber der Wert kann angepasst werden, um die Leistung zu optimieren.
* **super().init(...):** Ruft den Konstruktor der BaseModel-Klasse auf, um das Random Forest-Modell zu initialisieren und für das Speichern zu konfigurieren.

#### PyTorch

Als Ergänzung zu den Scikit-Learn-Modellen wurde PyTorch als zweiter Modell-Anbieter in das System integriert. PyTorch bietet eine flexible und leistungsstarke Bibliothek für die Entwicklung und das Training von tiefen neuronalen Netzwerken. Im Vergleich zu Scikit-Learn unterscheidet sich die Verwaltung von PyTorch-Modellen in mehreren wesentlichen Aspekten, insbesondere in Bezug auf die Architekturdefinition, das Training und die Anpassung der Eingabedaten beziehungsweise der Modellarchitektur selbst. Diese Unterschiede werden im Folgenden ersichtlich.

##### Basismodell

Das Basismodell für PyTorch stellt eine Wrapper-Klasse für die instanziierten Modelle dar und definiert die Schnittstellen nach außen. Durch sie erhält das Modell die eigentlichen angebotenen Funktionen und kommuniziert mit Klassen, die eine Aggregation mit ihr eingehen.

###### Imports

Für die Implementierung der Basismodell-Klasse für die PyTorch-Modelle werden folgende Module benötigt:

* **os**: Wird verwendet, um zu überprüfen, ob ein Modell bereits gespeichert wurde, und um mit Dateipfaden zu arbeiten.
* **torch**: Die Hauptbibliothek für PyTorch, die für das Erstellen und Trainieren von Modellen verwendet wird.
* **torch.optim**: Beinhaltet Optimierungsalgorithmen wie Adam.
* **torch.nn**: Enthält die Grundbausteine für neuronale Netzwerke.
* **train\_test\_split aus sklearn.model\_selection**: Wird verwendet, um den Datensatz in Trainings- und Testdaten zu unterteilen.
* **classification\_report aus sklearn.metrics**: Wird verwendet, um den Klassifikationsbericht zu erstellen, der Metriken wie Präzision und F1-Score enthält.
* **LabelEncoder aus sklearn.preprocessing**: Wird verwendet, um die Zielwerte (Labels) in numerische Werte zu kodieren.
* **TensorDataset und DataLoader aus torch.utils.data**: Diese helfen beim Erstellen von Datasets und deren Batch-Verarbeitung.
* **preprocess\_image aus modules.ImageProcessor**: Eine benutzerdefinierte Funktion zur Vorverarbeitung von Bildern, damit sie im richtigen Format für das Modell vorliegen.
* **adjust\_model\_output\_layer\_pytorch, adjust\_input\_channels\_pytorch\_tensor, adjust\_input\_channels\_pytorch aus modules.ModelProcessor**: Diese benutzerdefinierten Funktionen stellen sicher, dass das Modell die richtigen Dimensionen und Eingabekonfigurationen hat.
* **IModel aus ..IModel**: Eine benutzerdefinierte Schnittstelle/Basisklasse, die von jedem Modell innerhalb des Systems implementiert wird.

###### training\_basemodel\_pytorch.py

Folgende Klasse implementiert oben genanntes IModel-Interface und stellt die logische Funktionalität für die PyTorch-Modelle bereit.

Code-Implementierung:

class BaseModelTorch(IModel):

def \_\_init\_\_(self, model, model\_save\_path, model\_name\_extension=""):

self.model = model

self.model\_save\_path = model\_save\_path

self.model\_name = self.model.\_\_class\_\_.\_\_name\_\_ + model\_name\_extension

self.device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

self.model.to(self.device)

Konstruktor

Der Konstruktor \_\_init\_\_ initialisiert das Modell und speichert den Pfad, unter dem es gespeichert werden soll, sowie den Modellnamen. Es wird auch überprüft, ob eine GPU verfügbar ist, und das Modell wird auf das entsprechende Gerät (CPU oder GPU) verschoben.

Parameter

* **model (object)**: Das initialisierte PyTorch-Modell.
* **model\_save\_path (str)**: Der Pfad, unter dem das Modell gespeichert werden soll.
* **model\_name\_extension (str, optional)**: Ein optionaler Zusatz für den Modellnamen (z.B. zur Unterscheidung ähnlicher Modelle).

Methoden

In diesem Abschnitt wird die Implementierung der oben beschriebenen Klasse detailliert analysiert. Dabei werden die einzelnen Methoden, ihre Funktionsweise sowie ihre spezifische Rolle innerhalb der KI-Komponente erläutert.

check(self)

Die check-Methode wird verwendet, um zu überprüfen, ob ein trainiertes Modell bereits gespeichert ist und ob es erfolgreich geladen werden kann. Sie stellt sicher, dass das Modell bereit ist, um Vorhersagen zu treffen oder weiter trainiert zu werden.

**Schritt 1: Erstellen des Modellpfads**

def check(self):  
 try:  
 model\_path = os.path.join(self.model\_save\_path, f"trained\_{self.model\_name}.pt")

Der Pfad zur Modell-Datei wird erstellt, indem der model\_save\_path mit dem Namen des Modells und der Erweiterung .pt kombiniert wird. Das Modell wird unter dem Namen trained\_<model\_name>.pt gespeichert. Zum Beispiel könnte dies trained\_NeuralNetwork.pt sein. Dies gibt uns den vollständigen Pfad, wo das Modell gespeichert ist, sodass wir die Datei später laden können.

**Schritt 2: Überprüfen, ob das Modell existiert**

if os.path.isfile(model\_path):

Es wird überprüft, ob die Datei, die das Modell enthält, tatsächlich auf dem angegebenen Pfad existiert. Die Methode os.path.isfile() gibt True zurück, wenn die Datei vorhanden ist, andernfalls False. Wenn das Modell vorhanden ist, wird der Rest der Methode ausgeführt; andernfalls wird der Fehler abgefangen und False zurückgegeben.

**Schritt 3: Laden der Modellmetadaten**

num\_classes = len(self.load\_model\_metadata(self.model\_save\_path, self.model\_name)[1])

Falls das Modell vorhanden ist, lädt die Methode self.load\_model\_metadata() die Metadaten des Modells (wie z.B. die Anzahl der Klassen, die das Modell lernen soll). Die load\_model\_metadata-Methode gibt in diesem Fall mindestens zwei Werte zurück, wobei der zweite Wert (Index 1) die Klassenlabels oder die Anzahl der Klassen enthält. Diese Information wird verwendet, um die Modellarchitektur (insbesondere die Ausgabeschicht) korrekt zu konfigurieren.

**Schritt 4: Anpassen der Ausgabeschicht des Modells**

adjust\_model\_output\_layer\_pytorch(self.model, num\_classes, self.device, self.model\_name)

Diese Methode passt die Ausgabeschicht des Modells an, je nachdem, wie viele Klassen (Zielklassen) das Modell zu klassifizieren hat. Der Parameter num\_classes wird verwendet, um die Ausgabeschicht so zu konfigurieren, dass sie die richtige Anzahl von Ausgabekanälen für die Klassifikation hat. Die Methode berücksichtigt auch das Gerät (CPU oder GPU), auf dem das Modell laufen soll.

**Schritt 5: Anpassen der Eingabekanäle des Modells**

adjust\_input\_channels\_pytorch(self.model)

Diese Methode stellt sicher, dass die Eingabedaten des Modells korrekt an die Architektur angepasst werden, z.B. dass die Anzahl der Eingabekanäle mit den Trainingsdaten übereinstimmt. Beispielsweise muss bei einem Bildklassifizierungsmodell geprüft werden, ob die Eingabe die erwartete Anzahl von Kanälen (z.B. 3 für RGB-Bilder) hat.

**Schritt 6: Laden der Modellgewichtungen**

self.model.load\_state\_dict(torch.load(model\_path))

Nachdem die Modellarchitektur und Eingabekanäle angepasst wurden, werden nun die tatsächlichen Gewichtungen des Modells aus der gespeicherten Datei geladen. torch.load(model\_path) lädt die Gewichtungen aus der .pt-Datei, und self.model.load\_state\_dict() wendet diese Gewichtungen auf das Modell an.

**Schritt 7: Verschieben des Modells auf das richtige Gerät**

self.model.to(self.device)

Nachdem das Modell mit den Gewichtungen geladen wurde, wird es auf das Gerät verschoben, das in der \_\_init\_\_-Methode festgelegt wurde (entweder die CPU oder GPU). Dies stellt sicher, dass das Modell auf der richtigen Hardware ausgeführt wird, was für die Berechnungen wichtig ist.

**Schritt 8: Erfolgreiches Laden des Modells**

return True

Wenn alle oben genannten Schritte erfolgreich ausgeführt wurden, wird True zurückgegeben, was bedeutet, dass das Modell erfolgreich überprüft und geladen wurde.

**Schritt 9: Fehlerbehandlung**

except Exception as e:  
 return False

return False

Falls bei irgendeinem Schritt ein Fehler auftritt (z.B. Datei nicht gefunden, Fehler beim Laden der Gewichtungen oder Anpassung der Architektur), wird der Fehler abgefangen und False zurückgegeben. Dies stellt sicher, dass der Benutzer informiert wird, dass etwas schief gelaufen ist, ohne dass die Anwendung abstürzt. Sollte kein gespeichertes Modell auffindbar sein, so wird ebenfalls False zurückgegeben.

train(self, dataset, epochs, reshape\_size, batch\_size=16, lr=0.001)

Die train-Methode wird verwendet, um ein Modell auf einem gegebenen Datensatz zu trainieren. Sie umfasst die Vorbereitung der Eingabedaten, das Setzen der Trainingsparameter und das Durchführen des Trainingsprozesses über eine bestimmte Anzahl von Epochen.

**Schritt 1: Initialisierung der train-Methode und Start des try-Blocks**

def train(self, dataset, epochs, reshape\_size, batch\_size=16, lr=0.001):

try:

Die Methode train wird aufgerufen, um das Modell zu trainieren. Der gesamte Trainingscode wird in einem try-Block ausgeführt, um Fehler während des Trainings zu fangen.

**Schritt 1: Aufteilen des Datensatzes in Eingabedaten (x) und Labels (y)**

x, y = dataset

Der Datensatz wird in zwei Teile aufgeteilt: x enthält die Eingabedaten (z. B. Bilder) und y enthält die zugehörigen Labels (z. B. Klassennamen oder Kategorien). Der Datensatz wird erwartet, als Tuple (x, y) vorzuliegen.

**Schritt 2: Kodieren der Labels mit LabelEncoder**

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(y)

num\_classes = len(label\_encoder.classes\_)

Der LabelEncoder wird verwendet, um die Labels (y) in numerische Werte umzuwandeln. Dies ist notwendig, da Modelle mit numerischen Werten arbeiten. Die Anzahl der verschiedenen Klassen (num\_classes) wird berechnet, um später die Ausgabeschicht des Modells korrekt anzupassen.

**Schritt 3: Umwandeln der Eingabedaten in Tensoren**

x\_tensor = torch.tensor(x, dtype=torch.float32).to(self.device)

y\_tensor = torch.tensor(y\_encoded, dtype=torch.long).to(self.device)

Die Eingabedaten x und die kodierten Labels y\_encoded werden in torch.Tensor-Objekte umgewandelt. x\_tensor enthält die Eingabedaten als Float32-Tensor, und y\_tensor enthält die Labels als Long-Tensor. Diese Tensoren werden auf das entsprechende Gerät (entweder CPU oder GPU) verschoben, das in der \_\_init\_\_-Methode definiert wurde.

**Schritt 4: Anpassen der Eingabekanäle des Modells**

x\_tensor = adjust\_input\_channels\_pytorch\_tensor(self.model, x\_tensor, reshape\_size)

Diese Methode sorgt dafür, dass die Eingabedaten die korrekte Form und die richtigen Kanäle haben, um vom Modell verarbeitet zu werden. Es wird das Modell angepasst, damit die Eingabedaten die korrekte Größe (reshape\_size) und Form haben, die während des Trainings erwartet werden.

**Schritt 5: Erstellen des DataLoader für das Training**

dataset = TensorDataset(x\_tensor, y\_tensor)

train\_loader = DataLoader(dataset, batch\_size=batch\_size, shuffle=True)

Ein TensorDataset wird erstellt, das die Eingabedaten (x\_tensor) und Labels (y\_tensor) zusammenführt. Ein DataLoader wird verwendet, um die Daten in Mini-Batches zu laden. batch\_size definiert die Größe jedes Batches, und shuffle=True sorgt dafür, dass die Daten vor jedem Epochendurchgang zufällig gemischt werden.

**Schritt 6: Anpassen der Ausgabeschicht des Modells**

self.model = adjust\_model\_output\_layer\_pytorch(self.model, num\_classes, self.device, self.model\_name)

Die Ausgabeschicht des Modells wird so angepasst, dass sie mit der Anzahl der Klassen (num\_classes) übereinstimmt. Dies sorgt dafür, dass die Anzahl der Ausgabekanäle im Modell der Anzahl der Zielklassen entspricht. Das Modell wird auch auf das richtige Gerät verschoben.

**Schritt 7: Definition des Kriteriums und des Optimierers**

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(self.model.parameters(), lr=lr)

Der Verlust (Loss) wird mit CrossEntropyLoss definiert. Diese Verlustfunktion wird häufig für Klassifikationsprobleme verwendet, bei denen die Eingabedaten mehreren Klassen zugeordnet werden müssen. Der Optimierer Adam wird verwendet, um die Gewichtungen des Modells zu aktualisieren. Der Lernratenwert (lr) kann als Parameter an die Methode übergeben werden und bestimmt, wie schnell die Gewichtungen während des Trainings angepasst werden.

**Schritt 9: Initialisierung der Trainingsschleife über mehrere Epochen**

for epoch in range(epochs):

Der Trainingsprozess wird für eine bestimmte Anzahl von Epochen (epochs) wiederholt.

**Schritt 10: Setzen des Modells in den Trainingsmodus und Initialisierung des Verlusts**

self.model.train()

running\_loss = 0.0

Das Modell wird in den Trainingsmodus versetzt, sodass es während des Trainings korrekt funktioniert (z. B. Dropout, BatchNorm). Der running\_loss wird auf 0 gesetzt, um den kumulierten Verlust während der Epoche zu berechnen.

**Schritt 11: Iteration über Mini-Batches im Trainings-DataLoader**

for inputs, labels in train\_loader:

optimizer.zero\_grad()

Für jedes Mini-Batch (inputs, labels) im train\_loader wird der Gradientenpuffer des Optimierers mit optimizer.zero\_grad() zurückgesetzt.

**Schritt 12: Vorwärtsdurchlauf und Vorhersagen des Modells**

try:

outputs = self.model(inputs)

Ein innerer try-Block wird verwendet, um den Vorwärtsdurchlauf durch das Modell auszuführen. Wenn ein Fehler auftritt (z. B. eine falsche Eingabegröße), wird dieser Fehler im inneren except-Block behandelt.

**Schritt 13: Fehlerbehandlung beim Vorwärtsdurchlauf**

except Exception as e:

error = (f'{self.model\_name} could not be trained. Try using a different reshape size. '

f'Error: {str(e)}')

return False, error

Falls beim Vorwärtsdurchlauf ein Fehler auftritt, wird eine detaillierte Fehlermeldung erzeugt, die angibt, dass das Modell möglicherweise aufgrund einer falschen Eingabegröße nicht trainiert werden konnte. Das Training wird mit der Rückgabe False und der Fehlermeldung abgebrochen.

**Schritt 14: Umgang mit Modellausgaben, die als Tuple zurückgegeben werden**

if isinstance(outputs, tuple):

outputs = outputs[0]

Falls das Modell ein Tuple zurückgibt (dies passiert bei bestimmten Architekturen), wird nur der erste Wert aus dem Tuple verwendet, um mit den Vorhersagen weiterzuarbeiten.

**Schritt 15: Berechnung des Verlustes (Loss)**

loss = criterion(outputs, labels)

Der Verlust (Fehler) zwischen den Vorhersagen (outputs) und den echten Labels (labels) wird mit der Verlustfunktion (CrossEntropyLoss) berechnet.

**Schritt 16: Backpropagation und Optimierung**

loss.backward()

optimizer.step()

Der Verlust wird zurückpropagiert, um die Gradienten zu berechnen (loss.backward()). Die Modellparameter werden basierend auf den berechneten Gradienten aktualisiert (optimizer.step()).

**Schritt 17: Akkumulieren des Verlusts für die aktuelle Epoche**

running\_loss += loss.item()

Der Verlust für das aktuelle Mini-Batch wird zum kumulierten Verlust (running\_loss) der aktuellen Epoche hinzugefügt.

**Schritt 18: Ausgabe des Trainingsfortschritts**

print(f"Epoch [{epoch + 1}/{epochs}], Loss: {running\_loss / len(train\_loader)}")

Nach jedem Epochendurchlauf wird der durchschnittliche Verlust des gesamten Mini-Batches in dieser Epoche ausgegeben.

**Schritt 19: Speichern des trainierten Modells**

model\_path = os.path.join(self.model\_save\_path, f"trained\_{self.model\_name}.pt")

torch.save(self.model.state\_dict(), model\_path)

Nach dem Training wird das Modell mit den gelernten Gewichtungen gespeichert. Der Speicherort wird durch den model\_save\_path und den Modellnamen bestimmt.

**Schritt 20: Speichern der Modellmetadaten**

self.save\_model\_metadata(self.model\_save\_path, self.model\_name, reshape\_size, list(label\_encoder.classes\_))

Die Modellmetadaten, wie die Eingabegröße und die Klassenbezeichner, werden ebenfalls gespeichert.

**Schritt 21: Erfolgreiches Training abschließen**

print('Trained and dumped model: ', model\_path)

return True, None

Eine Bestätigung wird ausgegeben, dass das Modell erfolgreich trainiert und gespeichert wurde. Es wird True zurückgegeben, um anzuzeigen, dass das Training erfolgreich war.

**Schritt 22: Fehlerbehandlung während des Trainings (äußere except-Klausel)**

except Exception as e:

error = f'{self.model\_name} could not be trained. Error: {str(e)}'

return False, error

Wenn während des Trainings ein Fehler auftritt, wird dieser abgefangen und eine Fehlermeldung wird ausgegeben. Es wird False zurückgegeben, um anzuzeigen, dass das Training fehlgeschlagen ist.

make\_prediction(self, image)

Die Methode make\_prediction dient der Vorhersage für ein gegebenes Eingabebild. Dabei wird das trainierte Modell verwendet, um eine Vorhersage zu treffen und gegebenenfalls Wahrscheinlichkeiten für jede mögliche Klasse zu berechnen. Der Ablauf dieser Methode kann in die folgenden Schritte unterteilt werden:

Schritt 1: Definition der Methode make\_prediction

def make\_prediction(self, image):

Diese Methode wird verwendet, um eine Vorhersage für ein einzelnes Eingabebild zu machen. Sie erwartet ein Bild als Eingabeparameter (image).

Schritt 2: Überprüfen, ob das Modell trainiert wurde

if self.check() is False:

raise ValueError('Model is not trained')

Die Methode self.check() wird aufgerufen, um zu überprüfen, ob ein trainiertes Modell vorhanden ist. Falls das Modell nicht trainiert ist (self.check() == False), wird eine **Fehlermeldung geworfen** (raise ValueError), die den Benutzer darauf hinweist, dass das Modell nicht trainiert ist.

Schritt 3: Laden der Modell-Metadaten

reshape\_size, class\_labels = self.load\_model\_metadata(self.model\_save\_path, self.model\_name)

Hier werden die **gespeicherten Metadaten** des Modells geladen, die folgende Informationen enthalten:

1. **reshape\_size** – Die erwartete Größe, auf die das Eingabebild skaliert werden muss.
2. **class\_labels** – Die Liste der Klassennamen, die das Modell vorhersagen kann.

Schritt 4: Vorverarbeitung des Eingabebildes

image\_array = preprocess\_image(image, reshape\_size)

Das Eingabebild (image) wird mit der Methode preprocess\_image auf die erforderliche Größe (reshape\_size) gebracht und normalisiert. Das Ergebnis wird in der Variable **image\_array** gespeichert.

Schritt 5: Öffnen eines try-Blocks zur Durchführung der Vorhersage

try:

Hier wird ein **try-Block** geöffnet, um sicherzustellen, dass Fehler während der Vorhersage erkannt und behandelt werden können.

Schritt 6: Setzen des Modells in den Evaluierungsmodus

self.model.eval()

Das Modell wird in den **Evaluierungsmodus** (eval()) versetzt, um sicherzustellen, dass keine Trainings-spezifischen Mechanismen (wie Dropout oder Batch-Normalisierung) aktiv sind.

Schritt 7: Öffnen eines with torch.no\_grad()-Blocks zur Deaktivierung des Gradienten-Trackings

with torch.no\_grad():

torch.no\_grad() wird verwendet, um das **Gradienten-Tracking** zu deaktivieren. Dadurch wird Speicher gespart und die Vorhersagegeschwindigkeit erhöht.

Schritt 8: Umwandlung des Bildarrays in einen PyTorch-Tensor

image\_tensor = torch.tensor(image\_array, dtype=torch.float32).to(self.device)

Das vorverarbeitete Bild (image\_array) wird in einen PyTorch-Tensor (image\_tensor) umgewandelt. Der Tensor wird als float32 deklariert und auf das richtige Gerät (self.device, also CPU oder GPU) verschoben.

Schritt 9: Anpassung der Eingabekanäle des Bildes an das Modell

image\_tensor = adjust\_input\_channels\_pytorch\_tensor(self.model, image\_tensor, reshape\_size)

Da verschiedene Modelle eine unterschiedliche Anzahl von Eingabekanälen (z. B. 1 für Graustufenbilder, 3 für RGB) erwarten, wird hier sichergestellt, dass das Bild die richtige Kanalanzahl hat.

Schritt 10: Durchführung der Vorhersage mit dem Modell

prediction = self.model(image\_tensor)

Das Modell wird mit dem verarbeiteten Bild (image\_tensor) gefüttert, um eine Vorhersage zu erzeugen. Das Ergebnis wird in der Variable **prediction** gespeichert.

Schritt 11: Berechnung der Wahrscheinlichkeiten mit der Softmax-Funktion

probabilities = f.softmax(prediction, dim=1).cpu().detach().numpy()[0]

Da das Modell rohe Werte (**Logits**) ausgibt, werden diese mit der **Softmax-Funktion** (f.softmax) in Wahrscheinlichkeiten umgerechnet. Anschließend wird das Ergebnis auf die **CPU** übertragen (.cpu()), vom Rechen-Graphen **getrennt** (.detach()) und in ein **NumPy-Array** umgewandelt (.numpy()). Die Wahrscheinlichkeiten für jede Klasse befinden sich im ersten Element [0] des Arrays.

Schritt 12: Standardmäßige Klassennamen generieren, falls class\_labels None ist

if class\_labels is None:

class\_labels = [f"class\_{i}" for i in range(len(probabilities))]

Falls keine Klassennamen (class\_labels) in den Metadaten gespeichert sind, wird eine Liste mit generischen Namen (class\_0, class\_1, etc.) erstellt.

Schritt 13: Erstellung eines Wörterbuchs für die Vorhersagen

prediction\_dict = {label: float(prob) for label, prob in zip(class\_labels, probabilities)}

Die vorhergesagten Wahrscheinlichkeiten werden den entsprechenden Klassennamen zugeordnet und in einem Wörterbuch (prediction\_dict) gespeichert.

Schritt 14: Ausgabe der Summe der Wahrscheinlichkeiten

print(f"Sum of probabilities: {probabilities.sum()}")

Die Summe aller Wahrscheinlichkeiten wird ausgegeben, um sicherzustellen, dass sie sich auf **1.0** belaufen (was ein korrektes Softmax-Ergebnis bestätigt).

Schritt 15: Rückgabe der Vorhersage

return prediction\_dict

Die Methode gibt das **prediction\_dict** zurück, das die Wahrscheinlichkeiten für jede Klasse enthält.

Schritt 16: Ausnahmebehandlung (except-Block), falls ein Fehler auftritt

except Exception as e:

raise ValueError(f"Error during prediction: {str(e)}")

Falls während der Vorhersage **irgendein Fehler** auftritt, wird er hier abgefangen. Ein **ValueError** wird ausgelöst, damit der Benutzer weiß, dass die Vorhersage fehlgeschlagen ist.

get\_classification\_report(self, dataset, reshape\_size)

Die Methode führt eine Evaluierung des Modells durch, indem sie den **Klassifikationsbericht** (classification report) auf einem Testdatensatz berechnet. Sie teilt sich in mehrere Schritte:

Schritt 1: Definition der Methode get\_classification\_report

def get\_classification\_report(self, dataset, reshape\_size):

Diese Methode berechnet und gibt einen **Klassifikationsbericht** (Precision, Recall, F1-Score) für das trainierte Modell zurück.

Sie benötigt:

* dataset: Ein Tupel (x, y) mit **Eingabedaten** (x) und **Zielwerten** (y).
* reshape\_size: Die **Größe**, auf die die Eingabebilder skaliert werden müssen.

Schritt 2: Öffnen eines try-Blocks

try:

Hier wird ein try-Block geöffnet, um Fehler während der Modellbewertung abzufangen.

Schritt 3: Laden der Eingabedaten und Aufteilen in Trainings- und Testdaten

x, y = dataset

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

Die Eingabedaten (x) und Zielwerte (y) werden aus dem dataset-Tupel entpackt.

Dann wird das Dataset in **Trainings- (80%)** und **Testdaten (20%)** aufgeteilt mit der Funktion train\_test\_split():

* x\_train, y\_train: **Trainingsdaten** (wird hier nicht genutzt, da nur das Testset für die Evaluation benötigt wird).
* x\_test, y\_test: **Testdaten**, die für die Modellbewertung verwendet werden.

Schritt 4: Kodierung der Zielwerte (Labels)

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(y\_test)

Die **Kategorien** in y\_test werden mit LabelEncoder in numerische Werte umgewandelt (y\_encoded), damit sie für PyTorch verwendbar sind.

Schritt 5: Umwandlung der Testdaten in PyTorch-Tensoren

x\_test\_tensor = torch.tensor(x\_test, dtype=torch.float32).to(self.device)

y\_test\_tensor = torch.tensor(y\_encoded, dtype=torch.long).to(self.device)

Die Testdaten (x\_test) und deren Labels (y\_encoded) werden in **PyTorch-Tensoren** konvertiert. Sie werden außerdem auf das richtige Gerät (self.device, also CPU oder GPU) übertragen.

Schritt 6: Anpassung der Eingabekanäle des Modells

x\_test\_tensor = adjust\_input\_channels\_pytorch\_tensor(self.model, x\_test\_tensor, reshape\_size)

Falls das Modell eine andere Anzahl von **Eingabekanälen** erwartet (z. B. 1 für Graustufen, 3 für RGB), wird dies hier korrigiert.

Schritt 7: Erstellung eines DataLoader für das Testset

dataset = TensorDataset(x\_test\_tensor, y\_test\_tensor)

test\_loader = DataLoader(dataset, batch\_size=16, shuffle=False)

Ein **Tensor-Dataset** wird mit den Testdaten (x\_test\_tensor, y\_test\_tensor) erstellt. Der DataLoader lädt die Daten in Batches von **16** Bildern (batch\_size=16) und **mischt die Reihenfolge nicht** (shuffle=False).

Schritt 8: Setzen des Modells in den Evaluierungsmodus

self.model.eval()

Das Modell wird mit eval() in den **Evaluierungsmodus** versetzt, um sicherzustellen, dass Dropout oder Batch-Normalisierung deaktiviert sind.

Schritt 9: Initialisierung von Listen für Vorhersagen und wahre Klassen

y\_pred\_classes = []

y\_true\_classes = []

Zwei **leere Listen** werden erstellt:

* y\_pred\_classes: Speichert die vorhergesagten Klassen.
* y\_true\_classes: Speichert die echten Labels.

Schritt 10: Öffnen eines with torch.no\_grad()-Blocks

with torch.no\_grad():

Die **Gradientenberechnung** wird deaktiviert (torch.no\_grad()), um Speicher zu sparen und die Vorhersage schneller zu machen.

Schritt 11: Durchlaufen des Testsets und Vorhersage der Klassen

for inputs, labels in test\_loader:

outputs = self.model(inputs)

Das **Testset wird batchweise durchlaufen**. Die **Eingaben (inputs)** werden durch das Modell geleitet, um die **Rohwerte (outputs)** zu berechnen.

Schritt 12: Falls das Modell ein Tupel zurückgibt, nur den ersten Wert nehmen

if isinstance(outputs, tuple):

outputs = outputs[0]

Falls outputs ein **Tupel** ist (z. B. weil das Modell mehrere Ausgaben hat), wird nur das **erste Element** verwendet.

Schritt 13: Umwandlung der Modell-Ausgabe in Klassenvorhersagen

\_, predicted = torch.max(outputs, 1)

torch.max(outputs, 1) wählt die **Klasse mit der höchsten Wahrscheinlichkeit** aus. Die Variable predicted enthält die vorhergesagten Klassen als **Tensor**.

Schritt 14: Speichern der Vorhersagen und der echten Klassen

y\_pred\_classes.extend(predicted.cpu().numpy())

y\_true\_classes.extend(labels.cpu().numpy())

Die vorhergesagten Klassen (predicted) und echten Labels (labels) werden in **Listen gespeichert**:

* predicted.cpu().numpy() konvertiert die Vorhersagen in eine **NumPy-Liste** und fügt sie y\_pred\_classes hinzu.
* labels.cpu().numpy() konvertiert die echten Klassen in eine **NumPy-Liste** und fügt sie y\_true\_classes hinzu.

Schritt 15: Berechnung des Klassifikationsberichts

report = classification\_report(y\_true\_classes, y\_pred\_classes, zero\_division=0)

Die classification\_report()-Funktion aus sklearn.metrics wird verwendet, um den **Klassifikationsbericht** zu erstellen. zero\_division=0 stellt sicher, dass es bei Klassen mit **keiner einzigen positiven Vorhersage** nicht zu Fehlern kommt.

Schritt 16: Rückgabe des Klassifikationsberichts

return True, report

Falls alles erfolgreich war, wird **True** und der Klassifikationsbericht (report) zurückgegeben.

Schritt 17: Ausnahmebehandlung (except-Block)

except Exception as e:

error = f"Error during model evaluation: {str(e)}"

return False, error

Falls ein Fehler während der Ausführung auftritt, wird er hier abgefangen. Der Fehler wird als String gespeichert und mit **False** zurückgegeben, um anzuzeigen, dass die Auswertung fehlgeschlagen ist.

##### Modellinstanzen

Theorie

Aufbau und Funktionsweise

Anwendungen

Verlässlichkeit und Performance

Imports

training\_\_sklearn.py

#### Tensorflow

Als Ergänzung zu den Scikit-Learn- und PyTorch-Modellen wurde TensorFlow als weiterer Modell-Anbieter in das System integriert. TensorFlow ist eine leistungsstarke und weit verbreitete Bibliothek für die Entwicklung und das Training von tiefen neuronalen Netzwerken. Im Vergleich zu Scikit-Learn und PyTorch unterscheidet sich die Verwaltung von TensorFlow-Modellen in mehreren wesentlichen Aspekten, insbesondere in Bezug auf die Architekturdefinition, das Training und die Optimierung der Modelle. Diese Unterschiede werden im Folgenden ersichtlich.

##### Basismodell

Das Basismodell für Tensorflow stellt eine Wrapper-Klasse für die instanziierten Modelle dar und definiert die Schnittstellen nach außen. Durch sie erhält das Modell die eigentlichen angebotenen Funktionen und kommuniziert mit Klassen, die eine Aggregation mit ihr eingehen.

###### Imports

Für die Implementierung der Basismodell-Klasse für die Tensorflow-Modelle werden folgende Module benötigt:

 **os**: Wird für Datei- und Pfadoperationen genutzt, z. B. zum Überprüfen oder Speichern von Modellen.

 **cv2**: Ermöglicht die Verarbeitung von Bildern, insbesondere für die Vorverarbeitung der Eingabedaten.

 **sklearn.metrics.classification\_report**: Wird verwendet, um eine detaillierte Auswertung der Modellleistung anhand von Metriken wie Präzision, Recall und F1-Score zu erstellen.

 **tensorflow.keras**: Bietet die notwendigen Funktionen zum Erstellen, Trainieren und Laden von neuronalen Netzwerken in TensorFlow.

 **numpy**: Ermöglicht effiziente numerische Berechnungen und Datenmanipulationen, insbesondere für Arrays und Matrizen.

 **sklearn.preprocessing.LabelEncoder**: Dient zur Umwandlung von Klassennamen in numerische Werte für das Training des Modells.

 **sklearn.model\_selection.train\_test\_split**: Ermöglicht die Aufteilung der Daten in Trainings- und Testsets, um die Modellleistung zu validieren.

 **modules.ImageProcessor.preprocess\_image**: Eine benutzerdefinierte Funktion zur Vorverarbeitung der Eingabebilder, um sie für das Modell nutzbar zu machen.

 **modules.ModelProcessor.adjust\_model\_output\_layer\_keras**: Eine Hilfsfunktion zur Anpassung der letzten Schicht eines Keras-Modells an die benötigte Anzahl von Klassen.

 **IModel**: Eine übergeordnete Schnittstelle, die die Struktur der Basismodell-Klasse definiert und sicherstellt, dass die Implementierung einheitlich bleibt.

###### training\_basemodel\_tensorflow.py

Folgende Klasse implementiert oben genanntes IModel-Interface und stellt die logische Funktionalität für die Tensorflow-Modelle bereit.

Code-Implementierung:

class BaseModelTF(IModel):  
 def \_\_init\_\_(self, model, build\_model\_fn, model\_save\_path, model\_name\_extension=""):  
 self.model = model self.model\_build\_fn = build\_model\_fn self.model\_save\_path = model\_save\_path self.model\_name = self.model.name + model\_name\_extension

Konstruktor

Die \_\_init\_\_-Methode ist der Konstruktor der BaseModelTF-Klasse und dient zur Initialisierung eines TensorFlow-Modell-Objekts. Sie speichert das übergebene Modell, die Funktion zur Modell-Erstellung sowie den Speicherpfad und generiert automatisch einen Modellnamen, der durch eine optionale Erweiterung ergänzt werden kann.

Parameter

 model **(object)**  
Das initialisierte TensorFlow-Modell, das von dieser Instanz verwaltet wird.

 **build\_model\_fn (function)**  
Eine Funktion, die das Modell erstellt oder rekonstruiert. Diese Funktion wird benötigt, falls das Modell neu aufgebaut werden muss.

 **model\_save\_path (str)**  
Der Pfad, unter dem das trainierte Modell gespeichert oder geladen wird.

 **model\_name\_extension (str, optional, Standard: "")**  
Ein optionaler String, der an den Modellnamen angehängt wird. Dies ist besonders nützlich, falls mehrere Modelle denselben Namen besitzen, sich aber durch unterschiedliche Konfigurationen unterscheiden.

Methoden

In diesem Abschnitt wird die Implementierung der oben beschriebenen Klasse detailliert analysiert. Dabei werden die einzelnen Methoden, ihre Funktionsweise sowie ihre spezifische Rolle innerhalb der KI-Komponente erläutert.

check(self)

Die check-Methode prüft, ob ein Modell bereits gespeichert wurde und lädt es, falls es vorhanden ist. Sie teilt sich in mehrere Schritte:

Schritt 1: Definition der Methode check

def check(self):

Diese Methode prüft, ob das Modell im angegebenen Speicherpfad existiert und lädt es, wenn dies der Fall ist.

Schritt 2: Öffnen eines try-Blocks

try:

Ein try-Block wird geöffnet, um Fehler während des Ladens des Modells abzufangen und eine ordnungsgemäße Fehlerbehandlung zu ermöglichen.

Schritt 3: Erstellen des vollständigen Pfades für das gespeicherte Modell

model\_path = os.path.join(self.model\_save\_path, f"trained\_{self.model\_name}.keras")

Der vollständige Pfad zum gespeicherten Modell wird unter Verwendung des Basisverzeichnisses (self.model\_save\_path), des Modellnamens (self.model\_name) und der Erweiterung .keras erstellt. Das Modell wird in diesem Format gespeichert, um es später mit keras.models.load\_model laden zu können.

Schritt 4: Überprüfen, ob das Modell existiert

if os.path.isfile(model\_path):

Die Methode überprüft, ob der erstellte Pfad (model\_path) auf eine existierende Datei zeigt. Dies stellt sicher, dass das Modell gespeichert wurde, bevor es geladen wird.

Schritt 5: Laden des Modells

self.model = keras.models.load\_model(model\_path)

Falls das Modell existiert, wird es mit keras.models.load\_model geladen und der Instanzvariablen self.model zugewiesen. Das Modell wird nun in der aktuellen Instanz verfügbar gemacht und kann für Vorhersagen oder andere Aufgaben verwendet werden.

Schritt 6: Rückgabe True bei erfolgreichem Laden

return True

Nachdem das Modell erfolgreich geladen wurde, gibt die Methode True zurück. Dies signalisiert, dass das Modell erfolgreich geladen wurde und für die Verwendung bereit ist.

Schritt 7: Fehlerbehandlung (Exception-Block)

except Exception:

return False

return False

Falls ein Fehler beim Laden des Modells auftritt (z. B. wenn die Datei nicht existiert oder beschädigt ist), wird der Fehler im except-Block abgefangen und False zurückgegeben. Dies zeigt an, dass das Modell nicht geladen werden konnte. Falls ein Fehler während des try-Blocks auftritt (z. B. das Modell existiert nicht oder ist beschädigt), gibt die Methode False zurück, um anzuzeigen, dass das Modell nicht geladen werden konnte.

train(self, dataset, epochs, reshape\_size, batch\_size=16, lr=None)

Die Methode train trainiert das Modell mit den angegebenen Trainingsdaten und speichert das trainierte Modell anschließend. Sie führt mehrere Schritte durch, die hier im Detail erläutert werden:

Schritt 1: Definition der Methode train

def train(self, dataset, epochs, reshape\_size, batch\_size=16, lr=None):

Die Methode train ist für das Training des Modells verantwortlich. Sie benötigt die folgenden Eingabewerte:

* dataset: Ein Tupel (x, y), wobei x die Eingabebilder und y die Zielwerte (Labels) sind.
* epochs: Die Anzahl der Trainingsdurchläufe (Epochs).
* reshape\_size: Die Größe, auf die die Bilder skaliert werden sollen.
* batch\_size: Die Größe der Batches, die beim Training verwendet werden (Standardwert: 16).
* lr: Die Lernrate des Optimierers (optional).

Schritt 2: Öffnen eines try-Blocks

try:

Ein try-Block wird geöffnet, um Fehler während des Trainingsprozesses zu fangen.

Schritt 3: Erstellen des Modells

self.model = self.model\_build\_fn(reshape\_size)

Das Modell wird mithilfe der model\_build\_fn-Funktion unter Verwendung der angegebenen reshape\_size-Größe erstellt. Diese Funktion gibt das initialisierte Modell zurück, das dann für das Training verwendet wird.

Schritt 4: Entpacken der Eingabedaten

x, y = dataset

Die Eingabedaten (x) und Zielwerte (y) werden aus dem übergebenen Dataset-Tupel entpackt.

Schritt 5: Kodierung der Zielwerte (Labels)

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(y)

num\_classes = len(label\_encoder.classes\_)

Die Zielwerte y werden mit LabelEncoder in numerische Werte (y\_encoded) umgewandelt. Zudem wird die Anzahl der Klassen (num\_classes) ermittelt, die später für die Anpassung der Modellarchitektur benötigt wird.

Schritt 6: Vorverarbeitung der Eingabedaten

x = x / 255.0

x = np.expand\_dims(x, axis=-1)

Die Eingabebilder x werden skaliert, indem sie durch 255 geteilt werden, sodass die Werte zwischen 0 und 1 liegen. Anschließend wird die Dimension der Bilder um eine zusätzliche Achse erweitert, um sie in das Modell einzuführen (z. B. für Graustufenbilder).

Schritt 7: Anpassung der Bildgröße

x = np.array([cv2.resize(img, (reshape\_size, reshape\_size)) for img in x])

Die Eingabebilder werden auf die angegebene reshape\_size-Größe skaliert, um sicherzustellen, dass alle Bilder dieselbe Größe haben. Diese Änderung erfolgt mit der OpenCV-Funktion cv2.resize.

Schritt 8: Anpassung der Ausgabeschicht des Modells

self.model = adjust\_model\_output\_layer\_keras(self.model, num\_classes, self.model\_name)

Die Ausgabeschicht des Modells wird an die Anzahl der Klassen (num\_classes) angepasst. Dies ist notwendig, damit das Modell korrekt für das Klassifizierungsproblem trainiert werden kann.

Schritt 9: Trainieren des Modells

self.model.fit(x, y\_encoded, epochs=epochs, batch\_size=batch\_size, verbose=1)

Das Modell wird mit den vorbereiteten Eingabedaten (x) und den kodierten Zielwerten (y\_encoded) trainiert. Das Training erfolgt für die angegebene Anzahl von epochs mit der spezifizierten batch\_size. Der Parameter verbose=1 sorgt dafür, dass der Trainingsfortschritt während des Trainings angezeigt wird.

Schritt 10: Speichern des trainierten Modells

model\_path = os.path.join(self.model\_save\_path, f"trained\_{self.model\_name}.keras")

self.model.save(model\_path)

Nach Abschluss des Trainings wird das Modell im angegebenen Verzeichnis (self.model\_save\_path) unter dem Namen trained\_{self.model\_name}.keras gespeichert.

Schritt 11: Speichern der Modell-Metadaten

self.save\_model\_metadata(self.model\_save\_path, self.model\_name, reshape\_size, list(label\_encoder.classes\_))

Die Metadaten des Modells, einschließlich der Modellarchitektur, der Bildgröße (reshape\_size) und der Klassenbezeichner (Label-Kodierungen), werden ebenfalls gespeichert, damit das Modell später wiederverwendet werden kann.

Schritt 12: Ausgabe des Speicherorts des Modells

print('Trained and dumped model: ', model\_path)

Schritt 13: Erfolgreiche Rückgabe

return True, None

Falls das Training und das Speichern des Modells erfolgreich waren, gibt die Methode True zurück. Der Fehlerparameter ist None, da keine Fehler aufgetreten sind.

Schritt 14: Fehlerbehandlung (Exception-Block)

except Exception as e:

error = f'{self.model\_name} could not be trained. Error: {str(e)}'

return False, error

Falls während des Trainings ein Fehler auftritt, wird dieser im except-Block abgefangen. Der Fehler wird als String (error) gespeichert und False zurückgegeben, um anzuzeigen, dass das Modell nicht erfolgreich trainiert wurde. Der Fehlertext wird als zweite Rückgabe mitgeliefert.

make\_prediction(self, image)

Die Methode make\_prediction nimmt ein Bild als Eingabe, verarbeitet es und gibt eine Vorhersage für das trainierte Modell zurück. Sie führt mehrere Schritte aus, die hier detailliert erklärt werden:

Schritt 1: Definition der Methode make\_prediction

def make\_prediction(self, image):

Die Methode make\_prediction nimmt ein Bild (image) als Eingabewert entgegen, um eine Vorhersage auf diesem Bild durch das Modell zu machen.

Schritt 2: Überprüfung, ob das Modell trainiert ist

if not self.check():

raise ValueError('Model is not trained')

Bevor die Vorhersage gemacht wird, wird überprüft, ob das Modell trainiert wurde. Dies geschieht durch den Aufruf der Methode check(). Falls das Modell nicht trainiert ist, wird eine ValueError-Ausnahme ausgelöst.

Schritt 3: Laden der Modell-Metadaten

reshape\_size, class\_labels = self.load\_model\_metadata(self.model\_save\_path, self.model\_name)

Die Methode load\_model\_metadata wird aufgerufen, um die Metadaten des Modells zu laden. Dazu gehören die Größe (reshape\_size), auf die die Eingabebilder skaliert werden müssen, und die Klassennamen (class\_labels), die das Modell verwenden soll.

Schritt 4: Vorverarbeitung des Eingabebildes

image\_array = preprocess\_image(image, reshape\_size)

Das Eingabebild (image) wird mit der Funktion preprocess\_image vorverarbeitet, wobei es auf die erforderliche Größe (reshape\_size) skaliert wird. Die vorverarbeitete Bilddatei wird als image\_array gespeichert.

Schritt 5: Öffnen eines try-Blocks

try:

Ein try-Block wird geöffnet, um Fehler während der Vorhersage abzufangen.

Schritt 6: Skalierung der Pixelwerte und Resizing

image\_array = image\_array / 255.0

image\_array = np.array([cv2.resize(image\_array, (reshape\_size, reshape\_size))])

Die Pixelwerte des Bildes werden durch 255 geteilt, um sie auf den Bereich von 0 bis 1 zu normalisieren. Anschließend wird das Bild auf die Dimension reshape\_size x reshape\_size skaliert, um sicherzustellen, dass das Bild die richtige Eingabedimension für das Modell hat.

Schritt 7: Vorhersage mit dem Modell

prediction = self.model.predict(image\_array)

Das vorverarbeitete Bild (image\_array) wird durch das Modell (self.model) geleitet, um eine Vorhersage zu erzeugen. Die predict-Methode des Modells gibt eine Wahrscheinlichkeitsverteilung über alle Klassen zurück.

Schritt 8: Setzen von Standard-Klassenbezeichnern, falls keine vorhanden sind

if class\_labels is None:

class\_labels = [f"class\_{i}" for i in range(len(prediction[0]))]

Falls beim Laden der Modell-Metadaten keine Klassennamen (class\_labels) vorgegeben wurden, werden standardmäßige Klassennamen generiert (z. B. class\_0, class\_1, etc.), basierend auf der Anzahl der Klassen im Modell (die gleich der Anzahl der Elemente in der Vorhersage ist).

Schritt 9: Erstellen des Vorhersage-Dictionaries

prediction\_dict = {label: float(prob) for label, prob in zip(class\_labels, prediction[0])}

Ein Dictionary (prediction\_dict) wird erstellt, das für jede Klasse (label) die entsprechende Vorhersagewahrscheinlichkeit (prob) enthält. Die Vorhersagewahrscheinlichkeiten werden als Fließkommazahlen (float) gespeichert.

Schritt 10: Rückgabe der Vorhersage

return prediction\_dict

Das Vorhersage-Dictionary wird zurückgegeben, das die Klassennamen und ihre zugehörigen Wahrscheinlichkeiten enthält.

Schritt 11: Fehlerbehandlung (Exception-Block)

except Exception as e:

raise ValueError(f"Error during prediction: {str(e)}")

Falls während des Vorhersageprozesses ein Fehler auftritt, wird dieser im except-Block abgefangen. Der Fehler wird ausgegeben und als ValueError mit einer entsprechenden Fehlermeldung erneut ausgelöst, um dem Benutzer den Fehler mitzuteilen.

get\_classification\_report(self, dataset, reshape\_size)

Die Methode get\_classification\_report wird verwendet, um die Leistung des Modells zu evaluieren, indem ein Klassifikationsbericht auf einem Testdatensatz erstellt wird. Sie berechnet die Metriken wie Präzision, Recall und F1-Score für das Modell. Die Methode geht durch mehrere Schritte, die im Folgenden detailliert beschrieben werden:

Schritt 1: Definition der Methode get\_classification\_report

def get\_classification\_report(self, dataset, reshape\_size):

Die Methode get\_classification\_report nimmt zwei Eingabewerte entgegen:

* dataset: Ein Tupel (x, y), wobei x die Eingabedaten (Bilder) und y die zugehörigen Zielwerte (Labels) sind.
* reshape\_size: Die Zielgröße, auf die jedes Bild skaliert werden muss.

Schritt 2: Öffnen eines try-Blocks

try:

Ein try-Block wird geöffnet, um Fehler während der Auswertung abzufangen.

Schritt 3: Aufteilen des Datensatzes in Trainings- und Testdaten

x, y = dataset

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

Der Datensatz wird entpackt, und die Daten (x) sowie die Zielwerte (y) werden in Trainings- und Testdaten aufgeteilt. Der Testdatensatz beträgt 20% des gesamten Datensatzes, der Trainingsdatensatz 80%.

Schritt 4: Kodierung der Zielwerte

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(y\_test)

Die Zielwerte im Testdatensatz (y\_test) werden mit LabelEncoder in numerische Werte (y\_encoded) umgewandelt, da das Modell mit numerischen Labels arbeitet.

Schritt 5: Normalisierung und Resizing der Testbilder

x\_test = x\_test / 255.0

x\_test = np.array([cv2.resize(img, (reshape\_size, reshape\_size)) for img in x\_test])

x\_test = np.expand\_dims(x\_test, axis=-1)

Die Testbilder (x\_test) werden normalisiert, indem die Pixelwerte durch 255 geteilt werden, sodass sie in den Bereich [0, 1] fallen. Dann wird jedes Bild auf die Zielgröße reshape\_size skaliert. Anschließend wird eine zusätzliche Dimension hinzugefügt, um die Eingabeform des Modells zu entsprechen (z. B. für Graustufenbilder).

Schritt 6: Durchführung der Vorhersage mit dem Modell

y\_pred = self.model.predict(x\_test, batch\_size=16)

Die normalisierten und skalierten Testbilder (x\_test) werden durch das Modell geleitet, um Vorhersagen zu erhalten. Die Vorhersagen werden als Wahrscheinlichkeiten für jede Klasse zurückgegeben.

Schritt 7: Umwandlung der Vorhersagen in Klassennamen

if len(y\_pred.shape) > 1 and y\_pred.shape[1] > 1:

y\_pred\_classes = np.argmax(y\_pred, axis=1)

else:

y\_pred\_classes = (y\_pred > 0.5).astype("int32")

Falls die Vorhersage mehrdimensional ist und mehr als eine Klasse vorliegt, wird für jede Vorhersage die Klasse mit der höchsten Wahrscheinlichkeit ausgewählt (via np.argmax). Wenn es nur eine Klasse gibt (z. B. binäre Klassifikation), wird die Vorhersage als 0 oder 1 basierend auf einem Schwellenwert von 0.5 interpretiert.

Schritt 8: Berechnung des Klassifikationsberichts

report = classification\_report(y\_encoded, y\_pred\_classes, zero\_division=0)

Der Klassifikationsbericht wird mit der Funktion classification\_report aus sklearn.metrics berechnet. Er enthält Metriken wie Präzision, Recall, F1-Score und Accuracy für jede Klasse. Der Parameter zero\_division=0 sorgt dafür, dass keine Division durch Null erfolgt, falls eine Klasse keine positiven Vorhersagen hat.

Schritt 9: Rückgabe des Klassifikationsberichts

return True, report

Wenn alles erfolgreich war, wird True und der berechnete Klassifikationsbericht (report) zurückgegeben.

Schritt 10: Fehlerbehandlung (Exception-Block)

except Exception as e:

error = f"Error during model evaluation: {str(e)}"

return False, error

Falls während der Auswertung ein Fehler auftritt, wird dieser im except-Block abgefangen. Die Fehlermeldung wird formatiert und zusammen mit False zurückgegeben, um anzuzeigen, dass die Evaluierung fehlgeschlagen ist.

##### Modellinstanzen

Theorie

Aufbau und Funktionsweise

Anwendungen

Verlässlichkeit und Performance

Imports

training\_\_sklearn.py

### Modelltrainer

#### Basisklasse

#### Implementierung

### API

## Jonas Maier (Datenbank, Backend)

*Kapitel verfasst von Jonas Maier*

## Jonas Bogensberger (Frontend)

*Kapitel verfasst von Jonas Bogensberger*

# Installation / Software Deployment

*Kapitel verfasst von Daniel Jessner*

Die erfolgreiche Nutzung des Systems setzt das Starten aller drei Hauptkomponenten voraus: das **Frontend**, das **Backend** und das **AI-Gateway**. Erst wenn diese Dienste aktiv sind und miteinander kommunizieren können, ist die vollständige Funktionalität gewährleistet.

## Manuelle Ausführung der Komponenten

Der aktuelle Deployment-Ansatz erfordert, dass die drei Systemkomponenten separat gestartet werden:

1. **Backend**: Startet den Server, der für die Verarbeitung von Anfragen sowie die Kommunikation mit der Datenbank verantwortlich ist.
2. **AI-Gateway**: Dient als Schnittstelle zwischen dem Backend und den KI-gestützten Analysemodellen.
3. **Frontend**: Die Benutzeroberfläche, über die Nutzer mit dem System interagieren können.

Jede dieser Komponenten kann unabhängig voneinander ausgeführt werden, solange die notwendigen Netzwerkverbindungen bestehen.

## Automatisierung mit Docker Compose oder Kubernetes

Zur Vereinfachung des Deployments könnte eine Containerisierungslösung wie **Docker Compose** oder **Kubernetes** genutzt werden. Dadurch ließen sich alle drei Komponenten in einer einheitlichen Umgebung orchestrieren und mit nur einem Befehl starten. Dies würde insbesondere die Bereitstellung auf Servern oder Cloud-Plattformen erleichtern.

Ein möglicher **Docker Compose**-Ansatz könnte folgende Struktur haben:

* Ein Container für das Backend
* Ein Container für das AI-Gateway
* Ein Container für das Frontend
* Optional: Ein Container für die Datenbank

Durch diese Struktur würden Abhängigkeiten automatisch konfiguriert und das gesamte System mit einem einzigen Befehl gestartet werden können.

## Nutzung durch den Endnutzer

Für den Endnutzer bleibt der Installationsprozess jedoch weitgehend unverändert. Die einzige Voraussetzung ist die **Installation der Android-App**, die als Schnittstelle zum System dient. Solange das AI-Gateway und der Backend-Server erreichbar sind, kann der Nutzer problemlos auf die Funktionen der Anwendung zugreifen.

In zukünftigen Erweiterungen könnte das Deployment weiter optimiert werden, um eine noch benutzerfreundlichere Bereitstellung und Verwaltung der Systemkomponenten zu ermöglichen.

# Projektabschluss

*Kapitel verfasst von Daniel Jessner*

## Projektzusammenfassung

Was lief gut?

* **Teamarbeit und Engagement:** Das Team zeigte ein hohes Maß an Engagement und Zusammenhalt. Die Zusammenarbeit unter den Teammitgliedern war konstruktiv und unterstützend.
* **Zielerreichung:** Trotz der Herausforderungen wurden die Hauptziele des Projekts grundlegend erreicht.

Was lief schlecht?

* **Kommunikation:** Die Kommunikation war oftmals nicht ausreichend. Es gab Missverständnisse und Verzögerungen, weil Informationen nicht rechtzeitig oder nicht klar weitergegeben wurden.
* **Zeitliche Unterschätzung:** Der Zeitaufwand für bestimmte Projektphasen wurde unterschätzt. Dies führte zu Stress in den Endphasen des Projekts.
* **Ressourcenplanung:** Die Ressourcenplanung war nicht immer optimal. Es gab Phasen, in denen wichtige Ressourcen nicht verfügbar waren, was zu Verzögerungen führte.

Welche Erkenntnisse wurden während der Durchführung des Projektes gewonnen?

* **Bedeutung klarer Kommunikation:** Eine klare und regelmäßige Kommunikation ist entscheidend für den Projekterfolg. Alle Teammitglieder müssen über den gleichen Informationsstand verfügen, um effizient arbeiten zu können.
* **Realistische Zeitplanung:** Es ist wichtig, den Zeitaufwand realistisch einzuschätzen und Pufferzeiten einzuplanen, um unerwartete Verzögerungen auffangen zu können.
* **Flexibilität und Anpassungsfähigkeit:** Flexibilität im Umgang mit Änderungen und Anpassungsfähigkeit an unvorhergesehene Herausforderungen sind essenziell für den Projekterfolg.

Was würde man nun anders machen bzw. wieder gleich machen?

* **Verbesserung der Kommunikation:** Künftige Projekte würden von einer klareren Kommunikationsstrategie profitieren. Regelmäßige „Meetings“ und klare Dokumentationsrichtlinien sollen eingeführt werden, nicht nur für die Stakeholder.
* **Realistischere Zeitplanung:** Eine gründlichere Analyse und Planung des Zeitaufwands für jede Projektphase würde sicherstellen, dass alle Aufgaben in der vorgegebenen Zeit erledigt werden können.
* **Erfolgreiche Aspekte beibehalten:** Der hohe Teamgeist und das Engagement sollen weiterhin gefördert werden. Team-building-Aktivitäten und Anerkennung für gute Leistungen würden beibehalten und weiter ausgebaut werden.
* **Optimierung der Ressourcenplanung:** Eine sorgfältigere und flexiblere Ressourcenplanung würde sicherstellen, dass alle notwendigen Ressourcen zur richtigen Zeit verfügbar sind.

## Attachments

# Literaturverzeichnis

*Kapitel verfasst von Daniel Jessner*

# Abbildungsverzeichnis

*Kapitel verfasst von Daniel Jessner*