**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 nur 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), Mobile-Anwendung (Registrierungssystem), Server (Datenbank für Nutzerdaten), KI-Komponente (Gateway)

**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 (über das Gateway) 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:**  
Mobile-Anwendung, Datenbank-Server, Gateway, Internetverbindung

##### Anmeldung eines bestehenden Benutzers

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

**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 (über das Gateway) 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.

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

**Systemgrenzen:**  
Mobile-Anwendung, Datenbank-Server, Gateway, Internetverbindung

##### Einblick in vergangene Analysen

**Akteure:**  
Benutzer, Mobile-Anwendung, Server (Datenbank mit analysierten Bildern), KI-Komponente (Gateway)

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

**Kurzbeschreibung:**  
Der Benutzer kann eine Liste aller 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 (über das Gateway), 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:**  
Mobile-Anwendung, Datenbank-Server, Gateway, Internetverbindung

##### Bild analysieren lassen

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

**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. Die Anwendung sendet das Bild an das Gateway zur Analyse mit dem gewählten Modell.
8. Falls das gewählte Modell nicht verfügbar ist (z. B. nicht trainiert), erhält der Benutzer eine Mitteilung und kann ein alternatives Modell auswählen.
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 KI-Komponente verfügt über mindestens ein trainiertes KI-Modell.

Nachbedingung: Die Analyseergebnisse werden auf dem Bildschirm angezeigt.

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

##### Automatische Bildzuschnitt-Funktion

**Akteure:**  
Benutzer, Mobile-Anwendung (Analyse-Modul & Kamera), KI-Komponente (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 das System 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:**  
Mobile-Anwendung, Kamera, Server, Gateway, Internetverbindung

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

**Akteure:**  
Benutzer, Mobile-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:**  
Mobile-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, Mobile-Anwendung (Trainingsmodul), KI-Komponente, Datenbank-Server

**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 (z.B. 224\*224 Pixel)
   * **Epochen:** Anzahl der Trainingsdurchläufe über den Datensatz
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 (mit gegebenen Parametern) nicht trainierbare Modelle erstellt.

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

##### Classifier-Reports abrufen

**Akteure:**  
Administrator, Mobile-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 Trainieren 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.

**Beschreibung alternativer Schritte:**

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

## Nicht Funktionale Anforderungen

Dieses Kapitel enthält die Anforderungen, die an die Anwendung gestellt werden, welche nicht die direkte 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 Mobile-App. 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 seinen 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:

##### Maschinelles Lernen

**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.

##### Deep Learning

**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:

##### 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. Dieser Punkt ist für dieses Projekt jedoch nicht von Nöten.

##### 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.

##### FastAPI

**FastAPI** ist ein modernes, leistungsstarkes Python-Framework zur Entwicklung von APIs. Es überzeugt durch seine hohe Geschwindigkeit (dank ASGI und Starlette), automatische, interaktive Dokumentation (Swagger UI & ReDoc), sowie einfache Bedienbarkeit durch Python-Typannotationen und Pydantic-Modelle. Asynchrone Verarbeitung ermöglicht die effiziente Handhabung vieler gleichzeitiger Anfragen. FastAPI bietet umfassende Sicherheitsfunktionen (OAuth2, JWT, CORS etc.), unterstützt moderne Python-Funktionen, ist leichtgewichtig, skalierbar und offen für Erweiterungen. Die gute Dokumentation und breite Community erleichtern den Einstieg und die Integration mit Tools wie SQLAlchemy, Celery oder Redis.

#### Installation der KI-Bilbiotheken

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

Folgender Befehl installiert die Scikit-Learn-Bilbiothek:

pip install 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**.

##### Pytorch

Die PyTorch-Bibliothek kann mit oder ohne GPU-Unterstützung installiert werden:

###### Ohne GPU-Unterstützung (CPU-Only)

Folgender Befehl installiert die PyTorch-Bibliothek ohne GPU-Unterstützung:

pip install torch torchvision

Was wird installiert?

**PyTorch (torch)** 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).

**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?

**Torch:**

* 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.

**Torchvision:**

* 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.

###### Mit GPU-Unterstützung

Folgender Befehl installiert die PyTorch-Bibliothek mit GPU-Unterstützung:

pip install torch torchvision cudatoolkit=11.3

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.

##### TensorFlow

Ähnlich wie bei PyTorch kann auch die TensorFlow-Bibliothek mit oder ohne GPU-Unterstützung installiert werden:

###### Ohne GPU-Unterstützung (CPU-Only)

Folgender Befehl installiert die TensorFlow-Bibliothek ohne GPU-Unterstützung:

pip install 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).

###### Mit GPU-Unterstützung

Folgender Befehl installiert die TensorFlow-Bibliothek mit GPU-Unterstützung:

pip install 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:

pip install fastapi

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:

pip install uvicorn

##### 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.

##### Imports

Für dieses Testexperiment sind die unten extra angeführten Importe sowie die eigentlichen Modelle aus dem **sklearn**-Modul an sich notwendig:

import matplotlib.pyplot as plt

import numpy as np

from matplotlib.colors import ListedColormap

from sklearn.datasets import make\_circles, make\_classification, make\_moons

from sklearn.gaussian\_process.kernels import RBF

from sklearn.inspection import DecisionBoundaryDisplay

from sklearn.model\_selection import train\_test\_split

from sklearn.pipeline import make\_pipeline

##### classifier\_comparison.py

Folglich werden die einzelnen Schritte, die zur Durchführung dieses Vergleichs gemacht werden müssen, veranschaulicht:

**1. Classifier deklarieren**

Zuerst wird eine Liste von Klassifikatoren und eine mit ihren entsprechenden Namen erstellt. Die Klassifikatoren werden später verwendet, um das Training mit einem Datensatz durchzuführen:

names = [  
 "Nearest Neighbors",  
 "Linear SVM",  
 "RBF SVM",  
 "Gaussian Process",  
 "Decision Tree",  
 "Random Forest",  
 "Neural Net",  
 "AdaBoost",  
 "Naive Bayes",  
 "QDA",  
]  
  
classifiers = [  
 KNeighborsClassifier(3),  
 SVC(kernel="linear", C=0.025, random\_state=42),  
 SVC(gamma=2, C=1, random\_state=42),  
 GaussianProcessClassifier(1.0 \* RBF(1.0), random\_state=42),  
 DecisionTreeClassifier(max\_depth=5, random\_state=42),  
 RandomForestClassifier(  
 max\_depth=5, n\_estimators=10, max\_features=1, random\_state=42 ),  
 MLPClassifier(alpha=1, max\_iter=1000, random\_state=42),  
 AdaBoostClassifier(random\_state=42),  
 GaussianNB(),  
 QuadraticDiscriminantAnalysis(),  
]

* **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 **6.1.3 Modelle** mit ihren Parametern genauer beschrieben.

**2. Datensätze generieren**

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

X, y = make\_classification(  
 n\_features=2, n\_redundant=0, n\_informative=2, random\_state=1, n\_clusters\_per\_class=1  
)

rng = np.random.RandomState(2)  
X += 2 \* rng.uniform(size=X.shape)  
linearly\_separable = (X, y)  
  
datasets = [  
 make\_moons(noise=0.3, random\_state=0),  
 make\_circles(noise=0.2, factor=0.5, random\_state=1),  
 linearly\_separable,  
]

* **make\_classification** erzeugt einen Datensatz für eine Klassifikationsaufgabe mit 2 informativen Features (= Labels).
* **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:

figure = plt.figure(figsize=(27, 9))  
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 dargestellt werden:

for ds\_cnt, ds in enumerate(datasets):  
 X, y = ds X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=0.4, random\_state=42 )  
  
 x\_min, x\_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5 y\_min, y\_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5  
  
 cm = plt.cm.RdBu cm\_bright = ListedColormap(["#FF0000", "#0000FF"])  
 ax = plt.subplot(len(datasets), len(classifiers) + 1, i)  
 if ds\_cnt == 0:  
 ax.set\_title("Input data")  
  
 ax.scatter(X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap=cm\_bright, edgecolors="k")  
  
 ax.scatter(  
 X\_test[:, 0], X\_test[:, 1], c=y\_test, cmap=cm\_bright, alpha=0.6, edgecolors="k" )  
 ax.set\_xlim(x\_min, x\_max)  
 ax.set\_ylim(y\_min, y\_max)  
 ax.set\_xticks(())  
 ax.set\_yticks(())  
 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:

for name, clf in zip(names, classifiers):  
 ax = plt.subplot(len(datasets), len(classifiers) + 1, i)  
  
 clf = make\_pipeline(StandardScaler(), clf)  
 clf.fit(X\_train, y\_train)  
 score = clf.score(X\_test, y\_test)  
 DecisionBoundaryDisplay.from\_estimator(  
 clf, X, cmap=cm, alpha=0.8, ax=ax, eps=0.5 )  
  
 ax.scatter(  
 X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap=cm\_bright, edgecolors="k" )  
  
 ax.scatter(  
 X\_test[:, 0],  
 X\_test[:, 1],  
 c=y\_test,  
 cmap=cm\_bright,  
 edgecolors="k",  
 alpha=0.6,  
 )  
  
 ax.set\_xlim(x\_min, x\_max)  
 ax.set\_ylim(y\_min, y\_max)  
 ax.set\_xticks(())  
 ax.set\_yticks(())  
 if ds\_cnt == 0:  
 ax.set\_title(name)  
 ax.text(  
 x\_max - 0.3,  
 y\_min + 0.3,  
 ("%.2f" % score).lstrip("0"),  
 size=15,  
 horizontalalignment="right",  
 )  
 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**

plt.tight\_layout()  
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:

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

KI-generierte Inhalte können fehlerhaft sein.**Das Testexperiment zeigt als Ergebnis folgende Visualisierung:

Das gezeigte Bild stellt eine Visualisierung der verschiedenen Klassifikationsalgorithmen dar, die auf drei verschiedene Datensätze angewendet wurden (zwecks Platzgründen nur die Hälfte der getesteten Klassifizierer angezeigt). Jeder Algorithmus versucht, eine Entscheidungsgrenze zu lernen, um die beiden Klassen (rot und blau) korrekt zu trennen.

###### . Aufbau der Visualisierung

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.

###### Analyse der Algorithmen und Entscheidungsgrenzen

Als Abschluss des Testexperiments wird noch kurz auf die Leistung der gezeigten Klassifikationsalgorithmen eingegangen.

###### Nearest Neighbors

* **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

* **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

* **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

* **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

* **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.

##### 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 und dem Use-Case, also für was man ein Modell verwenden will, 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.

#### Redesign

(Von dem zu Befüllen, der es nötig hat.)

### 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

Die Module bilden die grundlegenden Bausteine der KI-Komponente und sind für die verschiedenen Verarbeitungsschritte innerhalb des Systems verantwortlich. Sie setzen sich aus zwei wesentlichen Aspekten zusammen:

* **ImageProcessor.py**  
  Verantwortlich für die Bildverarbeitung, stellt dieses Modul sicher, dass eingehende Bilddaten korrekt erfasst und für die weiteren Verarbeitungsprozesse vorbereitet werden. Außerdem ist dieses Modul zuständig, die Daten aus der Datenbank zu lesen.
* **ModelProcessor.py**  
  Dieses Modul ist für die Anpassung und Verarbeitung der Modelle verantwortlich. Es sorgt dafür, dass die zugrunde liegenden Algorithmen und Modelle optimal an die Struktur der Eingabedaten angepasst werden. Gleichzeitig stellt es sicher, dass die Daten in die passende Form gebracht werden, um mit den Modellen effektiv arbeiten zu können. Auf diese Weise gewährleistet das Modul, dass Vorhersagen oder Klassifikationen korrekt und effizient durchgeführt werden, indem es eine nahtlose Integration von Daten und Modellen ermöglicht.

#### ImageProcessor

Der ImageProcessor bietet grundlegend zwei Funktionen:

* Daten aus der Datenbank lesen
* Bilder für den Eingang in die Modelle vorverarbeiten

##### loadImagesAs1DVectorFromAPI(api\_uri, reshape\_size)

Die Funktion loadImagesAs1DVectorFromAPI lädt die medizinischen Bilddaten der Datenbank-API, verarbeitet diese Bilder (konvertiert sie in Graustufen, skaliert und bringt sie auf eine einheitliche Größe) und gibt sie als 1D-Vektoren zurück, die für maschinelles Lernen oder andere Analysezwecke verwendet werden können. Jedes Bild wird zusammen mit einem zugehörigen Label verarbeitet, das als Diagnose bezeichnet wird.

Schritt 1: Abruf der Daten von der API

data = \_\_fetchDataFromAPI(api\_uri)

 **Beschreibung:**  
Zuerst wird die Funktion \_\_fetchDataFromAPI aufgerufen, um die Daten von der API zu laden. Dabei wird die URL (api\_uri) an die Funktion übergeben, die die Verbindung zur API herstellt und die Daten (z. B. Bilder und zugehörige Labels) zurückgibt.

 **Erklärung:**  
Diese Zeile erwartet eine Liste von Objekten, wobei jedes Objekt ein Bild und ein Label enthält.

Schritt 2: Überprüfung, ob Daten vorhanden sind

if not data:

raise ValueError("Could not fetch from database")

 **Beschreibung:**  
Es wird überprüft, ob die Daten erfolgreich abgerufen wurden. Falls keine Daten abgerufen werden konnten (z. B. wegen eines API-Fehlers), wird eine ValueError-Ausnahme ausgelöst.

 **Erklärung:**  
Hier wird sicherstellt, dass der weitere Code nur ausgeführt wird, wenn tatsächlich Daten verfügbar sind. Andernfalls gibt es eine klare Fehlermeldung.

Schritt 3: Initialisierung der Listen für Merkmale und Labels

features\_list = []

labels\_list = []

 **Beschreibung:**  
Zwei leere Listen werden initialisiert: features\_list für die Merkmale der Bilder (nach der Umwandlung in 1D-Vektoren) und labels\_list für die zugehörigen Labels (Diagnosen).

 **Erklärung:**  
Diese Listen werden später verwendet, um die verarbeiteten Bildmerkmale und die entsprechenden Labels zu speichern.

Schritt 4: Schleife über alle Datenpunkte

for item in data:

 **Beschreibung:**  
Eine Schleife wird gestartet, die durch jedes Element in den abgerufenen Daten (data) iteriert. Jedes Element (Bild) enthält ein Bild und ein zugehöriges Label.

 **Erklärung:**  
Jedes Element (Bild) wird durchlaufen, damit es verarbeitet und in den 1D-Vektor umgewandelt werden kann.

Schritt 5: Laden des Bildes und des Labels

picture = np.array(item["picture"], dtype=np.uint8)

label = item["diagnosis"]

 **Beschreibung:**  
Hier wird das Bild aus dem item-Dictionary extrahiert und als numpy-Array im uint8-Datenformat gespeichert. Außerdem wird das Label des Bildes (die Diagnose) extrahiert.

 **Erklärung:**  
Das Bild wird in ein numpy-Array umgewandelt, um es später zu bearbeiten. Das Label wird extrahiert, um es in labels\_list zu speichern.

Schritt 6: Umwandlung in Graustufen, falls erforderlich

if len(picture.shape) == 2: # Already grayscale

gray\_image = picture

elif len(picture.shape) == 3: # Convert only if it's RGB or RGBA

gray\_image = cv2.cvtColor(picture, cv2.COLOR\_BGR2GRAY)

else:

raise ValueError(f"Unexpected image shape: {picture.shape}")

 **Beschreibung:**  
Abhängig von der Form des Bildes wird überprüft, ob das Bild bereits in Graustufen vorliegt (2D). Falls das Bild in Farbe vorliegt (3D, z. B. RGB), wird es mit der OpenCV-Funktion cv2.cvtColor() in Graustufen umgewandelt. Falls das Bild eine unerwartete Form hat, wird ein Fehler ausgelöst.

 **Erklärung:**  
Dieser Schritt stellt sicher, dass alle Bilder in Graustufen vorliegen, was für die meisten Bildverarbeitungs- und maschinellen Lernaufgaben sinnvoll ist.

Schritt 7: Skalierung des Bildes auf die angegebene Größe

resized\_image = cv2.resize(gray\_image, (reshape\_size, reshape\_size))

* **Beschreibung:**  
  Das Bild wird auf die angegebene Größe (reshape\_size) skaliert. Das Bild wird auf eine quadratische Form skaliert (d. h. Höhe = Breite = reshape\_size).
* **Erklärung:**  
  Diese Operation stellt sicher, dass alle Bilder dieselbe Größe haben, was für die weitere Verarbeitung und das Training von Modellen erforderlich ist.

Schritt 8: Umwandlung in einen 1D-Vektor und Skalierung

features = resized\_image.flatten()

features\_scaled = StandardScaler().fit\_transform(features.reshape(-1, 1)).flatten()

 **Beschreibung:**

* Zuerst wird das Bild, das jetzt die gewünschte Größe hat, mit der Methode .flatten() in einen 1D-Vektor umgewandelt.
* Danach wird der Vektor mit StandardScaler() normalisiert, um sicherzustellen, dass alle Pixelwerte in einem standardisierten Bereich liegen. Dies ist wichtig, damit der Lernalgorithmus besser mit den Daten arbeiten kann.

 **Erklärung:**  
Das Umwandeln eines Bildes in einen 1D-Vektor ist ein häufiger Schritt in der Bildverarbeitung, insbesondere wenn das Bild als Eingabe in maschinelle Lernmodelle verwendet werden soll. Die Skalierung sorgt dafür, dass die Pixelwerte innerhalb eines Standardbereichs liegen, was die Konvergenz des Modells beschleunigt.

Schritt 9: Hinzufügen der Merkmale und Labels zur Liste

features\_list.append(features\_scaled)

labels\_list.append(label)

 **Beschreibung:**  
Der skalierte 1D-Vektor des Bildes wird zur Liste features\_list hinzugefügt, und das zugehörige Label wird zur Liste labels\_list hinzugefügt.

 **Erklärung:**  
Am Ende des Schleifenprozesses werden die Merkmale und Labels des Bildes in den Listen gespeichert, um sie später als numpy-Arrays zurückzugeben.

Schritt 10: Rückgabe der Ergebnisse als numpy-Arrays

return np.array(features\_list), np.array(labels\_list)

 **Beschreibung:**  
Die Listen features\_list und labels\_list werden in numpy-Arrays umgewandelt und zurückgegeben. features\_list enthält die 1D-Vektoren der Bilder, und labels\_list enthält die zugehörigen Diagnosen.

 **Erklärung:**  
Am Ende der Funktion werden die verarbeiteten Daten als numpy-Arrays zurückgegeben, die dann für maschinelles Lernen oder weitere Verarbeitung verwendet werden können.

###### \_\_fetchDataFromAPI(api\_uri)

Diese Funktion ruft Daten von einer API ab, die paginierte Antworten liefert. Sie lädt die Daten Seite für Seite und gibt die gesammelten Daten zurück.

Schritt 1: Initialisierung

print('\n# ----------- FETCHING DATA FROM DATABASE ----------- #\n')

all\_data = []

page = 1

 **Beschreibung:**  
Zu Beginn wird eine Nachricht ausgegeben, um anzuzeigen, dass der Abruf von Daten gestartet wird. Außerdem werden zwei Variablen initialisiert:

* all\_data: Eine leere Liste, die später alle abgerufenen Daten speichern wird.
* page: Eine Zählvariable, die die aktuelle Seite der API-Antwort verfolgt (beginnend mit Seite 1).

 **Erklärung:**  
Die Daten werden seitenweise abgerufen, daher wird page verwendet, um den Fortschritt der Anforderung zu verfolgen.

Schritt 2: Abruf der Daten mit einer while-Schleife

try:

while True:

response = requests.get(f"{api\_uri}?page={page}&limit=5")

response.raise\_for\_status()

data = response.json()

 **Beschreibung:**

* **requests.get**: Eine GET-Anfrage wird an die API gestellt, wobei die page-Nummer und ein limit (hier 5) für die Anzahl der Datensätze pro Seite angegeben werden.
* **raise\_for\_status**: Diese Methode prüft, ob die Antwort einen Fehlerstatuscode zurückgibt (z. B. 4xx oder 5xx). Falls ja, wird eine Ausnahme ausgelöst.
* **response.json()**: Wandelt die Antwort in ein Python-Dictionary um.

 **Erklärung:**  
Die while True-Schleife wird so lange ausgeführt, bis alle Seiten abgerufen wurden. Nach jeder Anfrage wird geprüft, ob die Anfrage erfolgreich war und die Antwort in ein JSON-Datenformat umgewandelt.

Schritt 3: Speichern der abgerufenen Daten

# Append the current page's data

all\_data.extend(data.get("docs", []))

 **Beschreibung:**  
Die aktuellen Daten von der Seite (abgerufen aus dem Feld "docs" im JSON) werden zur all\_data-Liste hinzugefügt. Falls das Feld "docs" nicht existiert, wird eine leere Liste verwendet.

 **Erklärung:**  
Der extend-Befehl fügt die Elemente der aktuellen Seite zu der Liste der bereits gesammelten Daten hinzu.

Schritt 4: Prüfen, ob es eine nächste Seite gibt

# Check if there's a next page

if not data.get("hasNextPage", False):

break

 **Beschreibung:**  
Das Feld "hasNextPage" wird überprüft, um zu sehen, ob es noch eine weitere Seite gibt. Falls der Wert False ist, wird die Schleife beendet.

 **Erklärung:**  
Die Schleife wird so lange fortgesetzt, wie eine nächste Seite existiert. Sobald keine weiteren Seiten vorhanden sind, wird die Schleife beendet.

Schritt 5: Wechsel zur nächsten Seite

# Move to the next page

print("Loaded Page ", page)

page += 1

 **Beschreibung:**  
Nachdem die aktuelle Seite erfolgreich geladen wurde, wird der Zähler page um eins erhöht, um zur nächsten Seite zu wechseln. Eine Nachricht wird gedruckt, die anzeigt, dass die Seite erfolgreich geladen wurde.

 **Erklärung:**  
Dies stellt sicher, dass jede Seite nacheinander abgerufen wird, und gibt dem Benutzer eine Rückmeldung über den Fortschritt.

Schritt 6: Abschluss und Rückgabe der gesammelten Daten

print('Received data with length of: ', all\_data.\_\_len\_\_())

return all\_data

 **Beschreibung:**  
Sobald alle Seiten abgerufen wurden, wird die Länge der gesammelten Daten (all\_data.\_\_len\_\_()) ausgegeben, und die vollständige Liste der gesammelten Daten wird zurückgegeben.

 **Erklärung:**  
Der Benutzer wird darüber informiert, wie viele Datensätze insgesamt abgerufen wurden. Anschließend werden die gesammelten Daten an den Aufrufer zurückgegeben.

Schritt 7: Fehlerbehandlung

except requests.exceptions.HTTPError as errh:

print(f"HTTP-Error: {errh}")

except requests.exceptions.ConnectionError as errc:

print(f"Connection-Error: {errc}")

except requests.exceptions.Timeout as errt:

print(f"Timeout-Error: {errt}")

except requests.exceptions.RequestException as err:

print(f"Error: {err}")

 **Beschreibung:**  
Diese except-Blöcke fangen verschiedene Arten von Fehlern ab, die während des Abrufs von Daten auftreten können:

* **HTTPError**: Fehler im Zusammenhang mit dem HTTP-Statuscode (z. B. 404, 500).
* **ConnectionError**: Netzwerk- oder Verbindungsfehler.
* **Timeout**: Fehler, wenn die Anfrage zu lange dauert.
* **RequestException**: Ein allgemeiner Fehler für alle anderen Ausnahmefälle.

 **Erklärung:**  
Durch die Fehlerbehandlung wird sichergestellt, dass die Anwendung nicht abstürzt, wenn Probleme beim Abrufen der Daten auftreten. Stattdessen wird eine klare Fehlermeldung ausgegeben.

Schritt 8: Rückgabe bei Fehlern

return None

 **Beschreibung:**  
Falls ein Fehler auftritt, wird None zurückgegeben, um anzuzeigen, dass keine Daten abgerufen werden konnten.

 **Erklärung:**  
Dies gibt dem Aufrufer die Möglichkeit, auf das Fehlen von Daten zu reagieren, wenn ein Fehler aufgetreten ist.

##### preprocess\_image(image, reshape\_size)

Diese Funktion nimmt ein Bild (im PIL.Image-Format) und wandelt es in ein geeignetes Format um, indem es es in Graustufen konvertiert, auf eine bestimmte Größe skaliert und in einen 1D-Vektor umwandelt.

Schritt 1: Umwandlung des Bildes in Graustufen

image = image.convert("L")

 **Beschreibung:**  
Der erste Schritt besteht darin, das Bild in Graustufen zu konvertieren. Die Methode convert("L") von PIL.Image wandelt das Bild in den Graustufenmodus um, wobei jeder Pixelwert zwischen 0 (schwarz) und 255 (weiß) liegt.

 **Erklärung:**  
Graustufenbilder sind oft einfacher zu verarbeiten und benötigen weniger Rechenleistung als Farbbilder (RGB), weshalb sie in vielen Bildverarbeitungs- und maschinellen Lernaufgaben bevorzugt werden.

Schritt 2: Skalierung des Bildes auf eine einheitliche Größe

image = image.resize((reshape\_size, reshape\_size), resample=PIL.Image.BICUBIC)

 **Beschreibung:**  
Das Bild wird auf die angegebenen Dimensionen (reshape\_size, reshape\_size) skaliert. Die Methode resize() von PIL.Image wird verwendet, um die Bildgröße zu ändern. Der Parameter resample=PIL.Image.BICUBIC gibt an, dass beim Skalieren eine bikubische Interpolation verwendet wird, die hochwertige Ergebnisse liefert.

 **Erklärung:**  
Die Bildgröße wird angepasst, um sicherzustellen, dass alle Bilder die gleiche Größe haben, was für viele maschinelle Lernmodelle erforderlich ist. Bikubische Interpolation wird häufig verwendet, da sie bei der Skalierung von Bildern eine glattere Ausgabe liefert.

Schritt 3: Umwandlung des Bildes in ein NumPy-Array

image\_array = np.array(image)

 **Beschreibung:**  
Das PIL.Image-Objekt wird in ein numpy-Array umgewandelt. Dies ist notwendig, weil viele Bildverarbeitungsbibliotheken (einschließlich OpenCV und scikit-learn) mit NumPy-Arrays arbeiten. Die Umwandlung ermöglicht es, mit den Pixelwerten direkt zu arbeiten.

 **Erklärung:**  
Das NumPy-Array repräsentiert das Bild als Matrix, in der jeder Wert einen Pixelwert im Bild darstellt. Nach der Umwandlung kann man auf jedes Pixel im Bild zugreifen und es weiterverarbeiten.

Schritt 4: Umwandlung in einen 1D-Vektor

image\_array = image\_array.reshape(1, -1)

 **Beschreibung:**  
Das image\_array wird in einen 1D-Vektor umgewandelt. Der Befehl reshape(1, -1) sorgt dafür, dass das Bild als 1D-Vektor mit einer Länge von reshape\_size \* reshape\_size gespeichert wird. Dabei wird die ursprüngliche 2D-Struktur (Höhe x Breite) in einen flachen Vektor umgewandelt.

 **Erklärung:**  
In vielen Anwendungen (z. B. maschinelles Lernen) werden Bilder als Vektoren anstelle von Matrizen verarbeitet. Dieser Schritt ist erforderlich, um das Bild als Eingabe in viele maschinelle Lernalgorithmen zu überführen.

Schritt 5: Rückgabe des vorverarbeiteten Bildes

return image\_array

 **Beschreibung:**  
Am Ende wird das vorverarbeitete Bild als 1D-Numpy-Array zurückgegeben.

 **Erklärung:**  
Das Bild im geeigneten Format (als 1D-Vektor) wird an den Aufrufer der Funktion zurückgegeben, sodass es für weitere Verarbeitung, z. B. als Eingabe in ein Modell, verwendet werden kann.

#### Model Processor

Der ModelProcessor bietet grundlegend vier Funktionen:

* Anpassen der Eingabekanäle für ein PyTorch-Modell
* Anpassen der Eingabekanäle für einen PyTorch-Tensor
* Anpassen der Ausgabeschicht für ein PyTorch-Modell
* Anpassen der Ausgabeschicht für ein Tensorflow-Modell

##### adjust\_input\_channels\_pytorch(model)

Diese Funktion überprüft den Typ des übergebenen Modells und ändert die Eingabekanäle der ersten Convolutional-Schicht so, dass sie nur einen Kanal (Graustufen) erwartet. Sie wird für gängige vorgefertigte Modelle wie ResNet, VGG, DenseNet, etc. verwendet.

Schritt 1: Initialisierung der Variablen

modified = False

 **Beschreibung:**  
Zu Beginn wird eine Variable modified auf False gesetzt. Diese Variable wird später verwendet, um zu überprüfen, ob das Modell geändert wurde.

 **Erklärung:**  
modified gibt an, ob eine der Bedingungen in der Funktion zutrifft und das Modell tatsächlich modifiziert wurde. Falls keine Änderungen vorgenommen werden, bleibt modified auf False.

Schritt 2: Prüfung auf ResNet, EfficientNet, VisionTransformer (ViT)

if hasattr(model, "conv1") and isinstance(model.conv1, nn.Conv2d):

model.conv1 = nn.Conv2d(1, model.conv1.out\_channels, kernel\_size=model.conv1.kernel\_size,

stride=model.conv1.stride, padding=model.conv1.padding, bias=False)

modified = True

 **Beschreibung:**  
Wenn das Modell ein Attribut conv1 hat und conv1 eine Convolutional Layer (Instanz von nn.Conv2d) ist, dann wird die erste Convolutional-Schicht (conv1) so geändert, dass sie nur 1 Eingangskanal anstelle von 3 erwartet. Die Anzahl der Ausgangskanäle bleibt unverändert.

 **Erklärung:**

* conv1: Die erste Convolutional-Schicht eines Modells wie ResNet oder EfficientNet.
* Es wird eine neue nn.Conv2d-Schicht erstellt, bei der die Eingabekanäle auf 1 gesetzt werden (für Graustufenbilder), während die anderen Parameter wie kernel\_size, stride und padding aus der bestehenden Schicht übernommen werden.

 **Änderung:**  
Die Eingabekanäle der ersten Convolutional-Schicht werden von 3 auf 1 geändert.

Schritt 3: Prüfung auf DenseNet, MobileNet, SqueezeNet

elif hasattr(model, "features") and hasattr(model.features, "conv0") and isinstance(model.features.conv0, nn.Conv2d):

model.features.conv0 = nn.Conv2d(1, model.features.conv0.out\_channels,

kernel\_size=model.features.conv0.kernel\_size,

stride=model.features.conv0.stride, padding=model.features.conv0.padding,

bias=False)

modified = True

 **Beschreibung:**  
Wenn das Modell die features-Eigenschaft hat und die conv0-Schicht eine Convolutional Layer ist, wird die erste Convolutional-Schicht (conv0) so geändert, dass sie nur 1 Eingangskanal anstelle von 3 erwartet.

 **Erklärung:**  
Das gleiche Prinzip wie bei conv1 gilt auch für Modelle wie DenseNet oder MobileNet, bei denen die erste Convolutional-Schicht unter der features-Eigenschaft zu finden ist. Die Eingabekanäle werden auf 1 gesetzt.

Schritt 4: Prüfung auf AlexNet, VGG

elif hasattr(model, "features") and isinstance(model.features[0], nn.Conv2d):

model.features[0] = nn.Conv2d(1, model.features[0].out\_channels,

kernel\_size=model.features[0].kernel\_size,

stride=model.features[0].stride, padding=model.features[0].padding,

bias=False)

modified = True

 **Beschreibung:**  
Wenn das Modell eine features-Eigenschaft hat und das erste Element in features eine Convolutional Layer ist, wird diese so geändert, dass sie nur 1 Eingangskanal erwartet.

 **Erklärung:**  
AlexNet und VGG haben ihre Convolutional-Schichten unter der features-Eigenschaft gespeichert, daher wird hier das erste Element (features[0]) überprüft und angepasst.

Schritt 5: Prüfung auf Inception v3

elif hasattr(model, "Conv2d\_1a\_3x3") and isinstance(model.Conv2d\_1a\_3x3.conv, nn.Conv2d):

model.Conv2d\_1a\_3x3.conv = nn.Conv2d(1, model.Conv2d\_1a\_3x3.conv.out\_channels,

kernel\_size=model.Conv2d\_1a\_3x3.conv.kernel\_size,

stride=model.Conv2d\_1a\_3x3.conv.stride,

padding=model.Conv2d\_1a\_3x3.conv.padding, bias=False)

modified = True

 **Beschreibung:**  
Wenn das Modell ein Attribut Conv2d\_1a\_3x3 hat und Conv2d\_1a\_3x3.conv eine Convolutional Layer ist, wird diese so geändert, dass sie nur 1 Eingangskanal erwartet.

 **Erklärung:**  
Inception v3 verwendet eine spezielle Schicht namens Conv2d\_1a\_3x3, daher wird diese Schicht auf 1 Eingangskanal umgestellt.

Schritt 6: Prüfung auf ConvNeXt

elif hasattr(model, "stem") and isinstance(model.stem[0], nn.Conv2d):

model.stem[0] = nn.Conv2d(1, model.stem[0].out\_channels,

kernel\_size=model.stem[0].kernel\_size,

stride=model.stem[0].stride, padding=model.stem[0].padding,

bias=False)

modified = True

 **Beschreibung:**  
Wenn das Modell ein Attribut stem hat und stem[0] eine Convolutional Layer ist, wird diese so geändert, dass sie nur 1 Eingangskanal erwartet.

 **Erklärung:**  
Bei ConvNeXt befindet sich die erste Convolutional-Schicht unter der stem-Eigenschaft. Auch hier wird die Eingabekanäle auf 1 geändert.

Schritt 7: Fehlerbehandlung

except Exception as e:

print(f"⚠️ Model modification failed: {e}")

 **Beschreibung:**  
Falls bei der Modifikation des Modells ein Fehler auftritt, wird eine Ausnahme abgefangen und eine Fehlermeldung ausgegeben.

 **Erklärung:**  
Dies hilft, Fehler zu diagnostizieren, falls das Modell nicht wie erwartet bearbeitet werden kann.

Schritt 8: Rückgabe des Status

return modified

 **Beschreibung:**  
Die Funktion gibt den Wert von modified zurück, der angibt, ob das Modell erfolgreich angepasst wurde oder nicht.

 **Erklärung:**  
Der Rückgabewert ermöglicht es dem Aufrufer zu wissen, ob das Modell verändert wurde, um mit Graustufenbildern zu arbeiten.

##### adjust\_input\_channels\_pytorch\_tensor(model, x\_tensor, reshape\_size)

Diese Funktion nimmt drei Eingabewerte an:

* model: Das PyTorch-Modell, dessen Eingabekanäle angepasst werden sollen.
* x\_tensor: Der Tensor, der die Bilddaten enthält.
* reshape\_size: Die Zielgröße, auf die die Bilder skaliert werden sollen.

Die Funktion stellt sicher, dass der Eingabekanal des Modells und der Tensor im richtigen Format vorliegen, insbesondere dass der Tensor die erwartete Kanalanzahl hat (1 Kanal für Graustufenbilder oder mehr).

Schritt 1: Ermittlung der erwarteten Kanäle des Modells

expected\_channels = detect\_input\_channels\_pytorch(model, reshape\_size)

 **Beschreibung:**  
Hier wird eine Funktion detect\_input\_channels\_pytorch aufgerufen, die die Anzahl der Kanäle ermittelt, die das Modell für die Eingabe erwartet. Der Parameter reshape\_size gibt die Größe des Bildes an, um sicherzustellen, dass die Eingabedaten korrekt angepasst werden.

 **Erklärung:**  
Diese Funktion könnte auf den ersten Layer des Modells zugreifen (z. B. die erste Convolutional-Schicht) und den erwarteten Eingabekanal auslesen. Zum Beispiel könnte das Modell 3 Kanäle für RGB-Bilder oder 1 Kanal für Graustufenbilder erwarten.

Schritt 2: Anpassung der Eingabekanäle des Modells

modified = adjust\_input\_channels\_pytorch(model)

* **Beschreibung:**  
  Hier wird die Funktion adjust\_input\_channels\_pytorch aufgerufen, die das Modell so anpasst, dass es mit Graustufenbildern (1 Kanal) arbeiten kann. Wenn das Modell bereits angepasst wurde (z. B. wenn es 1 Eingangskanal erwartet), gibt diese Funktion True zurück.
* **Erklärung:**  
  Falls das Modell nicht die erwartete Anzahl von Kanälen (1 für Graustufen) hat, wird die Eingabeschicht des Modells angepasst, um nur 1 Kanal zu erwarten. Diese Funktion gibt dann zurück, ob eine Modifikation stattgefunden hat.

Schritt 3: Umwandlung des Tensors in das passende Format

x\_tensor = x\_tensor.view(-1, 1, reshape\_size, reshape\_size)

 **Beschreibung:**  
Der Tensor x\_tensor wird umgeformt, sodass er die Form (-1, 1, reshape\_size, reshape\_size) hat. Dies bedeutet:

* -1: Der Wert wird automatisch so angepasst, dass die Gesamtzahl der Elemente des Tensors unverändert bleibt.
* 1: Der Tensor wird auf 1 Kanal (Graustufen) gesetzt.
* reshape\_size, reshape\_size: Die Größe des Bildes wird auf reshape\_size x reshape\_size geändert.

 **Erklärung:**  
Dies ist wichtig, da der Tensor möglicherweise ursprünglich in einer anderen Form vorliegt (z. B. als flacher Vektor oder mit einer anderen Kanalanzahl). Mit view wird die Form des Tensors so geändert, dass sie mit den Anforderungen des Modells übereinstimmt.

Schritt 4: Wiederholung der Kanäle, wenn nötig

if not modified and expected\_channels != 1:

print(f"🔹 Model expects {expected\_channels} channels → Updating gray scale images to {expected\_channels} channels.")

x\_tensor = x\_tensor.repeat(1, expected\_channels, 1, 1)

 **Beschreibung:**  
Wenn das Modell nicht modifiziert wurde und die Anzahl der Kanäle, die das Modell erwartet (expected\_channels), nicht 1 ist (also mehr als 1 Kanal erwartet wird), wird der Tensor wiederholt, um die erforderliche Anzahl an Kanälen zu erreichen.

 **Erklärung:**  
Falls das Modell mehr als 1 Kanal erwartet (z. B. RGB mit 3 Kanälen), wird der Graustufenbild-Tensor so wiederholt, dass er die richtige Anzahl von Kanälen hat. Die Methode repeat(1, expected\_channels, 1, 1) wiederholt den Tensor entlang der Kanalachse, sodass der Tensor für das Modell geeignet ist.

Schritt 5: Rückgabe des modifizierten Tensors

return x\_tensor

 **Beschreibung:**  
Die Funktion gibt den (möglicherweise modifizierten) Tensor zurück.

 **Erklärung:**  
Der Tensor wird nun in der richtigen Form (mit der richtigen Anzahl an Kanälen) zurückgegeben, sodass er als Eingabe für das Modell verwendet werden kann.

###### detect\_input\_channels\_pytorch(model, reshape\_size)

Die Funktion detect\_input\_channels\_pytorch ermittelt die Anzahl der Eingabekanäle, die ein PyTorch-Modell erwartet, indem sie das Modell mit einem Dummy-Tensor (mit der richtigen Form) ausführt und die Fehlermeldung nach der Anzahl der Kanäle durchsucht, wenn ein Fehler auftritt.

**Eingabewerte:**

* **model**: Das PyTorch-Modell, dessen Eingabekanäle wir erkennen möchten.
* **reshape\_size**: Die Zielgröße der Eingabebilder (z.B. 224x224 für ResNet).

Schritt 1: Versuch, das Modell mit einem Dummy-Tensor auszuführen

model(torch.empty(1, 0, reshape\_size, reshape\_size))

 **Beschreibung:**  
Ein Dummy-Tensor wird erstellt, der die Form (1, 0, reshape\_size, reshape\_size) hat.

* 1: Ein Batch von Bildern (1 Bild im Batch).
* 0: Eine Platzhalterzahl für die Anzahl der Kanäle (diese wird später durch die Fehlermeldung extrahiert).
* reshape\_size, reshape\_size: Die Bildgröße, die auf reshape\_size gesetzt wird (z. B. 224x224).

 **Erklärung:**  
Wir verwenden einen Dummy-Tensor, der eine gültige Batch-Größe und Bildgröße hat, aber keine Kanäle enthält. Das Modell kann normalerweise nicht mit diesem Tensor arbeiten und wird daher eine Fehlermeldung ausgeben. Wir nutzen diese Fehlermeldung, um die Anzahl der Kanäle zu extrahieren.

Schritt 2: Fehlerbehandlung und Analyse der Fehlermeldung

except RuntimeError as e:

search = re.search(r'(\d+)[^\d]+channels', str(e))

if search:

return int(search.group(1))

 **Beschreibung:**

* Wenn das Modell mit dem Dummy-Tensor nicht funktioniert, wird ein RuntimeError ausgelöst. Der Fehler wird durch den except-Block abgefangen.
* Der Fehlertext wird mit einem regulären Ausdruck (re.search) durchsucht, um die Anzahl der Kanäle zu extrahieren.

 **Erklärung:**

* Der reguläre Ausdruck sucht nach einer Zahl, die vor dem Wort "channels" steht (z. B. 3 channels für ein Modell, das RGB-Bilder erwartet). Der Ausdruck (\d+)[^\d]+channels extrahiert die Zahl (Anzahl der Kanäle) aus der Fehlermeldung.
* Falls eine Zahl gefunden wird, wird diese Zahl zurückgegeben.

Schritt 3: Rückgabe des Standardwertes (1 Kanal)

return 1

 **Beschreibung:**  
Falls die Anzahl der Kanäle nicht aus der Fehlermeldung extrahiert werden kann (d.h., wenn der reguläre Ausdruck keine Übereinstimmung findet), wird 1 als Standardwert zurückgegeben. Das bedeutet, dass das Modell wahrscheinlich Graustufenbilder erwartet (also 1 Kanal).

 **Erklärung:**  
In diesem Fall gibt die Funktion 1 zurück, was darauf hinweist, dass das Modell für Graustufenbilder (mit nur 1 Kanal) konzipiert ist.

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

Die Funktion adjust\_model\_output\_layer\_pytorch passt die Ausgangsschicht eines PyTorch-Modells an, um sicherzustellen, dass die Anzahl der Ausgabeklassen (num\_classes) korrekt ist. Diese Funktion erkennt verschiedene Architekturen von Modellen (wie AlexNet, VGG, ResNet, EfficientNet usw.) und aktualisiert entsprechend die finale Schicht (meist eine Fully Connected Layer oder eine Klassifikationsschicht), um die richtige Anzahl an Ausgabeklassen zu berücksichtigen.

**Eingabeparameter:**

* **model**: Das PyTorch-Modell, dessen Ausgangsschicht angepasst werden soll.
* **num\_classes**: Die gewünschte Anzahl an Ausgabeklassen (z. B. 10 für CIFAR-10, 1000 für ImageNet).
* **device**: Das Gerät (z. B. cuda oder cpu), auf dem das Modell laufen soll.
* **model\_name**: Der Name des Modells (als String), um spezifische Modelle wie EfficientNet zu erkennen.

Die Funktion prüft, ob das Modell eine bestimmte Schichtstruktur hat (z. B. classifier, fc, head, etc.). Basierend auf dieser Struktur wird dann die finale Schicht des Modells angepasst, um die korrekte Anzahl von Ausgabeklassen zu verwenden. Sie geht für verschiedene Modellarchitekturen wie AlexNet, VGG, ResNet, ConvNeXt und EfficientNet durch und modifiziert die finale Schicht.

Schritt 1: AlexNet und VGG (mit classifier)

if hasattr(model, "classifier"): # AlexNet, VGG

if isinstance(model.classifier, nn.Sequential) and len(model.classifier) > 6:

current\_classes = model.classifier[6].out\_features

if current\_classes != num\_classes:

print(f"Update classifier[6]: {current\_classes} → {num\_classes}")

model.classifier[6] = nn.Linear(model.classifier[6].in\_features, num\_classes).to(device)

**Beschreibung:**

* Überprüft, ob das Modell eine classifier-Schicht hat (was für AlexNet und VGG typisch ist).
* Falls classifier ein nn.Sequential-Modul ist und mehr als 6 Schichten enthält, wird angenommen, dass sich die finale Fully Connected (FC)-Schicht an Index 6 befindet (dies ist für viele gängige Modelle wie VGG und AlexNet der Fall).
* Wenn die Anzahl der Ausgabeklassen (out\_features) der aktuellen FC-Schicht nicht mit num\_classes übereinstimmt, wird diese Schicht auf die richtige Größe angepasst und auf das angegebene Gerät (device) verschoben.

Schritt 2: ResNet, ConvNeXt und EfficientNet (mit fc)

elif hasattr(model, "fc"): # ResNet, ConvNeXt, EfficientNet

current\_fc = model.fc.out\_features

if current\_fc != num\_classes:

print(f"Update fc-Layer: {current\_fc} → {num\_classes}")

model.fc = nn.Linear(model.fc.in\_features, num\_classes).to(device)

**Beschreibung:**

* Für Modelle wie ResNet, ConvNeXt und EfficientNet wird überprüft, ob das Modell eine fc-Schicht hat.
* Wenn fc existiert, wird die Anzahl der Ausgabeklassen (d.h., out\_features) der aktuellen FC-Schicht überprüft.
* Wenn die Anzahl der Klassen nicht mit num\_classes übereinstimmt, wird die Schicht aktualisiert, um die richtige Anzahl von Ausgabeklassen zu liefern.

Schritt 3: ConvNeXt und bestimmte Architekturen (mit head)

elif hasattr(model, "head"): # ConvNeXt & bestimmte Architekturen

current\_head = model.head.out\_features

if current\_head != num\_classes:

print(f"Update head-Layer: {current\_head} → {num\_classes}")

model.head = nn.Linear(model.head.in\_features, num\_classes).to(device)

**Beschreibung:**

* Einige Modelle (wie ConvNeXt) verwenden eine head-Schicht als finale Klassifikationsschicht.
* Falls das Modell diese head-Schicht hat und die Anzahl der Ausgabeklassen (out\_features) nicht mit num\_classes übereinstimmt, wird diese Schicht angepasst.

Schritt 4: Spezifische Behandlung für EfficientNet

elif isinstance(model, nn.Module): # Allgemeiner Check für PyTorch Modelle

# Überprüfe den Namen des Modells und passe die finale Schicht an

if "efficientnet" in model\_name: # Spezifische Behandlung für EfficientNet

if hasattr(model, 'classifier'):

current\_classifier = model.classifier[-1]

if current\_classifier.out\_features != num\_classes:

print(f"Update classifier[-1]: {current\_classifier.out\_features} → {num\_classes}")

model.classifier[-1] = nn.Linear(model.classifier[-1].in\_features, num\_classes).to(device)

**Beschreibung:**

* Für Modelle wie EfficientNet wird der Name des Modells (model\_name) überprüft. Falls der Name efficientnet enthält, wird die finale Klassifikationsschicht (meistens die letzte Schicht im classifier-Modul) angepasst.
* Wenn die Anzahl der Ausgabeklassen (out\_features) nicht mit num\_classes übereinstimmt, wird die Schicht aktualisiert.

Schritt 5: Fehlerbehandlung

else:

raise ValueError("Unknown model architecture. Final Layer could not get updated.")

**Beschreibung:**  
Falls das Modell nicht in eine der bekannten Kategorien (wie classifier, fc oder head) fällt, wird eine Fehlermeldung ausgegeben, dass die Architektur des Modells unbekannt ist und die finale Schicht nicht angepasst werden kann.

Schritt 6: Rückgabe des modifizierten Modells

return model

**Beschreibung:**  
Die Funktion gibt das angepasste Modell zurück, bei dem die finale Schicht so geändert wurde, dass die korrekte Anzahl an Ausgabeklassen (num\_classes) verwendet wird.

##### adjust\_model\_output\_layer\_keras(model, num\_classes, model\_name)

Die Funktion adjust\_model\_output\_layer\_keras passt die Ausgangsschicht eines Keras-Modells an, um sicherzustellen, dass die Anzahl der Ausgabeklassen (num\_classes) korrekt ist. Diese Funktion erkennt verschiedene Architekturen von Keras-Modellen wie Dense-Netzwerke, ResNet, VGG, Inception, EfficientNet, Xception und MobileNetV2 und aktualisiert die finale Schicht, um die richtige Anzahl an Ausgabeklassen zu berücksichtigen.

**Eingabeparameter:**

* **model**: Das Keras-Modell, dessen Ausgangsschicht angepasst werden soll.
* **num\_classes**: Die gewünschte Anzahl an Ausgabeklassen (z. B. 10 für CIFAR-10, 1000 für ImageNet).
* **model\_name**: Der Name des Modells (als String), um spezifische Modelle wie EfficientNet oder Inception zu erkennen.

**Funktionsweise:**

Die Funktion überprüft, ob das Modell eine bestimmte Schichtstruktur hat (z. B. Dense, fc, classifier, etc.). Basierend auf dieser Struktur wird dann die finale Schicht des Modells angepasst, um die richtige Anzahl von Ausgabeklassen zu verwenden. Sie geht für verschiedene Modellarchitekturen wie Dense-Netzwerke, ResNet, VGG, Inception, EfficientNet, Xception und MobileNetV2 durch und modifiziert entsprechend die finale Schicht.

Schritt 1: Dense-Schicht als letzte Schicht (typisch für viele Modelle)

if isinstance(model.layers[-1], layers.Dense):

current\_classes = model.layers[-1].units

if current\_classes != num\_classes:

print(f"🔹 Update Dense output layer: {current\_classes} → {num\_classes}")

model.layers[-1] = layers.Dense(num\_classes, activation='softmax')(model.layers[-2].output)

model = models.Model(inputs=model.inputs, outputs=model.layers[-1].output)

**Beschreibung:**

* Überprüft, ob die letzte Schicht des Modells eine Dense-Schicht ist, was für viele Modelle wie einfache Feedforward-Netzwerke typisch ist.
* Wenn die Anzahl der Ausgabeklassen (units der letzten Schicht) nicht mit num\_classes übereinstimmt, wird die Dense-Schicht auf die richtige Anzahl an Ausgabeklassen angepasst. Die activation='softmax'-Funktion wird verwendet, um die Wahrscheinlichkeiten der Klassen zu berechnen.
* Das Modell wird mit der aktualisierten letzten Schicht neu erstellt.

Schritt 2: ResNet, VGG, Inception und ähnliche Architekturen (mit fc)

elif hasattr(model, 'fc'): # Einige Modelle wie ResNet, ConvNeXt, EfficientNet

current\_fc = model.fc.units # In Keras sind diese Schichten als Dense zu finden

if current\_fc != num\_classes:

print(f"🔹 Update fc-Layer: {current\_fc} → {num\_classes}")

model.fc = layers.Dense(num\_classes, activation='softmax')(model.fc.input)

model = models.Model(inputs=model.inputs, outputs=model.fc.output)

**Beschreibung:**

* Überprüft, ob das Modell eine fc-Schicht hat (typisch für Modelle wie ResNet und ConvNeXt).
* Falls die Anzahl der Ausgabeklassen der fc-Schicht nicht mit num\_classes übereinstimmt, wird sie entsprechend angepasst.
* Das Modell wird mit der neuen fc-Schicht neu erstellt.

Schritt 3: Spezifische Behandlung für EfficientNet

elif 'efficientnet' in model\_name.lower():

if hasattr(model, 'classifier'):

current\_classifier = model.classifier[-1] # Für Keras Modelle

if current\_classifier.units != num\_classes:

print(f"🔹 Update classifier[-1]: {current\_classifier.units} → {num\_classes}")

model.classifier[-1] = layers.Dense(num\_classes, activation='softmax')(model.classifier[-2].output)

model = models.Model(inputs=model.inputs, outputs=model.classifier[-1].output)

**Beschreibung:**

* Wenn der Modellname efficientnet enthält, wird angenommen, dass es sich um ein EfficientNet-Modell handelt.
* Es wird überprüft, ob das Modell eine classifier-Schicht hat, und wenn die Anzahl der Ausgabeklassen in der letzten Schicht (classifier[-1]) nicht mit num\_classes übereinstimmt, wird sie entsprechend angepasst.

Schritt 4: Spezifische Behandlung für InceptionV3

elif 'inceptionv3' in model\_name.lower():

if hasattr(model, 'output'):

if model.output.shape[-1] != num\_classes:

print(f"🔹 Update InceptionV3 output layer: {model.output.shape[-1]} → {num\_classes}")

x = layers.GlobalAveragePooling2D()(model.output)

output = layers.Dense(num\_classes, activation='softmax')(x)

model = models.Model(inputs=model.inputs, outputs=output)

**Beschreibung:**

* Wenn der Modellname inceptionv3 enthält, wird die Ausgangsschicht überprüft.
* Es wird sichergestellt, dass die Anzahl der Ausgabeklassen mit num\_classes übereinstimmt.
* Wenn nicht, wird eine GlobalAveragePooling2D-Schicht hinzugefügt, gefolgt von einer Dense-Schicht, um die Ausgabeklassen anzupassen.

Schritt 5: Spezifische Behandlung für Xception

elif 'xception' in model\_name.lower():

if hasattr(model, 'output'):

if model.output.shape[-1] != num\_classes:

print(f"🔹 Update Xception output layer: {model.output.shape[-1]} → {num\_classes}")

x = layers.GlobalAveragePooling2D()(model.output)

output = layers.Dense(num\_classes, activation='softmax')(x)

model = models.Model(inputs=model.inputs, outputs=output)

**Beschreibung:**

* Wenn der Modellname xception enthält, wird ähnlich wie bei Inception die Ausgangsschicht überprüft und bei Bedarf angepasst.

Schritt 6: Spezifische Behandlung für MobileNetV2

elif 'mobilenetv2' in model\_name.lower():

if hasattr(model, 'output'):

if model.output.shape[-1] != num\_classes:

print(f"🔹 Update MobileNetV2 output layer: {model.output.shape[-1]} → {num\_classes}")

x = layers.GlobalAveragePooling2D()(model.output)

output = layers.Dense(num\_classes, activation='softmax')(x)

model = models.Model(inputs=model.inputs, outputs=output)

**Beschreibung:**

* Für mobilenetv2-Modelle wird ebenfalls die Ausgangsschicht überprüft und angepasst, falls nötig.

Schritt 7: Fehlerbehandlung

else:

raise ValueError("Unknown model architecture. Final Layer could not get updated.")

**Beschreibung:**  
Falls das Modell nicht in eine der bekannten Kategorien fällt, wird eine Fehlermeldung ausgegeben, dass die Architektur des Modells unbekannt ist und die finale Schicht nicht angepasst werden kann.

Rückgabe des Modells:

return model

**Beschreibung:**  
Die Funktion gibt das angepasste Modell zurück, bei dem die finale Schicht so geändert wurde, dass die korrekte Anzahl an Ausgabeklassen (num\_classes) verwendet wird.

### 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

In diesem Abschnitt wird erklärt, wie die Modellinstanzen in Verbindung mit der **Basisklasse** instanziiert werden, um spezifische Modelle aus der PyTorch-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.

###### AlexNet

AlexNet ist ein tiefes neuronales Netzwerk, das insbesondere für Bildklassifikationsaufgaben verwendet wird. Es wurde von Alex Krizhevsky et al. entwickelt und gewann 2012 den ImageNet-Wettbewerb, was einen bedeutenden Durchbruch im Deep Learning darstellte.

Theorie

AlexNet basiert auf einer mehrschichtigen Architektur, die Bilder durch eine Kombination aus Faltungsschichten (Convolutional Layers), nicht-linearen Aktivierungsfunktionen und voll verbundenen Schichten (Fully Connected Layers) verarbeitet.

Ziel ist es, automatisch Merkmale aus Bildern zu extrahieren und anhand dieser Merkmale eine Klassifikation vorzunehmen. Dabei werden zunächst einfache Merkmale wie Kanten erkannt, während tiefere Schichten komplexe Muster wie Objekte oder Gesichter identifizieren können.

Aufbau und Funktionsweise

AlexNet besteht aus insgesamt **8 Schichten**, die trainierbare Parameter enthalten:

* **5 Convolutional Layers**: Diese Schichten extrahieren Merkmale aus dem Bild, indem sie verschiedene Filter anwenden.
* **3 Fully Connected Layers**: Hier werden die extrahierten Merkmale genutzt, um eine Entscheidung über die Klassenzugehörigkeit des Bildes zu treffen.

Weitere wichtige Aspekte:

* **ReLU-Aktivierungsfunktion**: Beschleunigt das Training und sorgt für nicht-lineare Modellierung.
* **Max-Pooling**: Reduziert die Größe der Merkmalsdarstellung und erhöht die Robustheit gegenüber Variationen im Bild.
* **Dropout**: Verhindert Überanpassung (Overfitting) an das Trainingsset, indem zufällig Neuronen deaktiviert werden.
* **Softmax-Ausgangsschicht**: Berechnet Wahrscheinlichkeiten für jede mögliche Klasse und trifft basierend darauf eine Vorhersage.

Anwendungen

AlexNet wird in vielen Bereichen der Bildverarbeitung eingesetzt:

* **Objektklassifikation**: Erkennung und Unterscheidung von Objekten in Bildern.
* **Gesichtserkennung**: Identifikation von Gesichtern in Fotos oder Videos.
* **Medizinische Bildverarbeitung**: Diagnoseunterstützung durch Analyse von Röntgenbildern oder MRT-Scans.
* **Autonomes Fahren**: Erkennung von Straßenschildern und Hindernissen in Kamerabildern.

Verlässlichkeit und Performance

AlexNet ist ein leistungsstarkes neuronales Netzwerk für Bildklassifikationsaufgaben und hat sich insbesondere durch seinen Erfolg beim ImageNet-Wettbewerb bewährt. Dank seiner tiefen Architektur kann es komplexe Muster und Objekte in Bildern erkennen, was es zu einem zuverlässigen Modell für viele Anwendungen macht. Durch die Nutzung von GPUs wird das Training erheblich beschleunigt, was gerade bei großen Datensätzen von Vorteil ist.

Allerdings bringt die Architektur auch Herausforderungen mit sich. AlexNet benötigt eine große Menge an Trainingsdaten, um sein volles Potenzial auszuschöpfen, und ist im Vergleich zu modernen Architekturen wie ResNet oder EfficientNet weniger effizient. Zudem erfordert es eine hohe Rechenleistung, insbesondere für das Training, was den Einsatz in Umgebungen mit begrenzten Ressourcen erschwert. Dennoch bleibt es ein solides Modell für Bildklassifikationen, das insbesondere in Szenarien mit ausreichend Rechenkapazität und großen Datensätzen eine hohe Genauigkeit liefert.

Imports

Die notwendigen Imports für dieses Modell sind:

from .training\_basemodel import BaseModelTorch

from torchvision import models

 **BaseModelTorch**: Wird aus dem Modul training\_basemodel importiert und stellt die Basisklasse für PyTorch-Modelle dar. Diese Klasse enthält allgemeine Funktionen für das Training, das Speichern und das Laden von Modellen.

 **models**: Stammt aus der torchvision-Bibliothek und enthält vortrainierte Deep-Learning-Modelle. Hier wird models.alexnet verwendet, um eine Instanz des AlexNet-Modells zu erstellen.

training\_alexnet\_pytorch.py

In diesem Fall wird AlexNet mit einer spezifischen Anzahl an Klassen initialisiert. Die Architektur des Modells bleibt dabei erhalten, aber die Anzahl der Ausgabeneinheiten wird an die gewünschte Klassenzahl angepasst.

class AlexNetModel(BaseModelTorch):

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

# AlexNet wird mit der angegebenen Anzahl an Klassen instanziiert

super().\_\_init\_\_(models.alexnet(weights=None, num\_classes=classes), model\_save\_path)

 **models.alexnet(weights=None, num\_classes=classes)**: Erstellt eine Instanz des AlexNet-Modells ohne vortrainierte Gewichte (weights=None). Die Anzahl der Ausgabeklassen wird auf den übergebenen classes-Wert gesetzt.

 **super().\_\_init\_\_(models.alexnet(...), model\_save\_path)**: Ruft den Konstruktor der Basisklasse BaseModelTorch auf und übergibt das erstellte AlexNet-Modell sowie den Speicherpfad für das trainierte Modell. Dadurch übernimmt die Basisklasse die Verantwortung für das Training, Speichern und Laden des Modells.

###### ConvNext

ConvNeXt ist ein modernes Convolutional Neural Network (CNN), das als eine Weiterentwicklung traditioneller CNN-Architekturen entwickelt wurde. Es kombiniert bewährte Techniken aus der CNN-Forschung mit Konzepten aus Vision Transformer (ViT)-Modellen, um eine leistungsfähige und effiziente Architektur für Bildklassifikationsaufgaben bereitzustellen.

Theorie

ConvNeXt wurde entwickelt, um die Vorteile von tiefen CNNs mit den Verbesserungen aus der Transformer-Forschung zu kombinieren. Im Gegensatz zu klassischen CNNs wie AlexNet oder ResNet wurde ConvNeXt optimiert, um mit modernen Trainingsmethoden und Hardware besser zu funktionieren.

Das Modell nutzt **Tiefe und Breite**, um eine verbesserte Repräsentationsfähigkeit zu erreichen, während es gleichzeitig auf bewährte Architekturprinzipien wie **Residual-Verbindungen**, **Layer Normalization** und **Depthwise Convolutions** setzt. Dadurch kann ConvNeXt eine hohe Genauigkeit bei geringerer Rechenkomplexität erreichen.

Aufbau und Funktionsweise

ConvNeXt basiert auf einer hierarchischen Struktur und beinhaltet mehrere wichtige Komponenten:

1. **Convolutional Blöcke:**
   * Verwendet Depthwise-Separable Convolutions, um die Rechenlast zu reduzieren.
   * Ersetzt klassische Batch-Normalization durch Layer-Normalization für stabileres Training.
2. **Residual-Verbindungen:**
   * Ähnlich wie bei ResNet werden Informationen über Sprungverbindungen weitergeleitet, um das Verschwinden des Gradienten zu vermeiden.
3. **Patchify-Strategie:**
   * Inspiriert von Vision Transformers werden Bilder in kleinere Blöcke (Patches) zerlegt, um effizientere Merkmalsextraktionen durchzuführen.
4. **Flexible Architektur:**
   * Unterstützt unterschiedliche Tiefen und Breiten, um verschiedene Anwendungsfälle abzudecken.

Anwendungen

ConvNeXt wird in vielen Bereichen eingesetzt, darunter:

* **Bildklassifikation:** Erkennung und Kategorisierung von Objekten in Bildern.
* **Medizinische Diagnostik:** Analyse von Röntgenbildern oder MRT-Scans zur Identifikation von Krankheiten.
* **Autonomes Fahren:** Verarbeitung von Kameradaten zur Erkennung von Objekten und Hindernissen.
* **Industrie 4.0:** Qualitätskontrolle durch visuelle Inspektion in der Produktion.

Verlässlichkeit und Performance

ConvNeXt bietet eine hohe Genauigkeit bei gleichzeitig effizienter Rechenleistung. Dank der verbesserten Architektur kann das Modell auf modernen GPUs schneller trainiert werden als klassische CNNs.

Ein großer Vorteil ist die Fähigkeit von ConvNeXt, auch auf großen Bilddatensätzen effizient zu arbeiten. Die Kombination aus Residual-Verbindungen und Depthwise-Separable Convolutions macht das Modell besonders leistungsfähig bei der Bildklassifikation.

Dennoch kann das Modell in bestimmten Szenarien mehr Rechenleistung benötigen als einfachere Architekturen wie MobileNet oder ResNet. In Anwendungen mit begrenzter Hardware (z. B. auf mobilen Geräten) kann es daher notwendig sein, kleinere Varianten von ConvNeXt zu verwenden.

Imports

Die notwendigen Imports für dieses Modell sind:

from .training\_basemodel import BaseModelTorch

from torchvision import models

 **BaseModelTorch**: Wird aus dem Modul training\_basemodel importiert und stellt die Basisklasse für PyTorch-Modelle dar. Diese Klasse enthält allgemeine Funktionen für das Training, das Speichern und das Laden von Modellen.

 **models**: Stammt aus der torchvision-Bibliothek und enthält vortrainierte Deep-Learning-Modelle. Hier wird models.convnext\_base verwendet, um eine Instanz des ConvNeXt-Modells zu erstellen.

training\_convnext\_pytorch.py

In diesem Fall wird ConvNeXt mit einer spezifischen Anzahl an Klassen initialisiert. Die Architektur des Modells bleibt dabei erhalten, aber die Anzahl der Ausgabeneinheiten wird an die gewünschte Klassenzahl angepasst.

class ConvNeXtModel(BaseModelTorch):

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

# ConvNeXt wird mit der angegebenen Anzahl an Klassen instanziiert

super().\_\_init\_\_(models.convnext\_base(weights=None, num\_classes=classes), model\_save\_path)

 **models.convnext\_base(weights=None, num\_classes=classes)**: Erstellt eine Instanz des ConvNeXt-Modells ohne vortrainierte Gewichte (weights=None). Die Anzahl der Ausgabeklassen wird auf den übergebenen classes-Wert gesetzt.

 **super().\_\_init\_\_(models.convnext\_base(...), model\_save\_path)**: Ruft den Konstruktor der Basisklasse BaseModelTorch auf und übergibt das erstellte ConvNeXt-Modell sowie den Speicherpfad für das trainierte Modell. Dadurch übernimmt die Basisklasse die Verantwortung für das Training, Speichern und Laden des Modells.

###### DenseNet

DenseNet (Densely Connected Convolutional Network) ist eine fortschrittliche CNN-Architektur, die durch eine dichte Verbindung zwischen den Schichten eine effiziente Merkmalsextraktion ermöglicht. Durch diese Verbindungen können Informationen besser zwischen den Schichten fließen, wodurch das Modell weniger Parameter benötigt und trotzdem eine hohe Genauigkeit erreicht.

Theorie

Im Gegensatz zu klassischen CNNs, bei denen jede Schicht nur mit der vorherigen Schicht verbunden ist, verwendet DenseNet eine sogenannte **dichte Konnektivität**. Das bedeutet, dass jede Schicht mit allen vorherigen Schichten verbunden ist. Dadurch kann das Modell bereits gelernte Merkmale wiederverwenden, was zu einer verbesserten Effizienz und Genauigkeit führt.

DenseNet zeichnet sich durch folgende Eigenschaften aus:

* **Dichte Verbindungen:** Jede Schicht erhält als Eingabe die Ausgabe aller vorherigen Schichten.
* **Geringere Anzahl an Parametern:** Durch die Wiederverwendung von Merkmalen wird die Anzahl der benötigten Parameter reduziert.
* **Effiziente Merkmalsextraktion:** Durch die dichteren Verbindungen lernt das Modell tiefere und detailliertere Merkmale.

Aufbau und Funktionsweise

DenseNet besteht aus mehreren **Dense Blocks**, in denen jede Schicht direkt mit allen vorherigen Schichten verbunden ist. Das bedeutet:

* **Jede Schicht empfängt die Feature-Maps aller vorherigen Schichten.**
* **Jede Schicht gibt ihre eigenen Feature-Maps an alle nachfolgenden Schichten weiter.**
* **Durch diese Architektur kann das Modell mit weniger Parametern eine hohe Genauigkeit erreichen.**

Zwischen den Dense Blocks befinden sich sogenannte **Transition Layers**, die die Größe der Feature-Maps durch **1x1 Convolutions und Pooling** reduzieren. Dadurch bleibt das Modell effizient und speicherschonend.

Anwendungen

DenseNet wird in verschiedenen Bereichen eingesetzt, darunter:

* **Bildklassifikation:** Erkennung und Klassifikation von Objekten in Bildern.
* **Medizinische Bildverarbeitung:** Analyse von Röntgenbildern oder MRT-Scans zur Diagnose von Krankheiten.
* **Autonomes Fahren:** Objekterkennung in Straßenverkehrsdaten.
* **Gesichtserkennung:** Identifikation von Personen durch Deep-Learning-gestützte Bilderkennung.

Verlässlichkeit und Performance

DenseNet bietet eine hohe Genauigkeit bei gleichzeitig geringem Speicherverbrauch, da Feature-Maps wiederverwendet werden. Dies führt zu einer effizienten Nutzung der Netzwerkkapazität, sodass das Modell schneller trainiert und weniger Rechenressourcen benötigt als klassische Architekturen wie ResNet.

Allerdings kann die dichte Verbindung zwischen den Schichten dazu führen, dass das Modell mit sehr großen Bildern speicherintensiv wird. In solchen Fällen kann es notwendig sein, kleinere Varianten wie DenseNet-121 anstelle von tieferen Modellen wie DenseNet-201 zu verwenden. Insgesamt ist DenseNet jedoch eine sehr leistungsfähige Architektur, die insbesondere für Bildklassifikationsaufgaben gut geeignet ist.

Imports

Die notwendigen Imports für dieses Modell sind:

from .training\_basemodel import BaseModelTorch

from torchvision import models

 **BaseModelTorch**: Wird aus dem Modul training\_basemodel importiert und stellt die Basisklasse für PyTorch-Modelle dar. Diese Klasse enthält allgemeine Funktionen für das Training, das Speichern und das Laden von Modellen.

 **models**: Stammt aus der torchvision-Bibliothek und enthält vortrainierte Deep-Learning-Modelle. Hier wird models.densenet121 verwendet, um eine Instanz des DenseNet-Modells zu erstellen.

training\_densenet\_pytorch.py

In diesem Fall wird DenseNet mit einer spezifischen Anzahl an Klassen initialisiert. Die Architektur des Modells bleibt dabei erhalten, aber die Anzahl der Ausgabeneinheiten wird an die gewünschte Klassenzahl angepasst.

class DenseNetModel(BaseModelTorch):

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

# DenseNet wird mit der angegebenen Anzahl an Klassen instanziiert

super().\_\_init\_\_(models.densenet121(weights=None, num\_classes=classes), model\_save\_path)

 **models.densenet121(weights=None, num\_classes=classes)**: Erstellt eine Instanz des DenseNet-Modells ohne vortrainierte Gewichte (weights=None). Die Anzahl der Ausgabeklassen wird auf den übergebenen classes-Wert gesetzt.

 **super().\_\_init\_\_(models.densenet121(...), model\_save\_path)**: Ruft den Konstruktor der Basisklasse BaseModelTorch auf und übergibt das erstellte DenseNet-Modell sowie den Speicherpfad für das trainierte Modell. Dadurch übernimmt die Basisklasse die Verantwortung für das Training, Speichern und Laden des Modells.

###### EfficientNet

EfficientNet ist eine moderne CNN-Architektur, die mit weniger Parametern eine höhere Genauigkeit erreicht als viele ältere Modelle. Es verwendet eine spezielle Skalierungsstrategie, um Tiefe, Breite und Auflösung gleichzeitig zu optimieren. Dadurch wird das Modell effizienter und leistungsfähiger als klassische Architekturen wie ResNet oder VGG.

Theorie

EfficientNet basiert auf der Idee des **Compound Scaling**, einer Methode zur gleichmäßigen Skalierung der drei Hauptfaktoren eines neuronalen Netzwerks:

* **Tiefe** (mehr Schichten verbessern die Feature-Extraktion)
* **Breite** (mehr Kanäle pro Schicht erfassen detailliertere Informationen)
* **Auflösung** (größere Eingangsgrößen bewahren mehr Bilddetails)

Anstatt diese Faktoren unabhängig voneinander zu optimieren, nutzt EfficientNet eine **gleichmäßige Skalierung**, um das Modell effizienter zu machen. Dadurch erreicht es eine hohe Genauigkeit bei gleichzeitig reduziertem Rechenaufwand.

Aufbau und Funktionsweise

EfficientNet verwendet **MBConv-Blöcke**, eine Weiterentwicklung der Inverted Residual Blocks aus MobileNetV2. Diese Blöcke kombinieren **Tiefenkonvolutionen, Expansionsschichten und Squeeze-and-Excitation-Mechanismen**, um die Rechenkomplexität zu reduzieren, während gleichzeitig die Erkennungsleistung verbessert wird.

Es gibt verschiedene Varianten von EfficientNet (B0 bis B7), wobei **EfficientNet-B0** das Basismodell ist. Höhere Varianten sind skalierte Versionen mit mehr Parametern und höherer Auflösung.

EfficientNet-B0 zeichnet sich durch:

* **Effiziente Architektur** durch optimierte Blockstrukturen
* **Geringere Anzahl von Parametern** bei hoher Genauigkeit
* **Automatische Skalierung** für unterschiedliche Hardware-Ressourcen

Anwendungen

EfficientNet wird in vielen Bereichen eingesetzt, darunter:

* **Bildklassifikation**: Kategorisierung von Objekten in Bildern
* **Medizinische Diagnostik**: Analyse von Röntgenbildern oder MRT-Scans
* **Autonomes Fahren**: Erkennung von Verkehrsschildern und Hindernissen
* **Edge Computing**: KI-Anwendungen auf mobilen Geräten mit begrenzten Ressourcen

Durch seine optimierte Architektur eignet sich EfficientNet besonders gut für Anwendungen, bei denen **hohe Genauigkeit mit geringer Rechenlast** erforderlich ist.

Verlässlichkeit und Performance

EfficientNet bietet eine hervorragende Balance zwischen **Genauigkeit und Effizienz**. Es erreicht eine höhere Leistung als ältere Modelle mit weniger Rechenaufwand. Dank des Compound-Scaling-Ansatzes kann das Modell für verschiedene Einsatzgebiete skaliert werden, ohne übermäßig viele Parameter zu benötigen.

Ein möglicher Nachteil von EfficientNet ist, dass **größere Varianten (B4–B7) mehr Speicher und Rechenleistung benötigen**, wodurch sie sich weniger für Echtzeit-Anwendungen eignen. Dennoch ist EfficientNet in den meisten Fällen eine der besten Wahlmöglichkeiten für Bildklassifikationsaufgaben.

Imports

Die notwendigen Imports für dieses Modell sind:

from .training\_basemodel import BaseModelTorch

from torchvision import models

 **BaseModelTorch**: Stammt aus dem Modul training\_basemodel und enthält allgemeine Funktionen für das Training, das Speichern und das Laden von PyTorch-Modellen.

 **models**: Wird aus torchvision importiert und stellt vortrainierte Deep-Learning-Modelle bereit. In diesem Fall wird models.efficientnet\_b0 verwendet.

training\_efficientnet\_pytorch.py

In diesem Fall wird EfficientNet-B0 mit einer bestimmten Anzahl von Klassen initialisiert.

class EfficientNetModel(BaseModelTorch):

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

# EfficientNet-B0 wird mit der angegebenen Anzahl an Klassen instanziiert

super().\_\_init\_\_(models.efficientnet\_b0(weights=None, num\_classes=classes), model\_save\_path, model\_name\_extension="\_v1")

 **models.efficientnet\_b0(weights=None, num\_classes=classes)**: Erstellt eine Instanz von EfficientNet-B0 ohne vortrainierte Gewichte (weights=None). Die Anzahl der Klassen wird durch den Parameter classes festgelegt.

 **super().\_\_init\_\_(models.efficientnet\_b0(...), model\_save\_path, model\_name\_extension="\_v1")**: Ruft den Konstruktor der Basisklasse BaseModelTorch auf und übergibt das erstellte EfficientNet-Modell, den Speicherpfad sowie eine Modellversionsbezeichnung (\_v1).

EfficientNetV2

EfficientNetV2 ist eine weiterentwickelte Version des ursprünglichen EfficientNet-Modells und verbessert dessen Effizienz und Genauigkeit. Es wurde speziell für eine schnellere Trainingszeit und eine bessere Skalierbarkeit entwickelt, indem es Änderungen an der Architektur vornimmt, um Rechenkosten zu reduzieren und die Leistung zu steigern.

Theorie

EfficientNetV2 baut auf den Prinzipien des ursprünglichen EfficientNet auf, verwendet jedoch mehrere Optimierungen, um die Leistung zu verbessern:

* **Verbesserte Skalierungsstrategie**: Statt des ursprünglichen Compound Scaling setzt EfficientNetV2 auf eine optimierte Architektur mit schrittweiser Vergrößerung von Tiefe, Breite und Auflösung.
* **Fused-MBConv-Blöcke**: Diese ersetzen in den ersten Stufen des Netzwerks die herkömmlichen MBConv-Blöcke, was zu schnelleren Berechnungen führt.
* **Reduzierte Anzahl von Downsampling-Schritten**: Dies verringert die Rechenkomplexität und beschleunigt das Training.

Durch diese Verbesserungen kann EfficientNetV2 Bilder schneller und mit höherer Genauigkeit verarbeiten als sein Vorgänger.

Aufbau und Funktionsweise

EfficientNetV2 verwendet eine Kombination aus **Fused-MBConv- und MBConv-Blöcken**, die je nach Tiefe des Netzwerks variieren. Die Hauptbestandteile sind:

* **Fused-MBConv-Blöcke in frühen Stufen**: Diese ersetzen herkömmliche Tiefenkonvolutionen und sind effizienter für kleinere Auflösungen.
* **MBConv-Blöcke in späteren Stufen**: Sobald das Feature-Extraktionsniveau höher ist, kommen klassische MBConv-Blöcke zum Einsatz.
* **Adaptive Skalierung**: EfficientNetV2 verwendet eine verbesserte Skalierungsmethode, um das Modell besser an verschiedene Rechenkapazitäten anzupassen.

Es gibt verschiedene Versionen von EfficientNetV2, darunter **EfficientNetV2-S, -M und -L**, wobei **EfficientNetV2-S** die kleinste und effizienteste Version ist.

Anwendungen

EfficientNetV2 findet Anwendung in vielen Bereichen, darunter:

* **Bildklassifikation**: Automatische Kategorisierung von Objekten in Bildern
* **Objekterkennung**: Erkennung und Lokalisierung von Objekten in Bildern
* **Medizinische Diagnostik**: Analyse von Röntgenbildern zur Krankheitsdetektion
* **Echtzeit-Anwendungen**: Ideal für mobile Geräte und Edge-Computing aufgrund der verbesserten Effizienz

Durch die schnellere Trainingszeit und geringere Speicheranforderungen ist EfficientNetV2 besonders nützlich für Aufgaben, bei denen **Echtzeitverarbeitung und hohe Genauigkeit** erforderlich sind.

Verlässlichkeit und Performance

EfficientNetV2 bietet eine höhere **Trainingsgeschwindigkeit** und **bessere Genauigkeit** als sein Vorgänger, ohne dabei an Effizienz einzubüßen. Durch die optimierte Architektur kann das Modell schneller konvergieren und ist robuster gegenüber verschiedenen Eingangsgrößen.

Ein möglicher Nachteil ist, dass größere Varianten von EfficientNetV2 (z. B. M und L) immer noch viel Rechenleistung benötigen, insbesondere für Inferenzaufgaben mit hoher Auflösung. Dennoch bleibt es eine der besten Wahlmöglichkeiten für Bildklassifikationsaufgaben mit begrenzten Ressourcen.

Imports

Die notwendigen Imports für dieses Modell sind:

from .training\_basemodel import BaseModelTorch

from torchvision import models

 **BaseModelTorch**: Stammt aus dem Modul training\_basemodel und enthält allgemeine Funktionen für das Training, das Speichern und das Laden von PyTorch-Modellen.

 **models**: Wird aus torchvision importiert und stellt vortrainierte Deep-Learning-Modelle bereit. In diesem Fall wird models.efficientnet\_v2\_s verwendet.

training\_efficientnetv2\_pytorch.py

In diesem Fall wird EfficientNetV2-S mit einer bestimmten Anzahl von Klassen initialisiert.

class EfficientNetV2Model(BaseModelTorch):

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

# EfficientNetV2-S wird mit der angegebenen Anzahl an Klassen instanziiert

super().\_\_init\_\_(models.efficientnet\_v2\_s(weights=None, num\_classes=classes), model\_save\_path, model\_name\_extension="\_v2")

 **models.efficientnet\_v2\_s(weights=None, num\_classes=classes)**: Erstellt eine Instanz von EfficientNetV2-S ohne vortrainierte Gewichte (weights=None). Die Anzahl der Klassen wird durch den Parameter classes festgelegt.

 **super().\_\_init\_\_(models.efficientnet\_v2\_s(...), model\_save\_path, model\_name\_extension="\_v2")**: Ruft den Konstruktor der Basisklasse BaseModelTorch auf und übergibt das erstellte EfficientNetV2-Modell, den Speicherpfad sowie eine Modellversionsbezeichnung (\_v2).

###### GoogleNet

GoogLeNet ist ein tiefes neuronales Netzwerk, das speziell für die Bildklassifikation entwickelt wurde. Es wurde von Google im Jahr 2014 eingeführt und zeichnet sich durch seine innovative Architektur, die sogenannte **Inception-Architektur**, aus. Das Modell wurde entwickelt, um die Effizienz und Genauigkeit zu maximieren und dabei die Anzahl der Parameter zu minimieren.

Theorie

GoogLeNet basiert auf der **Inception-Architektur**, die die Idee verfolgt, verschiedene Filtergrößen in einem einzigen Layer zu kombinieren. Anstatt sich nur auf einen Filtertyp (z. B. 3x3) zu beschränken, verwendet GoogLeNet eine Kombination aus verschiedenen Filtergrößen wie 1x1, 3x3 und 5x5. Diese Filter werden parallel angewendet und die Ergebnisse werden zusammengeführt. Die Architektur enthält außerdem einen **auxiliary classifier** (Nebenklassifikator), der während des Trainings zusätzliche Gradienten bereitstellt und so die Lernrate des Modells verbessert.

GoogLeNet hat im Vergleich zu anderen CNN-Architekturen wie AlexNet und VGG eine deutlich geringere Anzahl an Parametern, was zu einer besseren Rechenleistung führt.

Aufbau und Funktionsweise

GoogLeNet verwendet mehrere wichtige Architekturelemente:

* **Inception-Module**: Diese Module bestehen aus parallelen Convolutional-Schichten mit unterschiedlichen Filtergrößen, die gleichzeitig auf den Eingabedaten angewendet werden. Die Ergebnisse werden anschließend zusammengeführt (konkateniert).
* **1x1 Convolution**: Dieses spezielle Convolutional-Element wird eingesetzt, um die Anzahl der Kanäle zu reduzieren, was die Berechnungen effizienter macht, ohne dabei die Modellleistung zu beeinträchtigen.
* **Auxiliary Classifier**: Ein zusätzlicher Klassifikator wird in die tiefen Schichten des Modells eingefügt, um das Training zu unterstützen und die Gradientenzurückpropagierung zu verbessern.

Die Architektur von GoogLeNet ist somit sehr tief, aber aufgrund des Einsatzes von **1x1 Convolutionen und Inception-Modulen** bleibt die Anzahl der Parameter im Vergleich zu anderen Modellen wie VGG relativ gering.

Anwendungen

GoogLeNet wird in verschiedenen Bereichen der Bildverarbeitung eingesetzt:

* **Bildklassifikation**: GoogLeNet kann für die Klassifikation von Bildern in vordefinierte Kategorien verwendet werden.
* **Objekterkennung und -lokalisierung**: GoogLeNet kann auch dazu verwendet werden, Objekte in Bildern zu erkennen und ihre Position zu bestimmen.
* **Medizinische Bildanalyse**: GoogLeNet wird in der medizinischen Bildanalyse eingesetzt, z. B. bei der Erkennung von Tumoren oder anderen Auffälligkeiten in Röntgenbildern.
* **Selbstfahrende Autos**: In der Fahrzeugerkennung und der Umweltanalyse wird GoogLeNet ebenfalls verwendet, um die Bildinformationen aus Kameras zu analysieren.

Verlässlichkeit und Performance

GoogLeNet ist bekannt für seine **hohe Leistung bei relativ niedrigen Rechenkosten**. Durch die effiziente Nutzung von **1x1 Convolutionen** und **Inception-Modulen** bietet GoogLeNet eine ausgezeichnete Genauigkeit, während die Anzahl der Parameter im Vergleich zu anderen CNN-Architekturen reduziert bleibt. Dies führt zu einer schnelleren Verarbeitung und einem geringeren Speicherbedarf.

Ein Vorteil von GoogLeNet ist, dass es **sehr tief** ist und in der Lage ist, komplexe Bildmerkmale zu extrahieren. Dies verbessert die Leistung bei komplexen Bildklassifikationsaufgaben. Allerdings kann GoogLeNet für kleinere Datensätze oder weniger komplexe Aufgaben möglicherweise überdimensioniert sein, was zu einem Überanpassungsrisiko führen könnte. Zudem sind die **Auxiliary Classifier**-Module eine zusätzliche Herausforderung, da ihre Wirksamkeit vom Anwendungsfall abhängt.

Insgesamt bleibt GoogLeNet jedoch eine **leistungsstarke Wahl für große, komplexe Bildklassifikationsaufgaben**, insbesondere wenn die Rechenressourcen begrenzt sind.

Imports

Die notwendigen Imports für dieses Modell sind:

from .training\_basemodel import BaseModelTorch

from torchvision import models

 **BaseModelTorch**: Dies ist die Basisklasse, die allgemeine Funktionen für das Modell-Training und -Speichern bietet und aus dem Modul training\_basemodel importiert wird.

 **models**: Wird aus torchvision importiert und stellt vortrainierte Deep-Learning-Modelle zur Verfügung. In diesem Fall wird models.googlenet verwendet, um GoogLeNet zu laden.

training\_googlenet\_pytorch.py

In diesem Fall wird GoogLeNet ohne vortrainierte Gewichte instanziiert.

class GoogLeNetModel(BaseModelTorch):

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

# GoogLeNet wird mit der angegebenen Anzahl an Klassen instanziiert

super().\_\_init\_\_(models.googlenet(weights=None, init\_weights=True, num\_classes=classes), model\_save\_path)

 **models.googlenet(weights=None, init\_weights=True, num\_classes=classes)**: Erstellt eine Instanz von GoogLeNet ohne vortrainierte Gewichte und mit der spezifizierten Anzahl von Klassen (durch classes), was bedeutet, dass das Modell für das gegebene Klassifikationsproblem angepasst wird.

 **super().\_\_init\_\_(models.googlenet(...), model\_save\_path)**: Der Aufruf an die super()-Methode übergibt das Modell an die Basisklasse BaseModelTorch, die das Modell dann trainiert, speichert und verwaltet.

###### InceptionV3

Das **InceptionV3**-Modell ist eine leistungsstarke Architektur für die Bildklassifikation, die auf Convolutional Neural Networks (CNNs) basiert. Es wurde von Google entwickelt und gehört zur **Inception-Serie** von Modellen, die dafür bekannt sind, verschiedene Arten von Convolutional-Schichten in einer einzigen Schicht zu kombinieren, um eine höhere Effizienz und Genauigkeit zu erreichen. Durch den Einsatz von **Inception-Modulen** und anderen Optimierungen ist das Modell besonders gut geeignet, um auf komplexen Bilddatensätzen wie ImageNet hervorragende Ergebnisse zu erzielen.

Das Modell nutzt fortschrittliche Techniken, um die Anzahl der Parameter zu minimieren und gleichzeitig die Genauigkeit zu maximieren. Aufgrund seiner Flexibilität und Effizienz wird InceptionV3 in einer Vielzahl von Anwendungsbereichen eingesetzt, insbesondere in der Bildklassifikation und der Objekterkennung. In dieser Sektion werden die theoretischen Grundlagen des Modells, seine Funktionsweise sowie seine Anwendungen und Leistungsmerkmale erläutert.

Theorie

Das **InceptionV3**-Modell nutzt die **Inception-Architektur**, bei der mehrere Convolutional-Schichten mit unterschiedlichen Filtergrößen und Pooling-Schichten parallel ausgeführt werden. Dies ermöglicht es dem Modell, Merkmale auf verschiedenen Skalen und mit unterschiedlichen Auflösungen gleichzeitig zu extrahieren. Dieser parallele Ansatz erhöht die Fähigkeit des Modells, komplexe Muster und Strukturen in den Eingabebildern zu erkennen, was besonders nützlich bei großen und hochdimensionalen Datensätzen ist.

Im Gegensatz zu traditionellen CNN-Architekturen, die nur eine bestimmte Filtergröße verwenden, fügt InceptionV3 **verschiedene Filtergrößen** in einem einzigen Layer zusammen, sodass das Modell mehr Informationen aus den Bildern extrahieren kann. Dies wird durch den Einsatz von **1x1-Convolutions** und **Max-Pooling** erreicht, die die Komplexität des Modells reduzieren und gleichzeitig die Leistung steigern.

Aufbau und Funktionsweise

Das InceptionV3-Modell ist in mehrere **Inception-Module** unterteilt. Ein Inception-Modul besteht aus mehreren parallelen Convolutional-Schichten, wobei jede Schicht eine andere Filtergröße hat. Diese Filtergrößen reichen von 1x1 über 3x3 bis zu 5x5, und es wird auch **Max-Pooling** verwendet. Die Ergebnisse dieser unterschiedlichen Filtergrößen werden dann kombiniert, um die besten Merkmale aus den Eingabebildern zu extrahieren.

Die Architektur von InceptionV3 verwendet auch **Batch Normalization** zur Stabilisierung des Trainingsprozesses, indem sie die internen Datenwerte während des Trainings standardisiert. Dies hilft dabei, das Training schneller und stabiler zu machen. Zusätzlich werden Techniken wie **Dropout** und **Label Smoothing** verwendet, um Überanpassung (Overfitting) zu verhindern.

Am Ende der Convolutional-Schichten gibt es eine **Fully Connected-Schicht**, die die extrahierten Merkmale in eine Klassifikationsvorhersage umwandelt. Das Modell kann daher in verschiedenen Anwendungsbereichen eingesetzt werden, bei denen eine präzise und schnelle Klassifikation erforderlich ist.

Anwendungen

Das **InceptionV3**-Modell wird in einer Vielzahl von Anwendungsbereichen der Computer Vision eingesetzt, insbesondere in der Bildklassifikation und Objekterkennung. Zu den häufigsten Anwendungen gehören:

* **Bildklassifikation:** InceptionV3 wird häufig für die Klassifikation von Objekten in Bildern verwendet. Dies umfasst Anwendungen wie die Erkennung von Tieren, Fahrzeugen oder anderen spezifischen Objekten in Bildern.
* **Medizinische Bildverarbeitung:** Das Modell wird auch in der medizinischen Bildanalyse verwendet, um etwa Tumore oder andere Anomalien in Röntgenbildern und MRI-Scans zu erkennen.
* **Autonomes Fahren:** In der Automobilindustrie wird InceptionV3 zur Erkennung von Verkehrsschildern, Fußgängern, Fahrzeugen und anderen wichtigen Objekten auf der Straße eingesetzt.
* **Gesichtserkennung:** Durch seine Fähigkeit zur genauen Bildklassifikation wird es auch in Systemen zur Gesichtserkennung eingesetzt.

Verlässlichkeit und Performance

Das **InceptionV3**-Modell hat sich aufgrund seiner **genauen Klassifikationsfähigkeit** und **Effizienz** in der Praxis als sehr zuverlässig erwiesen. Es hat hohe Performance-Werte auf verschiedenen Standard-Datensätzen wie **ImageNet** erzielt und ist aufgrund seiner Architektur besonders gut geeignet, komplexe Bilddaten mit vielen unterschiedlichen Merkmalen zu verarbeiten.

Allerdings ist das Modell auch recht rechenintensiv, insbesondere bei großen Bildgrößen und Datensätzen. Dennoch hat es sich als eines der leistungsfähigsten Modelle für die Bildklassifikation etabliert, da es in der Lage ist, tiefere, komplexere Merkmale zu extrahieren, ohne dass die Anzahl der Parameter exponentiell wächst. In der Praxis hat es sich als weniger anfällig für Überanpassung gezeigt, vor allem bei Verwendung von **Transfer Learning**, wo vortrainierte Modelle auf neuen Datensätzen feinabgestimmt werden.

Imports

Die notwendigen Imports für dieses Modell sind:

from torchvision import models

from .training\_basemodel import BaseModelTorch

 **models.inception\_v3**: Wird aus dem torchvision-Modul importiert und stellt das vortrainierte **InceptionV3**-Modell zur Verfügung, das speziell für die Bildklassifikation entwickelt wurde.

 **BaseModelTorch**: Ein benutzerdefiniertes Basismodell, das grundlegende Funktionen für das Training und Speichern des Modells bereitstellt.

training\_inception\_pytorch.py

In diesem Fall wird das **InceptionV3-Modell** mit der folgenden Struktur instanziiert:

class InceptionModel(BaseModelTorch):

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

super().\_\_init\_\_(models.inception\_v3(weights=None, num\_classes=classes), model\_save\_path)

 **models.inception\_v3(weights=None, num\_classes=classes)**: Instanziiert das InceptionV3-Modell ohne vortrainierte Gewichte (weights=None) und legt die Anzahl der Zielklassen (num\_classes) fest. Wenn das Modell mit neuen Daten trainiert wird, können die Gewichte auf den Datensatz angepasst werden.

 **super().init(...)**: Der Aufruf an super() ruft den Konstruktor der Basisklasse **BaseModelTorch** auf, die für das Setup, Training und Speichern des Modells verantwortlich ist.

###### MaxVit

**MaxViT** ist ein modernes neuronales Netzwerk, das ursprünglich für die Bildklassifikation entwickelt wurde und dabei eine neue Art der Architektur einführt, die die Leistung von Vision Transformer (ViT)-Modellen mit der Effizienz von Convolutional Neural Networks (CNNs) kombiniert. MaxViT zeichnet sich durch seine innovative Nutzung von **Max-Pooling** in Verbindung mit der Transformer-Architektur aus, wodurch es in der Lage ist, sowohl lokale als auch globale Informationen aus den Eingabebildern zu extrahieren. Dieses Modell hat sich als besonders leistungsfähig bei der Arbeit mit großen Bilddatensätzen und komplexen Bildklassifikationsaufgaben erwiesen.

In dieser Sektion wird das **MaxViT-Modell** detailliert erläutert, einschließlich seiner Funktionsweise, der zugrunde liegenden Theorie, der Anwendungen und seiner Performance-Eigenschaften.

Theorie

MaxViT ist eine Weiterentwicklung von **Vision Transformers (ViT)**, wobei anstelle der klassischen globalen und lokalen **Self-Attention-Mechanismen** die **Max-Pooling**-Techniken in der Verarbeitung von Bildinformationen integriert werden. MaxViT kombiniert den **Transformer**-Ansatz mit lokalen Bildmerkmalen, wodurch das Modell in der Lage ist, sowohl globale als auch lokale Zusammenhänge im Bild zu erkennen.

Im Gegensatz zu den traditionellen ViT-Architekturen, die eine lineare Projektion der Bildteile verwenden, ermöglicht MaxViT durch seine **maximalen Pooling-Schichten** eine bessere Erfassung der lokalen Bildmerkmale und eine effizientere Nutzung der globalen Bildinformationen. Dadurch wird die Rechenleistung verbessert und die Modellgenauigkeit in vielen Anwendungsfällen gesteigert.

Aufbau und Funktionsweise

MaxViT ist eine hybride Architektur, die Transformer-Mechanismen mit **maximalen Pooling-Operationen** kombiniert. Diese Kombination ermöglicht es dem Modell, die Vorteile beider Ansätze zu nutzen: Die Transformer-Architektur bietet eine starke Fähigkeit zur Modellierung langfristiger Abhängigkeiten und globaler Kontextinformationen, während das Max-Pooling dazu beiträgt, die detaillierten lokalen Informationen beizubehalten und effizienter zu verarbeiten.

Die Architektur besteht aus mehreren Schichten, die eine **Multiskalen-Aufmerksamkeit** (multi-scale attention) ermöglichen. Dabei werden Bildteile auf unterschiedlichen Skalen verarbeitet, was die Fähigkeit des Modells zur Erkennung von Objekten in unterschiedlichen Auflösungen verbessert. In der Praxis bedeutet dies, dass MaxViT in der Lage ist, sowohl große Objekte als auch feine Details in Bildern präzise zu erfassen.

Anwendungen

Das **MaxViT-Modell** findet Anwendung in verschiedenen Bereichen der **Bildklassifikation** und **Objekterkennung**. Zu den häufigsten Anwendungsgebieten gehören:

* **Bildklassifikation:** MaxViT wird häufig für die Klassifikation von Objekten in Bildern verwendet. Dies umfasst Anwendungen wie die Erkennung von Tieren, Fahrzeugen oder anderen spezifischen Objekten in Bildern.
* **Medizinische Bildverarbeitung:** Wie viele andere moderne Modelle wird MaxViT auch in der medizinischen Bildverarbeitung verwendet, z.B. zur Erkennung von Tumoren oder Anomalien in Röntgenbildern oder MRI-Scans.
* **Autonomes Fahren:** In der autonomen Fahrzeugtechnik wird MaxViT für die Erkennung von Straßenzeichen, Fußgängern und anderen relevanten Objekten verwendet.
* **Verkehrsüberwachung und Sicherheitsanwendungen:** MaxViT wird auch zur Objekterkennung in Echtzeit-Videos und Überwachungsaufnahmen eingesetzt.

Verlässlichkeit und Performance

Das **MaxViT-Modell** hat sich als äußerst leistungsfähig erwiesen, besonders bei großen und komplexen Datensätzen. Es nutzt die Stärken der Transformer-Architektur in Verbindung mit einer besseren Verarbeitung von lokalen Bilddetails durch Max-Pooling, was zu einer verbesserten Modellgenauigkeit führt.

In Bezug auf die **Verlässlichkeit** ist MaxViT weniger anfällig für das klassische Overfitting-Problem, das häufig bei anderen großen Transformer-Modellen auftritt, da es durch die Mischung von lokalen und globalen Bildmerkmalen robuster wird. Die **Performance** des Modells ist auf verschiedenen Benchmark-Datensätzen hervorragend und übertrifft oft andere Modelle, die nur auf Transformer-Architekturen basieren. Bei der Verwendung von **Transfer Learning** erzielt das Modell in der Regel noch bessere Ergebnisse.

Dennoch ist es auch ein rechenintensives Modell, das bei sehr großen Bilddatensätzen oder realzeitfähigen Anwendungen ressourcenintensiv sein kann. Die Bereitstellung des Modells auf Ressourcenkonfigurationen mit begrenztem Speicher oder Rechenleistung erfordert oft eine Optimierung des Modells.

Imports

Die notwendigen Imports für dieses Modell sind:

from torchvision import models

from .training\_basemodel import BaseModelTorch

 **models.maxvit\_t**: Wird aus dem torchvision-Modul importiert, um das vortrainierte **MaxViT**-Modell zu verwenden.

 **BaseModelTorch**: Ein benutzerdefiniertes Basismodell, das grundlegende Funktionen für das Training und Speichern des Modells bereitstellt.

Training\_maxvit\_pytorch.py

In diesem Fall wird das **MaxViT-Modell** mit der folgenden Struktur instanziiert:

class MaxVitModel(BaseModelTorch):

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

super().\_\_init\_\_(models.maxvit\_t(weights=None, num\_classes=classes), model\_save\_path)

 **models.maxvit\_t(weights=None, num\_classes=classes)**: Instanziiert das MaxViT-Modell ohne vortrainierte Gewichte (weights=None) und legt die Anzahl der Zielklassen (num\_classes) fest. Bei der Verwendung mit neuen Daten kann das Modell für spezifische Klassifikationen angepasst werden.

 **super().init(...)**: Der Aufruf an super() ruft den Konstruktor der Basisklasse **BaseModelTorch** auf, der für das Setup, Training und Speichern des Modells verantwortlich ist.

###### MNASNet

**MNASNet** ist ein effizientes neuronales Netzwerk, das speziell für mobile Geräte und ressourcenbeschränkte Umgebungen entwickelt wurde. Es wurde mit dem Ziel entworfen, eine hohe Genauigkeit bei gleichzeitig geringerem Rechenaufwand zu erzielen, sodass es für den Einsatz auf Geräten mit begrenzten Hardware-Ressourcen geeignet ist. Die Architektur von MNASNet basiert auf dem **MobileNet**-Ansatz und verwendet eine **Neural Architecture Search (NAS)**-Technik, um das Modell zu optimieren und die Effizienz zu maximieren.

In diesem Abschnitt wird das **MNASNet-Modell** erläutert, einschließlich seiner Funktionsweise, der zugrunde liegenden Theorie, Anwendungen und Performance-Eigenschaften.

Theorie

MNASNet wurde durch den Einsatz von **Neural Architecture Search (NAS)** entwickelt, einer Methode zur automatisierten Suche nach der besten Architektur für ein Modell. Die NAS-Technik ermöglicht es, Modelle zu entwickeln, die speziell für bestimmte Zielplattformen und -ressourcen optimiert sind. Im Fall von MNASNet liegt der Fokus auf der Verbesserung der Effizienz und Genauigkeit von Mobilgeräten, die typischerweise eine geringere Rechenleistung und weniger Speicher zur Verfügung haben als Desktop-Computer oder Server.

Die Architektur von MNASNet basiert auf den Prinzipien der **Depthwise Separable Convolutions**, ähnlich wie bei anderen mobilen Architekturen wie **MobileNetV2**. Dies ermöglicht es, die Anzahl der Parameter und die Berechnungen signifikant zu reduzieren, während die Leistung erhalten bleibt. Der Hauptvorteil von NAS in diesem Fall liegt in der Fähigkeit, die Struktur des Netzwerks zu optimieren, ohne auf manuelle Modellanpassung angewiesen zu sein.

Aufbau und Funktionsweise

MNASNet verwendet die **Depthwise Separable Convolutions**, eine Technik, bei der die Convolution in zwei Schritte unterteilt wird: eine Tiefen- und eine Punktwellen-Faltung. Dies ermöglicht eine erhebliche Reduzierung der Berechnungen im Vergleich zu Standard-Convolutions. Das Modell optimiert die **Kompromisse zwischen Genauigkeit und Effizienz**, was es ideal für den Einsatz auf mobilen Geräten macht.

Das Modell wird durch **Neural Architecture Search (NAS)** optimiert, wobei eine spezialisierte Suchstrategie verwendet wird, um die besten Hyperparameter und Netzwerkarchitekturen zu finden. Die Suchmethoden konzentrieren sich auf die Minimierung der Anzahl der Rechenoperationen und die Maximierung der Genauigkeit, was zu einem optimierten Modell führt, das sowohl effizient als auch leistungsstark ist.

Anwendungen

Das **MNASNet-Modell** ist besonders nützlich in Szenarien, in denen **Rechenressourcen** begrenzt sind und dennoch eine hohe Modellgenauigkeit erforderlich ist. Zu den häufigsten Anwendungsgebieten gehören:

* **Mobile Bildklassifikation:** MNASNet wird häufig auf mobilen Geräten eingesetzt, um Aufgaben der Bildklassifikation zu lösen, z. B. bei der Objekterkennung in Bildern, die durch Smartphones aufgenommen wurden.
* **Echtzeit-Bildverarbeitung:** Aufgrund seiner Effizienz wird MNASNet auch für die Echtzeit-Bildverarbeitung auf mobilen Geräten verwendet, zum Beispiel für Augmented Reality (AR)-Anwendungen oder die Bilderkennung in Live-Videos.
* **Autonome Geräte:** MNASNet wird in autonomer Robotik oder Geräten mit begrenzter Rechenleistung verwendet, da es eine hohe Effizienz und gleichzeitig eine präzise Modellierung bietet.
* **Gesichtserkennung:** In mobilen Geräten und Sicherheitsanwendungen wird MNASNet auch zur Gesichtserkennung und zur Durchführung von Sicherheitsüberprüfungen eingesetzt.

Verlässlichkeit und Performance

MNASNet ist für seine Effizienz bekannt und bietet **eine hohe Performance auf mobilen Geräten** bei gleichzeitig geringem Ressourcenverbrauch. Es zeichnet sich durch eine geringere **Latenzzeit** aus, was es besonders für Echtzeitanwendungen geeignet macht. Die Leistung des Modells ist jedoch von der spezifischen Hardware abhängig, da es optimiert wurde, um mit den begrenzten Rechenressourcen von Mobilgeräten kompatibel zu sein.

Im Vergleich zu anderen mobilen Architekturen wie **MobileNetV2** oder **EfficientNet** bietet MNASNet ähnliche oder sogar bessere Ergebnisse bei einer geringeren Anzahl von Operationen. Die **Verlässlichkeit** des Modells ist hoch, da es speziell für den Einsatz in ressourcenbeschränkten Umgebungen entwickelt wurde, jedoch muss beachtet werden, dass es bei größeren Datensätzen oder höherkomplexen Aufgaben nicht immer die gleiche Genauigkeit wie größere, rechenintensive Modelle bietet.

Imports

Die notwendigen Imports für dieses Modell sind:

from torchvision import models

from .training\_basemodel import BaseModelTorch

 **models.mnasnet1\_0**: Wird aus dem torchvision-Modul importiert, um das vortrainierte **MNASNet1.0**-Modell zu verwenden.

 **BaseModelTorch**: Ein benutzerdefiniertes Basismodell, das grundlegende Funktionen für das Training und Speichern des Modells bereitstellt.

Training\_mnasnet\_pytorch.py

In diesem Fall wird das **MNASNet-Modell** mit der folgenden Struktur instanziiert:

class MNASNetModel(BaseModelTorch):

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

super().\_\_init\_\_(models.mnasnet1\_0(weights=None, num\_classes=classes), model\_save\_path)

 **models.mnasnet1\_0(weights=None, num\_classes=classes)**: Instanziiert das MNASNet1.0-Modell ohne vortrainierte Gewichte (weights=None) und legt die Anzahl der Zielklassen (num\_classes) fest.

 **super().init(...)**: Der Aufruf an super() ruft den Konstruktor der Basisklasse **BaseModelTorch** auf, der für das Setup, Training und Speichern des Modells verantwortlich ist.

###### MobileNetV2

**MobileNetV2** ist eine weiterentwickelte Version des ursprünglichen **MobileNet-Modells**, das für den Einsatz auf mobilen Geräten und in ressourcenbeschränkten Umgebungen optimiert wurde. Es verwendet die **Depthwise Separable Convolutions** und führt zusätzlich neue Architekturen wie die **Inverted Residuals** ein, um die Effizienz und Leistung zu verbessern. Die Hauptmotivation hinter MobileNetV2 ist es, die Genauigkeit zu maximieren, während gleichzeitig die Anzahl der Berechnungen und Parameter minimiert wird, um eine schnelle Ausführung auf mobilen Geräten zu gewährleisten.

In diesem Abschnitt wird das **MobileNetV2-Modell** erläutert, einschließlich seiner Funktionsweise, der zugrunde liegenden Theorie, Anwendungen und Performance-Eigenschaften.

Theorie

**MobileNetV2** baut auf den Prinzipien des ursprünglichen **MobileNet** auf, jedoch mit verbesserten Architekturen für eine noch größere Effizienz. Ein wesentliches Merkmal von MobileNetV2 ist die Einführung von **Inverted Residuals**. Diese verwenden **depthwise separable convolutions** und eine **linear bottleneck architecture**, um die Effizienz und Leistung zu steigern, während die Modellgröße und Berechnungen gering gehalten werden.

Das Modell ist darauf optimiert, **schnell** und **speichereffizient** auf mobilen Geräten zu arbeiten, ohne die Genauigkeit signifikant zu beeinträchtigen. Diese Architektur hat MobileNetV2 zu einem beliebten Modell für den Einsatz in mobilen Anwendungen gemacht, bei denen Ressourcen wie Rechenleistung und Speicher begrenzt sind.

Aufbau und Funktionsweise

MobileNetV2 verwendet eine **lineare Bottleneck-Architektur**, bei der die Eingabe durch eine **1x1-Konvolution** und dann durch eine **depthwise separable convolution** verarbeitet wird. Dies reduziert die Anzahl der Parameter und Rechenoperationen erheblich. Darüber hinaus verwendet das Modell **Inverted Residuals**, bei denen die Convolutionen zuerst durch eine Expansionsphase laufen und dann durch eine Kompressionsphase gehen. Diese Architektur ermöglicht es, komplexe Funktionen effizient zu lernen und gleichzeitig die Größe des Modells zu minimieren.

Die **Depthwise Separable Convolutions** sind eine Methode, um die Berechnungen in Convolutional Neural Networks (CNNs) zu optimieren, indem jede Faltung nur über die Tiefe der Eingabedaten durchgeführt wird, anstatt über die gesamte Eingabe. Dadurch wird der Rechenaufwand verringert, ohne die Modellleistung erheblich zu beeinträchtigen.

Anwendungen

**MobileNetV2** ist aufgrund seiner Effizienz und geringen Ressourcenanforderungen in verschiedenen Bereichen weit verbreitet:

* **Bildklassifikation**: MobileNetV2 wird häufig auf mobilen Geräten für die Klassifikation von Bildern und die Erkennung von Objekten in Echtzeit verwendet.
* **Gesichtserkennung**: Das Modell wird zur Gesichtserkennung in mobilen Anwendungen und Sicherheitslösungen eingesetzt.
* **Echtzeit-Bildverarbeitung**: Aufgrund seiner geringen Latenz eignet sich MobileNetV2 hervorragend für Echtzeit-Bildverarbeitungsaufgaben, z. B. in Augmented Reality (AR)-Anwendungen und mobilen Kameras.
* **Objekterkennung**: In Anwendungen wie der automatischen Objekterkennung in Videos und der intelligenten Überwachung wird MobileNetV2 aufgrund seiner Effizienz bevorzugt.

Verlässlichkeit und Performance

**MobileNetV2** bietet eine hervorragende **Performance** und ist für **mobilitätseffiziente** Berechnungen optimiert. Das Modell weist eine geringe **Latenz** auf und liefert dennoch eine hohe **Genauigkeit**, insbesondere in mobilen Anwendungen mit begrenzten Hardware-Ressourcen. Es ist jedoch wichtig zu beachten, dass die Leistung des Modells je nach Anwendungsfall und eingesetztem Gerät variieren kann. In bestimmten Szenarien könnte MobileNetV2 bei sehr großen Datensätzen oder besonders komplexen Aufgaben durch leistungsfähigere Modelle ersetzt werden.

Trotz seiner Optimierung für mobile Geräte zeigt MobileNetV2 auch **auf Desktop-Geräten** oder leistungsfähigeren Maschinen eine beeindruckende Performance, insbesondere bei der Analyse von großen Bilddaten.

Imports

Die notwendigen Imports für dieses Modell sind:

from torchvision import models

from .training\_basemodel import BaseModelTorch

 **models.mobilenet\_v2**: Wird aus dem torchvision-Modul importiert, um das vortrainierte **MobileNetV2-Modell** zu verwenden.

 **BaseModelTorch**: Ein benutzerdefiniertes Basismodell, das grundlegende Funktionen für das Training und Speichern des Modells bereitstellt.

training\_mobilenetv2\_pytorch.py

In diesem Fall wird das **MobileNetV2-Modell** mit der folgenden Struktur instanziiert:

class MobileNetV2Model(BaseModelTorch):

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

super().\_\_init\_\_(models.mobilenet\_v2(weights=None, num\_classes=classes), model\_save\_path)

 **models.mobilenet\_v2(weights=None, num\_classes=classes)**: Instanziiert das MobileNetV2-Modell ohne vortrainierte Gewichte (weights=None) und legt die Anzahl der Zielklassen (num\_classes) fest.

 **super().init(...)**: Der Aufruf an super() ruft den Konstruktor der Basisklasse **BaseModelTorch** auf, der für das Setup, Training und Speichern des Modells verantwortlich ist.

MobileNetV3

**MobileNetV3** ist eine Weiterentwicklung des MobileNet-Ansatzes, der für mobile und ressourcenbeschränkte Geräte optimiert wurde. Es kombiniert die Vorteile der vorherigen MobileNet-Versionen und führt neue Techniken zur Leistungssteigerung ein, um die Effizienz weiter zu erhöhen. MobileNetV3 verwendet eine verbesserte Architektur, die durch **Automated Neural Architecture Search (NAS)** und **Netzwerkquantisierung** zur Optimierung des Modells beiträgt. Dies macht MobileNetV3 zu einem sehr leistungsfähigen Modell, das auch auf Geräten mit begrenzten Ressourcen exzellente Ergebnisse liefert.

In diesem Abschnitt wird das **MobileNetV3-Modell** detailliert erläutert, einschließlich der zugrunde liegenden Theorie, seiner Funktionsweise, typischen Anwendungen und Leistungsmerkmale.

Theorie

**MobileNetV3** baut auf den Prinzipien der **MobileNetV1** und **MobileNetV2** auf und verwendet zusätzlich eine **Netzwerkarchitektur-Suche (NAS)**, um die Architektur für das spezifische Ziel von mobilen Geräten zu optimieren. Dabei werden sowohl **Depthwise Separable Convolutions** als auch **Inverted Residuals** verwendet, um ein Modell zu schaffen, das sowohl **effizient** als auch **leistungsfähig** ist. MobileNetV3 integriert auch **Squeeze-and-Excitation (SE)-Blöcke**, um die Modellkapazität zu erweitern, ohne die Berechnungen zu erhöhen.

Durch den Einsatz von **NAS** wird die optimale Architektur für die Zielhardware automatisch ermittelt, sodass MobileNetV3 in vielen mobilen und eingebetteten Systemen eine der besten Optionen ist. Das Modell ist bekannt für seine hohe Leistung bei gleichzeitig geringem Ressourcenverbrauch.

Aufbau und Funktionsweise

MobileNetV3 kombiniert zwei verschiedene Varianten für unterschiedliche Anforderungen:

1. **MobileNetV3-Large**: Diese Version ist für leistungsfähigere Geräte und Anwendungen optimiert, bei denen eine höhere Genauigkeit und eine größere Modellgröße erforderlich sind.
2. **MobileNetV3-Small**: Diese Version ist für kleinere Geräte mit niedrigeren Leistungsanforderungen konzipiert und bietet eine optimierte Leistung bei geringerem Ressourcenverbrauch.

Das Modell nutzt **depthwise separable convolutions**, um die Berechnungen effizient zu gestalten. Die **Inverted Residuals** und **Squeeze-and-Excitation-Blöcke** helfen dabei, die Repräsentationsfähigkeit des Modells zu verbessern, ohne die Effizienz zu beeinträchtigen.

MobileNetV3 hat auch den Vorteil, dass es in Echtzeit-Computing-Anwendungen eine hohe **Rechenleistung** bei gleichzeitig **geringem Stromverbrauch** bietet.

Anwendungen

**MobileNetV3** wird in einer Vielzahl von Anwendungen verwendet, in denen **mobile Geräte** oder **eingebettete Systeme** eingesetzt werden, und die Anforderungen an Rechenleistung und Speicher begrenzt sind. Typische Anwendungsbereiche sind:

* **Bildklassifikation**: MobileNetV3 wird in mobilen Apps zur Klassifikation von Bildern oder Objekten verwendet, insbesondere in der medizinischen Bildverarbeitung, der Bildsuche und der Echtzeit-Erkennung.
* **Gesichtserkennung**: Wie andere MobileNet-Versionen wird MobileNetV3 auch für die Gesichtserkennung in mobilen Sicherheits- und Kamerasystemen eingesetzt.
* **Objekterkennung und Tracking**: MobileNetV3 wird häufig in Überwachungs- und Smart-City-Anwendungen verwendet, in denen Objekte in Echtzeit erkannt und verfolgt werden müssen.
* **Augmented Reality (AR)**: Durch seine hohe Leistung bei geringer Latenz eignet sich MobileNetV3 auch hervorragend für AR-Anwendungen, bei denen eine schnelle Verarbeitung von Live-Video erforderlich ist.

Verlässlichkeit und Performance

**MobileNetV3** bietet eine **hohe Genauigkeit** bei gleichzeitig **geringem Ressourcenverbrauch**. Das Modell zeigt in vielen mobilen Anwendungen eine **hervorragende Performance** und eignet sich für den Einsatz auf Geräten mit geringem Stromverbrauch und begrenzten Rechenressourcen. Besonders in **Echtzeit-Anwendungen** ist MobileNetV3 aufgrund seiner schnellen Verarbeitung und geringen Latenz äußerst zuverlässig.

Da MobileNetV3 auf die Leistung von mobilen Geräten ausgelegt ist, bietet es eine **schnelle Modellverarbeitung**, jedoch könnte die Leistung in sehr komplexen Aufgaben oder mit großen Datensätzen durch leistungsstärkere Modelle übertroffen werden.

Die **Optimierung für mobile Geräte** macht MobileNetV3 zu einer bevorzugten Wahl für viele praktische Anwendungen im Bereich der Computer Vision und Echtzeit-Bildverarbeitung.

Imports

Die notwendigen Imports für dieses Modell sind:

from torchvision import models

from .training\_basemodel import BaseModelTorch

 **models.mobilenet\_v3\_large**: Wird aus dem torchvision-Modul importiert, um das **MobileNetV3-Large-Modell** zu verwenden.

 **BaseModelTorch**: Ein benutzerdefiniertes Basismodell, das grundlegende Funktionen für das Training und Speichern des Modells bereitstellt.

training\_mobilenetv3\_pytorch.py

In diesem Fall wird das **MobileNetV3-Modell** mit der folgenden Struktur instanziiert:

class MobileNetV3Model(BaseModelTorch):

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

super().\_\_init\_\_(models.mobilenet\_v3\_large(weights=None, num\_classes=classes), model\_save\_path)

* **models.mobilenet\_v3\_large(weights=None, num\_classes=classes)**: Instanziiert das **MobileNetV3-Large-Modell** ohne vortrainierte Gewichte (weights=None) und legt die Anzahl der Zielklassen (num\_classes) fest.
* **super().init(...)**: Der Aufruf an super() ruft den Konstruktor der Basisklasse **BaseModelTorch** auf, der für das Setup, Training und Speichern des Modells verantwortlich ist.

###### RegNet

**RegNet** ist eine Architektur für Convolutional Neural Networks (CNNs), die ursprünglich von Facebook AI Research (FAIR) entwickelt wurde. Die RegNet-Modelle basieren auf der Idee der **regulären Netzwerkanpassung** (hence the name "RegNet") und wurden entworfen, um die Modellkomplexität mit einer **einfacheren, systematischen Herangehensweise** zu verbessern, ohne die Leistung zu beeinträchtigen. Im Gegensatz zu anderen Architekturen, die eine **explizite Handhabung von Hyperparametern** wie Kanalanzahl und Netzwerkbreite erfordern, konzentriert sich RegNet darauf, die **Architektur durch automatisierte Verfahren** zu optimieren und zu regulieren.

RegNet zeichnet sich durch eine **einheitliche Struktur** aus, bei der die Netzwerkarchitektur aus **modularen Bausteinen** besteht, die miteinander kombiniert werden, um eine leistungsfähige und skalierbare Architektur zu erhalten.

Theorie

**RegNet** verwendet eine systematische Methode zur Modifikation der Architektur eines Netzwerks, indem es **Kanalbreite** und **Tiefe** dynamisch und gleichmäßig anpasst. Ein weiteres zentrales Konzept von RegNet ist die **Skalierung der Modellgröße**, was es ermöglicht, Netzwerke mit einer Vielzahl von Größen zu entwickeln, die dann auf unterschiedliche Rechenressourcen und Anwendungsfälle abgestimmt werden können.

Die Schlüsselidee von RegNet ist die Optimierung der Architektur mit Hilfe eines algorithmischen Ansatzes, der sowohl **effizient** als auch **skalierbar** ist. Dies geschieht durch eine regulierte Variation der Kanalanzahl und -struktur über verschiedene Blöcke des Modells.

Aufbau und Funktionsweise

Das RegNet-Modell verwendet eine **modulare Architektur**, bei der die Struktur des Netzwerks auf den spezifischen Anforderungen an Rechenleistung und Modellgröße basiert. RegNet ermöglicht eine **automatische Architekturwahl**, sodass das Modell je nach den Anforderungen der Zielhardware angepasst werden kann.

Ein einzigartiges Merkmal der RegNet-Architektur ist die **Verwendung von separaten Blöcken**, die sich in ihrer Anzahl und Breite unterscheiden können, aber gleichzeitig eine **einheitliche Struktur** behalten. Dies führt zu einer verbesserten **Skalierbarkeit**, was bedeutet, dass es leicht an verschiedene Konfigurationen angepasst werden kann.

Anwendungen

**RegNet** ist besonders gut für Anwendungen geeignet, bei denen **Skalierbarkeit** und **Effizienz** von entscheidender Bedeutung sind. Häufige Anwendungsgebiete sind:

* **Bildklassifikation**: RegNet bietet exzellente Ergebnisse in der Bildklassifikation, sowohl auf kleinen als auch auf großen Datensätzen.
* **Objekterkennung**: Aufgrund seiner Flexibilität und Leistungsfähigkeit wird RegNet auch in anspruchsvollen Aufgaben der Objekterkennung verwendet.
* **Transfer Learning**: Durch die Vielseitigkeit der Architektur eignet sich RegNet hervorragend für den Einsatz in Transfer-Learning-Szenarien, bei denen vortrainierte Modelle auf neue Aufgaben angewendet werden.
* **Verteilte Systeme**: RegNet-Modelle sind durch ihre modulare und skalierbare Struktur besonders gut für den Einsatz in verteilten Rechensystemen und Cloud-Umgebungen geeignet.

Verlässlichkeit und Performance

RegNet hat sich als **stabil und zuverlässig** in vielen gängigen Aufgaben der Bildverarbeitung erwiesen. Die Modelle bieten nicht nur **hohe Leistung** auf gängigen Benchmarks wie **ImageNet**, sondern zeigen auch eine exzellente **Skalierbarkeit**, was sie ideal für den Einsatz in verschiedenen **Hardware-Konfigurationen** macht. Besonders in Bezug auf die **Rechenleistung** bietet RegNet einen sehr guten Kompromiss zwischen **Modellgröße** und **Leistungsfähigkeit**.

Die **Skalierbarkeit** von RegNet ermöglicht es, das Modell für Anwendungen von mobilen Geräten bis hin zu High-Performance-Computing-Plattformen zu optimieren. Ein weiterer Vorteil von RegNet ist seine **Robustheit** gegenüber **Überanpassung**, was es zu einem zuverlässigen Modell für eine Vielzahl von praktischen Anwendungen macht.

Imports

Die notwendigen Imports für dieses Modell sind:

from torchvision import models

from .training\_basemodel import BaseModelTorch

 **models.regnet\_y\_400mf**: Wird aus dem torchvision-Modul importiert, um das **RegNetY 400MF-Modell** zu verwenden.

 **BaseModelTorch**: Ein benutzerdefiniertes Basismodell, das grundlegende Funktionen für das Training und Speichern des Modells bereitstellt.

Training\_regnet\_pytorch.py

In diesem Fall wird das **RegNet-Modell** mit der folgenden Struktur instanziiert:

class RegNetModel(BaseModelTorch):

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

super().\_\_init\_\_(models.regnet\_y\_400mf(weights=None, num\_classes=classes), model\_save\_path)

 **models.regnet\_y\_400mf(weights=None, num\_classes=classes)**: Instanziiert das **RegNetY 400MF-Modell** ohne vortrainierte Gewichte (weights=None) und legt die Anzahl der Zielklassen (num\_classes) fest.

 **super().init(...)**: Der Aufruf an super() ruft den Konstruktor der Basisklasse **BaseModelTorch** auf, der für das Setup, Training und Speichern des Modells verantwortlich ist.

###### ResNet

**ResNet (Residual Network)** ist eine der bekanntesten Architekturen im Bereich des tiefen Lernens. Sie wurde von Microsoft Research entwickelt und stellt einen **Meilenstein** bei der Entwicklung von Convolutional Neural Networks (CNNs) dar. Das ResNet-Modell ist besonders dafür bekannt, dass es durch die **Verwendung von Residualverbindungen** (Kurzschlussverbindungen) die Probleme des **Vanishing Gradients** über tiefere Netzwerke hinweg löst. Dies ermöglicht das Training von sehr tiefen Netzwerken und verbessert deren Leistung auf verschiedenen Aufgaben der Bildverarbeitung, wie **Bildklassifikation**, **Objekterkennung** und **Semantische Segmentierung**.

Theorie

Die **ResNet-Architektur** basiert auf dem Konzept der **Residualverbindungen**, die es ermöglichen, Informationen direkt durch das Netzwerk zu übertragen, ohne dass sie durch jedes einzelne Layer transformiert werden müssen. Dies verbessert den **Gradientenfluss** und erleichtert das Training von sehr tiefen Netzwerken.

Ein Residualblock besteht aus einem Hauptpfad, der die gewöhnliche Convolutional-Operation durchführt, sowie einem **Shortcut-Pfad**, der die Eingabe direkt an den Ausgang des Blocks weitergibt. Diese Residualverbindungen ermöglichen es, tiefere Netzwerke zu trainieren, da die Gradienten während der Backpropagation nicht so stark verschwinden.

ResNet wurde in verschiedenen Varianten entwickelt, die auf die spezifischen Anforderungen und Ressourcen von Anwendungen abgestimmt sind, darunter **ResNet-18**, **ResNet-34**, **ResNet-50** und **ResNet-152**.

Aufbau und Funktionsweise

Das **ResNet-Model** besteht aus einer Reihe von **Residualblöcken**, die sich über das gesamte Netzwerk hinweg wiederholen. Jeder dieser Blöcke enthält eine **Konvolution**, eine **Batch-Normalisierung** und eine **ReLU-Aktivierung**, gefolgt von einer **Residualverbindung**. Die Residualverbindung sorgt dafür, dass der Eingabewert direkt zum Ausgang des Blocks weitergeleitet wird, wodurch der Gradientenfluss verbessert wird und das Modell tiefere Netzwerke effizienter trainieren kann.

Ein zentraler Bestandteil von ResNet ist die **tiefe Architektur** mit vielen Schichten, was es dem Modell ermöglicht, sehr komplexe Funktionen zu erlernen. Beispielsweise verwendet ResNet50 50 Schichten, und ResNet152 verwendet 152 Schichten, was die Flexibilität und Leistungsfähigkeit des Modells erhöht.

Anwendungen

**ResNet** hat sich in verschiedenen Anwendungen der Computer Vision als äußerst leistungsfähig erwiesen:

* **Bildklassifikation**: ResNet ist ein sehr häufig eingesetztes Modell für Bildklassifikationsaufgaben, sowohl auf Standarddatensätzen wie **ImageNet** als auch auf spezifischen, maßgeschneiderten Datensätzen.
* **Objekterkennung**: Durch die starke Leistung in der Bildklassifikation eignet sich ResNet auch hervorragend für komplexe Aufgaben der **Objekterkennung**, wie **Faster R-CNN**.
* **Semantische Segmentierung**: Die Flexibilität von ResNet bei der Verarbeitung von Bildern und die Tiefe der Architektur machen es auch für die **semantische Segmentierung** von Bildern geeignet.
* **Medizinische Bildverarbeitung**: ResNet wird zunehmend in der medizinischen Bildverarbeitung verwendet, z.B. bei der Analyse von **Röntgenbildern** oder der **Krebsdiagnose**.

Verlässlichkeit und Performance

Die **ResNet-Architektur** hat sich in vielen Benchmark-Tests als äußerst zuverlässig und leistungsfähig erwiesen. Das Modell bietet eine **exzellente Leistung** auf einer Vielzahl von Aufgaben, insbesondere wenn es um tiefere Netzwerke geht. Aufgrund der **Residualverbindungen** ist es nicht nur in der Lage, tiefere Modelle effizient zu trainieren, sondern zeigt auch **hohe Genauigkeit** in der Vorhersage.

**ResNet50** hat sich als besonders geeignet für große Datensätze und komplexe Aufgaben erwiesen. Durch die Residualverbindungen kann es die **Überanpassung** verhindern und die **Stabilität** während des Trainings gewährleisten, was es zu einem robusten Modell für den praktischen Einsatz macht.

Imports

Die notwendigen Imports für dieses Modell sind:

from torchvision import models

from .training\_basemodel import BaseModelTorch

 **models.resnet50**: Wird aus dem torchvision-Modul importiert, um das **ResNet50-Modell** zu verwenden.

 **BaseModelTorch**: Ein benutzerdefiniertes Basismodell, das grundlegende Funktionen für das Training und Speichern des Modells bereitstellt.

training\_resnet\_pytorch.py

In diesem Fall wird das **ResNet50-Modell** mit der folgenden Struktur instanziiert:

class ResNetModel(BaseModelTorch):

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

super().\_\_init\_\_(models.resnet50(weights=None, num\_classes=classes), model\_save\_path, model\_name\_extension="\_resnet")

* **models.resnet50(weights=None, num\_classes=classes)**: Instanziiert das **ResNet50-Modell** ohne vortrainierte Gewichte (weights=None) und legt die Anzahl der Zielklassen (num\_classes) fest.
* **super().init(...)**: Der Aufruf an super() ruft den Konstruktor der Basisklasse **BaseModelTorch** auf, der für das Setup, Training und Speichern des Modells verantwortlich ist.

ResNeXt

**ResNeXt** ist eine Erweiterung der bekannten **ResNet**-Architektur und wurde entwickelt, um die Leistung und Effizienz in tiefen neuronalen Netzwerken zu verbessern. Es basiert auf dem Prinzip der **gruppierten Convolution**, was zu einer verbesserten Parametereffizienz und einer besseren Modellleistung führt. Das ResNeXt-Modell nutzt eine **modulare Architektur**, die es ermöglicht, die Anzahl der Operationen in einem Netzwerk effizient zu skalieren, ohne dabei die Modellkomplexität unnötig zu erhöhen.

ResNeXt führt das Konzept der **Cardinality** (die Anzahl der Gruppen innerhalb eines Blocks) ein, wodurch die Netzwerkarchitektur flexibler und leistungsfähiger wird. Es wurde gezeigt, dass ResNeXt in vielen Computer Vision Aufgaben überlegene Ergebnisse erzielt und gleichzeitig die **Rechenkomplexität** reduziert.

Theorie

Die **ResNeXt-Architektur** basiert auf der **gruppierten Convolution**, einem Konzept, bei dem die Convolutional-Schichten in mehrere Gruppen unterteilt werden. Anstatt die gesamte Eingabe durch die gleiche Convolution zu verarbeiten, wird sie in verschiedene Gruppen aufgeteilt, die parallel verarbeitet werden. Dies führt zu einer **reduzierten Rechenlast** und erhöht gleichzeitig die **Kapazität** des Modells.

Die **Cardinality** ist eine entscheidende Eigenschaft von ResNeXt. Sie bezeichnet die Anzahl der parallelen "Pfadgruppen", die innerhalb eines Blocks existieren. Eine höhere Cardinality ermöglicht es dem Netzwerk, mehr **unabhängige** Merkmale zu extrahieren und dadurch eine **größere Vielfalt an Repräsentationen** zu lernen. Im Vergleich zu traditionellen Netzwerken wie ResNet bietet ResNeXt eine bessere Skalierbarkeit, da die Performance des Modells effizienter verbessert werden kann, ohne die Anzahl der Parameter drastisch zu erhöhen.

Aufbau und Funktionsweise

Das ResNeXt-Modell nutzt eine **modulare Struktur**, bei der die Grundbausteine des Netzwerks sogenannte **ResNeXt-Blöcke** sind. Jeder dieser Blöcke besteht aus mehreren parallel ausgeführten Convolutional-Schichten (die in Gruppen unterteilt sind), was zu einer signifikanten Effizienzsteigerung führt.

* **Cardinality**: Die Anzahl der parallelen Convolutional-Pfade in jedem Block. Eine höhere Cardinality führt zu einer besseren Modellleistung bei gleichzeitig reduzierter Rechenzeit.
* **Skalierbarkeit**: Im Vergleich zu klassischen Netzwerken ist ResNeXt einfach skalierbar, da durch das Hinzufügen von weiteren Gruppen die Modellleistung gesteigert werden kann, ohne die Netzwerkstruktur grundlegend ändern zu müssen.
* **Gruppierte Convolution**: Diese Technik ermöglicht eine **erhöhte Repräsentationskapazität** ohne eine signifikante Zunahme der Anzahl der Parameter.

Anwendungen

**ResNeXt** hat sich als sehr effektiv in verschiedenen Bereichen der **Computer Vision** erwiesen:

* **Bildklassifikation**: ResNeXt hat exzellente Ergebnisse auf großen Datensätzen wie **ImageNet** erzielt und wird für Klassifikationsaufgaben auf Bilddaten weit verbreitet eingesetzt.
* **Objekterkennung**: Durch die Möglichkeit, die Modellgröße zu variieren und gleichzeitig eine hohe Leistung zu erzielen, wird ResNeXt häufig für anspruchsvolle Aufgaben der **Objekterkennung** verwendet.
* **Semantische Segmentierung**: Die Architektur wird auch für **Segmentierungsaufgaben** verwendet, bei denen es darauf ankommt, jedes Pixel in einem Bild einer bestimmten Klasse zuzuordnen.

Verlässlichkeit und Performance

Die **Leistung von ResNeXt** ist bemerkenswert, da das Modell **hochgradig skalierbar** ist und durch die Verwendung der gruppierten Convolution effizienter arbeitet. Im Vergleich zu traditionellen Modellen wie ResNet bietet ResNeXt eine bessere **Leistung pro Parameter**, was zu einer besseren **Modellgenauigkeit** bei vergleichbarer Rechenleistung führt.

* **Konsistenz**: Das Modell hat eine hohe Konsistenz bei der Modellgenauigkeit und zeigt auf vielen Datensätzen zuverlässige Ergebnisse.
* **Robustheit**: ResNeXt ist besonders robust bei der Verarbeitung komplexer Datensätze und kann auch bei weniger idealen Trainingsbedingungen eine hohe Leistung erzielen.

Imports

Die notwendigen Imports für dieses Modell sind:

from torchvision import models

from .training\_basemodel import BaseModelTorch

 **models.resnext50\_32x4d**: Wird aus dem torchvision-Modul importiert, um das **ResNeXt50-Modell** zu verwenden, das 50 Schichten und 32 Gruppen mit jeweils 4 Kanälen verwendet.

 **BaseModelTorch**: Ein benutzerdefiniertes Basismodell, das grundlegende Funktionen für das Training und Speichern des Modells bereitstellt.

training\_resnext\_pytorch.py

In diesem Fall wird das **ResNeXt50-Modell** mit der folgenden Struktur instanziiert:

class ResNeXtModel(BaseModelTorch):

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

super().\_\_init\_\_(models.resnext50\_32x4d(weights=None, num\_classes=classes), model\_save\_path, model\_name\_extension="\_resnext")

 **models.resnext50\_32x4d(weights=None, num\_classes=classes)**: Instanziiert das **ResNeXt50-Modell** ohne vortrainierte Gewichte und legt die Anzahl der Zielklassen (num\_classes) fest.

 **super().init(...)**: Der Aufruf an super() ruft den Konstruktor der Basisklasse **BaseModelTorch** auf, der für das Setup, Training und Speichern des Modells verantwortlich ist.

WideResNet

**WideResNet** ist eine Erweiterung der klassischen **ResNet**-Architektur, die durch eine breitere Netzstruktur eine bessere Leistung und höhere Effizienz erzielen soll. Die Idee hinter WideResNet ist es, die Netzwerkbreite zu erhöhen, anstatt die Tiefe zu erweitern, was in vielen Fällen zu besseren Ergebnissen führt, ohne die Rechenkosten exponentiell zu steigern. Diese Architektur hat sich insbesondere bei der Bildklassifikation als leistungsfähig erwiesen und wird häufig für Aufgaben wie Objekterkennung und semantische Segmentierung verwendet.

Theorie

WideResNet basiert auf dem klassischen **ResNet**, das für seine Residual-Blöcke bekannt ist, die den Gradientenfluss verbessern und das Training von sehr tiefen Netzwerken ermöglichen. Im Unterschied zu ResNet erhöht WideResNet jedoch die **Breite** der einzelnen Blöcke, anstatt die **Tiefe** zu erhöhen. Dies führt zu einer besseren Modellleistung und verringert das Risiko der Überanpassung.

1. **Residual-Verbindungen**: Wie bei ResNet verwendet auch WideResNet Residual-Verbindungen, die den Gradientenfluss verbessern und es dem Modell ermöglichen, schneller zu konvergieren.
2. **Erhöhte Breite**: Anstelle der Erhöhung der Tiefe des Netzwerks wird die Anzahl der Filter in den Convolutional-Schichten erhöht. Dies hilft, mehr Features zu lernen, was zu einer besseren Leistung führen kann.
3. **Tiefe des Netzwerks**: WideResNet nutzt die gleiche grundlegende Architektur wie ResNet, jedoch mit breiteren Schichten. Dies kann die Komplexität des Modells reduzieren und gleichzeitig die Leistung verbessern.

Aufbau und Funktionsweise

WideResNet verwendet wie ResNet **Residual-Blöcke**, die es dem Modell ermöglichen, tiefer zu werden, ohne dass es zu Problemen mit dem Gradientenfluss kommt. Die wichtigsten Merkmale dieser Architektur sind:

1. **Residual-Verbindungen**: In einem Residual-Block wird die Eingabe direkt an die Ausgangsschicht des Blocks weitergegeben, wodurch die Trainingszeit und die Konvergenzgeschwindigkeit verbessert werden.
2. **Breitere Blöcke**: Die breitere Netzstruktur von WideResNet bedeutet, dass mehr Filter in den Convolutional-Schichten verwendet werden. Dies ermöglicht es dem Netzwerk, mehr Merkmale zu lernen und damit eine bessere Leistung zu erzielen.
3. **Kompromiss zwischen Tiefe und Breite**: Im Vergleich zu klassischen tiefen Netzwerken wie ResNet ist WideResNet flacher, aber breiter, was eine effizientere Berechnung bei gleichzeitig hoher Leistung ermöglicht.

Anwendungen

WideResNet findet Anwendung in einer Vielzahl von Bereichen der **Computer Vision**, darunter:

* **Bildklassifikation**: Wie bei anderen Convolutional Neural Networks wird WideResNet oft für die Klassifikation von Bildern verwendet und hat in vielen Benchmark-Datensätzen wie ImageNet exzellente Ergebnisse erzielt.
* **Objekterkennung**: Das Modell wird auch bei der Objekterkennung eingesetzt, um verschiedene Objekte in Bildern zu identifizieren und zu klassifizieren.
* **Semantische Segmentierung**: Die breitere Architektur von WideResNet hat sich ebenfalls bei der semantischen Segmentierung als vorteilhaft erwiesen, da sie in der Lage ist, detaillierte Merkmale zu lernen und zu extrahieren.

Verlässlichkeit und Performance

WideResNet hat sich als **sehr leistungsfähig** bei Bildklassifikationsaufgaben erwiesen und bietet durch die erhöhte Breite des Netzwerks eine verbesserte Leistung im Vergleich zu traditionellen CNNs. Ein großer Vorteil von WideResNet ist, dass es mit einer geringeren Anzahl an Schichten eine ähnliche oder sogar bessere Leistung als tiefere Netzwerke erzielt. Allerdings hat die erhöhte Breite auch ihre Kosten, insbesondere in Bezug auf den Speicherbedarf und die Rechenleistung. Daher sollte das Modell je nach Anwendungsfall und verfügbaren Ressourcen sorgfältig ausgewählt werden.

Imports

Die notwendigen Imports für dieses Modell sind:

from torchvision import models

from .training\_basemodel import BaseModelTorch

 **models.wide\_resnet50\_2**: Wird aus dem torchvision-Modul importiert, um das WideResNet-Modell mit der Basisversion von 50 Schichten und einer breiten Netzwerkstruktur zu verwenden.

 **BaseModelTorch**: Ein benutzerdefiniertes Basismodell, das grundlegende Funktionen für das Training und Speichern des Modells bereitstellt.

training\_\_sklearn.py

In diesem Fall wird der **WideResNet** wie folgt instanziiert:

class WideResNetModel(BaseModelTorch):

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

super().\_\_init\_\_(models.wide\_resnet50\_2(weights=None, num\_classes=classes), model\_save\_path)

 **models.wide\_resnet50\_2(weights=None, num\_classes=classes)**: Instanziiert das **WideResNet**-Modell mit einer Breite von 50 Schichten und einer breiten Netzwerkstruktur, wobei die Anzahl der Zielklassen (num\_classes) festgelegt wird.

 **super().init(...)**: Der Aufruf an super() ruft den Konstruktor der Basisklasse **BaseModelTorch** auf, der für das Setup, Training und Speichern des Modells verantwortlich ist.

###### ShuffleNet

**ShuffleNet** ist eine effiziente Architektur für neuronale Netzwerke, die speziell entwickelt wurde, um sowohl eine hohe Leistung als auch eine geringe Rechenlast zu gewährleisten, was es ideal für mobile und eingebettete Systeme macht. Es nutzt eine innovative Technik namens **Channel Shuffle**, um die Effizienz von Convolutional-Neural-Networks (CNNs) zu verbessern. ShuffleNet zielt darauf ab, eine niedrige Modellkomplexität bei gleichzeitig hoher Genauigkeit zu bieten, wodurch es eine ausgezeichnete Wahl für Echtzeit- und ressourcenbegrenzte Anwendungen darstellt.

Theorie

Die **ShuffleNet-Architektur** basiert auf der Idee, die Effizienz von Convolutional-Neural-Networks durch zwei Haupttechniken zu verbessern:

1. **Group Convolution**: Anstatt die Eingabe durch ein einziges Convolutional-Filter zu leiten, wird sie in Gruppen unterteilt, und jede Gruppe wird unabhängig voneinander verarbeitet. Dies reduziert die Anzahl der Parameter und verbessert die Effizienz.
2. **Channel Shuffle**: Diese Technik wurde entwickelt, um die Informationsflüsse zwischen den Gruppen zu kombinieren und so die **modellinterne Kommunikation** zu verbessern, ohne die Rechenkomplexität stark zu erhöhen. Es sorgt dafür, dass die verschiedenen Gruppen ihre Merkmale besser austauschen können, was zu einer besseren Leistung führt.

Durch diese beiden Methoden kann ShuffleNet die Leistung eines Netzwerks steigern und gleichzeitig den Rechenaufwand deutlich reduzieren, was es ideal für den Einsatz auf Geräten mit begrenzten Ressourcen macht.

Aufbau und Funktionsweise

**ShuffleNet** nutzt eine Architektur, die auf **Group Convolutions** und **Channel Shuffle** aufbaut. Die Architektur besteht aus mehreren **ShuffleNet-Blöcken**, die für die Parallelverarbeitung von Eingabedaten verantwortlich sind. Jeder Block enthält:

* **Group Convolutions**: Hierbei werden Convolutional-Operationen in Gruppen unterteilt, die parallel ausgeführt werden.
* **Channel Shuffle**: Nach der Gruppenkonvolution werden die Kanäle zwischen den Gruppen neu gemischt, um die Kommunikation zwischen den Gruppen zu verbessern.

Die **Effizienz** von ShuffleNet basiert darauf, dass es bei einer geringen Modellgröße und einem reduzierten Rechenaufwand dennoch in der Lage ist, **hohe Genauigkeit** zu erreichen. Diese Eigenschaften machen das Modell besonders geeignet für **mobile Geräte** und Anwendungen, bei denen sowohl Geschwindigkeit als auch geringen Energieverbrauch erforderlich sind.

Anwendungen

**ShuffleNet** wird in vielen Bereichen eingesetzt, insbesondere dort, wo Rechenressourcen begrenzt sind und Echtzeitanwendungen erforderlich sind:

* **Mobile Bildklassifikation**: Dank seiner Effizienz ist ShuffleNet eine beliebte Wahl für die Bildklassifikation auf mobilen Geräten, bei denen die Rechenleistung und der Energieverbrauch optimiert werden müssen.
* **Objekterkennung in Echtzeit**: Durch die schnelle und ressourcenschonende Verarbeitung eignet sich ShuffleNet auch hervorragend für die **Objekterkennung** in Echtzeit, etwa in autonomen Fahrzeugen oder mobilen Kamerasystemen.
* **Gesichtserkennung**: ShuffleNet hat sich auch als nützlich für **Gesichtserkennungssoftware** erwiesen, bei der eine schnelle und genaue Identifizierung von Gesichtern erforderlich ist.

Verlässlichkeit und Performance

Die **Leistung von ShuffleNet** ist beeindruckend, insbesondere wenn man die reduzierte Anzahl der Parameter und den optimierten Rechenaufwand berücksichtigt. Obwohl es auf **geringe Modellkomplexität** und **schnelle Berechnungen** ausgelegt ist, erzielt es dennoch eine **hohe Genauigkeit** bei Aufgaben der Bildklassifikation und Objekterkennung.

* **Hohe Effizienz**: Durch den Einsatz von **Group Convolutions** und **Channel Shuffle** ist ShuffleNet in der Lage, eine hohe Rechenleistung bei gleichzeitig niedrigem Energieverbrauch zu bieten.
* **Skalierbarkeit**: Das Modell lässt sich problemlos auf verschiedene Geräte und Anforderungen skalieren, was es zu einer vielseitigen Wahl für verschiedene Anwendungen macht.
* **Limitierte Anfälligkeit für Überanpassung**: Durch seine effiziente Architektur zeigt ShuffleNet eine gute Robustheit und ist weniger anfällig für Überanpassung, da es mit weniger Parametern und einer besseren Informationsverteilung arbeitet.

Imports

Die notwendigen Imports für dieses Modell sind:

from torchvision import models

from .training\_basemodel import BaseModelTorch

 **models.shufflenet\_v2\_x1\_0**: Wird aus dem torchvision-Modul importiert, um das **ShuffleNetV2-Modell** zu verwenden. Hierbei handelt es sich um eine Version des ShuffleNet-Modells mit einer **Base-Architektur** für schnelle und effiziente Berechnungen.

 **BaseModelTorch**: Ein benutzerdefiniertes Basismodell, das grundlegende Funktionen für das Training und Speichern des Modells bereitstellt.

training\_shufflenet\_pytorch.py

In diesem Fall wird das **ShuffleNetV2-Modell** mit der folgenden Struktur instanziiert:

class ShuffleNetModel(BaseModelTorch):

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

super().\_\_init\_\_(models.shufflenet\_v2\_x1\_0(weights=None, num\_classes=classes), model\_save\_path)

 **models.shufflenet\_v2\_x1\_0(weights=None, num\_classes=classes)**: Instanziiert das **ShuffleNetV2-Modell** ohne vortrainierte Gewichte und legt die Anzahl der Zielklassen (num\_classes) fest.

 **super().init(...)**: Der Aufruf an super() ruft den Konstruktor der Basisklasse **BaseModelTorch** auf, der für das Setup, Training und Speichern des Modells verantwortlich ist.

###### SqueezeNet

**SqueezeNet** ist eine kompakte und effiziente Convolutional Neural Network (CNN)-Architektur, die entwickelt wurde, um die Modellgröße zu minimieren und gleichzeitig eine hohe Klassifikationsleistung zu erzielen. Durch den Einsatz von **Fire-Modulen** und der Reduzierung der Anzahl der Parameter hat SqueezeNet den Vorteil einer signifikant kleineren Modellgröße im Vergleich zu traditionellen CNN-Architekturen, ohne dabei zu viel an Genauigkeit zu verlieren. Diese Eigenschaften machen SqueezeNet besonders geeignet für den Einsatz auf Geräten mit begrenzten Rechenressourcen, wie etwa mobilen Geräten oder eingebetteten Systemen.

Theorie

**SqueezeNet** ist eine leichte Architektur, die speziell entwickelt wurde, um mit einer minimalen Anzahl von Parametern zu arbeiten, ohne die Genauigkeit erheblich zu beeinträchtigen. Die Architektur verwendet **Fire-Module**, eine spezielle Schicht, die die Anzahl der Parameter reduziert. Ein **Fire-Module** besteht aus zwei Teilen:

1. **Squeeze**-Teil: Dieser Teil reduziert die Anzahl der Eingabekanäle mithilfe einer **1x1 Convolution**, was zu einer geringeren Modellgröße führt.
2. **Expand**-Teil: Dieser Teil erhöht die Anzahl der Kanäle wieder mit **1x1 Convolutions** und **3x3 Convolutions**, um die Komplexität und Ausdruckskraft des Modells zu gewährleisten.

Die Hauptidee hinter SqueezeNet ist es, den Anteil der Parameter, die durch die **1x1 Convolution** erzeugt werden, zu maximieren, während gleichzeitig die **3x3 Convolutions** nur in einem kleinen Maßstab angewendet werden, um die Komplexität zu steuern.

Aufbau und Funktionsweise

Die Architektur von **SqueezeNet** setzt auf eine effiziente Verwendung von **1x1 Convolutions** und **Fire-Modulen**, um das Modell zu komprimieren und gleichzeitig eine akzeptable Leistung zu gewährleisten. Im Wesentlichen kann man sagen, dass SqueezeNet eine **kleine Modellgröße** hat, was es zu einem bevorzugten Modell für **Echtzeit- und mobile Anwendungen** macht, bei denen die Ressourcen begrenzt sind.

Der Aufbau des Modells umfasst mehrere Fire-Module, die zusammen die Eingabedaten durchlaufen, um die Ausgabe zu berechnen. Dabei wird die Anzahl der Parameter durch die Verwendung der **Squeeze-Operation** auf eine minimalistische Weise kontrolliert, was den Speicherbedarf und die Rechenleistung reduziert.

Anwendungen

**SqueezeNet** ist aufgrund seiner geringen Modellgröße und hohen Effizienz besonders geeignet für die folgenden Anwendungen:

* **Echtzeit-Bildklassifikation**: Dank seiner geringen Größe und schnellen Verarbeitung eignet sich SqueezeNet hervorragend für mobile Geräte und eingebettete Systeme, bei denen schnelle Bildklassifikationen erforderlich sind.
* **Gesichtserkennung**: SqueezeNet wird auch häufig für **Gesichtserkennungsanwendungen** auf mobilen Geräten und in Kamerasystemen eingesetzt.
* **Objekterkennung auf mobilen Geräten**: Durch seine Effizienz und geringe Modellgröße ist SqueezeNet eine beliebte Wahl für die **Objekterkennung in Echtzeit** auf mobilen Plattformen.

Verlässlichkeit und Performance

SqueezeNet bietet eine **gute Balance** zwischen Modellgröße und Performance. Im Vergleich zu traditionellen CNN-Architekturen wie AlexNet oder VGGNet erzielt SqueezeNet mit deutlich weniger Parametern eine ähnliche Genauigkeit. Allerdings gibt es einige Einschränkungen, die bei der Auswahl dieses Modells berücksichtigt werden sollten:

* **Geringe Modellgröße**: Aufgrund der Verwendung von **1x1 Convolutions** und der kompakten **Fire-Module** hat SqueezeNet eine kleine Modellgröße, die es zu einer hervorragenden Wahl für ressourcenbeschränkte Geräte macht.
* **Energieeffizienz**: SqueezeNet benötigt weniger Rechenressourcen und verbraucht daher weniger Energie, was besonders für mobile und eingebettete Systeme von Vorteil ist.
* **Leistungseinbußen bei sehr komplexen Aufgaben**: SqueezeNet ist zwar für viele Anwendungen ausreichend, kann jedoch in sehr komplexen Aufgaben, bei denen größere Modelle erforderlich sind, eine geringere Leistung als größere Modelle wie ResNet oder DenseNet aufweisen.

Imports

Die notwendigen Imports für dieses Modell sind:

from torchvision import models

from .training\_basemodel import BaseModelTorch

 **models.squeezenet1\_0**: Wird aus dem torchvision-Modul importiert, um das **SqueezeNet v1.0-Modell** zu verwenden. Hierbei handelt es sich um die erste Version von SqueezeNet, die optimiert wurde, um mit einer kleinen Modellgröße eine vergleichbare Leistung zu erzielen.

 **BaseModelTorch**: Ein benutzerdefiniertes Basismodell, das grundlegende Funktionen für das Training und Speichern des Modells bereitstellt.

training\_squeezenet\_pytorch.py

In diesem Fall wird das **SqueezeNet-Modell** mit der folgenden Struktur instanziiert:

class SqueezeNetModel(BaseModelTorch):

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

super().\_\_init\_\_(models.squeezenet1\_0(weights=None, num\_classes=classes), model\_save\_path)

 **models.squeezenet1\_0(weights=None, num\_classes=classes)**: Instanziiert das **SqueezeNet-Modell** ohne vortrainierte Gewichte und legt die Anzahl der Zielklassen (num\_classes) fest.

 **super().init(...)**: Der Aufruf an super() ruft den Konstruktor der Basisklasse **BaseModelTorch** auf, der für das Setup, Training und Speichern des Modells verantwortlich ist.

###### SwinTransformer

**Swin Transformer** ist ein leistungsstarkes und skalierbares Modell, das die Architektur des klassischen **Transformers** für Computer Vision-Aufgaben adaptiert. Es wurde entwickelt, um die Vorteile der Transformer-Architektur, die ursprünglich für Natural Language Processing (NLP) entwickelt wurde, auf visuelle Aufgaben wie Bildklassifikation und Objekterkennung zu übertragen. Der "Shifted Window"-Ansatz des Swin Transformers hilft dabei, die Probleme der hohen Rechenkosten und des Speicherverbrauchs zu minimieren, die bei herkömmlichen Transformer-Modellen aufkommen können, wenn sie auf Bilder angewendet werden.

Theorie

Der **Swin Transformer** verwendet eine hierarchische Transformer-Architektur, die in mehreren Schichten aufbaut. Jede dieser Schichten besteht aus einem **Shifted Window Attention**-Mechanismus, bei dem die Bildinformationen in **Fenster (Windows)** unterteilt und innerhalb jedes Fensters Selbstaufmerksamkeit (Self-Attention) angewendet wird. Der "Shifted"-Teil des Ansatzes hilft dabei, die Probleme der lokalen Begrenzung der Selbstaufmerksamkeit zu überwinden und das Modell effizienter zu gestalten.

Im Vergleich zu traditionellen CNNs ist der Transformer in der Lage, **globale Abhängigkeiten** über das gesamte Bild hinweg zu modellieren, was ihm bei vielen Computer Vision-Aufgaben einen Vorteil verschafft. Der Swin Transformer hat den Vorteil, dass er **skalierbar** ist, was bedeutet, dass er problemlos an unterschiedliche Bildgrößen und -komplexitäten angepasst werden kann.

Aufbau und Funktionsweise

Der Aufbau des **Swin Transformers** lässt sich in mehrere Schichten unterteilen:

1. **Patch Partitioning**: Das Eingabebild wird zunächst in kleine **Patches** unterteilt.
2. **Shifted Window Attention**: In jedem Patch wird dann ein Selbstaufmerksamkeitsmechanismus angewendet. Der "Shifted"-Ansatz sorgt dafür, dass benachbarte Fenster miteinander interagieren können, was die Leistung des Modells verbessert.
3. **Hierarchische Architektur**: Die Architektur des Swin Transformers ist hierarchisch, was bedeutet, dass sie immer größere Merkmale und abstrakte Repräsentationen des Bildes in jeder Schicht lernt.

Das Modell verwendet auch **Leaky ReLU** als Aktivierungsfunktion, und in den späteren Schichten werden die Repräsentationen mit den **klassischen voll verbundenen Schichten** zusammengeführt, um die endgültige Klassifikation durchzuführen.

Anwendungen

**Swin Transformer** hat sich in mehreren Computer Vision-Aufgaben als äußerst leistungsfähig erwiesen und wird häufig in den folgenden Bereichen verwendet:

* **Bildklassifikation**: Dank seiner Fähigkeit, globale Bildbeziehungen zu modellieren, hat der Swin Transformer in großen Bildklassifikationsaufgaben wie ImageNet herausragende Ergebnisse erzielt.
* **Objekterkennung und Segmentierung**: Der Swin Transformer wird auch erfolgreich in **Objekterkennungsmodellen** wie **DEtection Transformers (DETR)** und **Mask R-CNN** eingesetzt.
* **Medizinische Bildverarbeitung**: Die Fähigkeit des Swin Transformers, subtile Bildmerkmale zu erfassen, macht ihn ideal für **medizinische Bildanalysen**, wie z. B. die Erkennung von Tumoren oder Anomalien in Röntgenbildern.

Verlässlichkeit und Performance

**Swin Transformer** hat sich als eines der leistungsfähigsten Modelle im Bereich der Computer Vision etabliert. Besonders hervorzuheben ist, dass es eine sehr **hohe Genauigkeit** erzielt, selbst bei komplexen Aufgaben wie der Objekterkennung. Das Modell bietet zudem **skalierbare Leistung**, was bedeutet, dass es in der Lage ist, mit unterschiedlich großen Datensätzen und Bildgrößen zu arbeiten, ohne die Leistung stark zu beeinträchtigen.

Jedoch gibt es auch Herausforderungen, insbesondere im Hinblick auf den **Rechenaufwand**. Transformer-Modelle im Allgemeinen sind tendenziell ressourcenintensiv, und der Swin Transformer bildet hier keine Ausnahme. Die **Shifted Window Attention**-Technik hilft zwar, den Speicherverbrauch zu reduzieren, aber dennoch bleibt die Verarbeitung von Bildern mit hohen Auflösungen auf weniger leistungsfähigen Geräten eine Herausforderung.

Imports

Die notwendigen Imports für dieses Modell sind:

from torchvision import models

from .training\_basemodel import BaseModelTorch

 **models.swin\_t**: Wird aus dem torchvision-Modul importiert, um das **Swin Transformer-Modell** zu verwenden.

 **BaseModelTorch**: Ein benutzerdefiniertes Basismodell, das grundlegende Funktionen für das Training und Speichern des Modells bereitstellt.

training\_\_sklearn.py

In diesem Fall wird das **Swin Transformer-Modell** wie folgt instanziiert:

class SwinTransformerModel(BaseModelTorch):

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

super().\_\_init\_\_(models.swin\_t(weights=None, num\_classes=classes), model\_save\_path)

 **models.swin\_t(weights=None, num\_classes=classes)**: Instanziiert das **Swin Transformer-Modell** ohne vortrainierte Gewichte und legt die Anzahl der Zielklassen (num\_classes) fest.

 **super().init(...)**: Der Aufruf an super() ruft den Konstruktor der Basisklasse **BaseModelTorch** auf, der für das Setup, Training und Speichern des Modells verantwortlich ist.

###### VGG

Das **VGG-Netzwerk** ist eines der bekanntesten Convolutional Neural Networks (CNN) und wurde 2014 von der Visual Geometry Group (VGG) der Universität Oxford entwickelt. Es hat sich in vielen Bildklassifikationsaufgaben als äußerst leistungsfähig erwiesen. Das Modell zeichnet sich durch seine einfache und einheitliche Architektur aus, die hauptsächlich auf **sehr tiefen Convolutional Layers** basiert, welche mit sehr kleinen Filtergrößen (3x3) arbeiten. Trotz der Einfachheit hat das VGG-Modell eine bemerkenswerte Leistung bei der Bildklassifikation erreicht und dient oft als Referenz in der Forschung und Industrie.

Theorie

**VGG-Netzwerk** verwendet eine sehr einfache, aber effektive Architektur, die sich auf tiefe, aufeinanderfolgende Convolutional Layers stützt. Die grundlegenden Prinzipien des VGG-Netzwerks umfassen:

1. **Kleine Filtergrößen (3x3)**: Anstatt größere Filter zu verwenden, wie es in vielen anderen CNN-Architekturen der Fall ist, verwendet VGG wiederholt 3x3-Filter, um Merkmale zu extrahieren. Diese Filter sind klein genug, um eine große Anzahl an Parametern zu vermeiden, bieten jedoch gleichzeitig genügend Ausdruckskraft für die Modellierung von Bildmerkmalen.
2. **Tiefe Architektur**: Das VGG-Modell besteht aus vielen Schichten, was bedeutet, dass das Modell tiefer ist als viele frühere Architekturen. Diese Tiefe ermöglicht es dem Netzwerk, mehr abstrakte Merkmale aus den Bilddaten zu extrahieren.
3. **Pooling Layers**: Die Architektur enthält mehrere **Max-Pooling-Schichten**, die die Dimensionen der Feature Maps nach jeder Convolution-Schicht reduzieren und so die Rechenlast verringern.
4. **Fully Connected Layers**: Nach den Convolutional- und Pooling-Schichten folgen die **voll verbundenen Schichten**, die dazu verwendet werden, die gewonnenen Merkmale zu kombinieren und die endgültige Klassifikation zu treffen.

Aufbau und Funktionsweise

Der **VGG-Netzwerkaufbau** kann als eine Reihe von Convolutional Layers und Max-Pooling-Schichten beschrieben werden, gefolgt von mehreren voll verbundenen Schichten. In der Regel wird das Modell in zwei Hauptteile unterteilt:

1. **Feature Extraction**: Dieser Teil des Netzwerks besteht aus einer Reihe von Convolutional Layers und Pooling-Schichten, die darauf abzielen, wichtige Merkmale im Bild zu extrahieren. Jede Convolutional-Schicht verwendet 3x3-Filter, und jede Pooling-Schicht reduziert die räumliche Dimension des Bildes.
2. **Klassifikation**: Nachdem die Merkmale extrahiert wurden, durchläuft das Modell mehrere voll verbundene Schichten, die die extrahierten Merkmale kombinieren und die endgültige Klassifikation vornehmen.

Die Architektur ist einfach und effektiv, was dazu beiträgt, dass sie auch heute noch eine Grundlage für viele Bildverarbeitungsaufgaben bildet.

Anwendungen

Das **VGG-Modell** wird in einer Vielzahl von Bildklassifikations- und Computer Vision-Aufgaben eingesetzt, darunter:

* **Bildklassifikation**: Aufgrund seiner hervorragenden Leistung bei der ImageNet-Bildklassifikation wird VGG häufig für allgemeine Bildklassifizierungsaufgaben verwendet.
* **Objekterkennung**: VGG wird auch als Basismodell für erweiterte Architekturen in der Objekterkennung verwendet.
* **Bildsegmentierung**: Das Modell wird für Aufgaben wie die Segmentierung von medizinischen Bildern und die Trennung von Bildobjekten verwendet.

Verlässlichkeit und Performance

Das VGG-Modell ist bekannt für seine **robuste Leistung** bei vielen Standard-Computer-Vision-Aufgaben. Es bietet eine gute Balance zwischen Einfachheit und Leistung, obwohl neuere Modelle wie ResNet oder Inception die Leistung in vielen Bereichen übertreffen.

Die **Verlässlichkeit** von VGG zeigt sich besonders in der Konsistenz der Ergebnisse bei der Bildklassifikation. Jedoch hat es einige Nachteile in Bezug auf die **Rechenanforderungen**, da es eine große Anzahl von Parametern enthält, was zu einem hohen Speicherverbrauch und längeren Trainingszeiten führen kann.

Trotz seiner Einfachheit ist das VGG-Netzwerk aufgrund seiner **effektiven Merkmalsrepräsentation** in vielen modernen Computer Vision-Aufgaben weiterhin weit verbreitet.

Imports

Die notwendigen Imports für dieses Modell sind:

from torchvision import models

from .training\_basemodel import BaseModelTorch

 **models.vgg16**: Wird aus dem torchvision-Modul importiert, um das **VGG-16-Modell** zu verwenden.

 **BaseModelTorch**: Ein benutzerdefiniertes Basismodell, das grundlegende Funktionen für das Training und Speichern des Modells bereitstellt.

training\_vgg\_pytorch.py

In diesem Fall wird das **VGG-Modell** wie folgt instanziiert:

class VGGModel(BaseModelTorch):

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

super().\_\_init\_\_(models.vgg16(weights=None, num\_classes=classes), model\_save\_path)

 **models.vgg16(weights=None, num\_classes=classes)**: Instanziiert das **VGG-16-Modell** ohne vortrainierte Gewichte und legt die Anzahl der Zielklassen (num\_classes) fest.

 **super().init(...)**: Der Aufruf an super() ruft den Konstruktor der Basisklasse **BaseModelTorch** auf, der für das Setup, Training und Speichern des Modells verantwortlich ist.

###### VisionTransformer

Der **Vision Transformer (ViT)** ist ein moderner Ansatz im Bereich der Computer Vision, der erstmals 2020 vorgestellt wurde. Im Gegensatz zu traditionellen Convolutional Neural Networks (CNNs), die speziell für Bilddaten entwickelt wurden, nutzt der Vision Transformer die Transformer-Architektur, die ursprünglich für Aufgaben im Bereich der natürlichen Sprachverarbeitung (NLP) entwickelt wurde. Der ViT hat die Fähigkeit, Bilddaten in Form von nicht überlappenden Patch-Sequenzen zu verarbeiten, was zu einer verbesserten Leistung in vielen Bildklassifikationsaufgaben führt.

Theorie

**Vision Transformer (ViT)** basiert auf der Idee, Bilder in kleinere Patches zu zerlegen und diese Patches als Eingabesequenzen für einen Transformer-Encoder zu verwenden. Der Grundgedanke ist, dass Transformer-Modelle, die sich in der NLP-Welt durchgesetzt haben, ebenfalls auf Bilddaten angewendet werden können, um globale Abhängigkeiten zwischen verschiedenen Bildbereichen zu erfassen.

1. **Patch-basierte Eingabe**: Ein Bild wird in kleine quadratische Patches (z. B. 16x16 Pixel) zerlegt. Jeder Patch wird in einen Vektor umgewandelt, der als Eingabe für das Modell dient. Dadurch wird das Bild in eine Sequenz von Patches umgewandelt, die ähnlich wie Wörter in einer Textsequenz behandelt werden.
2. **Transformer-Encoder**: Der Vision Transformer nutzt die Transformer-Architektur, die auf **Selbstaufmerksamkeit (Self-Attention)** basiert, um Beziehungen zwischen den Patches zu modellieren. Das Modell lernt, wie jeder Patch im Kontext des gesamten Bildes wichtig ist, wodurch es in der Lage ist, globale Bildinformationen zu erfassen.
3. **Klassifikation**: Am Ende des Transformer-Modells wird eine **Klassifikationsschicht** hinzugefügt, die die Merkmale aus den Patches nutzt, um eine Bildklassifikation zu erstellen.

Aufbau und Funktionsweise

Die **Funktionsweise des Vision Transformers** lässt sich wie folgt zusammenfassen:

1. **Patch-Extraktion**: Das Bild wird zunächst in kleine Patches unterteilt. Jeder Patch wird in einen Vektor umgewandelt, der dann als Eingabe in das Modell verwendet wird.
2. **Positionale Embeddings**: Da der Transformer keine inhärente Struktur für sequenzielle Daten hat, werden den Patches **positionale Embeddings** hinzugefügt, die die Position jedes Patches im Bild widerspiegeln. Diese Embeddings ermöglichen es dem Modell, räumliche Informationen zu nutzen.
3. **Selbstaufmerksamkeit (Self-Attention)**: Der Vision Transformer nutzt die **Self-Attention**-Mechanismus, der es dem Modell ermöglicht, den Einfluss jedes Patches auf alle anderen Patches zu berechnen. Dadurch kann das Modell globale Beziehungen zwischen verschiedenen Bereichen des Bildes erfassen.
4. **Klassifikation**: Nach der Verarbeitung durch den Transformer-Encoder wird eine Klassifikationsschicht hinzugefügt, die auf den erlernten Repräsentationen basiert, um die endgültige Klassifikation durchzuführen.

Anwendungen

Der Vision Transformer hat sich in einer Vielzahl von **Computer Vision**-Aufgaben als leistungsfähig erwiesen, darunter:

* **Bildklassifikation**: Der ViT wird häufig für Standard-Bildklassifikationsaufgaben verwendet und hat bei vielen Benchmark-Datensätzen wie ImageNet herausragende Leistungen erzielt.
* **Objekterkennung**: Vision Transformer-Modelle werden zunehmend auch für Aufgaben wie Objekterkennung eingesetzt, wo sie die Fähigkeit haben, verschiedene Objekte innerhalb eines Bildes zu lokalisieren und zu klassifizieren.
* **Semantische Segmentierung**: Auch in der Bildsegmentierung wird der ViT verwendet, um feine Details und komplexe Strukturen in Bildern zu erfassen.

Verlässlichkeit und Performance

Der Vision Transformer hat eine ausgezeichnete Leistung gezeigt, insbesondere wenn er mit **großen Datensätzen** und **vortrainierten Modellen** verwendet wird. Er bietet eine verbesserte Fähigkeit, globale Bildabhängigkeiten zu modellieren, und hat die Leistung vieler traditioneller CNN-Architekturen wie ResNet und Inception auf Benchmark-Datensätzen übertroffen.

Allerdings erfordert der Vision Transformer in der Regel **größere Datensätze und Rechenressourcen**, um eine konkurrenzfähige Leistung zu erzielen. Bei kleineren Datensätzen kann der ViT im Vergleich zu traditionellen CNNs möglicherweise schlechter abschneiden, da er die Fähigkeit zur Überanpassung (Overfitting) erhöht.

Imports

Die notwendigen Imports für dieses Modell sind:

from torchvision import models

from .training\_basemodel import BaseModelTorch

 **models.vit\_b\_16**: Wird aus dem torchvision-Modul importiert, um den **Vision Transformer** mit der Basisarchitektur und der Patch-Größe 16x16 zu verwenden.

 **BaseModelTorch**: Ein benutzerdefiniertes Basismodell, das grundlegende Funktionen für das Training und Speichern des Modells bereitstellt.

training\_visiontransformer\_pytorch.py

In diesem Fall wird der **Vision Transformer** wie folgt instanziiert:

class VisionTransformerModel(BaseModelTorch):

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

super().\_\_init\_\_(models.vit\_b\_16(weights=None, num\_classes=classes), model\_save\_path)

 **models.vit\_b\_16(weights=None, num\_classes=classes)**: Instanziiert den **Vision Transformer (ViT)** mit einer Basisarchitektur und einer Patch-Größe von 16x16 ohne vortrainierte Gewichte und legt die Anzahl der Zielklassen (num\_classes) fest.

 **super().init(...)**: Der Aufruf an super() ruft den Konstruktor der Basisklasse **BaseModelTorch** auf, der für das Setup, Training und Speichern des Modells verantwortlich ist.

#### 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

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.

###### CNN

Das **CNNModel** ist ein eigens erstelltes Convolutional Neural Network (CNN), das auf TensorFlow und Keras basiert. Es wurde entwickelt, um eine benutzerdefinierte Architektur zu implementieren, die auf die spezifischen Anforderungen des Projekts abgestimmt ist. Das Modell kombiniert mehrere Schichten wie Convolutional Layers, Max-Pooling, Batch Normalization und Dense Layers, um die Leistung zu optimieren und die Klassifikationsergebnisse zu verbessern.

Theorie

Im folgenden Abschnitt wird das eigens erstellte CNN-Modell erläutert. Es wird beschrieben, wie dieses Modell aufgebaut ist und wie die verwendeten Schichten zusammenwirken, um die Klassifikation zu ermöglichen.

Das Modell nutzt **Convolutional Neural Networks (CNNs)**, die aufgrund ihrer Effektivität in der Bildverarbeitung und Mustererkennung weit verbreitet sind. Bei diesem Modell handelt es sich um eine benutzerdefinierte Architektur, die für eine Vielzahl von Bildklassifikationsaufgaben anpassbar ist.

**Eigens erstellte Architektur:** Die Architektur des Modells basiert auf mehreren aufeinanderfolgenden Schichten, die dafür sorgen, dass das Netzwerk wichtige Merkmale aus den Eingabedaten extrahiert. Es wurden mehrere **Convolutional Layers** verwendet, um die räumlichen Merkmale der Bilder zu erfassen, die durch **Max-Pooling Layers** zusammengefasst und anschließend durch **Dense Layers** klassifiziert werden.

Aufbau und Funktionsweise

Das Modell besteht aus den folgenden wichtigen Komponenten:

* **Convolutional Layers:** Diese Schichten extrahieren Merkmale aus den Eingabedaten. Hier wird der **He-Initialisierer** für die Gewichtsmatrizen verwendet, um das Modell effizient zu trainieren und vanishing gradient issues zu vermeiden. Aktivierungsfunktionen wie **ReLU** (Rectified Linear Unit) werden eingesetzt, um nicht-lineare Transformationen zu ermöglichen.
* **Batch Normalization:** Diese Schicht sorgt für eine bessere und stabilere Lernrate, indem sie den Eingabewert jeder Schicht normalisiert, um den Gradientenabstieg zu stabilisieren.
* **Max-Pooling Layers:** Diese reduzieren die Dimensionsgröße der Merkmalskarten, um die Berechnungen zu vereinfachen und gleichzeitig die wichtigen Merkmale beizubehalten.
* **Global Average Pooling:** Nach den Convolutional und Pooling-Schichten wird das Feature Map auf eine kleinere Form reduziert, wodurch das Modell effizienter wird.
* **Fully Connected Dense Layers:** Diese Schichten nehmen die extrahierten Merkmale und führen sie in die finale Klassifikation.

Die Architektur wurde speziell für die Verarbeitung von Bilddaten mit einer festen Eingabedimension von 200x200 Pixeln erstellt.

Anwendungen

 **Bildklassifikation:** Das Modell ist besonders geeignet für die Klassifikation von Bilddaten und kann für eine Vielzahl von Anwendungen genutzt werden, wie etwa in der medizinischen Bildverarbeitung, der Gesichtserkennung oder der Objekterkennung.

 **Zeitnahe und effiziente Modellierung:** Durch die Reduktion der Bildgröße und die Optimierung der Modellarchitektur kann dieses Modell auch in realen Anwendungen verwendet werden, die schnelle Ergebnisse erfordern.

Verlässlichkeit und Performance

Das eigens erstellte CNN-Modell bietet eine solide Grundlage für Bildklassifikationsaufgaben. Die Leistung kann durch das Hinzufügen zusätzlicher Schichten oder die Verwendung verschiedener Optimierungsstrategien wie **Learning Rate Scheduling** und **Dropout** weiter verbessert werden. Das Modell ist jedoch auf die Handhabung von Bilddaten beschränkt und könnte bei komplexeren, höherdimensionalen Datensätzen weniger leistungsfähig sein.

Da das Modell speziell für kleinere Datensätze und einfachere Aufgaben konzipiert wurde, bietet es eine gute Balance zwischen Leistung und Trainingszeit. Es ist jedoch anfällig für **Overfitting**, wenn zu viele Trainingsdaten oder komplexe Aufgaben verwendet werden.

Imports

Die notwendigen Imports für dieses Modell sind:

from tensorflow import keras

from tensorflow.keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, Dense, BatchNormalization, Dropout, Input

from .training\_basemodel import BaseModelTF

 **keras**: Wird aus TensorFlow importiert, um das Keras-Modul zu nutzen, das die Modellarchitektur vereinfacht und beschleunigt.

 **Conv2D, MaxPooling2D, GlobalAveragePooling2D, Dense, BatchNormalization, Dropout, Input**: Diese Schichten sind die Kernbausteine des CNN-Modells, die zusammen eine vollständige Modellarchitektur bilden.

 **BaseModelTF**: Das Basismodell für TensorFlow, das für das Training und Speichern des Modells zuständig ist.

training\_cnn\_tensorflow.py

\_\_build\_model(self, reshape\_size)

Die Methode \_\_build\_model ist für die Definition und den Aufbau der Convolutional Neural Network (CNN)-Architektur zuständig.

def \_\_build\_model(self, reshape\_size):

model = keras.Sequential([

Input(shape=(reshape\_size, reshape\_size, 1)),

Conv2D(32, (3, 3), activation='relu', kernel\_initializer='he\_normal', padding='same'),

BatchNormalization(),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu', kernel\_initializer='he\_normal', padding='same'),

BatchNormalization(),

MaxPooling2D((2, 2)),

Conv2D(128, (3, 3), activation='relu', kernel\_initializer='he\_normal', padding='same'),

BatchNormalization(),

MaxPooling2D((2, 2)),

GlobalAveragePooling2D(),

Dense(128, activation='relu', kernel\_initializer='he\_normal'),

BatchNormalization(),

Dropout(0.3),

Dense(self.num\_classes, activation='softmax')

])

optimizer = keras.optimizers.AdamW(learning\_rate=0.001)

model.compile(optimizer=optimizer,

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

return model

1. **Erklärung der einzelnen Schritte**
2. **Input(shape=(reshape\_size, reshape\_size, 1))**:
   * Dies definiert die Eingabeschicht des Modells. Hier wird die Form der Eingabebilder festgelegt. Die Eingabebilder haben eine Größe von (reshape\_size, reshape\_size) und sind in Graustufen, daher ist der letzte Wert 1 (für einen Kanal).
   * **reshape\_size**: Die Bilder werden auf eine einheitliche Größe (z. B. 200x200 Pixel) umgeformt, bevor sie in das Modell eingespeist werden.
3. **Erste Convolutional Layer – Conv2D(32, (3, 3), activation='relu', kernel\_initializer='he\_normal', padding='same')**:
   * Dies ist die erste **Convolutional Layer**, die 32 Filter verwendet, um Merkmale aus den Eingabebildern zu extrahieren.
   * Der Filter hat eine Größe von **3x3**. Die **ReLU**-Aktivierungsfunktion wird verwendet, um Nichtlinearitäten einzuführen, und der **He-Initializer** hilft, das Problem des verschwindenden Gradienten zu vermeiden.
   * **Padding 'same'** stellt sicher, dass die Dimension der Eingabe auch nach der Faltung dieselbe bleibt.
4. **BatchNormalization()**:
   * Diese Schicht normalisiert die Ausgaben der vorherigen Convolutional Layer, um die Stabilität des Lernprozesses zu verbessern. Batch Normalization hilft, schneller zu konvergieren und das Modell robuster zu machen.
5. **MaxPooling2D((2, 2))**:
   * Diese **Pooling-Schicht** reduziert die räumlichen Dimensionen der Feature-Maps, indem sie nur den höchsten Wert innerhalb eines 2x2 Fensters auswählt. Das hilft dabei, die Modellkomplexität zu verringern und zu verhindern, dass das Modell überanpasst (Overfitting).
6. **Zweite Convolutional Layer und Pooling**:
   * Diese Schichten wiederholen das gleiche Prinzip wie die erste Convolutional-Schicht, aber diesmal mit 64 Filtern. Wiederum wird eine Batch-Normalisierung und Max-Pooling durchgeführt, um die Merkmale weiter zu extrahieren und die Dimensionen zu reduzieren.
7. **Dritte Convolutional Layer und Pooling**:
   * Die dritte **Convolutional Layer** verwendet 128 Filter, um tiefere und komplexere Merkmale zu extrahieren. Auch hier werden wieder **BatchNormalization** und **MaxPooling** angewendet.
8. **GlobalAveragePooling2D()**:
   * Diese Schicht reduziert die räumliche Dimension der Feature-Maps, indem sie für jedes Feature eine durchschnittliche Aktivierung berechnet. Dies führt zu einer einzigen Zahl pro Feature, die das Modell kompakter macht und die Lernrate stabilisiert.
9. **Dense Layer – Dense(128, activation='relu', kernel\_initializer='he\_normal')**:
   * Diese **Fully Connected Layer** (Dense-Schicht) hat 128 Neuronen und verwendet ebenfalls **ReLU** als Aktivierungsfunktion. Hier werden die extrahierten Merkmale aus den vorherigen Convolutional Layern in eine flache, einheitliche Form umgewandelt.
   * **He-Initializer** wird wieder verwendet, um die Gewichtsmatrizen zu initialisieren.
10. **BatchNormalization und Dropout**:
    * Die **BatchNormalization** hilft erneut, die Trainingsgeschwindigkeit zu erhöhen und das Modell robuster gegen Überanpassung zu machen.
    * **Dropout** mit einer Rate von 0,3 wird angewendet, um Überanpassung zu verhindern, indem bei jedem Training zufällig 30% der Neuronen in dieser Schicht deaktiviert werden.
11. **Ausgabeschicht – Dense(self.num\_classes, activation='softmax')**:
    * Dies ist die finale **Ausgabeschicht**, die die Anzahl der Neuronen auf **self.num\_classes** setzt. Das bedeutet, dass das Modell für eine Klassifikationsaufgabe mit mehreren Klassen ausgelegt ist.
    * Die **Softmax-Aktivierungsfunktion** wird verwendet, um eine Wahrscheinlichkeitsverteilung über die Klassen zu berechnen. Jede Zahl stellt die Wahrscheinlichkeit dar, dass das Bild einer bestimmten Klasse zugeordnet wird.
12. **Optimierung und Kompilierung**

* **Optimizer**: Der **AdamW**-Optimierer wird mit einer Lernrate von 0.001 verwendet. **AdamW** ist eine erweiterte Version des Adam-Optimierers, der die Regularisierung von Gewichtsnormen verbessert und die Leistung in vielen Fällen stabilisiert.
* **Loss Function**: Für das Training wird die **sparse\_categorical\_crossentropy** als Verlustfunktion gewählt. Diese ist für Aufgaben geeignet, bei denen es mehrere Klassen gibt, und sie funktioniert gut mit ganzzahligen Labels, die den Klassen entsprechen.
* **Metrics**: Als Metrik wird **accuracy** verwendet, um die Leistung des Modells während des Trainings und der Evaluation zu überwachen.

###### Xception

Das **XceptionModel** basiert auf dem **Xception**-Modell, einem Convolutional Neural Network (CNN) aus der Keras-Anwendungsliste. Xception ist eine Erweiterung des Inception-Modells und nutzt eine spezielle Architektur, die als "Depthwise Separable Convolutions" bekannt ist. Diese Architektur ermöglicht eine effiziente Nutzung von Rechenressourcen, indem sie die Anzahl der Parameter im Modell reduziert, während gleichzeitig die Modellleistung erhalten bleibt.

Theorie

**Xception** ist eine Erweiterung des Inception-Modells, bei der die Standard-Konvolution durch sogenannte "depthwise separable convolutions" ersetzt wurde. Diese Technik zerlegt eine Standard-Faltung (Convolution) in zwei Schritte: Eine Faltung für jede Eingabekanal (Depthwise) und eine Punkt-Faltung (Pointwise), die eine lineare Kombination der Faltungen berechnet. Diese Architektur hat gezeigt, dass sie sowohl eine höhere Effizienz als auch eine verbesserte Leistung bei der Bildklassifikation bietet.

Aufbau und Funktionsweise

Das XceptionModel nutzt die Keras-Anwendung **Xception** und passt sie an, um auf eine benutzerdefinierte Anzahl von Klassen zu klassifizieren.

1. **Eingabeschicht**: Das Modell erwartet Eingabebilder einer bestimmten Größe (z.B. 200x200 Pixel) und mit einem Kanal (Graustufenbilder). Diese Bilder werden dann an das Xception-Modell weitergegeben.
2. **Xception-Modell**: Das Kernstück dieses Modells ist das **Xception**-Netzwerk, das eine tiefere und optimierte Version des Inception-Netzwerks darstellt, indem es die Faltungsschichten effizienter gestaltet. Die Xception-Architektur basiert auf einer tiefen Hierarchie von separaten Convolutions und lässt sich gut auf eine Vielzahl von Bildklassifikationsaufgaben anwenden.
3. **Ausgabeschicht**: Am Ende des Modells befindet sich eine Dense-Schicht, die die Klassenzugehörigkeit des Eingabebildes auf der Basis der angegebenen Anzahl von Klassen ermittelt.

Anwendungen

 **Gesichtserkennung**: Xception wird oft in der Gesichtserkennung eingesetzt, um Merkmale aus Bildern zu extrahieren und diese zu klassifizieren.

 **Medizinische Bildklassifikation**: Das Modell kann in der medizinischen Bildverarbeitung verwendet werden, beispielsweise zur Erkennung von Anomalien oder Tumoren in Röntgen- oder CT-Bildern.

 **Allgemeine Objekterkennung**: Xception wird häufig in verschiedenen Bereichen der Objekterkennung verwendet, z.B. für die Klassifikation von Tieren, Pflanzen oder Objekten in der Umwelt.

Verlässlichkeit und Performance

Das **Xception**-Modell hat sich in vielen Bereichen der Bildklassifikation als äußerst leistungsfähig erwiesen. Dank seiner effizienten Architektur, die die Parameterzahl minimiert, ohne die Genauigkeit zu beeinträchtigen, eignet es sich besonders gut für ressourcenbegrenzte Umgebungen. Es bietet auch eine hohe Genauigkeit und Robustheit bei der Verarbeitung von Bilddaten.

### Modelltrainer

Der **Modelltrainer** ist eine zentrale Komponente, die dafür zuständig ist, mehrere Modelle zu verwalten, zu trainieren und deren Performance zu evaluieren. Dieser Trainer agiert als Wrapper für verschiedene Modelle, die in einem spezifischen Kontext, wie zum Beispiel Scikit-Learn, PyTorch oder TensorFlow, entwickelt wurden. Die Hauptaufgabe des Modelltrainers ist es, eine einheitliche Schnittstelle zu schaffen, über die man Modelle trainieren, Vorhersagen treffen und Klassifikationsberichte generieren kann. Er abstrahiert die spezifischen Details der unterschiedlichen Frameworks und vereinfacht die Arbeit mit mehreren Modellen, indem er die Verwaltung und das Training der Modelle optimiert.

Der Modelltrainer stellt sicher, dass alle Modelle einer bestimmten Architektur oder eines Frameworks effizient und auf die gleiche Weise gehandhabt werden können. Er ermöglicht eine einfache Integration neuer Modelle und stellt eine konsistente Schnittstelle für deren Nutzung zur Verfügung.

#### Basisklasse

Die **Basisklasse** des Modelltrainers stellt die Grundfunktionalitäten bereit, die für die Verwaltung der Modelle und deren Training erforderlich sind.

Code-Implementierung:

class ModelTrainer:  
 def \_\_init\_\_(self, models, trainer\_name):  
 self.trainer\_name = trainer\_name self.models = models

* + trainer\_name: Speichert den Namen des Modelltrainers
  + models: Speichert alle instanziierten Modelle des Trainers

##### Methoden

Die **Modelltrainer**-Klasse stellt eine allgemeine Schnittstelle für das Training, Testen und Evaluieren von Modellen zur Verfügung. Sie ermöglicht es, verschiedene Modelle zu verwalten, Vorhersagen zu treffen und die Modellleistung zu bewerten. Hier sind die Funktionen der Modelltrainer-Klasse, die das Arbeiten mit mehreren Modellen aus verschiedenen Frameworks ermöglichen:

* **get\_all\_models**: Gibt eine Liste aller Modelle zurück, die vom Modelltrainer verwaltet werden.
* **make\_prediction**: Macht Vorhersagen mit einem ausgewählten Modell anhand von Eingabedaten.
* **train\_model**: Trainiert ein Modell mit einem gegebenen Datensatz und einer bestimmten Anzahl von Epochen.
* **train\_all\_models**: Trainiert alle Modelle des Trainers mit einem gegebenen Datensatz.
* **get\_classification\_report**: Generiert einen Klassifikationsbericht für ein bestimmtes Modell.
* **get\_all\_classification\_reports**: Gibt Klassifikationsberichte für alle Modelle im Trainer zurück.

Die Flexibilität der Modelltrainer-Klasse erlaubt es, dass sie mit einer Vielzahl von Modelltypen und -architekturen arbeitet, ohne dass der Benutzer sich mit den Details des zugrunde liegenden Frameworks beschäftigen muss.

###### get\_all\_models(self)

Die Funktion get\_all\_models gibt eine Liste von Modellnamen zurück, die der Modelltrainer verwaltet.

def get\_all\_models(self):

return [model.model\_name for model in self.models]

###### make\_prediction(self, model\_int, image\_array)

Die Methode make\_prediction führt die Vorhersage eines Modells für ein gegebenes Bild durch. Sie ermöglicht es, basierend auf dem Modellindex (model\_int) das Modell auszuwählen und dann eine Vorhersage für das übergebene Bild (image\_array) zu machen.

def make\_prediction(self, model\_int, image\_array):

print(f'\n{self.trainer\_name}\n# ----------- STARTING PREDICTION OF IMAGE ----------- #\nTrainer: {self.\_\_class\_\_.\_\_name\_\_}\nModel: {model\_int}\n')

model = self.\_\_get\_model(model\_int)

prediction = model.make\_prediction(image\_array)

print('Finished Prediction\n')

return prediction

###### train\_model(self, model\_int, dataset, epochs, reshape\_size)

Die Methode train\_model dient dazu, ein spezifisches Modell zu trainieren. Sie akzeptiert einen Modellindex (model\_int), das Trainingsdatenset (dataset), die Anzahl der Epochen (epochs) und die Größe, auf die das Bild umgeformt werden soll (reshape\_size).

def train\_model(self, model\_int, dataset, epochs, reshape\_size):

error = ''

model = self.\_\_get\_model(model\_int)

print(f'\n{self.trainer\_name}\n# ----------- STARTING TRAINING OF {model.model\_name} ----------- #\n')

res = model.train(dataset, epochs, reshape\_size)

if res[0] is False:

error = res[1]

print(f'{error}\n')

return error

###### train\_all\_models(self, dataset, epochs, reshape\_size, batch\_size=32, lr=0.001)

Die Methode train\_all\_models ist dafür zuständig, alle Modelle, die im Trainer verwaltet werden, zu trainieren. Sie akzeptiert ein Datenset, die Anzahl der Epochen, die Größe der Bildumformung sowie optional Batch-Größe und Lernrate als Parameter.

def train\_all\_models(self, dataset, epochs, reshape\_size, batch\_size=32, lr=0.001):

errors = []

print(f'\n{self.trainer\_name}\n# ----------- STARTING TRAINING OF {len(self.models)} MODELS ----------- #\n')

x = 1

for model in self.models:

print(f'{x}. {model.model\_name}')

res = model.train(dataset, epochs, reshape\_size, batch\_size=batch\_size, lr=lr)

if res[0] is False:

errors.append(res[1])

print(f'{errors[len(errors)-1]}')

print('\n')

x += 1

print('\n')

return errors

###### get\_classification\_report(self, model\_int, dataset, reshape\_size)

Die Methode get\_classification\_report ist dafür verantwortlich, den Klassifizierungsbericht für ein bestimmtes Modell zu erhalten und auszugeben. Sie nimmt die Modellnummer (model\_int), das Datenset und die Bildgröße (reshape\_size) als Eingaben und gibt den Klassifizierungsbericht sowie einen Fehlerbericht zurück, falls einer auftritt.

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

report = ''

error = ''

model = self.\_\_get\_model(model\_int)

print(f'\n{self.trainer\_name}\n# ----------- CLASSIFIER REPORT OF {model.model\_name} ----------- #\n')

res = model.get\_classification\_report(dataset, reshape\_size)

if res[0] is False:

error = res[1]

else:

report = res[1]

print(f'{res[1]}\n')

return report, error

###### get\_all\_classification\_reports(self, dataset, reshape\_size)

Die Methode get\_all\_classification\_reports wird verwendet, um den Klassifizierungsbericht für alle Modelle im Trainer zu generieren und zurückzugeben.

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

print(f'\n{self.trainer\_name}\n# ----------- CLASSIFIER REPORTS OF {len(self.models)} MODELS ----------- #\n')

reports = {}

errors = []

# Iterate over all models and generate reports

x = 1

for model in self.models:

print(f'{x}. {model.model\_name}')

res = model.get\_classification\_report(dataset, reshape\_size)

if res[0] is False:

errors.append(res[1])

else:

reports[model.model\_name] = res[1]

print(f'{res[1]}\n')

x += 1

return reports, errors

###### \_\_get\_model(self, model\_int)

Die Methode \_\_get\_model wird verwendet, um ein Modell aus der Liste der Modelle im ModelTrainer zu holen.

def \_\_get\_model(self, model\_int):

try:

return self.models[model\_int]

except IndexError:

raise NotImplementedError("Model not found")

#### Scikit-Learn

Der Modelltrainer für Scikit-Learn gibt an seine Superklasse alle Scikit-Learn-Modelle und seinen Namen mit:

class ModelTrainerSKLearn(ModelTrainer):  
 def \_\_init\_\_(self, model\_save\_path):  
 super().\_\_init\_\_([  
 KNN(model\_save\_path),  
 SVM(model\_save\_path),  
 RBF\_SVM(model\_save\_path),  
 GaussianProcess(model\_save\_path),  
 DecisionTree(model\_save\_path),  
 RandomForest(model\_save\_path),  
 NeuralNet(model\_save\_path),  
 AdaBoost(model\_save\_path),  
 NaiveBayes(model\_save\_path)  
 ], self.\_\_class\_\_.\_\_name\_\_)

#### PyTorch

Der Modelltrainer für PyTorch gibt an seine Superklasse alle PyTorch-Modelle und seinen Namen mit:

class ModelTrainerPyTorch(ModelTrainer):  
 def \_\_init\_\_(self, model\_save\_path, num\_classes):  
 super().\_\_init\_\_([  
 AlexNetModel(model\_save\_path, num\_classes),  
 ConvNeXtModel(model\_save\_path, num\_classes),  
 DenseNetModel(model\_save\_path, num\_classes),  
 EfficientNetModel(model\_save\_path, num\_classes),  
 EfficientNetV2Model(model\_save\_path, num\_classes),  
 GoogLeNetModel(model\_save\_path, num\_classes),  
 InceptionModel(model\_save\_path, num\_classes),  
 MaxVitModel(model\_save\_path, num\_classes),  
 MNASNetModel(model\_save\_path, num\_classes),  
 MobileNetV2Model(model\_save\_path, num\_classes),  
 MobileNetV3Model(model\_save\_path, num\_classes),  
 RegNetModel(model\_save\_path, num\_classes),  
 ResNetModel(model\_save\_path, num\_classes),  
 ResNeXtModel(model\_save\_path, num\_classes),  
 ShuffleNetModel(model\_save\_path, num\_classes),  
 SqueezeNetModel(model\_save\_path, num\_classes),  
 SwinTransformerModel(model\_save\_path, num\_classes),  
 VGGModel(model\_save\_path, num\_classes),  
 VisionTransformerModel(model\_save\_path, num\_classes),  
 WideResNetModel(model\_save\_path, num\_classes)  
 ], self.\_\_class\_\_.\_\_name\_\_)

#### TensorFlow

Der Modelltrainer für TensorFlow gibt an seine Superklasse alle TensorFlow-Modelle und seinen Namen mit:

class ModelTrainerTensorFlow(ModelTrainer):  
 def \_\_init\_\_(self, model\_save\_path, num\_classes):  
 super().\_\_init\_\_(  
 [  
 CNNModel(model\_save\_path, num\_classes),  
 DenseNetModel(model\_save\_path, num\_classes),  
 EfficientNetModel(model\_save\_path, num\_classes),  
 InceptionModel(model\_save\_path, num\_classes),  
 MobileNetModel(model\_save\_path, num\_classes),  
 ResNetModel(model\_save\_path, num\_classes),  
 XceptionModel(model\_save\_path, num\_classes)  
 ], self.\_\_class\_\_.\_\_name\_\_)

### ModellTrainerContainer

Der **ModellTrainerContainer** dient als zentrale Verwaltungseinheit für die verschiedenen Modelltrainer, die mit verschiedenen Machine Learning Frameworks arbeiten, wie PyTorch, Scikit-Learn und TensorFlow. Diese Klasse bietet eine strukturierte Schnittstelle, um die verschiedenen Modelltrainer zu verwalten und deren Funktionen wie Modellvorhersage, Training und Evaluierung durchzuführen.

Der **ModellTrainerContainer** initialisiert und speichert Instanzen der Modelltrainer für PyTorch, Scikit-Learn und TensorFlow und stellt Funktionen bereit, um mit diesen Trainern zu interagieren. Hierbei können Modelle trainiert, Vorhersagen gemacht und Klassifikationsberichte abgefragt werden. Der Container verwaltet zudem den Zugriff auf den Datensatz und sorgt dafür, dass alle Trainer auf denselben Datensatz zugreifen, der aus einer JSON-Datei geladen wird.

Mit dieser Struktur können Modelle für verschiedene Frameworks parallel verwaltet werden, was eine hohe Flexibilität und Erweiterbarkeit ermöglicht. Der Container übernimmt dabei die Aufgabe, den richtigen Trainer anhand der Anforderungen auszuwählen und die entsprechenden Methoden zu delegieren, wodurch der Umgang mit den unterschiedlichen Modellen vereinfacht wird.

Code-Implementierung:

class ModelTrainerContainer :  
 def \_\_init\_\_(self, model\_save\_path, api\_base\_url, num\_classes):  
 self.api\_base\_url = api\_base\_url self.trainers = [  
 ModelTrainerPyTorch(model\_save\_path, num\_classes),  
 ModelTrainerSKLearn(model\_save\_path),  
 ModelTrainerTensorFlow(model\_save\_path, num\_classes)  
 ]

* + api\_base\_url: Speichert die URL zur Datenbank
  + trainers: Speichert alle Trainers, die der Container verwaltet

##### get\_all\_models(self)

Die Funktion get\_all\_models ist eine Methode, die in einer Klasse definiert ist und eine Sammlung aller Modelle aus verschiedenen Trainer-Objekten zurückgibt. Sie iteriert über eine Liste von trainers und ruft für jeden Trainer die Methode get\_all\_models() auf, um eine Liste von Modellnamen zu erhalten. Diese werden dann in einem Dictionary gespeichert, wobei der Schlüssel der Name des Trainers (als String) ist und der Wert eine Liste der Modellnamen.

def get\_all\_models(self):  
 all\_models = {}  
  
 for trainer in self.trainers:  
 trainer\_name = type(trainer).\_\_name\_\_ model\_names = trainer.get\_all\_models()  
 all\_models[trainer\_name] = model\_names  
  
 return all\_models

##### get\_all\_models(self)

Die Funktion make\_prediction ist eine Methode, die basierend auf dem angegebenen Trainer-String (trainer\_string), dem Modell-Index (model\_int) und einem Bild-Array (image\_array) eine Vorhersage trifft. Die Methode überprüft den Typ des Trainers und ruft dann die entsprechende make\_prediction-Methode des entsprechenden Trainers auf. Wenn der Trainer-String ungültig ist, wird eine Fehlermeldung ausgegeben.

def make\_prediction(self, trainer\_string, model\_int, image\_array):  
 if str(trainer\_string).lower() == 'modeltrainerpytorch':  
 return self.trainers[0].make\_prediction(model\_int, image\_array)  
 elif str(trainer\_string).lower() == 'modeltrainersklearn':  
 return self.trainers[1].make\_prediction(model\_int, image\_array)  
 elif str(trainer\_string).lower() == 'modeltrainertensorflow':  
 return self.trainers[2].make\_prediction(model\_int, image\_array)  
 else:  
 raise ValueError('Invalid trainer')

##### train\_model(self, trainer\_string, model\_int, num\_epochs, reshape\_size)

Die Methode train\_model ist dafür verantwortlich, ein Modell zu trainieren, basierend auf dem angegebenen Trainer (PyTorch, Scikit-Learn oder TensorFlow). Sie prüft den Trainer-String, lädt das entsprechende Dataset und ruft dann die train\_model-Methode des spezifischen Trainers auf.

def train\_model(self, trainer\_string, model\_int, num\_epochs, reshape\_size):  
 if str(trainer\_string).lower() == 'modeltrainerpytorch':  
 trainer = self.trainers[0]  
 elif str(trainer\_string).lower() == 'modeltrainersklearn':  
 trainer = self.trainers[1]  
 elif str(trainer\_string).lower() == 'modeltrainertensorflow':  
 trainer = self.trainers[2]  
 else:  
 raise ValueError('Invalid trainer')  
  
 dataset = self.\_\_fetch\_dataset(reshape\_size)  
 return trainer.train\_model(model\_int, dataset, num\_epochs, reshape\_size)

##### train\_all\_models(self, num\_epochs, reshape\_size)

Die Methode train\_all\_models ist dafür verantwortlich, alle Modelle in den verschiedenen Trainer-Objekten zu trainieren und dabei das Ergebnis (einschließlich möglicher Fehler) zu speichern. Sie geht durch jedes Trainer-Objekt in self.trainers, trainiert alle Modelle und speichert die Ergebnisse in einem Dictionary. Das Dictionary enthält den Namen des Trainers als Schlüssel und das Ergebnis des Trainings als Wert.

def train\_all\_models(self, num\_epochs, reshape\_size):  
 errors = {}  
 dataset = self.\_\_fetch\_dataset(reshape\_size)  
  
 for trainer in self.trainers:  
 res = trainer.train\_all\_models(dataset, num\_epochs, reshape\_size)  
 errors[trainer.\_\_class\_\_.\_\_name\_\_] = res return errors

##### get\_classifier\_report(self, trainer\_string, model\_int, reshape\_size)

Die Methode get\_classifier\_report wird verwendet, um einen Klassifikationsbericht für ein bestimmtes Modell zu erhalten. Der Bericht wird durch die Methode get\_classification\_report des entsprechenden Trainers (PyTorch, Scikit-Learn oder TensorFlow) generiert.

def get\_classifier\_report(self, trainer\_string, model\_int, reshape\_size):  
 if str(trainer\_string).lower() == 'modeltrainerpytorch':  
 trainer = self.trainers[0]  
 elif str(trainer\_string).lower() == 'modeltrainersklearn':  
 trainer = self.trainers[1]  
 elif str(trainer\_string).lower() == 'modeltrainertensorflow':  
 trainer = self.trainers[2]  
 else:  
 raise ValueError('Invalid trainer')  
  
 dataset = self.\_\_fetch\_dataset(reshape\_size)  
 return trainer.get\_classification\_report(model\_int, dataset, reshape\_size)

##### get\_all\_classifier\_reports(self, reshape\_size)

Die Methode get\_all\_classifier\_reports wird verwendet, um Klassifikationsberichte für alle Trainer in der self.trainers Liste zu generieren. Sie gibt sowohl die Berichte als auch etwaige Fehler für jeden Trainer zurück.

def get\_all\_classifier\_reports(self, reshape\_size):  
 reports = {}  
 errors = {}  
  
 dataset = self.\_\_fetch\_dataset(reshape\_size)  
  
 for trainer in self.trainers:  
 trainer\_name = trainer.\_\_class\_\_.\_\_name\_\_ res = trainer.get\_all\_classification\_reports(dataset, reshape\_size)  
 reports[trainer\_name] = res[0]  
 errors[trainer\_name] = res[1]  
 return reports, errors

##### \_\_fetch\_dataset(self, reshape\_size)

Die Methode \_\_fetch\_dataset wird verwendet, um ein Dataset zu laden und es in einem für das Modell trainierbaren Format bereitzustellen. In diesem Fall wird die Datei aus einem lokalen Verzeichnis geladen, und die Funktion loadImagesAs1DVectorFromJson wird verwendet, um die Daten aus der JSON-Datei zu extrahieren und in ein geeignetes Format zu bringen.

def \_\_fetch\_dataset(self, reshape\_size):  
 #dataset = loadImagesAs1DVectorFromAPI(self.apir\_base\_url + "/picture/picture", reshape\_size)  
 base\_dir = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "..", "test"))  
 file\_path = os.path.join(base\_dir, "IMAGES1.images.json")  
  
 # Prüfen, ob die Datei existiert  
 if not os.path.exists(file\_path):  
 raise FileNotFoundError(f"File not found: {file\_path}")  
  
 # Lade die Datei  
 dataset = loadImagesAs1DVectorFromJson(file\_path, reshape\_size)  
 return dataset

### API

Das API (Application Programming Interface) der KI-Komponente bildet jene Schnittstelle nach außen, die dem Benutzer (dem Frontend) den Zugang zum DermaAI-System gewährt. Sie bearbeitet sowohl alle Anfragen direkt für die KI-Komponente als auch jene, die nur die Datenbank benötigen (z.B. Login und Register), indem es die Anfragen an die entsprechenden Endpunkte der Datenbank weiterleitet, auf die Antwort wartet und diese schließlich dem Frontend preisgibt.

#### RequestModels

Hinter den Anfragen des Frontends steckt ein Datenmodell, sodass sowohl die KI-Komponente als auch die Datenbank einheitliche Daten zum Verarbeiten bekommen. Folglich werden die RequestModels, also die Datenmodelle der Requests, für oben genannte API beschrieben.

##### RequestModels.py

Dieses Modul enthält alle für die API benötigten Datenmodelle und sorgt für eine gebündelte, zentrale Verwaltung dieser.

###### Imports

Für entsprechendes Modul ist folgender Import von Nöten:

from pydantic import BaseModel

Importiert wird die **Basisklasse BaseModel** aus der pydantic-Bibliothek. Diese Klasse ermöglicht es, ganz einfach **strukturierte Datenmodelle mit Typüberprüfung** zu erstellen.

Features:

* Automatische **Typüberprüfung**
* Automatische **Validierung**
* Einfache **Konvertierung von z. B. JSON nach Python**
* Praktische **Hilfsmethoden** wie .dict() oder .json()

###### Berechtigungen

Folgenden Aktionen dürfen nur von berechtigten Personen (Admins) durchgeführt werden, da sie verlangen, dass der gesamte Datensatz aus der Datenbank geladen wird und somit entsprechende Zeit benötigen:

* Ein Modell trainieren
* Alle Modelle trainieren
* Einen Klassifikationsbericht erstellen
* Alle Klassifikationsberichte erstellen

Aufgrund dessen werden bei diesen Anfragen Email und Passwort des Benutzers mitgesendet, um zu prüfen, ob eine Berechtigung vorhanden ist.

###### UserRequest

Der UserRequest wird für den Login -und Registrierungsprozess verwendet. Er übermittelt lediglich die Benutzerdaten (Email und Passwort) und einen Bool, ob MFA (Mehrfaktorauthentifizierung) aktiviert ist oder nicht:

class UserRequest(BaseModel):  
 email: str

password: str

mfa: bool

###### PredictionRequest

Der PredictionRequest wird für den Analyse-Prozess verwendet. Der Benutzer sendet den Namen des gewünschten Trainers, den Index des ausgewählten Modells und das Bild als Base64-String mit:

class PredictionRequest(BaseModel):  
 model\_int: int

trainer\_string: str

image: str

###### ClassifierReportRequest

Der ClassifierReportRequest wird für das Erstellen eines Klassifikationsberichtes für ein spezifisches Modell verwendet. Der Benutzer sendet neben dem Trainer und dem Modell auch eine Reshape Size, welche angibt, auf welche Größe die Trainings -und Testdaten aus der Datenbank vor der Verwendung vom Modell angepasst werden:

class ClassifierReportRequest(BaseModel):  
 email: str

password: str

trainer\_string: str

model\_int: int

reshape\_size: int

###### ClassifierReportsRequest

Der ClassifierReportsRequest wird verwendet, um über alle Modelle aller Modelltrainer zu iterieren und für jedes Modell im System einen Klassifikationsbericht zu erstellen:

class ClassifierReportsRequest(BaseModel):  
 email: str

password: str

reshape\_size: int

###### RetrainRequest

Der RetrainRequest wird verwendet, um ein spezifisches Modell mit den aktuellen Trainingsdaten aus der Datenbank zu trainieren. Dabei wählt der Benutzer ein gewünschtes Modell aus und sendet die Reshape Size sowie die Anzahl der Epochen, also wie oft ein Modell über den Datensatz iterieren soll, mit:

class RetrainRequest(BaseModel):  
 email: str

password: str

trainer\_string: str

model\_int: int

num\_epochs: int

reshape\_size: int

###### RetrainAllRequest

Der RetrainAllRequest wird verwendet, um alle sich im System befindenden Modelle zu trainieren:

class RetrainAllRequest(BaseModel):  
 email: str

password: str

num\_epochs: int

reshape\_size: int

###### SavePredictionRequest

Der SavePredictionRequest wird verwendet, um nach der Analyse eines Bildes das Ergebnis für einen Benutzer zu speichern. Dabei sendet der Benutzer neben seinen Benutzerdaten das analysierte Bild als Base64-String sowie das entsprechende Ergebnis mit:

class SavePredictionRequest(BaseModel):  
 email: str

password: str

image: str

prediction: dict

###### ResizeRequest

Der ResizeRequest wird verwendet, um ein Bild an den Datenbank-Server für die automatische Bildzuschnitt-Funktion weiterzuleiten:

class ResizeRequest(BaseModel):  
 image: str

#### Controller

Um alle oben genannten Anfragen abzudecken, werden drei Controller implementiert, welche Anfragen des Frontends entgegennehmen.

##### ImageController

Der ImageController ist für Anfragen zuständig, die lediglich Bilder als Ressource betrachten. In diesem Fall zählt einzig und allein der ResizeRequest dazu.

Code-Implementierung:

class ImageController:  
 def \_\_init\_\_(self, api\_base\_url):  
 self.router = APIRouter()

###### POST /resize

Dieser Endpunkt empfängt einen ResizeRequest und leitet ihn an den Datenbank-Server weiter:

@self.router.post("/resize/")  
async def resize\_image(request: ResizeRequest):  
 request = request.dict()  
 try:  
 res = requests.post(api\_base\_url[1], json=request)  
 return Response(content=res.content, media\_type=res.headers["Content-Type"],  
 status\_code=res.status\_code)  
 except Exception as e:  
 raise HTTPException(status\_code=500, detail=f"Error: {str(e)}")

Beispiel für RequestBody:

{

"image": "string"

}

##### ModelController

Der ModelController ist für Anfragen zuständig, welche Zugriff auf Modellfunktionen benötigen. Deshalb verlangt er auch den ModelTrainerContainer als Parameter:

class ModelController:  
 def \_\_init\_\_(self, api\_base\_url, model\_trainer):  
 self.router = APIRouter()  
 self.model\_trainer = model\_trainer

###### User-Validation

Bei Anfragen auf Endpunkte, die nur von Admins gemacht werden dürfen, wird eine User-Validation durchgeführt. Hier beispielhaft für den ClassifierReportsRequest:

request = request.dict()  
try:  
 payload = {  
 "email": request["email"],  
 "password": request["password"]  
 }  
 res = requests.post(api\_base\_url[0] + '/user/validateUser', json=payload)  
 if res.status\_code != 200:  
 raise HTTPException(status\_code=res.status\_code, detail=res.text)  
  
 res\_json = res.json()  
 if res\_json["isAdmin"] is True:  
 res = self.model\_trainer.get\_all\_classifier\_reports(request["reshape\_size"])  
 else:  
 raise HTTPException(status\_code=401, detail="Unauthorized")

Es wird der externe Endpunkt **/user/validateUser** des Datenbank-Servers aufgefordert, einen Benutzer zu validieren. Als Antwort erhält man die Information, ob ein Benutzer ein Admin ist oder nicht. Sollte der Benutzer nicht berechtigt sein, eine Aktion auszuführen, wird der **Statuscode 401 „Unauthorized“** zurückgegeben. Sollte der Benutzer berechtigt sein, wird die entsprechende Aktion durchgeführt.

###### GET /models

Dieser Endpunkt gibt ein Dictionary zurück, welches für jeden ModelTrainer alle Namen seiner Modelle enthält:

@self.router.get("/models")  
async def get\_all\_models():  
 try:  
 models = self.model\_trainer.get\_all\_models()  
 return JSONResponse(content=models, status\_code=200)  
 except Exception as e:  
 raise HTTPException(status\_code=500, detail=f"Error while fetching models: {str(e)}")

Beispielantwort:

{

"ModelTrainerPyTorch": [

"VisionTransformer",

"ResNet"

],

"ModelTrainerSKLearn": [

"GaussianNB"

],

"ModelTrainerTensorFlow": [

"xception"

]

}

###### GET /classifier-report

Dieser Endpunkt erstellt einen Klassifikationsbericht für ein ausgewähltes Modell und gibt diesen zurück:

@self.router.get("/classifier-report/")  
def get\_classifier\_report(request: ClassifierReportRequest):  
 request = request.dict()  
 try:  
 payload = {  
 "email": request["email"],  
 "password": request["password"]  
 }  
 res = requests.post(api\_base\_url[0] + '/user/validateUser', json=payload)  
 if res.status\_code != 200:  
 raise HTTPException(status\_code=res.status\_code, detail=res.text)  
  
 res\_json = res.json()  
 if res\_json["isAdmin"] is True:  
 res = self.model\_trainer.get\_classifier\_report(request["trainer\_string"], request["model\_int"],  
 request["reshape\_size"])  
 else:  
 raise HTTPException(status\_code=401, detail="Unauthorized")  
 return JSONResponse(content={"report": res[0], "error": res[1]}, status\_code=200)  
 except NotImplementedError as nie:  
 raise HTTPException(status\_code=404, detail=f"Error: {str(nie)}")  
 except Exception as e:  
 raise HTTPException(status\_code=500, detail=f"Error while fetching report: {str(e)}")

###### GET /classifier-reports

Dieser Endpunkt erstellt alle Klassifikationsberichte für alle Modelle und gibt diese zurück:

@self.router.get("/classifier-reports/")  
def get\_all\_models(request: ClassifierReportsRequest):  
 request = request.dict()  
 try:  
 payload = {  
 "email": request["email"],  
 "password": request["password"]  
 }  
 res = requests.post(api\_base\_url[0] + '/user/validateUser', json=payload)  
 if res.status\_code != 200:  
 raise HTTPException(status\_code=res.status\_code, detail=res.text)  
  
 res\_json = res.json()  
 if res\_json["isAdmin"] is True:  
 res = self.model\_trainer.get\_all\_classifier\_reports(request["reshape\_size"])  
 else:  
 raise HTTPException(status\_code=401, detail="Unauthorized")  
 return JSONResponse(content={"reports": res[0], "errors": res[1]}, status\_code=200)  
 except Exception as e:  
 raise HTTPException(status\_code=500, detail=f"Error while fetching reports: {str(e)}")

###### POST /predict

Dieser Endpunkt führt eine Analyse eines Bildes durch und gibt diese zurück:

@self.router.post("/predict/")  
async def predict\_image(request: PredictionRequest):  
 request = request.dict()  
 try:  
 image\_data = base64.b64decode(request["image"])  
 image = PILImage.open(io.BytesIO(image\_data))  
 prediction = self.model\_trainer.make\_prediction(request["trainer\_string"], request["model\_int"], image)  
  
 return JSONResponse(content={  
 "trainer\_string": request["trainer\_string"],  
 "model\_id": request["model\_int"],  
 "prediction": prediction }, status\_code=200)  
  
 except NotImplementedError as nie:  
 raise HTTPException(status\_code=404, detail=f"Error: {str(nie)}")  
 except Exception as e:  
 print(str(e))  
 raise HTTPException(status\_code=500, detail=f"Error while predicting Image: {str(e)}")

###### POST /retrain-all

Dieser Endpunkt trainiert alle Modelle und gibt eine Fehlerliste zurück:

@self.router.post("/retrain-all/")  
async def retrain\_models(request: RetrainAllRequest):  
 request = request.dict()  
 try:  
 payload = {  
 "email": request["email"],  
 "password": request["password"]  
 }  
 res = requests.post(api\_base\_url[0] + '/user/validateUser', json=payload)  
 if res.status\_code != 200:  
 raise HTTPException(status\_code=res.status\_code, detail=res.text)  
  
 res\_json = res.json()  
 if res\_json["isAdmin"] is True:  
 errors = self.model\_trainer.train\_all\_models(request["num\_epochs"], request["reshape\_size"])  
 else:  
 raise HTTPException(status\_code=401, detail="Unauthorized")  
 return JSONResponse(content={"message": "Finished training", "errors": errors}, status\_code=200)  
 except Exception as e:  
 raise HTTPException(status\_code=500, detail=f"Error: {str(e)}")

###### POST /retrain

Dieser Endpunkt trainiert ein spezifisches Modell und gibt einen Fehler zurück:

@self.router.post("/retrain/")  
def retrain\_model(request: RetrainRequest):  
 request = request.dict()  
 try:  
 payload = {  
 "email": request["email"],  
 "password": request["password"],  
 }  
 res = requests.post(api\_base\_url[0] + '/user/validateUser', json=payload)  
 if res.status\_code != 200:  
 raise HTTPException(status\_code=res.status\_code, detail=res.text)  
  
 res\_json = res.json()  
 if res\_json["isAdmin"] is True:  
 error = self.model\_trainer.train\_model(request["trainer\_string"], request["model\_int"],  
 request["num\_epochs"], request["reshape\_size"])  
 else:  
 raise HTTPException(status\_code=401, detail="Unauthorized")  
  
 return JSONResponse(  
 content={"message": f"Finished training model with trainer {request['trainer\_string']} and ID" f" {request['model\_int']}", "error": error}, status\_code=200)  
 except NotImplementedError as nie:  
 raise HTTPException(status\_code=404, detail=f"Error: {str(nie)}")  
 except Exception as e:  
 raise HTTPException(status\_code=500, detail=f"Error: {str(e)}")

##### UserController

Der UserController ist für Anfragen zuständig, welche lediglich Zugriff auf die Datenbank benötigen.

Code-Implementierung:

class UserController:  
 def \_\_init\_\_(self, api\_base\_url):  
 self.router = APIRouter()  
 self.api\_base\_url = api\_base\_url

###### POST /create-user

Dieser Endpunkt leitet Benutzerdaten an die Datenbank weiter, um einen neuen Benutzer zu erstellen:

@self.router.post("/create-user/")  
def create\_user(user\_data: UserRequest):  
 user\_data = user\_data.dict()  
 try:  
 res = requests.post(f"{api\_base\_url[0]}/user/saveUser", json=user\_data)  
 return Response(content=res.content, media\_type=res.headers["Content-Type"],  
 status\_code=res.status\_code)  
 except Exception as e:  
 raise HTTPException(status\_code=500, detail=f"Error: {str(e)}")

###### POST /login

Dieser Endpunkt leitet Benutzerdaten an die Datenbank weiter, um einen bestehenden Benutzer zu überprüfen:

@self.router.post("/login/")  
def login\_user(user\_data: UserRequest):  
 user\_data = user\_data.dict()  
 try:  
 res = requests.post(f"{api\_base\_url[0]}/user/validateUser/", json=user\_data)  
 return Response(content=res.content, media\_type=res.headers["Content-Type"],  
 status\_code=res.status\_code)  
  
 except Exception as e:  
 raise HTTPException(status\_code=500, detail=f"Error: {str(e)}")

###### POST /save-prediction

Dieser Endpunkt leitet Benutzerdaten und ein Analyseergebnis an die Datenbank weiter, um es in der Historie des Benutzers zu speichern:

@self.router.post("/save-prediction/")  
async def save\_prediction(request: SavePredictionRequest):  
 request = request.dict()  
 try:  
 res = requests.post(f"{api\_base\_url[0]}/prediction/savePrediction", json=request)  
 return Response(content=res.content, media\_type=res.headers["Content-Type"],  
 status\_code=res.status\_code)  
 except Exception as e:  
 raise HTTPException(status\_code=500, detail=f"Error: {str(e)}")

###### POST /load-predictions

Dieser Endpunkt holt die vergangenen Analysen eines Benutzers aus der Datenbank, um diese im Frontend anzeigen zu lassen:

@self.router.post("/load-predictions/")  
async def load\_predictions(request: UserRequest):  
 request = request.dict()  
 try:  
 res = requests.post(f"{api\_base\_url[0]}/prediction/loadPrediction", json=request)  
 return Response(content=res.content, media\_type=res.headers["Content-Type"],  
 status\_code=res.status\_code)  
 except Exception as e:  
 raise HTTPException(status\_code=500, detail=f"Error: {str(e)}")

#### APIHandler

Der APIHandler ist die den Controllern übergeordnete Komponente, welche sie in eine FastAPI-App kombiniert und somit alle Endpunkte aller Controller für den Benutzer zur Verfügung stellt. Er dient als **zentraler Einstiegspunkt** zur Initialisierung der **FastAPI Anwendung** für das DermaAI-Projekt. Sie konfiguriert grundlegende API-Einstellungen und stellt eine statische Dateianbindung bereit, welche für angefragte Dokumente (z.B. Modellinformationen) verwendet wird.

Code-Implementierung:

class APIHandler:  
 def \_\_init\_\_(self, model\_save\_path, api\_base\_url):  
 self.app = FastAPI(title="DermaAI", description="API for DermaAI", version="1.0.0")  
 self.app.mount("/static", StaticFiles(directory="static"), name="static")

##### Abfrage der Labels

Bevor die FastAPI-Anwendung gestartet wird, werden von der Datenbank alle Labels (als unique-Liste) abgefragt. Das ist notwendig, da die Tensorflow-Modelle die Anzahl der Klassen, die das Modell verwendet, als Initialisierungsparameter verlangen:

labels = None try:  
 response = requests.get(f"{api\_base\_url[0]}/picture/labels")  
 if response.status\_code == 200:  
 data = response.json()  
 diagnoses = data.get("diagnoses", [])  
 labels = len(diagnoses)  
 print("Length of diagnoses:", labels)  
 else:  
 print("Failed to fetch data:", response.status\_code)  
except Exception:  
 labels = 1 print("Database API not reachable for labels")  
  
if labels == 0:  
 labels = 1

Falls die Labels nicht von der Datenbank abgefragt werden können, so wird labels auf 1 gesetzt. Ist dies der Fall, so müssen bei Abweichungen die verwendeten Labels aus den Metadaten geladen und das Modell erneut erstellt werden.

##### Einbinden der Komponenten

Nach der Grundkonfiguration der FastAPI-App werden hier die **Kernkomponenten für die API-Funktionalität** eingerichtet und in die App integriert.

###### ModelTrainerContainer

Hier wird der ModelTrainerContainer für die Verwaltung aller ModelTrainer und aller Modelle erstellt:

self.model\_trainer = ModelTrainerContainer(model\_save\_path, api\_base\_url[0], labels)

###### Router

Jeder Controller bringt seinen eigenen **Router** mit, der unter einem bestimmten **Pfad-Präfix** in die API eingebunden wird:

model\_controller = ModelController(api\_base\_url, self.model\_trainer)  
user\_controller = UserController(api\_base\_url)  
image\_controller = ImageController(api\_base\_url)  
  
self.app.include\_router(model\_controller.router, prefix="/model", tags=["Model"])  
self.app.include\_router(user\_controller.router, prefix="/user", tags=["User"])  
self.app.include\_router(image\_controller.router, prefix="/image", tags=["Image"])

##### GET /health

Der Health-Endpunkt dient der Überprüfung, ob die KI-Komponente bereit für Anfragen ist:

@self.app.get("/health/")  
def test():  
 return JSONResponse(content={"message": "Application is live"}, status\_code=200)

##### start(self)

Die Methode start startet den **FastAPI-Server** mit Hilfe von **Uvicorn**:

def start(self):  
 uvicorn.run(self.app, host="0.0.0.0", port=8000)

## 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*