

DBA63

Labor Datenanalyse und Auswertung

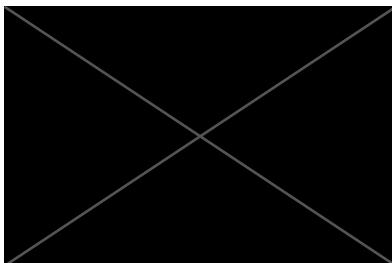
Assignment

eingereicht bei



von

Justin Stange-Heiduk



Datum: 18.11.2024

Einleitung

1. Business Understanding

- a. Geschäftsziele: Überlegen Sie sich Geschäftsziele für dieses Thema. Was könnten Sie über das Thema herausfinden wollen? (Recherchieren Sie gerne zu dieser Frage.)
- b. Ziele der Datenanalysen: Leiten Sie daraus die Ziele Ihrer Datenanalysen her.

2. Data Understanding

- a. Exploration: Explorieren Sie die Daten qualitativ, quantitativ und grafisch, um einen Eindruck davon zu bekommen.
- b. Analysen: Welche Analysen möchten bzw. können Sie durchführen, um Ihre Ziele zu erreichen?

3. Data Preparation

- a. Fehlende Daten: Gibt es fehlende oder nicht plausible Daten? Was tun Sie damit?
- b. Ausreißer: Werten Sie alle Daten aus oder schließen Sie Ausreißer oder irrelevante Daten aus? Warum?
- c. Diskretisierung: Wollen Sie aus diskreten Werten kontinuierliche machen oder umgekehrt?

- d. Normalisierung: Macht es Sinn, Daten zu normalisieren? Zu projizieren oder kalibrieren?
- e. Ausgewogenheit: Sind die Kategorien ausgewogen? Versuchen Sie es mit Undersampling oder Oversampling?
- f. Zusätzliche Daten: Möchten Sie noch weitere Daten ergänzen und hinzufügen?

4. Modelling

- a. Methoden: Wählen Sie die Methoden aus, die Sie benötigen, um mit Hilfe Ihrer Daten die gewählten Ziele zu erreichen und gestellten Fragen zu beantworten.
- b. Durchführung: Führen Sie die Auswertungen durch.

5. Evaluation

- a .Evaluierung: Ist bei den Auswertungen das Erwartete oder Richtige herausgekommen? Sind die Ergebnisse plausibel? Sind sie statistisch signifikant?
- b. Verbesserung: Verbessern und verfeinern Sie Ihre Auswertungen

6. Deployment

- a. Zielerreichung: Sind die Ziele der Datenanalysen erreicht worden? Können damit auch die Geschäftsziele erreicht werden? Wie?
- b. Monitoring und Wartung: Wie sollen zukünftig die Daten aktuell gehalten und die Auswertungen aktualisiert werden?
- c. Bericht: Schreiben Sie Ihren Bericht. Fassen Sie die wichtigsten Ergebnisse am Ende des Berichts zusammen und erstellen Sie einen halbseitigen Management Summary
- d. Kritische Reflexion / Lessons Learned: Was hätte man besser machen können? Was werden Sie das nächste Mal besser machen? Was haben Sie dazu gelernt?
- e. Ausblick: Was wären die nächsten Schritte?

1) Business Understanding

a) Geschäftsziele

Im Rahmen dieses Projekts wird eine Bilderkennungs- und Klassifikationslösung entwickelt, um Galaxien basierend auf ihrem visuellen Erscheinungsbild in verschiedene Kategorien einzurichten. Die Daten für dieses Projekt stammen aus dem Galaxy10 DECaLS Dataset, das aus dem ursprünglichen Galaxy10-Datensatz entwickelt wurde. Der ursprüngliche Galaxy10-Datensatz wurde mithilfe von Galaxy Zoo (GZ) Data Release 2 erstellt, bei dem Freiwillige etwa 270.000 Galaxienbilder aus dem Sloan Digital Sky Survey (SDSS) klassifizierten, von denen etwa 22.000 mithilfe von freiwilligen Stimmen in zehn breite Klassen unterteilt wurden. Später nutzte Galaxy Zoo Bilder aus den DESI Legacy Imaging Surveys (DECaLS), die eine deutlich bessere Auflösung und Bildqualität boten. (Leung und Bovy, 2024)

Galaxy10 DECaLS kombiniert die Ergebnisse von GZ DR2 mit DECaLS-Bildern anstelle der SDSS-Bilder sowie den Ergebnissen der DECaLS-Kampagnen (a, b, c), was zu insgesamt etwa 441.000 einzigartigen Galaxien führte, die von DECaLS abgedeckt werden. Von

diesen wurden etwa 18.000 in zehn breite Klassen eingeteilt, wobei strengere Filter angewendet wurden. Die Klassifikationen basieren auf den Stimmen von Freiwilligen, und die zehn Klassen wurden so angepasst, dass sie sich deutlicher voneinander unterscheiden. Beispielsweise wurde die Klasse "Edge-on Disk with Boxy Bulge", die im ursprünglichen Galaxy10-Datensatz nur 17 Bilder umfasste, aus dem Galaxy10 DECaLS-Datensatz entfernt. (Leung und Bovy, 2024)

Die Bilder stammen aus den DESI Legacy Imaging Surveys, und die Klassifikationslabels wurden von den Freiwilligen des Galaxy Zoo-Projekts bereitgestellt. Galaxy Zoo wird in Lintott et al. (2008) beschrieben, der Data Release 2 von Galaxy Zoo in Lintott et al. (2011) und die Galaxy Zoo DECaLS-Kampagne in Walmsley M. et al. (2021). Die DESI Legacy Imaging Surveys werden in Dey A. et al. (2019) beschrieben.

Die Klassifikation von Galaxien ist seit jeher ein wichtiger Bestandteil der astronomischen Forschung. Der erste Schritt zur Entwicklung der Galaxienklassifikation begann mit visuellen Beobachtungen und einfachen Beschreibungen von Galaxien durch Forscher wie Wolf (1908) und Vorontsov-Velyaminov (1960er Jahre). Später entwickelte Edwin Hubble (1926) ein systematisches Klassifikationssystem, das Galaxien in elliptische, spiralförmige und irreguläre Typen einteilte. Diese Klassifikationen ermöglichten es, physikalische Eigenschaften von Galaxien besser zu verstehen und wichtige Hypothesen zur Entstehung und Entwicklung von Galaxien aufzustellen. Moderne Ansätze, wie die Klassifikation durch maschinelles Lernen, bauen auf diesen Grundlagen auf, um die Effizienz und Genauigkeit bei der Analyse der großen Datenmengen zu verbessern, die heute zur Verfügung stehen.(Sandage, 1975, S. 4ff)

Die Klassifikation von Galaxien ist in der Astronomie von entscheidender Bedeutung, da die Galaxientypen Informationen darüber liefern, wie eine Galaxie entstanden ist und sich entwickelt hat. Die manuelle Durchführung der Klassifikationsaufgabe erfordert jedoch umfangreiches Fachwissen und ist sehr zeitaufwändig. Zudem werden Daten häufig auf zentralisierten Speicher-Servern gespeichert, was insbesondere in Bereichen wie der Fernerkundung und der Astronomie der Fall ist, wo jedes Jahr mehrere Petabyte an Daten produziert werden. Durch den Einsatz von Deep-Learning-Algorithmen kann dieser Prozess effizienter gestaltet werden, was Zeit spart und eine konsistenter Klassifikation ermöglicht. Das übergeordnete Geschäftsziel dieses Projekts ist es, einen automatisierten Ansatz zur Klassifikation von Galaxien zu entwickeln, der die Forschung im Bereich der kosmischen Evolution unterstützt und eine zeiteffiziente Methode zur Analyse großer Mengen von Galaxienbildern bereitstellt.(Hui et al., 2022, S. 1f; Oehmcke und Gieseke, 2022, S. 1)

b) Ziele der Datenanalysen

Die Datenanalyse in diesem Projekt zielt darauf ab, einen Proof of Concept (PoC) für die Bilderkennung von Galaxiedaten zu entwickeln. Im Mittelpunkt stehen die Identifikation und Durchführung relevanter Schritte der Datenvorbereitung sowie die Untersuchung potenzieller Herausforderungen und Probleme, die dabei auftreten können. Dabei liegt der Fokus nicht auf der Optimierung der Ergebnisse, sondern darauf, die notwendigen

Schritte herauszuarbeiten und deren Effektivität zu bewerten. Im Detail umfassen die Ziele der Datenanalyse:

Datenvorbereitung und Preprocessing:

- Die Bilder der Galaxien sollen so vorbereitet werden, dass sie für maschinelle Lernmodelle geeignet sind. Dazu gehören Normalisierung, Größenanpassung (z. B. Center Cropping) und Datenaugmentation.
- Identifikation der Herausforderungen bei der Datenvorbereitung, wie z. B. Datenungleichgewicht, Bildrauschen oder die Entfernung irrelevanter Bildbestandteile (z. B. Sterne).

Proof of Concept für Modellentwicklung:

- Aufbau eines grundlegenden Convolutional Neural Networks (CNN) zur Galaxienklassifikation.
- Nutzung eines vortrainierten Modells (Transfer Learning) als Orientierung, um die Ergebnisse aus der Literatur nachzustellen und die Schritte zu validieren.

2) Data Understanding

a) Exploration: Explorieren Sie die Daten qualitativ, quantitativ und grafisch, um einen Eindruck davon zu bekommen

```
In [14]: import gc
from astroNN.datasets import load_galaxy10
from tensorflow.keras import utils
import numpy as np
import psutil
import os
from astroNN.datasets.galaxy10 import galaxy10cls_lookup
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest
from skimage.morphology import remove_small_objects, binary_erosion
from skimage import img_as_ubyte
import cv2
import albumentations as A
from albumentations.core.composition import OneOf
from albumentations import Rotate, RandomCrop, RandomBrightnessContrast, HorizontalFlip
from albumentations.core.transforms_interface import ImageOnlyTransform
from sklearn.utils.class_weight import compute_class_weight
import tensorflow as tf
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import LearningRateScheduler
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import Callback
import seaborn as sns
```

Das erstmalige Laden der Daten

```
In [ ]: # To Load images and Labels (will download automatically at the first time)
# First time downloading location will be ~/.astroNN/datasets/
images, labels = load_galaxy10()

# To convert the labels to categorical 10 classes
labels = utils.to_categorical(labels, 10)

# To convert to desirable type
labels = labels.astype(np.uint8, copy=False)
#images = images.astype(np.float32, copy=False)
```

Erstmalige abspeichern der Numpy Daten für schnellers Laden.

```
In [2]: def save_to_numpy(image_file, images):
    np.save(image_file, images)
    print(f"Daten in {image_file} gespeichert.")

# Laden der Daten aus NumPy-Dateien
def load_from_numpy(image_file, ):
    images = np.load(image_file)
    print(f"Daten aus {image_file} geladen.")
    return images

# save_to_numpy('images.npy', images)
# save_to_numpy('Labels.npy', Labels)
```

```
In [ ]: # Laden
images = load_from_numpy('images.npy')
labels = load_from_numpy('labels.npy')

# Beispiel: Zugriff auf ein bestimmtes Bild
image_14258 = images[14258].copy()
```

```
In [34]: images[0][0][0]
```

```
Out[34]: array([68., 19., 23.], dtype=float32)
```

(68, 19, 23) ist der Farbcode für den Pixel, an der Stelle (0,0), im RGB-Farbraum. Die Bedeutung der einzelnen Werte:

68 ist der Rotwert. 19 ist der Grünwert. 23 ist der Blauwert. Diese Werte zusammen bestimmen die Farbe des Pixels. In diesem Fall ergibt die Kombination eine dunkle, rötlich-braune Farbe.

```
In [3]: # Images shape
images.shape
```

```
Out[3]: (17736, 256, 256, 3)
```

Anzahl von Bildern: 17736 Anzahl der Pixel für die Höhe: 256 Anzahl der Pixel für die Breite: 256 Anzahl der Farträume: 3

```
In [4]: # Labels shape
labels.shape
```

```
Out[4]: (17736, 10)
```

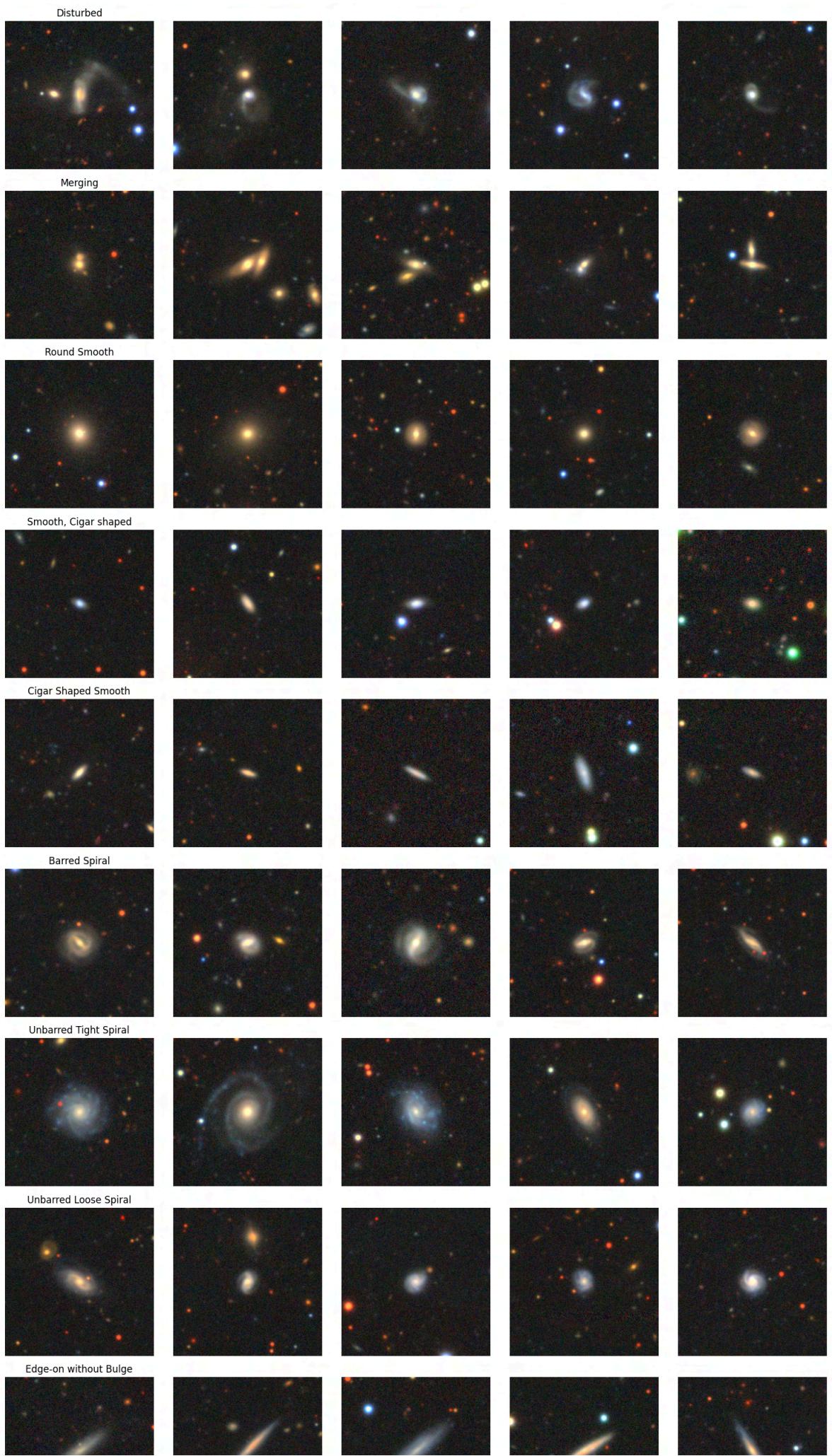
Anzahl der Bilder: 17736 Anzahl der Klassen: 10

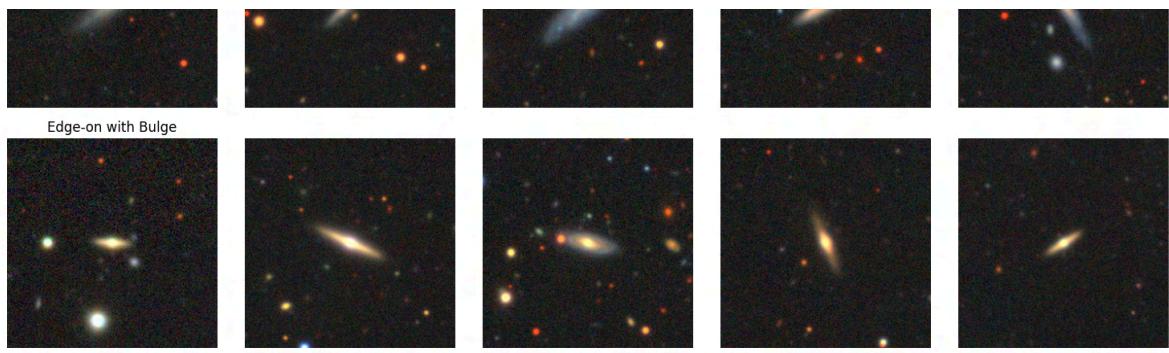
```
In [ ]: galaxy10cls_lookup(9)
```

```
Out[ ]: 'Edge-on with Bulge'
```

```
In [ ]:
```

```
# Ausgabe von fünf Bildern jeder Klasse
fig, ax = plt.subplots(10, 5, figsize=(15, 30))
for i in range(10):
    class_images = images[np.where(np.argmax(labels, axis=1) == i)]
    for j in range(5):
        ax[i, j].imshow(class_images[j] / 255.0)
        if j == 0:
            ax[i, j].set_title(galaxy10cls_lookup(i))
            ax[i, j].axis('off')
plt.tight_layout()
plt.show()
```





1. Disturbed: Diese Galaxien zeigen ein chaotisches Erscheinungsbild mit unregelmäßigen Strukturen. Sie könnten durch Kollisionen oder Verschmelzungen mit anderen Galaxien entstanden sein. Man erkennt oft verzerrte Formen, die auf äußere Einflüsse hindeuten.

2. Merging: Diese Galaxien befinden sich in einem Verschmelzungsprozess. Man kann zwei oder mehrere Galaxien erkennen, die zusammenstoßen oder bereits teilweise ineinander aufgegangen sind. Dies führt zu einem sehr komplexen und dynamischen Erscheinungsbild.

3. Round Smooth: Diese Galaxien sind glatt und rund, ohne erkennbare Spiralarme oder ausgeprägte Strukturen. Sie ähneln häufig elliptischen Galaxien und haben ein symmetrisches, einfaches Erscheinungsbild.

4. Smooth Cigar Shaped: Diese Galaxien haben eine glatte, längliche, zigarrenförmige Form. Sie wirken ähnlich wie elliptische Galaxien, aber mit einer gestreckten Struktur.

5. Cigar Shaped Smooth: Diese Klasse ähnelt der vorherigen Klasse und hat ebenfalls eine längliche, glatte Form. Der Unterschied in der Bezeichnung deutet oft nur auf subtile Abweichungen in der Erscheinung hin.

6. Barred Spiral: Diese Galaxien besitzen eine zentrale Balkenstruktur, die sich quer durch den Kern erstreckt. Von den Enden des Balkens gehen Spiralarme aus, was ihnen ein markantes Aussehen verleiht. Der Balken ist ein auffälliges Merkmal, das die Spiralarme strukturiert.

7. Unbarred Tight Spiral: Diese Spiralen besitzen keinen zentralen Balken, aber sehr eng gewundene Spiralarme. Die Spiralarme sind nah am Zentrum und verleihen der Galaxie ein kompaktes, dicht gewundenes Erscheinungsbild.

8. Unbarred Loose Spiral: Diese Galaxien haben keine zentrale Balkenstruktur und die Spiralarme sind lockerer gewunden, was ihnen ein ausgedehnteres und offeneres Erscheinungsbild verleiht. Die Spiralarme sind weniger eng um den Kern gewickelt.

9. Edge-on without Bulge: Diese Galaxien sind aus der Kante betrachtet (Kantenansicht) sichtbar, ohne einen ausgeprägten zentralen Bulge (helles, kugelförmiges Zentrum). Sie erscheinen als eine dünne, flache Linie ohne Verdickung in der Mitte.

10. Edge-on with Bulge: Auch hier sieht man die Galaxie aus der Kante, aber im Gegensatz zur vorherigen Klasse hat diese Galaxie einen zentralen Bulge. Dies verleiht ihr eine Verdickung in der Mitte der ansonsten flachen Linie.

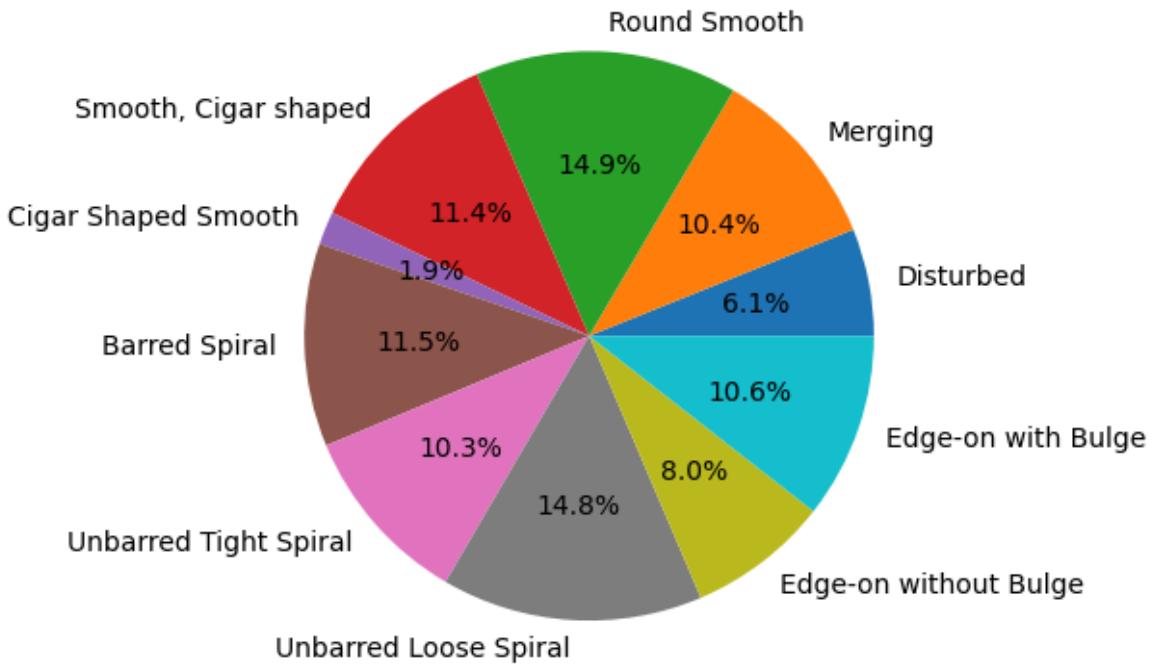
Ähnliche Klassen:

- Smooth Cigar Shaped und Cigar Shaped Smooth: Beide Klassen sind glatt und länglich, unterscheiden sich nur geringfügig in ihrer Beschreibung, was es schwer macht, sie zu unterscheiden.
- Unbarred Tight Spiral und Unbarred Loose Spiral: Beide haben keine Balkenstruktur und besitzen Spiralarme. Der Hauptunterschied liegt in der engen bzw. losen Windung der Spiralarme, was sie visuell ähnlich macht.
- Edge-on without Bulge und Edge-on with Bulge: Beide sind Galaxien, die aus der Kante betrachtet werden, wobei sich der Unterschied auf das Vorhandensein eines Bulges beschränkt.

```
In [ ]: # Anzahl der Datensätze pro Klasse
for i in range(10):
    print(f"galaxy10cls_lookup({i}): {len(images[np.where(np.argmax(labels, axis=1) == i)])}")

# Pie Chart
fig, ax = plt.subplots()
ax.pie([len(images[np.where(np.argmax(labels, axis=1) == i)])] for i in range(10))
plt.show()
```

Disturbed: 1081
Merging: 1853
Round Smooth: 2645
Smooth, Cigar shaped: 2027
Cigar Shaped Smooth: 334
Barred Spiral: 2043
Unbarred Tight Spiral: 1829
Unbarred Loose Spiral: 2628
Edge-on without Bulge: 1423
Edge-on with Bulge: 1873



- Ungleichmäßige Verteilung: Die Klassen sind nicht gleichmäßig verteilt, was eine Herausforderung für die Modellbildung darstellt. Zum Beispiel gibt es deutlich weniger Galaxien in der Klasse "Cigar Shaped Smooth" (334) im Vergleich zu "Round Smooth" (2645) oder "Unbarred Loose Spiral" (2628). Solche Ungleichgewichte können zu einem Problem der Klassifizierungsgenauigkeit führen, insbesondere wenn das Modell dazu neigt, häufiger vorkommende Klassen zu bevorzugen.
- Häufigste Klassen: Die häufigsten Klassen sind "Round Smooth" (2645), "Unbarred Loose Spiral" (2628), und "Smooth, Cigar Shaped" (2027). Diese Klassen haben eine große Anzahl von Beispielen, was positiv ist, da das Modell durch viele Trainingsdaten lernen kann, diese Kategorien gut zu erkennen. Allerdings kann dies auch dazu führen, dass seltener vertretene Klassen unterrepräsentiert bleiben und das Modell Schwierigkeiten hat, diese korrekt zu identifizieren.
- Seltene Klassen: Die Klasse "Cigar Shaped Smooth" ist mit nur 334 Instanzen die seltenste in diesem Datensatz. Auch "Disturbed" (1081) ist relativ unterrepräsentiert. Dies könnte bedeuten, dass diese Klassen schwieriger zu lernen sind, da das Modell möglicherweise nicht genügend Beispiele sieht, um die relevanten Merkmale gut zu verinnerlichen. Um das Modell darauf zu trainieren, auch diese Klassen gut zu erkennen, könnte eine Methode wie Oversampling, Datenaugmentation oder der Einsatz von gewichteten Verlustfunktionen hilfreich sein.
- Ähnliche Klassen mit ungleicher Verteilung: Wie bereits erwähnt, sind einige Klassen visuell sehr ähnlich, z.B. "Smooth, Cigar Shaped" und "Cigar Shaped Smooth". Interessanterweise gibt es hier eine große Ungleichverteilung zwischen diesen Klassen (2027 vs. 334), was das Modell dazu verleiten könnte, fälschlicherweise ähnliche Objekte in die dominantere Klasse einzusortieren. Dies könnte ein Hinweis darauf sein, dass diese Klassen stärker miteinander verwechselt werden könnten, es

sei denn, die Daten werden so vorbereitet, dass die Unterrepräsentierung ausgeglichen wird.

b) Analysen: Welche Analysen möchten bzw. können Sie durchführen, um Ihre Ziele zu erreichen?

- Explorative Datenanalyse : Zu Beginn wurde eine explorative Analyse der Galaxienbilder durchgeführt, um ein besseres Verständnis der Bilddaten zu gewinnen. Dabei wurde untersucht, wie die Verteilung der Klassen aussieht, ob es Ungleichgewichte gibt und wie die verschiedenen Kategorien sich visuell voneinander unterscheiden. Es umfasst auch das Visualisieren von Beispielbildern, um erste Erkenntnisse über Unterschiede zwischen den Klassen zu gewinnen.
- Vorverarbeitungsanalyse: Im nächsten Schritt wird eine detaillierte Analyse durchgeführt, um die optimalen Vorverarbeitungsschritte zu definieren, wie z.B. die Größenanpassung der Bilder. Dies stellt sicher, dass die Trainingsdaten qualitativ hochwertig sind.
- Modellvergleichsanalyse: Die Leistung des Modells wird analysiert. Dazu werden verschiedene Metriken wie die Genauigkeit und der Verlust während des Trainingsprozesses betrachtet.
- Fehleranalyse: Eine Analyse der Fehlklassifikationen wird durchgeführt, um zu verstehen, welche Galaxienbilder falsch klassifiziert wurden und mögliche Erklärung warum. Diese Analyse kann helfen, potenzielle Schwächen der Modelle zu identifizieren und die Trainingsdaten oder die Architektur der Modelle entsprechend anzupassen.

3) Data Preparation

a) Fehlende Daten: Gibt es fehlende oder nicht plausible Daten? Was tun Sie damit?

```
In [5]: # Missing values  
np.isnan(images).sum(), np.isnan(labels).sum()
```

```
Out[5]: (0, 0)
```

Es gibt keine fehlenden Werte.

```
In [7]: # Sum of all labels  
sum(np.sum(labels, axis=0))
```

```
Out[7]: 17736.0
```

Die Labels enthalten für jedes Bild nur eine Klasse.

```
In [ ]: # Plausibility check  
def check_image_dimensions(images: np.ndarray, expected_size=(256, 256), expecte
```

```

"""
Überprüft die Dimensionen und Farbkanäle der Bilder in einem Array.

Parameters:
- images (numpy.ndarray): Array von Bildern.
- expected_size (tuple): Erwartete Größe der Bilder.
- expected_channels (int): Erwartete Anzahl der Farbkanäle.

Returns:
- None
"""

inconsistent = []
for i, img in enumerate(images):
    if img.shape[:2] != expected_size or img.shape[2] != expected_channels:
        inconsistent.append(i)
if inconsistent:
    print(f"Inkonstante Bilder gefunden: {len(inconsistent)}")
    for idx in inconsistent:
        print(f"- Bildindex: {idx}, Größe: {images[idx].shape}")
else:
    print("Alle Bilder haben die erwarteten Dimensionen und Farbkanäle.")

check_image_dimensions(images)

```

Alle Bilder haben die erwarteten Dimensionen und Farbkanäle.

b) Ausreißer: Werten Sie alle Daten aus oder schließen Sie Ausreißer oder irrelevante Daten aus? Warum?

```

In [ ]: # Reshape the images to a 2D array (flatten the images)
features = images.reshape(images.shape[0], -1)

# Funktion zur Erkennung von Ausreißern
def detect_outliers(features: np.ndarray) -> np.ndarray:
    """
    Erkennt Ausreißer in einem Datensatz mit Hilfe des Isolation Forest Algorithmus.

    Parameters:
    - features (numpy.ndarray): Datensatz, in dem Ausreißer erkannt werden sollen

    Returns:
    - numpy.ndarray: Indizes der Ausreißer im Datensatz.
    """
    clf = IsolationForest(random_state=42)
    clf.fit(features)
    outliers = clf.predict(features)
    return np.where(outliers == -1)[0]

outlier_indices = detect_outliers(features)
print(f"Anzahl der Ausreißer: {len(outlier_indices)}")

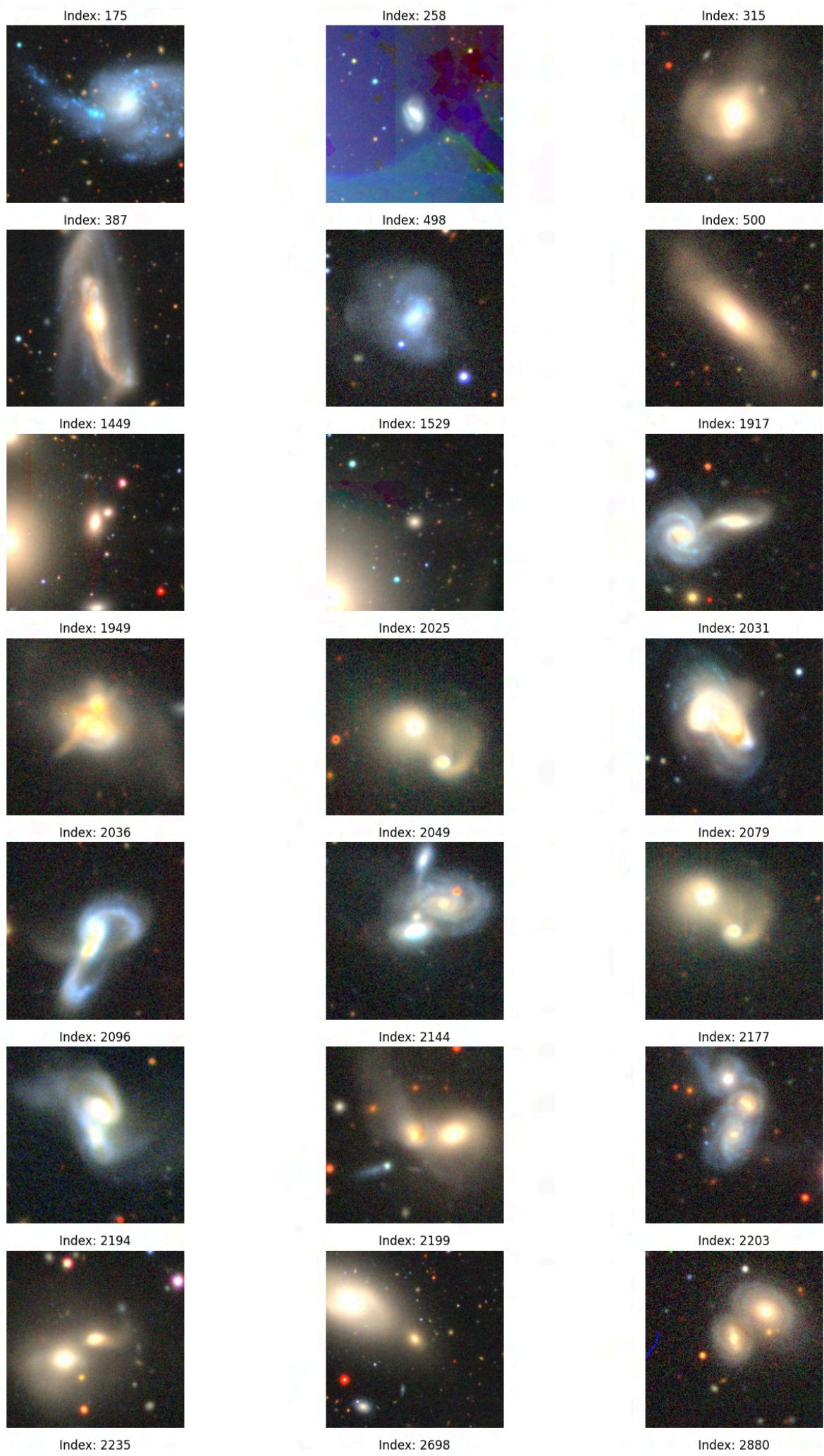
```

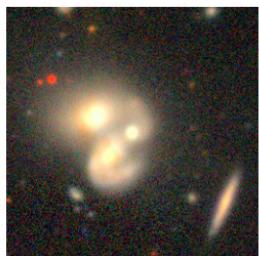
Anzahl der Ausreißer: 159

Der IsolationForest ist ein Algorithmus zur Anomalieerkennung, der in der Bibliothek scikit-learn implementiert ist. Er isoliert Beobachtungen, indem er zufällig ein Merkmal auswählt und dann einen zufälligen Schwellenwert zwischen dem maximalen und minimalen Wert des ausgewählten Merkmals festlegt. Dieser Prozess wird rekursiv

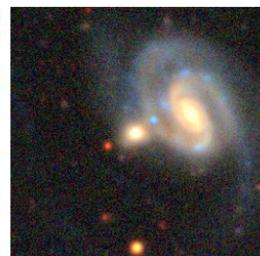
durchgeführt und kann als Baumstruktur dargestellt werden. Die Anzahl der Aufteilungen, die erforderlich sind, um eine Stichprobe zu isolieren, entspricht der Pfadlänge vom Wurzelknoten zum Endknoten. Kürzere Pfadlängen deuten auf Anomalien hin, da sie leichter zu isolieren sind. (Liu, 2009)

```
In [43]: # Show all outliers in a Matrix with 3 columns and 53 rows with their index
fig, ax = plt.subplots(53, 3, figsize=(15, 150))
for i, idx in enumerate(outlier_indices):
    ax[i // 3, i % 3].imshow(images[idx] / 255.0)
    ax[i // 3, i % 3].set_title(f"Index: {idx}")
    ax[i // 3, i % 3].axis('off')
plt.tight_layout()
plt.show()
```





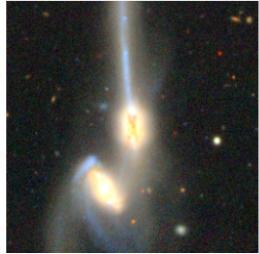
Index: 2891



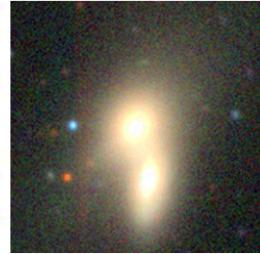
Index: 2897



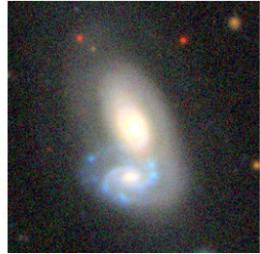
Index: 2899



Index: 2910



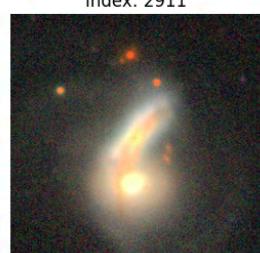
Index: 2911



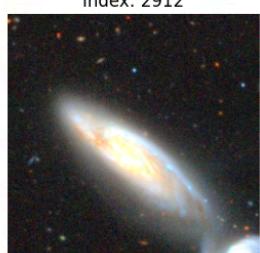
Index: 2912



Index: 2922



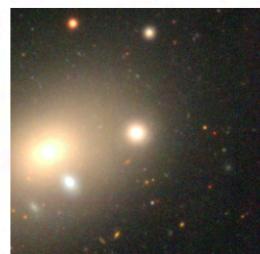
Index: 3367



Index: 3449



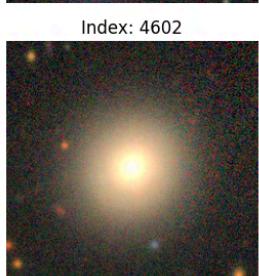
Index: 4602



Index: 5535



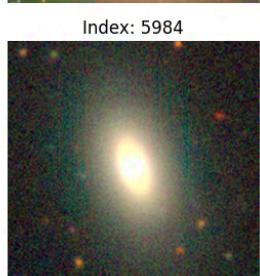
Index: 5984



Index: 5985



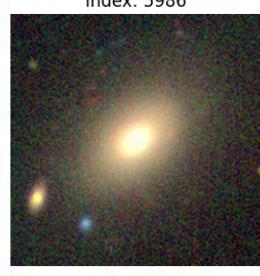
Index: 5986



Index: 7510



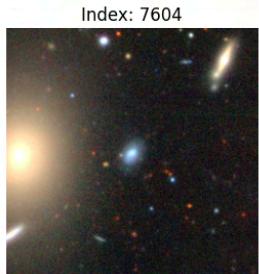
Index: 7604



Index: 7956



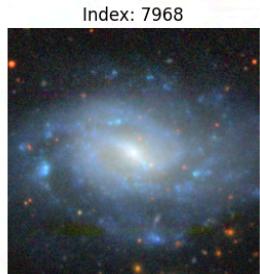
Index: 7968



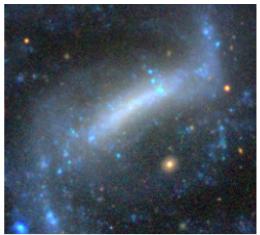
Index: 7986



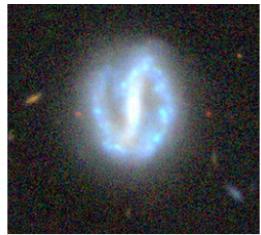
Index: 7998



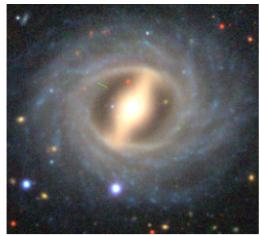
Index: 8153



Index: 8193



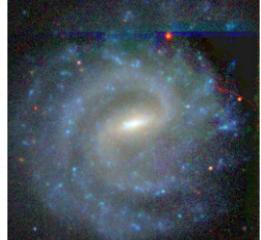
Index: 8214



Index: 8281



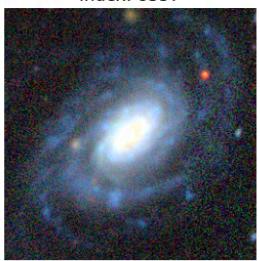
Index: 8357



Index: 8362



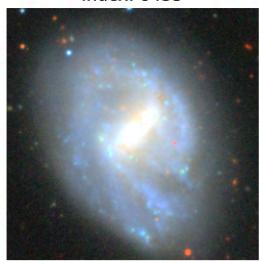
Index: 8433



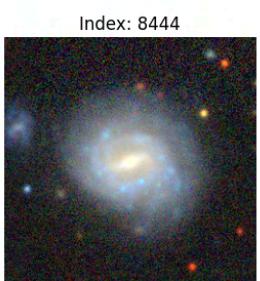
Index: 8444



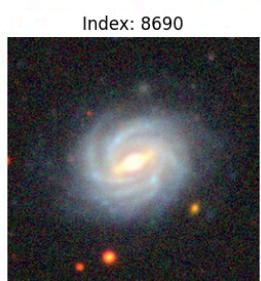
Index: 8690



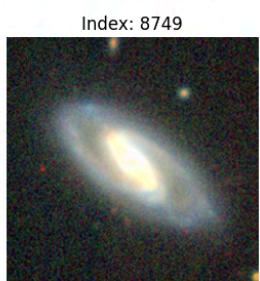
Index: 8749



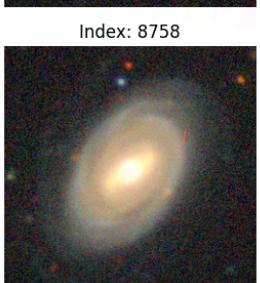
Index: 8758



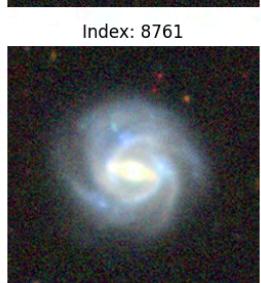
Index: 8761



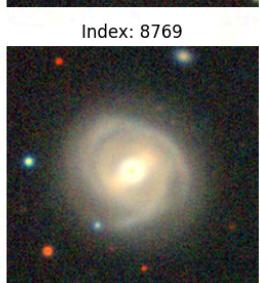
Index: 8769



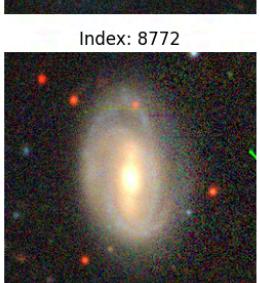
Index: 8772



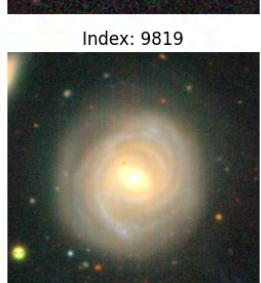
Index: 9819



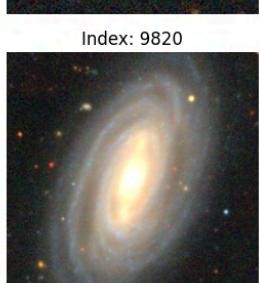
Index: 9820



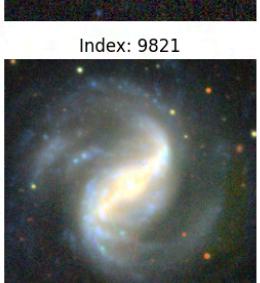
Index: 9821



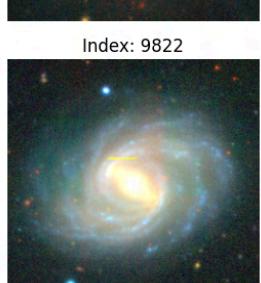
Index: 9822



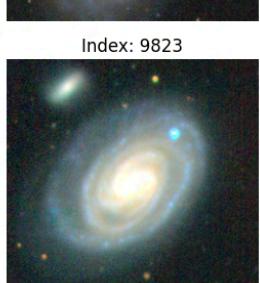
Index: 9823



Index: 9825



Index: 9826



Index: 9828



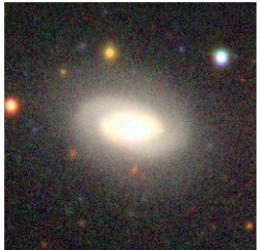
Index: 9867



Index: 9875



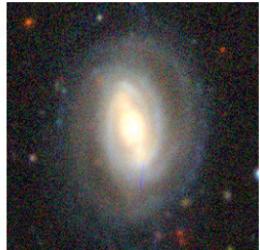
Index: 9878



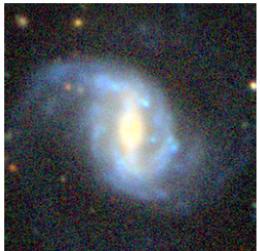
Index: 9879



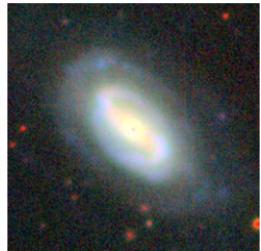
Index: 9887



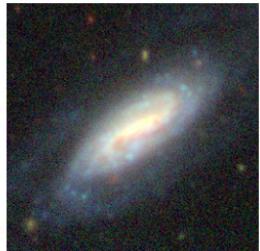
Index: 9902



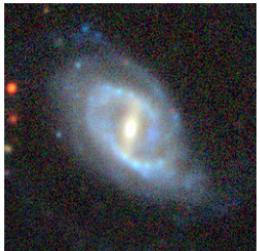
Index: 9913



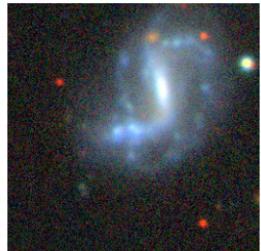
Index: 9981



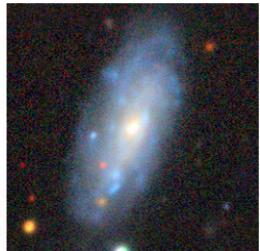
Index: 10144



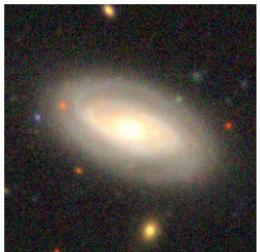
Index: 10315



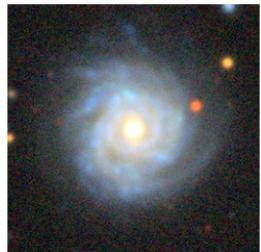
Index: 10350



Index: 10400



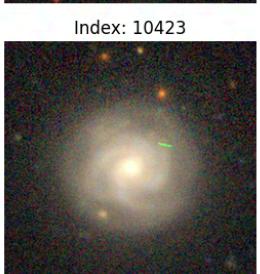
Index: 10423



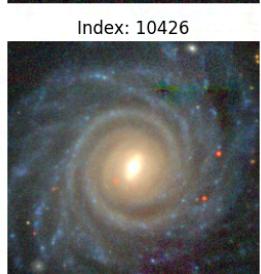
Index: 10426



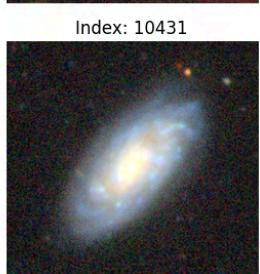
Index: 10431



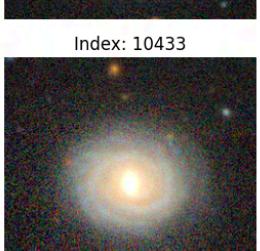
Index: 10433



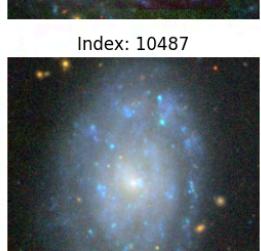
Index: 10487



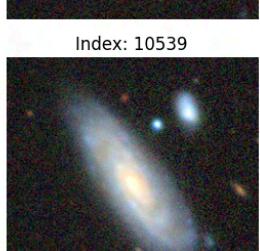
Index: 10539



Index: 10585



Index: 10603



Index: 10605



Index: 10609



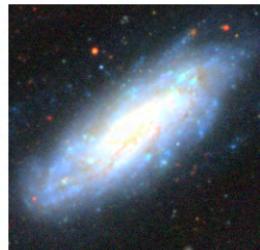
Index: 10709



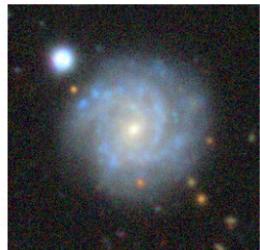
Index: 10760



Index: 10894



Index: 10999



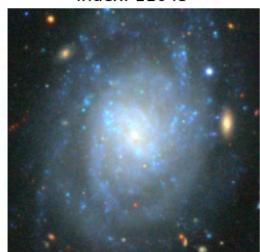
Index: 11045



Index: 11071



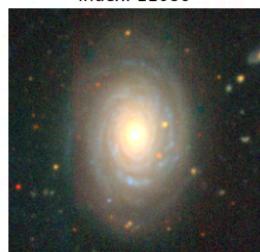
Index: 11086



Index: 11091



Index: 11092



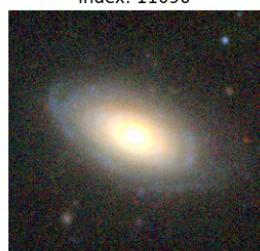
Index: 11096



Index: 11102



Index: 11111



Index: 11114



Index: 11134



Index: 11173



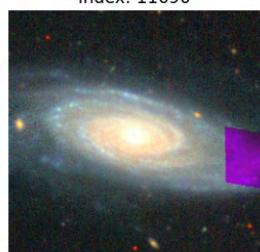
Index: 11696



Index: 11713



Index: 11723



Index: 11724



Index: 11729





Index: 11732



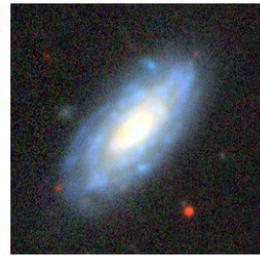
Index: 11733



Index: 11735



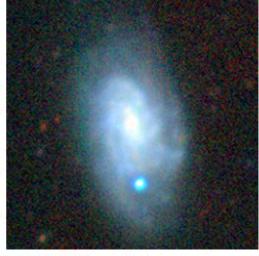
Index: 11738



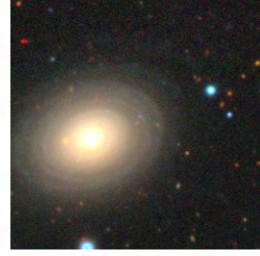
Index: 11784



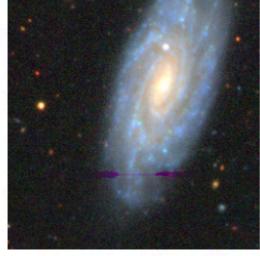
Index: 11786



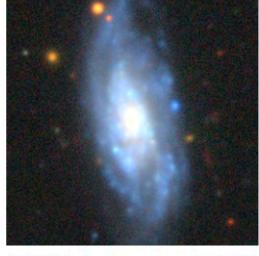
Index: 12420



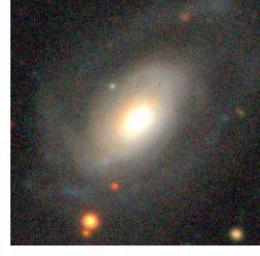
Index: 12494



Index: 12496



Index: 12498



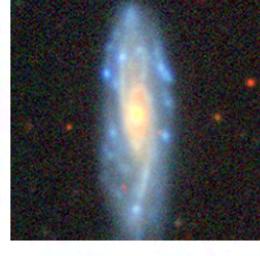
Index: 12507



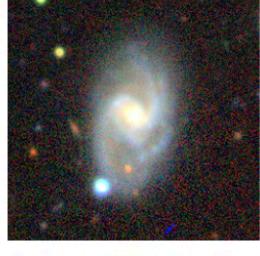
Index: 12512



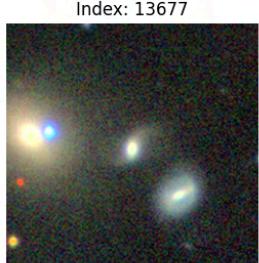
Index: 13677



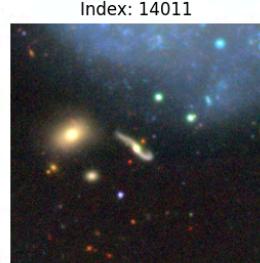
Index: 14011



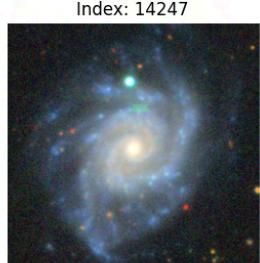
Index: 14247



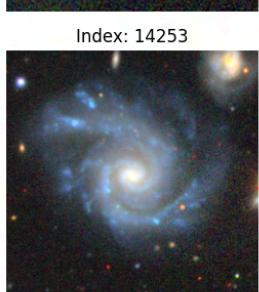
Index: 14253



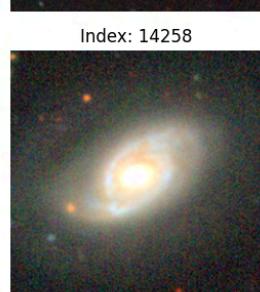
Index: 14258



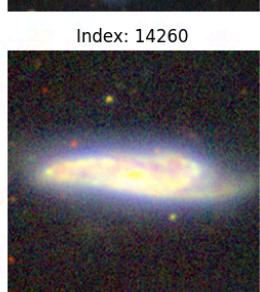
Index: 14260



Index: 14272



Index: 14292



Index: 14397





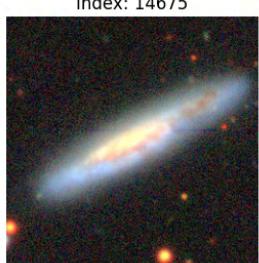
Index: 14675



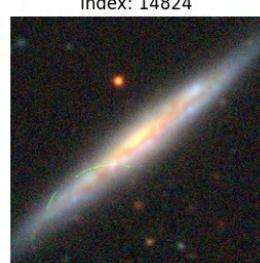
Index: 14824



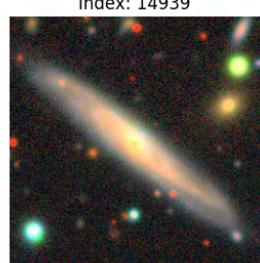
Index: 14939



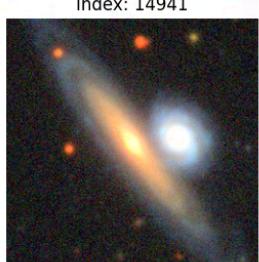
Index: 14941



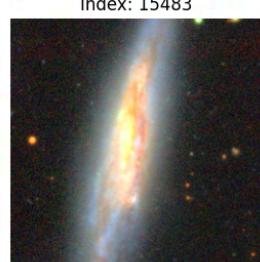
Index: 15483



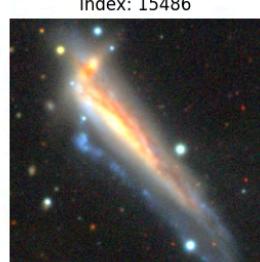
Index: 15486



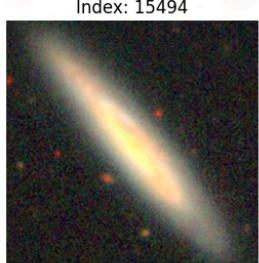
Index: 15494



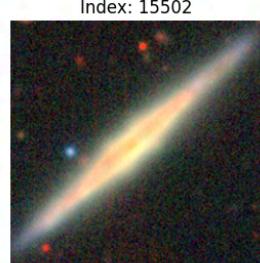
Index: 15502



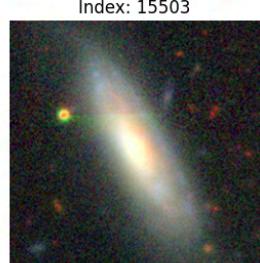
Index: 15503



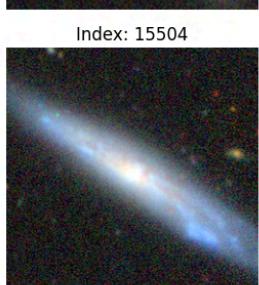
Index: 15504



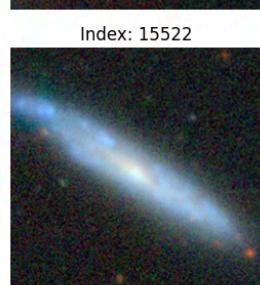
Index: 15522



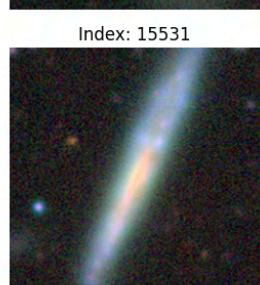
Index: 15531



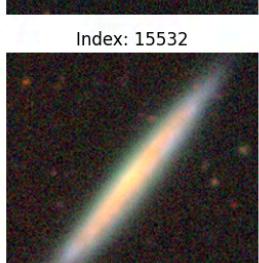
Index: 15532



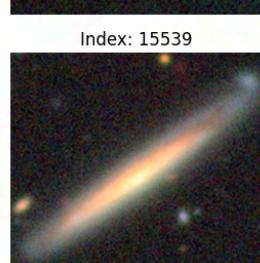
Index: 15539



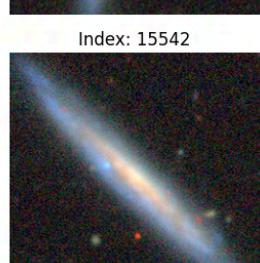
Index: 15542



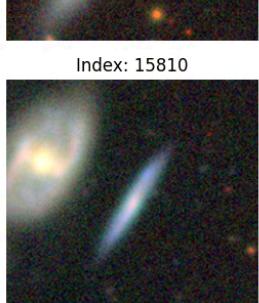
Index: 15810



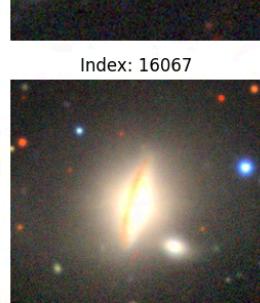
Index: 16067



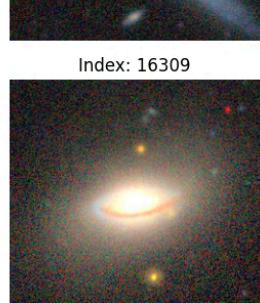
Index: 16309



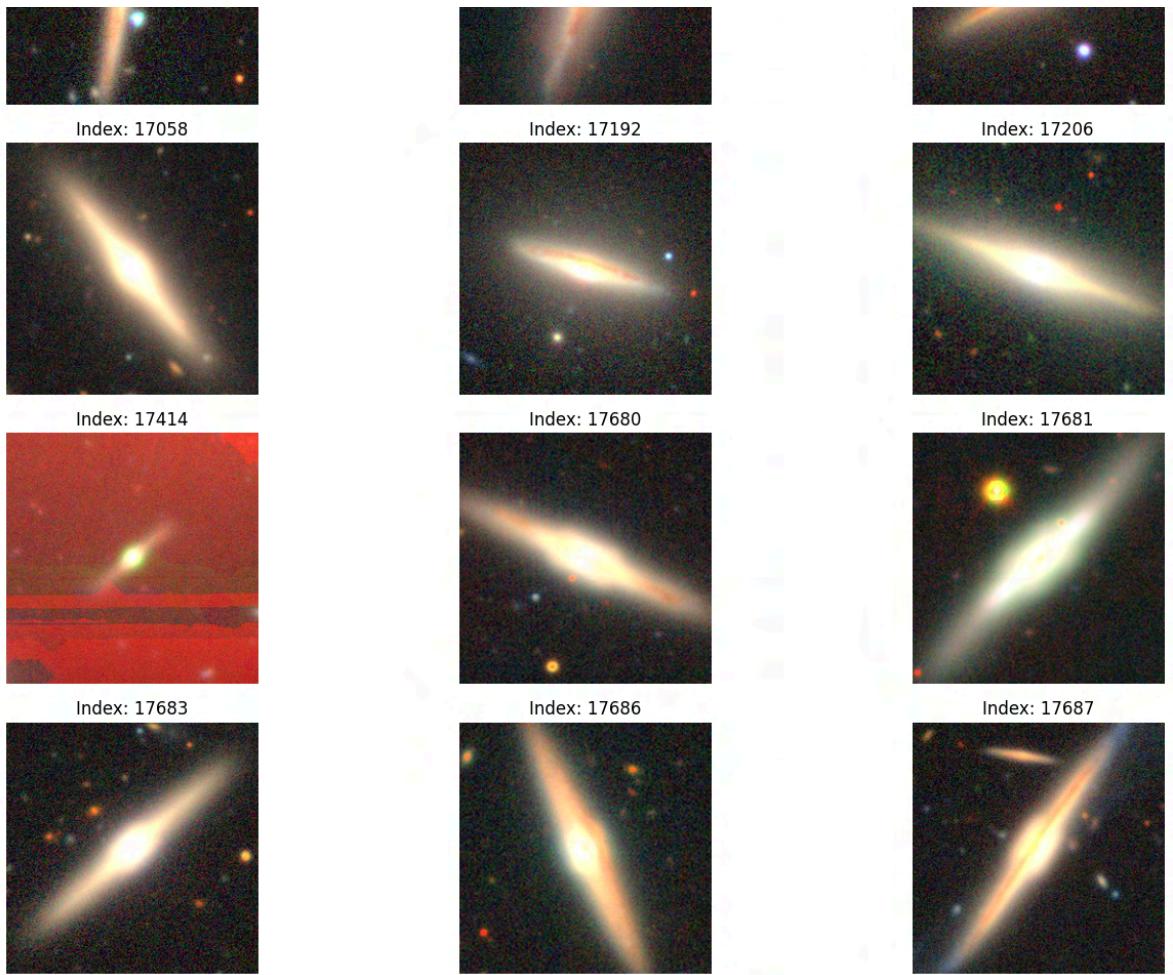
Index: 16389



Index: 16543



Index: 16757



Von den 159 Bildern die als Ausreißer identifiziert wurden, wurden nur drei als eindeutige Visuelle Ausreißer identifiziert.

```
In [44]: # Plt Image with index 258, 3449 and 17414
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
for i, idx in enumerate([258, 3449, 17414]):
    ax[i].imshow(images[idx] / 255.0)
    ax[i].set_title(f"Index: {idx}")
    ax[i].axis('off')
plt.tight_layout()
plt.show()
```



```
In [45]: # Remove outliers with index 258, 3449 and 17414
images = np.delete(images, [258, 3449, 17414], axis=0)
labels = np.delete(labels, [258, 3449, 17414], axis=0)
```

Diese Ausreißer wurden aus dem Datensatz entfernt da sie erhebliche Farbfehler aufweisen. Die anderen nicht entfernten Ausreißer werden später genauer untersucht ob es signifikante Unterschiede der Klassifikationsgenauigkeit im Vergleich zu den nicht identifizierten Ausreißer gibt.

c) Diskretisierung: Wollen Sie aus diskreten Werten kontinuierliche machen oder umgekehrt?

In den weiteren Data Preparation Schritte werden die vorgenommene Daten Änderungen von Hui et al. (2022) durchgeführt um die Bild Daten für den optimalen Einsatz vorzubereiten. In diesem Schritt werden die Bild Daten am äußeren Rand abgeschnitten um die Bildgröße zu verkleinern. Die Pixelanzahl wird von (256,256) zu (224,224) verkleinert. Es werden nicht viele Informationen verloren gegangen, da die Galaxie jeweils im Zentrum des Bildes liegen. (Hui et al., 2022, S. 4)

```
In [45]: def center_crop_all(images: np.ndarray, target_size=(224, 224)) -> np.ndarray:
    """
        Schneidet den zentralen Bereich aller Bilder aus.

    Parameters:
    - images (numpy.ndarray): Array von Bildern mit der Form (N, H, W, C).
    - target_size (tuple): Zielgröße als (Höhe, Breite).

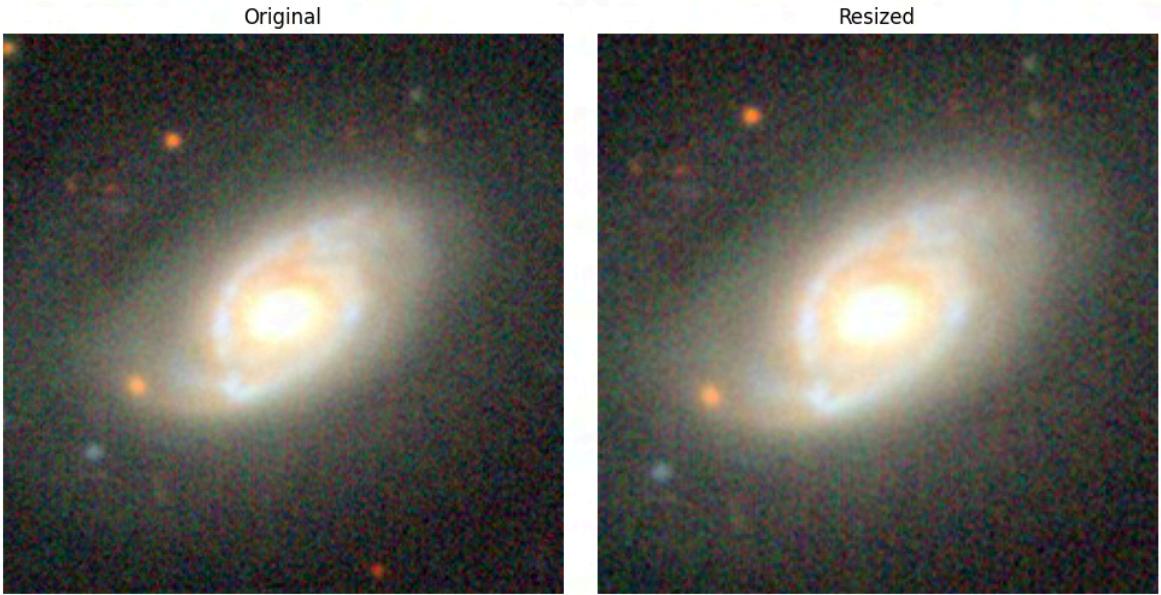
    Returns:
    - numpy.ndarray: Array von zugeschnittenen Bildern.
    """
    N, H, W, C = images.shape
    th, tw = target_size
    start_y = (H - th) // 2
    start_x = (W - tw) // 2
    cropped_images = images[:, start_y:start_y + th, start_x:start_x + tw, :].copy()
    return cropped_images

# Anwenden des Center Crop
images_cropped = center_crop_all(images, target_size=(224, 224))
print(f"Cropped Images Shape: {images_cropped.shape}, Dtype: {images_cropped.dtype}")
images_cropped_14258 = images_cropped[14258].copy()
```

Cropped Images Shape: (17736, 224, 224, 3), Dtype: float32

Die Bilder wurden erfolgreich verkleinert von 256x256 in 224x224 Pixel

```
In [ ]: # Compare the original and resized images
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].imshow(image_14258 / 255.0)
ax[0].set_title("Original")
ax[0].axis('off')
ax[1].imshow(images_cropped_14258 / 255.0)
ax[1].set_title("Resized")
ax[1].axis('off')
plt.tight_layout()
plt.show()
```



Der Rand mit den nicht relevanten Informationen wurde erfolgreich abgeschnitten. Die alten original Daten werden gelöscht um den Arbeitsspeicher zu entlasten.

```
In [46]: images = None
del images
gc.collect()
```

```
Out[46]: 11
```

d) Normalisierung: Macht es Sinn, Daten zu normalisieren? Zu projizieren oder kalibrieren?

Bevor die Daten normalisiert werden, wird eine morphologische Öffnung durchgeführt, die zu einer Verberessung der Modellleistung führt (Haralick et al., 1987). Hier wird eine morphologische Öffnung durchgeführt die teilweise unbedeutene kleine Objekte die nicht im Zentrum und vom dem eigentlich, die Galaxie, ablenken können und Rauschen wird entfernt (Hui et al., 2022, S. 5).

```
In [47]: def remove_large_objects(images: np.ndarray, kernel_size_opening=9, blur_kernel_
    """
        Entfernt Rauschen, Sterne und größere Punkte, ohne die Galaxie zu beeinträchtigen.

        Parameters:
        - images (numpy.ndarray): Array von Bildern.
        - kernel_size_opening (int): Größe des Kernels für morphologische Öffnung.
        - blur_kernel_size (tuple): Kernelgröße für minimalen Gaussian Blur.
        - min_area (int): Minimale Größe der Objekte, die berücksichtigt werden soll.
        - max_area (int): Maximale Größe der Objekte, die entfernt werden sollen.

        Returns:
        - numpy.ndarray: Array von verbesserten Bildern.
    """
    # Kernel für morphologische Öffnung
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (kernel_size_opening,
    processed_images = np.empty_like(images)

    for i, img in enumerate(images):
```

```

# Konvertieren zu uint8, falls erforderlich
if img.dtype != np.uint8:
    img = img.astype(np.uint8)

# Schritt 1: Minimale Glättung
blurred = cv2.GaussianBlur(img, blur_kernel_size, 0)

# Schritt 2: Morphologische Öffnung
opened = cv2.morphologyEx(blurred, cv2.MORPH_OPEN, kernel)

# Schritt 3: Erstellen einer Maske für größere Punkte
gray = cv2.cvtColor(opened, cv2.COLOR_BGR2GRAY) if len(opened.shape) == 3, binary = cv2.threshold(gray, 30, 255, cv2.THRESH_BINARY)

# Konturen finden
contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
mask = np.zeros_like(gray)

for contour in contours:
    area = cv2.contourArea(contour)
    if min_area < area < max_area:
        cv2.drawContours(mask, [contour], -1, 255, thickness=cv2.FILLED)

# Maske invertieren, um größere Punkte zu entfernen
mask = cv2.bitwise_not(mask)

# Schritt 4: Anwenden der Maske auf das Originalbild
result = cv2.bitwise_and(opened, opened, mask=mask)

# Ergebnis speichern
processed_images[i] = result

return processed_images

```

Beispieldaufruf

```

images_opened_cv2 = remove_large_objects(images_cropped, kernel_size_opening=9,
print(f"images_opened_cv2 Shape: {images_opened_cv2.shape}, Dtype: {images_opened_cv2.dtype}")
images_opened_cv2_14258 = images_opened_cv2[14258].copy()

```

images_opened_cv2 Shape: (17736, 224, 224, 3), Dtype: float32

In [11]:

```

# Compare the original and resized images
fig, ax = plt.subplots(1, 3, figsize=(10, 5))
ax[0].imshow(image_14258 / 255.0)
ax[0].set_title("Original")
ax[0].axis('off')
ax[1].imshow(images_cropped_14258 / 255.0)
ax[1].set_title("Resized")
ax[1].axis('off')
ax[2].imshow(images_opened_cv2_14258 / 255.0)
ax[2].set_title("Opened")
ax[2].axis('off')
plt.tight_layout()
plt.show()

```



Die cropped Daten werden gelöscht um den Arbeitsspeicher zu entlasten.

```
In [48]: images_cropped = None
del images_cropped
gc.collect()
```

```
Out[48]: 0
```

```
In [49]: save_to_numpy('images_opened_cv2.npy', images_opened_cv2)

Bilder in images_opened_cv2.npy gespeichert.
```

```
In [ ]: del images_opened_cv2
gc.collect()
```

```
In [3]: images_opened_cv2 = load_from_numpy('images_opened_cv2.npy')
labels = load_from_numpy('labels.npy')
```

Daten aus images_opened_cv2.npy geladen.
Daten aus labels.npy geladen.

Für die Overfitting prävention und realitätsnahe Praxis werden Schritte der Datenaugmentation durchgeführt. Die einzelnen Schritte (Hui et al., 2022, S. 4):

- RandomRotation

Das Bild wird um einen zufälligen Winkel (innerhalb eines bestimmten Bereichs, z. B. -20° bis +20°) gedreht. Diese Transformation hilft dem Modell, invariantes Verhalten gegenüber Rotation zu lernen. Effekt: Es simuliert verschiedene Perspektiven der Galaxien.

- RandomTranslation

Verschiebt das Bild zufällig horizontal oder vertikal (z. B. um $\pm 10\%$ der Bildgröße). Dies hilft dem Modell, mit leichten Positionsverschiebungen der Objekte umzugehen. Effekt: Erzeugt Variationen der Position im Bild.

- RandomContrast

Verändert zufällig den Kontrast des Bildes, um unterschiedliche Helligkeitsbedingungen zu simulieren. Dies kann wichtig sein, wenn die Bilder aus verschiedenen Quellen stammen. Effekt: Robustheit gegenüber Beleuchtungsvariationen.

- RandomFlip

Spiegelt das Bild zufällig horizontal oder vertikal. Dies hilft, Symmetrieffekte im Bild auszunutzen. Effekt: Variationen durch Spiegelung.

```
In [6]: images_opened_cv2_14258 = images_opened_cv2[14258].copy()

def data_augmentation_pipeline(image: np.ndarray) -> np.ndarray:
    """
    Wendet eine Datenaugmentierungspipeline auf ein Eingabebild an.
    """

    # Definieren der Augmentierungssequenz ohne RandomBrightnessContrast
    transform = A.Compose([
        A.Rotate(limit=20, p=0.5, border_mode=cv2.BORDER_CONSTANT, value=(128, 1),
        A.ShiftScaleRotate(shift_limit=0.1, scale_limit=0.0, rotate_limit=0, p=0,
        A.HorizontalFlip(p=0.5), # Zufällige horizontale Spiegelung
    ])

    # Anwenden der Augmentierung ohne Helligkeits- und Kontrastanpassung
    augmented = transform(image=image)
    augmented_image = augmented['image']

    # Zusätzliche Helligkeits- und Kontrastanpassung manuell durchführen
    if np.random.rand() < 0.5:
        augmented_image = adjust_brightness_contrast(augmented_image, brightness=0.1, contrast=0.1)

    # Sicherstellen, dass Pixelwerte im gültigen Bereich liegen
    augmented_image = np.clip(augmented_image, 0, 255).astype(np.float32)

    return augmented_image

def adjust_brightness_contrast(image: np.ndarray, brightness=0.1, contrast=0.1):
    """
    Manuelle Anpassung von Helligkeit und Kontrast.
    """

    beta = brightness * 255 # Skaliere Helligkeit
    alpha = 1 + contrast # Skaliere Kontrast
    adjusted = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)
    return adjusted

print(f"images_opened_cv2 Shape: {images_opened_cv2.shape}, Dtype: {images_opened_cv2.dtype}")
# Beispieldruck
augmented_image = data_augmentation_pipeline(images_opened_cv2_14258)
print(f"augmented_image Shape: {augmented_image.shape}, Dtype: {augmented_image.dtype}")
augmented_image_14258 = augmented_image
```

```
images_opened_cv2 Shape: (17736, 224, 224, 3), Dtype: float32
augmented_image Shape: (224, 224, 3), Dtype: float32
```

Im weiteren Verlauf wird die Datenaugmentation genutzt um zusätzliche Varianten von Bildern für die unterpräsentierten Klassen zu erstellen.

```
In [17]: # Compare the original and resized images
fig, ax = plt.subplots(1, 4, figsize=(10, 5))
ax[0].imshow(image_14258 / 255.0)
```

```

ax[0].set_title("Original")
ax[0].axis('off')
ax[1].imshow(images_cropped_14258 / 255.0)
ax[1].set_title("Resized")
ax[1].axis('off')
ax[2].imshow(images_opened_cv2_14258 / 255.0)
ax[2].set_title("Opened")
ax[2].axis('off')
ax[3].imshow(augmented_image_14258 / 255.0)
ax[3].set_title("Augmentation")
ax[3].axis('off')
plt.tight_layout()
plt.show()

```



```

In [13]: # Gesamtspeicher
def print_memory_usage():
    total_memory = psutil.virtual_memory().total / (1024 ** 3) # in GiB
    available_memory = psutil.virtual_memory().available / (1024 ** 3) # in GiB
    used_memory = psutil.virtual_memory().used / (1024 ** 3) # in GiB

    print(f"Gesamtspeicher: {total_memory:.2f} GiB")
    print(f"Verfügbarer Speicher: {available_memory:.2f} GiB")
    print(f"Genutzter Speicher: {used_memory:.2f} GiB")

    # Speicherverbrauch des aktuellen Python-Prozesses
    process = psutil.Process(os.getpid())
    process_memory = process.memory_info().rss / (1024 ** 3) # in GiB
    print(f"Speicherverbrauch des Python-Prozesses: {process_memory:.2f} GiB")

```

```

In [ ]: print_memory_usage()

```

e) Ausgewogenheit: Sind die Kategorien ausgewogen? Versuchen Sie es mit Undersampling oder Oversampling?

Die Klassenverteilung wurde im Data Understanding schon analysiert:

- **Disturbed:** 1081
- **Merging:** 1853
- **Round Smooth:** 2645
- **Smooth, Cigar Shaped:** 2027
- **Cigar Shaped Smooth:** 334
- **Barred Spiral:** 2043
- **Unbarred Tight Spiral:** 1829
- **Unbarred Loose Spiral:** 2628
- **Edge-on without Bulge:** 1423

- **Edge-on with Bulge:** 1873

Es werden signifikante Unterschiede zwischen den Klassenverteilungen festgestellt.

Deswegen werden zusätzliche Bilder mit der Augmentation für unterrepräsentierte Klassen erstellt und anschließend noch Klassengewichte verwendet um die weitere Ungleichheit in den Daten entgegenzusetzen. Dabei werden selteneren Klassen wie Cigar Shaped Smooth höhere Gewichte zugeordnet, damit die Verlustfunktion diese Klassen stärker berücksichtigt um die Robustheit der Klassifikation sowie die Konvergenzgeschwindigkeit beschleunigt. (Hui et al., 2022, 6) (Xu et al., 2020)

Es wird mit der Datenaugmentation neue Bilder generiert bis jede Klasse mindestens 1500 Bilder hat. Es wurde verzichtet so viele Bilder zu generieren das jede Klasse gleich viel Daten hat aufgrund fehlenden Arbeitsspeicher.

```
In [7]: def augment_class_images(images, target_count, augmentation_function):  
    """  
        Augmentiert Bilder einer bestimmten Klasse, um die Zielanzahl zu erreichen.  
        Die augmentierten Bilder werden direkt in das bestehende Array eingefügt.  
  
        Parameters:  
        - images (np.ndarray): Originalbilder der Klasse.  
        - target_count (int): Zielanzahl an Bildern nach Augmentation.  
        - augmentation_function (callable): Funktion zur Augmentation.  
  
        Returns:  
        - np.ndarray: Augmentierte Bilder.  
    """  
  
    # Anzahl der fehlenden Bilder berechnen  
    missing_count = target_count - len(images)  
    augmented_images = []  
  
    # Bilder augmentieren, bis die Zielanzahl erreicht ist  
    for _ in range(missing_count):  
        img = images[np.random.randint(0, len(images))] # Zufälliges Bild auswählen  
        augmented_images.append(augmentation_function(img))  
  
    return np.array(augmented_images)  
  
  
# Zielanzahl pro Klasse  
target_count = 1500  
  
# Augmentation und Labels direkt in die bestehenden Arrays einfügen  
for i in range(10): # Für jede Klasse  
    gc.collect()  
    class_images = images_opened_cv2[np.where(np.argmax(labels, axis=1) == i)]  
    print(f"Klasse {galaxy10cls_lookup(i)} hat {len(class_images)} Bilder."  
  
    if len(class_images) < target_count:  
        print(f"Augmentiere Klasse {galaxy10cls_lookup(i)} auf {target_count} Bilder")  
  
        # Fehlende augmentierte Bilder generieren  
        augmented_images = augment_class_images(class_images, target_count, data  
  
        # Zu den Originalbildern hinzufügen  
        images_opened_cv2 = np.concatenate([images_opened_cv2, augmented_images])
```

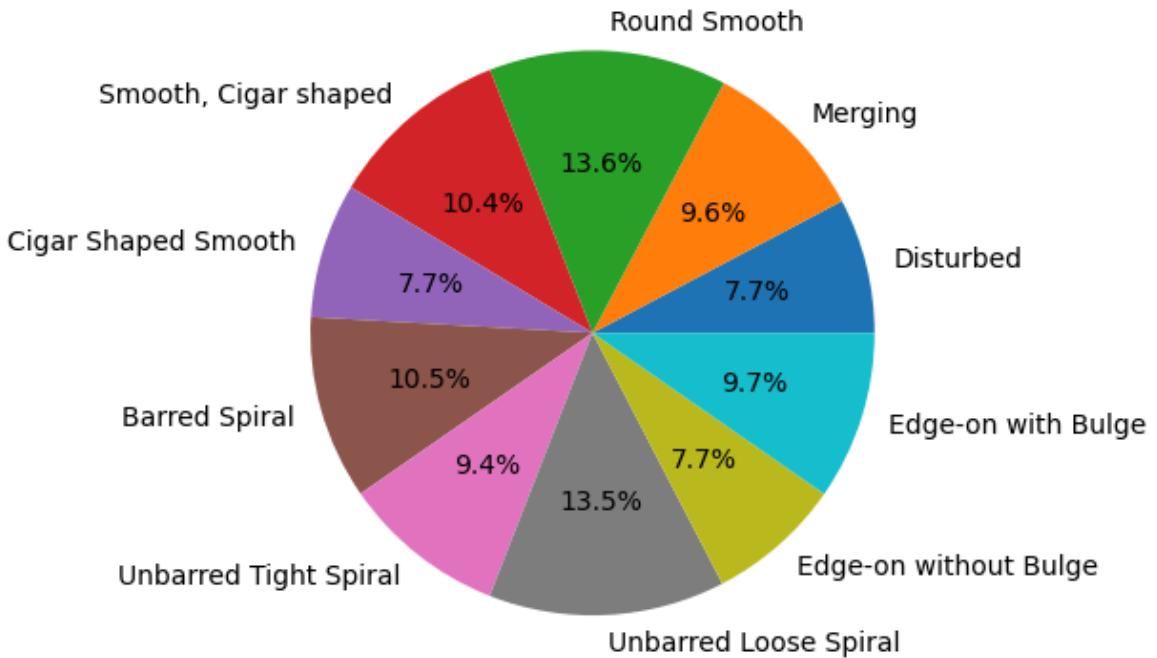
```
# Labels für die neuen Bilder generieren und hinzufügen
new_labels = np.zeros((len(augmented_images), labels.shape[1]))
del augmented_images
gc.collect()
new_labels[:, i] = 1 # One-Hot-Label für die Klasse setzen
labels = np.concatenate([labels, new_labels], axis=0)

# Ausgabe der neuen Formen
print(f"Neue Datenform: {images_opened_cv2.shape}, Neue Label-Form: {labels.shap

Klasse Disturbed hat 1081 Bilder.
Augmentiere Klasse Disturbed auf 1500 Bilder.
Klasse Merging hat 1853 Bilder.
Klasse Round Smooth hat 2645 Bilder.
Klasse Smooth, Cigar shaped hat 2027 Bilder.
Klasse Cigar Shaped Smooth hat 334 Bilder.
Augmentiere Klasse Cigar Shaped Smooth auf 1500 Bilder.
Klasse Barred Spiral hat 2043 Bilder.
Klasse Unbarred Tight Spiral hat 1829 Bilder.
Klasse Unbarred Loose Spiral hat 2628 Bilder.
Klasse Edge-on without Bulge hat 1423 Bilder.
Augmentiere Klasse Edge-on without Bulge auf 1500 Bilder.
Klasse Edge-on with Bulge hat 1873 Bilder.
Neue Datenform: (19398, 224, 224, 3), Neue Label-Form: (19398, 10)
```

```
In [58]: for i in range(10):
    print(f'{galaxy10cls_lookup(i)}: {len(images_opened_cv2[np.where(np.argmax(labels, axis=1) == i)])} for i in range(10)
fig, ax = plt.subplots()
ax.pie([len(images_opened_cv2[np.where(np.argmax(labels, axis=1) == i)]) for i in range(10)],
       labels=[galaxy10cls_lookup(i) for i in range(10)],
       autopct="%1.1f%%")
plt.show()
```

Disturbed: 1500
Merging: 1853
Round Smooth: 2645
Smooth, Cigar shaped: 2027
Cigar Shaped Smooth: 1500
Barred Spiral: 2043
Unbarred Tight Spiral: 1829
Unbarred Loose Spiral: 2628
Edge-on without Bulge: 1500
Edge-on with Bulge: 1873



Die Verteilung ist nach der Datenaugmentation deutlich ausgeglichener. Für die restliche nicht ausgeglichene Klassenverteilung werden Klassengewichte hinzugefügt.

```
In [ ]: # Klassenverteilung basierend auf den Labels
classes = np.argmax(labels, axis=1)

# Automatische Berechnung der Gewichte
weights = compute_class_weight('balanced', classes=np.unique(classes), y=classes)
class_weight_dict = dict(enumerate(weights))

for i in range(10):
    print(f"galaxy10cls_lookup({i}): {len(images_opened_cv2[np.where(np.argmax(l
```

Disturbed: 1500, 1.29
 Merging: 1853, 1.05
 Round Smooth: 2645, 0.73
 Smooth, Cigar shaped: 2027, 0.96
 Cigar Shaped Smooth: 1500, 1.29
 Barred Spiral: 2043, 0.95
 Unbarred Tight Spiral: 1829, 1.06
 Unbarred Loose Spiral: 2628, 0.74
 Edge-on without Bulge: 1500, 1.29
 Edge-on with Bulge: 1873, 1.04

d) Normalisierung: Macht es Sinn, Daten zu normalisieren? Zu projizieren

oder kalibrieren?

Jetzt werden die Daten noch normalisiert. Normalisierung erhöht nachweislich die Konvergenzgeschwindigkeit, verbessert die Modellgenauigkeit und verhindert explodierende Gradienten. Dabei werden die Daten mit der Formel normalisiert:

$$x = \frac{x}{127.5} - 1$$

wobei (x) der Wert für jedes Pixel ist. Diese Transformation wandelt die Pixelwerte in den Bereich $([-1, 1])$ um. (Hui et al., 2022, S. 6)

```
In [8]: # Normalisierung der Bilder mit x/127.5 - 1
images_opened_cv2 /= 127.5
images_opened_cv2 -= 1
```

```
In [ ]: save_to_numpy('images_final.npy', images_opened_cv2)
save_to_numpy('labels_final.npy', labels)
```

```
In [ ]: del images_opened_cv2
gc.collect()
```

```
In [3]: images = load_from_numpy('images_final.npy')
labels = load_from_numpy('labels_final.npy')
```

Daten aus `images_final.npy` geladen.
Daten aus `labels_final.npy` geladen.

f) Zusätzliche Daten: Möchten Sie noch weitere Daten ergänzen und hinzufügen?

Es war notwendig, zusätzliche Daten durch gezielte Datenaugmentation zu generieren, um das bestehende Klassenungleichgewicht im Galaxy10 DECaLS Dataset auszugleichen. Während die Originaldaten bereits eine hohe Vielfalt und eine solide Grundlage für das Modelltraining bieten, zeigten einige Klassen eine unzureichende Repräsentation. Mithilfe von Augmentationstechniken wie Rotation, Spiegelung und Helligkeitsanpassung wurden künstliche Variationen der unterrepräsentierten Klassen erzeugt. Dadurch konnte die Anzahl der Bilder für diese Klassen auf ein Minimum von 1.500 erhöht werden, ohne dass neue astronomische Daten beschafft werden mussten.

Die augmentierten Daten gewährleisten eine bessere Balance zwischen den Klassen, erhöhen die Robustheit des Modells und verbessern dessen Generalisierungsfähigkeit. Der Ansatz spart Kosten und ermöglicht ein effektives Training mit dem bestehenden Dataset.

4) Modelling

a) Methoden: Wählen Sie die Methoden aus, die Sie benötigen, um mit Hilfe Ihrer Daten die gewählten Ziele zu erreichen und gestellten Fragen zu beantworten.

Methodenwahl und Modellbegründung: DenseNet121

Für die Klassifikation von Galaxienbildern wird das Modell DenseNet121 gewählt, da es eine moderne Architektur mit mehreren Vorteilen bietet, die speziell für komplexe Datensätze wie das Galaxy10 DECaLS Dataset geeignet sind (Hui et al., 2022, S. 4):

- Effizienz und Wiederverwendung von Features:

Die Architektur von DenseNet121 basiert auf sogenannten Dense Blocks, in denen jede Schicht die Ausgaben aller vorherigen Schichten erhält. Dies ermöglicht eine effiziente Wiederverwendung von bereits gelernten Features**, anstatt diese neu zu lernen, wodurch das Modell ressourcenschonend und effektiv ist.

- Vermeidung des Vanishing Gradient Problems:

DenseNet121 verhindert den Verlust von Gradienten, der bei sehr tiefen Modellen häufig auftritt, indem es direkten Zugriff auf die Gradienten und Eingabefeatures ermöglicht. Dies verbessert die Stabilität und Trainingsfähigkeit des Modells.

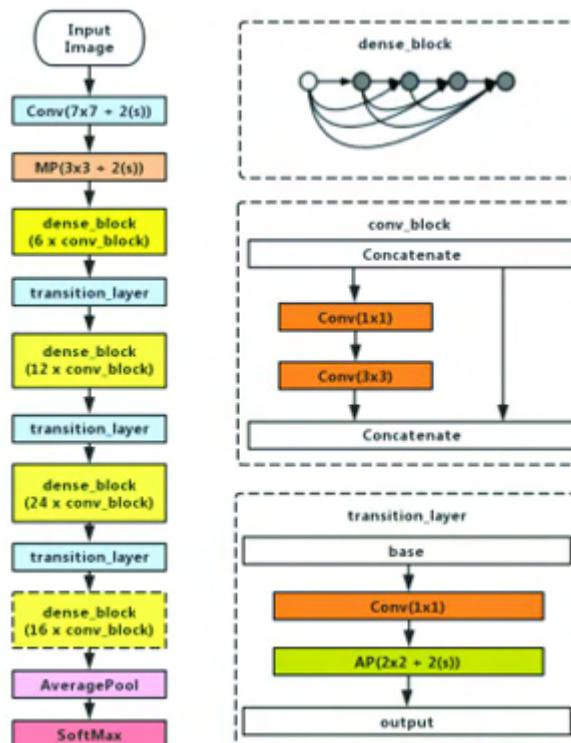
- Kompakte Architektur durch Transition Layers:

Zwischen den Dense Blocks werden Transition Layers eingefügt, die die Anzahl der Parameter und die Modellbreite kontrollieren. Dies verringert die Komplexität, ohne die Genauigkeit zu beeinträchtigen.

- Robustheit und Generalisierung:

Studien (z. B. von Huang et al.(2017, 4700 - 4708)) zeigen, dass DenseNet eine konsistente Verbesserung der Genauigkeit erzielt, auch bei steigender Parameteranzahl. Dies macht es besonders robust gegen Overfitting, selbst bei kleineren Datensätzen wie Galaxy10 DECaLS.

Aufbau von DenseNet121



(Quelle: (Hui et al., 2022, S. 43)

Die Architektur von DenseNet121 besteht aus folgenden Komponenten (Hui et al., 2022, S. 3):

- Eingabeschicht:

Eine 7x7-Convolution-Schicht mit 64 Filtern extrahiert erste Merkmale, gefolgt von einem 3x3 Max-Pooling, um die Bildgröße zu reduzieren.

- Dense Blocks:

Es gibt vier Dense Blocks mit 6, 12, 24 und 16 Convolution-Blöcken. Innerhalb jedes Blocks sind die Schichten durch dichte Verbindungen vollständig miteinander verbunden.

- Transition Layers:

Diese Schichten reduzieren die Dimensionen des Feature-Space durch 1x1-Convolutions und Average Pooling. Dadurch bleibt das Modell effizient, ohne an Genauigkeit zu verlieren.

- Klassifikationsschicht:

Nach dem letzten Dense Block folgen ein Global Average Pooling und eine Softmax-Schicht, die die Wahrscheinlichkeitswerte für jede Klasse ausgeben.

Transfer Learning mit DenseNet121 Um die Leistungsfähigkeit des Modells weiter zu optimieren, wird Transfer Learning mit vortrainierten Gewichten auf ImageNet verwendet (Hui et al., 2022, S. 4))--

- Warum Transfer Learning?

DenseNet121 ist auf ImageNet vortrainiert, einem Datensatz mit Millionen von Bildern. Die ersten Schichten des Modells haben dadurch gelernt, universelle Merkmale wie Kanten, Texturen und Formen zu extrahieren, die auch für die Galaxienklassifikation relevant sind.

- Implementierung von Transfer Learning:

Die ersten Schichten, die grundlegende Merkmale extrahieren, werden eingefroren (keine Backpropagation), da diese Features universell sind.

Nur die letzten Schichten werden auf das Galaxy10 DECaLS Dataset angepasst, wodurch das Modell effizient auf die spezifische Aufgabe abgestimmt wird.

- Vorteile:

Zeiteffizienz: Vortrainierte Gewichte beschleunigen das Training erheblich.

Generalisation: Durch die Nutzung der vortrainierten Features wird das Risiko von Overfitting auf den kleineren Datensatz reduziert.

- Ohne Transfer Learning:

Alternativ kann DenseNet121 von Grund auf trainiert werden. Dies erfordert jedoch eine größere Datenmenge und längere Trainingszeiten. Wie die Ablationsstudien zeigen, führt dieser Ansatz zu einer deutlich geringeren Genauigkeit (-9,86 %).

Ergebnisse der Ablationsstudien

Die Ablationsstudien belegen die Effektivität von DenseNet121 und der unterstützenden Methoden (Hui et al., 2022, S. 9):

- DenseNet121 als Basis:

Mit einer Genauigkeit von 88,64 % zeigt DenseNet121 in Kombination mit Techniken wie Datenaugmentierung, Transfer Learning und Class Weights die besten Ergebnisse.

- Wichtigkeit der Methoden:

Datenaugmentierung: Ohne Augmentation fällt die Genauigkeit um 5,40 %.
Augmentierung verbessert die Robustheit des Modells gegen Variationen in den Eingabebildern.

Class Weights: Durch Gewichtung der Klassen konnte das Klassenungleichgewicht ausgeglichen werden, was die Genauigkeit um 1,42 % steigerte.

Transfer Learning: Mit vortrainierten Gewichten (ImageNet) wurde die Genauigkeit um 9,86 % gegenüber dem Training von Grund auf erhöht.

b) Durchführung: Führen Sie die Auswertungen durch.

Die Bilddaten werden in Test, Trainings und Validierungsdatensätze eingeteilt.

```
In [4]: # Train/Test Split  
X_train, X_test, y_train, y_test = train_test_split(images, np.argmax(labels, axis=1), test_size=0.2)
```

```
In [7]: save_to_numpy('X_test.npy', X_test)  
save_to_numpy('y_test.npy', y_test)
```

Daten in X_test.npy gespeichert.
Daten in y_test.npy gespeichert.

```
In [ ]: X_test = load_from_numpy('X_test.npy')  
y_test = load_from_numpy('y_test.npy')
```

Nicht mehr gebrauchte Arrays werden gelöscht

```
In [5]: del images  
del labels  
gc.collect()
```

```
Out[5]: 130
```

```
In [6]: # Validation Split  
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
```

```
In [ ]: save_to_numpy('X_train.npy', X_train)
```

Durchführen des Trainings mit DenseNet mit transfer learning.

1. Parameterdefinition

Zunächst wurden die zentralen Parameter für das Modelltraining festgelegt:

- input_shape: Die Zielgröße der Eingabebilder beträgt 224x224 Pixel mit 3 Farbkanälen (RGB).
- num_classes: Das Modell soll 10 Klassen unterscheiden.

batch_size: Die Batch-Größe wurde auf 32 gesetzt, um eine effiziente Verarbeitung während des Trainings zu ermöglichen.

- epochs: Die Anzahl der Trainingsdurchläufe wurde auf 5 gesetzt.

initial_lr: Die anfängliche Lernrate wurde auf 0.001 festgelegt.

- decay_rate und decay_steps: Diese Parameter steuern die Lernratenreduktion, um die Konvergenz zu verbessern.

2. Weitere Data Preparation

- Die Labels wurden mit to_categorical in ein numerisches Format (One-Hot-Encoding) konvertiert, um sie für das Modell nutzbar zu machen.

3. Berechnung der Klassengewichte

Um die Klassenungleichheit im Datensatz auszugleichen, wurden die Klassengewichte berechnet:

- compute_class_weight: Die Funktion berechnet für jede Klasse ein Gewicht basierend auf der Häufigkeit im Datensatz.
- class_weight_dict: Dieses Wörterbuch speichert die berechneten Gewichte, die später im Modelltraining verwendet werden.

4. Laden des DenseNet121-Modells

Das vorgebildete DenseNet121-Modell wurde mit den vorgebildeten ImageNet-Gewichten geladen:

- `include_top=False`: Die letzte Klassifikationsschicht wurde entfernt, um das Modell für die spezifische Aufgabe anzupassen.
- `trainable=True`: Alle Schichten des DenseNet-Modells wurden trainierbar gemacht, um eine Feinabstimmung (Fine-Tuning) zu ermöglichen.

5. Anpassung des Modells

Der Kopf des Modells wurde speziell für die Klassifikationsaufgabe angepasst:

- Global Average Pooling: Diese Schicht aggregiert die Merkmale über die gesamte Feature-Map und reduziert die Parameteranzahl.
- Dense Layer: Eine Zwischenschicht mit 256 Neuronen und ReLU-Aktivierung wurde hinzugefügt.
- Klassifikationsschicht: Eine abschließende Dense-Schicht mit 10 Neuronen und einer Softmax-Aktivierungsfunktion wurde hinzugefügt, um Wahrscheinlichkeiten für jede Klasse auszugeben.

6. Modellkompilierung

Das Modell wurde mit den folgenden Einstellungen kompiliert:

- Optimizer: Der Adam-Optimizer mit einer initialen Lernrate von 0.001 wurde verwendet.

Loss: Die Funktion `categorical_crossentropy` wurde für Mehrklassenklassifikationen genutzt.

- Metrik: Die Genauigkeit (accuracy) wurde zur Bewertung des Modells verwendet.

7. Lernratensteuerung

Ein Learning Rate Scheduler wurde implementiert, um die Lernrate während des Trainings dynamisch anzupassen:

- Funktion `lr_schedule`: Die Lernrate wurde alle 10 Epochen um den Faktor 0.1 reduziert.

8. Modellbewertung

Nach dem Training wurde das Modell auf den Testdaten evaluiert:

Testgenauigkeit: Die Modellleistung wurde auf den zuvor zurückgehaltenen Testdaten bewertet und als Genauigkeit (`test_acc`) ausgegeben.

9. Callback

- Fortschrittausgabe während des Trainings: loss und accuracy

```
In [ ]: # 1. Parameter
input_shape = (224, 224, 3) # Zielgröße der Bilder
num_classes = 10 # Anzahl der Klassen
batch_size = 32
epochs = 5
initial_lr = 0.001
decay_rate = 0.1
decay_steps = 10

# 2. Labels zu kategorischen Daten
y_train = to_categorical(y_train, num_classes)
y_val = to_categorical(y_val, num_classes)
y_test = to_categorical(y_test, num_classes)

# Klassenverteilung basierend auf den Labels
labels = load_from_numpy('labels_final.npy')
classes = np.argmax(labels, axis=1)
del labels
gc.collect()

# 3. Automatische Berechnung der Gewichte
weights = compute_class_weight('balanced', classes=np.unique(classes), y=classes)
class_weight_dict = dict(enumerate(weights))
print("Class weights:", class_weight_dict)

# DenseNet121-Modell Laden
base_model = DenseNet121(weights='imagenet', include_top=False, input_shape=input_shape)

# 4. Backbone trainierbar machen
for layer in base_model.layers:
    layer.trainable = True

# 5. Kopf des Modells anpassen
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

# 6. Optimizer und Loss
model.compile(optimizer=Adam(learning_rate=initial_lr),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# 7. Learning Rate Scheduler
def lr_schedule(epoch):
    return initial_lr * (decay_rate ** (epoch // decay_steps))

# 8. Learning Rate Scheduler
lr_scheduler = LearningRateScheduler(lr_schedule)

# 9. Training Callback
```

```

class TrainingProgressCallback(Callback):
    def on_epoch_end(self, epoch, logs=None):
        """
        Diese Methode wird am Ende jeder Epoche aufgerufen.
        """
        print(f"Epoch {epoch + 1}/{self.params['epochs']}")
        print(f" - Loss: {logs['loss']:.4f}, Accuracy: {logs['accuracy']:.4f}")
        print(f" - Validation Loss: {logs['val_loss']:.4f}, Validation Accuracy:")

```

Training des Modells

Das Modell wurde mit den folgenden Einstellungen trainiert:

- Trainingsdaten (X_{train} , y_{train}): Die Trainingsbilder und -labels.
- Validierungsdaten (X_{val} , y_{val}): Zur Überwachung der Modellleistung während des Trainings.
- Klassengewichte: Mit `class_weight=class_weight_dict` wurde sichergestellt, dass Klassen mit weniger Daten

stärker gewichtet werden.

- Callbacks: Der Learning Rate Scheduler wurde als Callback hinzugefügt.

```

In [9]: # Training mit Klassengewichten (ohne datagen)
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    batch_size=batch_size,
    epochs=epochs,
    callbacks=[lr_scheduler, TrainingProgressCallback()],
    class_weight=class_weight_dict # Hier werden die Klassengewichte hinzugefügt
)

# Evaluation
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc * 100:.2f}%")

```

```

Epoch 1/5
388/388 0s 4s/step - accuracy: 0.4841 - loss: 1.4732Epoch 1/
5
- Loss: 1.2064, Accuracy: 0.5754
- Validation Loss: 1.6212, Validation Accuracy: 0.5190
388/388 1760s 4s/step - accuracy: 0.4844 - loss: 1.4725 - va
l_accuracy: 0.5190 - val_loss: 1.6212 - learning_rate: 0.0010
Epoch 2/5
388/388 0s 4s/step - accuracy: 0.6865 - loss: 0.8732Epoch 2/
5
- Loss: 0.8357, Accuracy: 0.7049
- Validation Loss: 1.1371, Validation Accuracy: 0.6253
388/388 1694s 4s/step - accuracy: 0.6865 - loss: 0.8731 - va
l_accuracy: 0.6253 - val_loss: 1.1371 - learning_rate: 0.0010
Epoch 3/5
388/388 0s 4s/step - accuracy: 0.7614 - loss: 0.6934Epoch 3/
5
- Loss: 0.6962, Accuracy: 0.7592
- Validation Loss: 0.9498, Validation Accuracy: 0.6959
388/388 1660s 4s/step - accuracy: 0.7614 - loss: 0.6934 - va
l_accuracy: 0.6959 - val_loss: 0.9498 - learning_rate: 0.0010
Epoch 4/5
388/388 0s 4s/step - accuracy: 0.7824 - loss: 0.6235Epoch 4/
5
- Loss: 0.6170, Accuracy: 0.7856
- Validation Loss: 1.0730, Validation Accuracy: 0.6466
388/388 1679s 4s/step - accuracy: 0.7824 - loss: 0.6235 - va
l_accuracy: 0.6466 - val_loss: 1.0730 - learning_rate: 0.0010
Epoch 5/5
388/388 0s 4s/step - accuracy: 0.8101 - loss: 0.5499Epoch 5/
5
- Loss: 0.5584, Accuracy: 0.8062
- Validation Loss: 0.7877, Validation Accuracy: 0.7262
388/388 1668s 4s/step - accuracy: 0.8100 - loss: 0.5499 - va
l_accuracy: 0.7262 - val_loss: 0.7877 - learning_rate: 0.0010
122/122 108s 854ms/step - accuracy: 0.7077 - loss: 0.8520
Test Accuracy: 72.09%

```

```
In [13]: # Speichern im SavedModel-Format
model.save('densenet121_model.keras')
```

5) Evaluation

a) Evaluierung: Ist bei den Auswertungen das Erwartete oder Richtige herausgekommen? Sind die Ergebnisse plausibel? Sind sie statistisch signifikant?

```
In [8]: # Laden des Modells
model = tf.keras.models.load_model('densenet121_model.keras')
```

```
d:\Anaconda\envs\meinenv\Lib\site-packages\keras\src\saving\saving_lib.py:719: Us
erWarning: Skipping variable loading for optimizer 'rmsprop', because it has 368
variables whereas the saved optimizer has 734 variables.
```

```
In [10]: # Confusion Matrix für die Testdaten
y_pred = model.predict(X_test)
```

```
122/122 106s 865ms/step
```

```
In [18]: import seaborn as sns

# Checken der Dimensionen
if len(y_test.shape) > 1:
    y_true = np.argmax(y_test, axis=1)
else:
    y_true = y_test

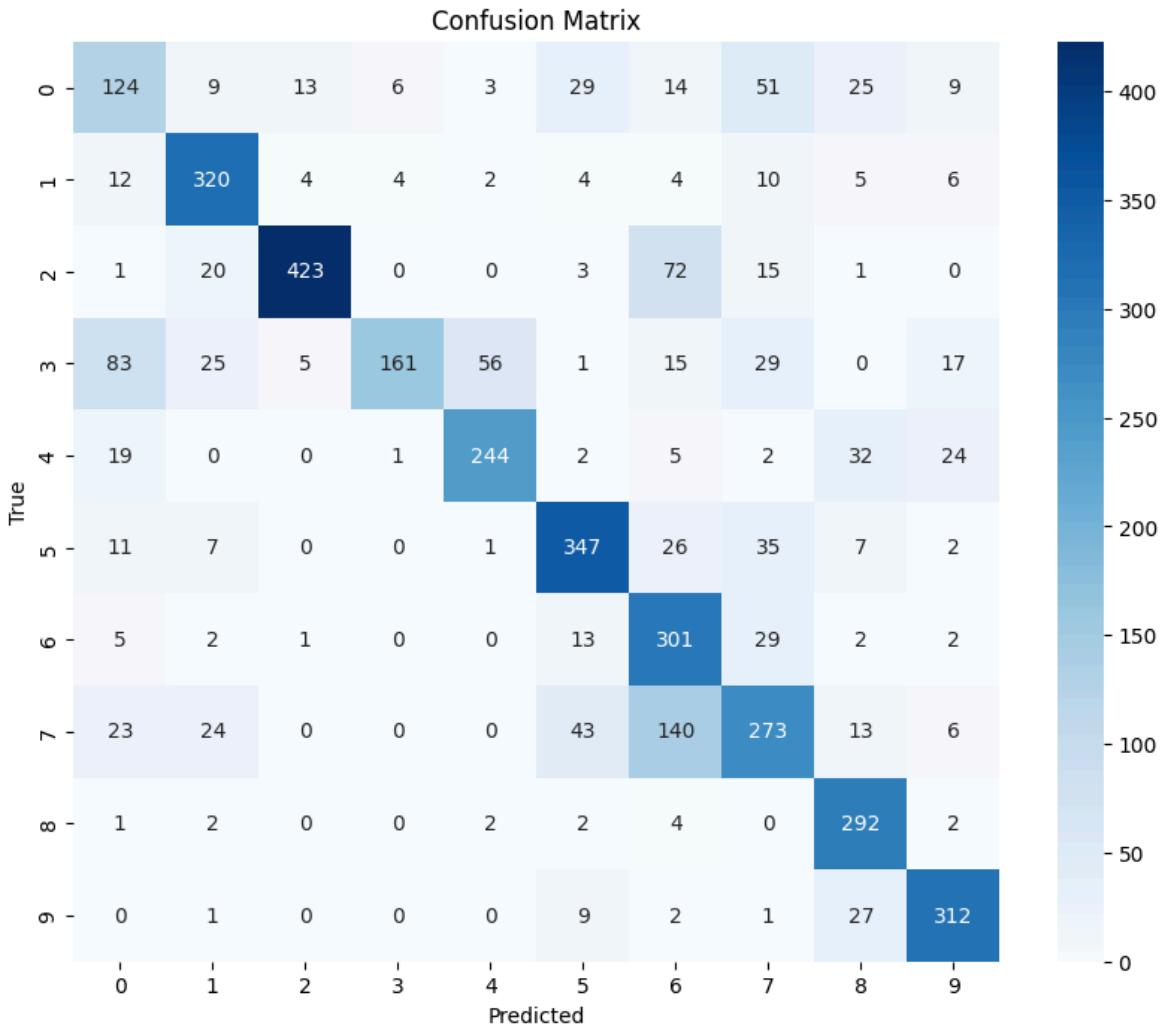
if len(y_pred.shape) > 1:
    y_pred_classes = np.argmax(y_pred, axis=1)
else:
    y_pred_classes = y_pred

confusion_matrix = tf.math.confusion_matrix(y_true, y_pred_classes)

# Ausgabe der Gesamtkorrektklassifizierungsrate
accuracy = np.trace(confusion_matrix) / np.sum(confusion_matrix)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Plotting
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

Accuracy: 72.09%



Als Klassifikationsqualität wurde die Korrektklassifikationsrate bzw. Genauigkeit herangezogen. Da jede Klasse gleich relevant ist für die Klassifikation.

```
In [17]: # Ausgaben der Klassennamen und Indizes
for i in range(10):
    print(f"{galaxy10cls_lookup(i)}: {i}")
```

```
Disturbed: 0
Merging: 1
Round Smooth: 2
Smooth, Cigar shaped: 3
Cigar Shaped Smooth: 4
Barred Spiral: 5
Unbarred Tight Spiral: 6
Unbarred Loose Spiral: 7
Edge-on without Bulge: 8
Edge-on with Bulge: 9
```

1. Überblick der Ergebnisse

- Hauptdiagonale: Die Hauptdiagonale enthält die korrekt klassifizierten Beispiele für jede Klasse.

Abweichungen von der Hauptdiagonale: Zeilen und Spalten abseits der Hauptdiagonalen zeigen - Fehlklassifikationen. Diese Fehler können Rückschlüsse auf die Ähnlichkeit zwischen Klassen zulassen.

2. Analyse der Klassen

Disturbed (Klasse 0)

- Korrekt klassifiziert: 124 Bilder.
- Fehlklassifikationen:
 - Häufig verwechselt mit Unbarred Loose Spiral (7) (51 Bilder) und Unbarred Tight Spiral (6) (25 Bilder).
- Interpretation:
 - Da die „Disturbed“-Galaxien chaotische und unregelmäßige Strukturen aufweisen, kann das Modell diese mit Spiralen verwechseln, deren Strukturen weniger klar definiert sind.

Merging (Klasse 1)

- Korrekt klassifiziert: 320 Bilder.
- Fehlklassifikationen:
 - Leichte Verwechslung mit Disturbed (0) (12 Bilder).
- Interpretation:
 - „Merging“-Galaxien haben ein dynamisches Erscheinungsbild, das gewisse Überschneidungen zu „Disturbed“ zeigt. Diese Klassen sind visuell ähnlich, da beide chaotische Merkmale aufweisen.

Round Smooth (Klasse 2)

- Korrekt klassifiziert: 423 Bilder (sehr gut).
- Fehlklassifikationen:
 - Verwechslung mit Smooth, Cigar Shaped (3) (20 Bilder) und Barred Spiral (5) (72 Bilder).
- Interpretation:
 - „Round Smooth“-Galaxien haben glatte und symmetrische Formen. Die Verwechslung mit „Smooth, Cigar Shaped“ liegt nahe, da beide glatt sind und nur in der Form variieren.

Smooth, Cigar Shaped (Klasse 3)

- Korrekt klassifiziert: 161 Bilder.
- Fehlklassifikationen:
 - Verwechslung mit Disturbed (0) (83 Bilder) und Cigar Shaped Smooth (4) (56 Bilder).
- Interpretation:
 - Hier zeigt sich eine klare Ähnlichkeit zu „Cigar Shaped Smooth“. Beide Klassen sind glatt und länglich, was es schwer macht, sie voneinander zu unterscheiden.

Cigar Shaped Smooth (Klasse 4)

- Korrekt klassifiziert: 244 Bilder.
- Fehlklassifikationen:
 - Leichte Verwechslung mit Smooth, Cigar Shaped (3) und Barred Spiral (5).
- Interpretation:
 - Die Ähnlichkeit zu „Smooth, Cigar Shaped“ führt zu Verwechslungen. Diese beiden Klassen unterscheiden sich visuell nur minimal. Dennoch wurde diese Klasse sehr gut klassifiziert. Diese Klasse hatte auch die meisten Bilder durch Augmentation dazu bekommen.

Barred Spiral (Klasse 5)

- Korrekt klassifiziert: 347 Bilder.
- Fehlklassifikationen:
 - Verwechslung mit Unbarred Loose Spiral (7) (35 Bilder) und Unbarred Tight Spiral (6) (26 Bilder).
- Interpretation:
 - Der Balken in „Barred Spiral“ kann vom Modell möglicherweise nicht eindeutig erkannt werden, wodurch eine Verwechslung mit anderen Spiralgalaxien ohne Balken entsteht.

Unbarred Tight Spiral (Klasse 6)

- Korrekt klassifiziert: 301 Bilder.
- Fehlklassifikationen:
 - Verwechslung mit Unbarred Loose Spiral (7) (29 Bilder).
- Interpretation:

- Die enge Windung der Spiralarme ist das Unterscheidungsmerkmal zu „Unbarred Loose Spiral“. Diese Merkmale können leicht falsch interpretiert werden. Dennoch sehr gut klassifiziert.

Unbarred Loose Spiral (Klasse 7)

- Korrekt klassifiziert: 273 Bilder.
- Fehlklassifikationen: Verwechslung mit Barred Spiral (5) (43 Bilder) und Unbarred Tight Spiral (6) (140 Bilder).

Interpretation: - Die lockeren Spiralarme können mit anderen Spiralformen verwechselt werden, besonders wenn der Balken nicht eindeutig erkennbar ist. Diese Klasse wurde am schlechtesten klassifiziert.

Edge-on without Bulge (Klasse 8)

- Korrekt klassifiziert: 292 Bilder.
- Fehlklassifikationen:
 - Verwechslung mit Edge-on with Bulge (9) (27 Bilder).
- Interpretation:
 - Beide Klassen werden aus der Kante betrachtet, der einzige Unterschied ist der zentrale Bulge. Dies führt zu häufigen Fehlklassifikationen.

Edge-on with Bulge (Klasse 9)

- Korrekt klassifiziert: 312 Bilder.
 - Fehlklassifikationen:
 - Leichte Verwechslung mit Edge-on without Bulge (8).
 - Interpretation:
 - Wie bei der vorherigen Klasse ist die feine Unterscheidung des Bulges eine Herausforderung. Aber wurde sehr gut gemeistert.
-

3. Zusammenfassung der Beobachtungen

- Ähnliche Klassen:
 - Klassen mit visueller Ähnlichkeit wurden häufiger miteinander verwechselt:
 - „Smooth, Cigar Shaped“ und „Cigar Shaped Smooth“.
 - „Unbarred Tight Spiral“ und „Unbarred Loose Spiral“.
 - „Edge-on without Bulge“ und „Edge-on with Bulge“.
 - Dies deutet darauf hin, dass das Modell Schwierigkeiten hat, subtile Unterschiede zwischen diesen Klassen zu erkennen.
- Klassen mit chaotischem Erscheinungsbild:
 - Klassen wie „Disturbed“ und „Merging“ wurden ebenfalls häufig falsch klassifiziert, was auf ihre unregelmäßige Struktur zurückzuführen ist.

Für den Signifikantstest wird der Chi-Quadrat-Unabhängigkeitstest verwendet. Die Bilddaten sind nominal da sie Kategorien sind ohne natürliche Reihenfolge und

unabhängig von einander. Es wird getestet ob die tatsächlichen Klassen (True Labels) und die vorhergesagten Klassen (Predictions) unabhängig voneinander sind.

Hypothesen für den Chi-Quadrat-Test:

- Nullhypothese (H_0): Die tatsächlichen Klassen und die vorhergesagten Klassen sind unabhängig voneinander (das Modell hat keine Aussagekraft).
- Alternativhypothese (H_1): Es gibt eine Abhängigkeit zwischen den tatsächlichen Klassen und den vorhergesagten Klassen (das Modell klassifiziert signifikant besser als zufällige Vorhersagen).

```
In [21]: # Chi-Quadrat-Unabhängigkeitstest der Klassen auf Basis der Confusion Matrix
from scipy.stats import chi2_contingency

chi2, p, _, _ = chi2_contingency(confusion_matrix)
print(f"Chi-Squared: {chi2:.2f}, p-value: {p:.5f}")

# Interpretation
if p < 0.05:
    print("Ergebnis: Die Nullhypothese ( $H_0$ ) wird abgelehnt.")
    print("Interpretation: Es gibt eine signifikante Abhängigkeit zwischen den t")
    print("Das Modell klassifiziert besser als Zufall.")
else:
    print("Ergebnis: Die Nullhypothese ( $H_0$ ) kann nicht abgelehnt werden.")
    print("Interpretation: Es gibt keine signifikante Abhängigkeit zwischen den")

Chi-Squared: 18071.37, p-value: 0.00000
Ergebnis: Die Nullhypothese ( $H_0$ ) wird abgelehnt.
Interpretation: Es gibt eine signifikante Abhängigkeit zwischen den tatsächlichen Klassen (True Labels) und den vorhergesagten Klassen (Predictions).
Das Modell klassifiziert besser als Zufall.
```

b) Verbesserung: Verbessern und verfeinern Sie Ihre Auswertungen.

Validität und Reliabilität der Ergebnisse

1. Konstruktvalidität (Construct Validity)

Messen die Daten genau das, was untersucht werden sollte?

- Die Daten stammen aus dem Galaxy10 DECaLS-Datensatz, der speziell zur Klassifizierung von Galaxien nach ihrem visuellen Erscheinungsbild erstellt wurde.
- Die Klassen (z. B. "Disturbed", "Barred Spiral") sind klar definiert und basieren auf wissenschaftlichen Kriterien der Astronomie.
- Das verwendete DenseNet121-Modell wurde trainiert, um die visuelle Struktur der Galaxien zu analysieren, was dem Ziel entspricht, die Bilder korrekt zu klassifizieren.
- Die Ergebnisse der Confusion Matrix sowie der Chi-Quadrat-Test zeigen, dass das Modell zwischen den Klassen signifikant unterscheiden kann, wodurch bestätigt wird, dass die Daten und Methoden das untersuchen, was sie sollen.

2. Interne Validität (Internal Validity)

Wenn Abhängigkeiten zwischen Daten beobachtet werden, gibt es dafür auch mögliche andere Erklärungen? Falls ja, wie können Sie ausschließen, dass diese alternative Erklärung gilt?

- Mögliche alternative Erklärungen für beobachtete Abhängigkeiten könnten sein:
 - Datenbias: Ungleichmäßige Klassenverteilung führt zu besserer Vorhersage für Klassen mit mehr Daten. Dies wurde durch Klassengewichtung (Class Weights) während des Trainings berücksichtigt sowie durch Datenaugmentation zur Generierung neuer Bilder für unterrepräsentierte Klassen.
 - Bildähnlichkeit: Visuell ähnliche Klassen (z. B. "Smooth, Cigar Shaped" und "Cigar Shaped Smooth") führen zu Fehlklassifikationen. Dies wurde durch die Analyse der Confusion Matrix identifiziert und erklärt.
 - Modellkomplexität: Ein vorgebildetes Modell (DenseNet121) wurde verwendet, um Overfitting zu minimieren.
 - Durch statistische Tests (z. B. Chi-Quadrat-Test) wurde ausgeschlossen, dass die Ergebnisse zufällig sind, wodurch die interne Validität gestärkt wird.
-

3. Externe Validität (External Validity)

Wie ist das Umfeld Ihrer Daten charakterisiert? Glauben Sie, dass die Ergebnisse Ihrer Untersuchung auch in einem anderen Umfeld gelten? Welchen Gültigkeitsbereich haben Ihre Ergebnisse? (Wie) können Sie dies begründen?

- Datenumfeld: Die Bilder stammen aus dem Galaxy10 DECaLS-Datensatz, der eine repräsentative Auswahl an Galaxienklassen bietet.
 - Die Ergebnisse sind spezifisch für astronomische Bilddaten, die unter kontrollierten Bedingungen aufgenommen wurden.
 - Verallgemeinerbarkeit:
 - Für ähnliche Datensätze aus astronomischen Surveys (z. B. SDSS, Hubble-Aufnahmen) könnten die Ergebnisse weitgehend übertragbar sein.
 - Bei Daten aus anderen Bildbereichen oder unterschiedlichen Auflösungen müssten zusätzliche Anpassungen (z. B. Vorverarbeitung, Training) erfolgen.
 - Gültigkeitsbereich: Das Modell und die Pipeline sind besonders geeignet für Aufgaben der astronomischen Bildklassifikation und könnten in ähnlichen Forschungsumgebungen verwendet werden.
-

4. Reliabilität (Wiederholbarkeit) der Ergebnisse

Sind die Ergebnisse objektiv und nachvollziehbar, so dass andere Forscher aufgrund derselben Daten zu denselben Schlussfolgerungen gelangen würden? Wie stellen Sie dies sicher?

- Die Ergebnisse sind reproduzierbar, da folgende Schritte dokumentiert wurden:
 - Datenvorbereitung: Schritte wie Normalisierung, Größenanpassung (Center Cropping) und Datenaugmentation sind klar definiert.

- Modelltraining: Die Parameter (z. B. DenseNet121, Learning Rate, Batch Size) sowie der Trainingsprozess wurden beschrieben.
- Code: Der gesamte Code zur Modellimplementierung und Evaluation wurde bereitgestellt, sodass andere Forscher den Prozess nachvollziehen und wiederholen können.
- Bewertung: Die Confusion Matrix und statistische Tests (z. B. Chi-Quadrat-Test) wurden zur objektiven Bewertung verwendet.
- Verfahren zur Sicherstellung der Wiederholbarkeit:
 - Verwendung einer festen Random-Seed während des Trainings und der Aufteilung der Daten.
 - Nutzung standardisierter Bibliotheken wie `TensorFlow` und `Keras` zur Modellerstellung.
 - Klar definierte Trainings- und Validierungsdaten gewährleisten eine konsistente Vergleichsbasis.

6) Deployment

a) Zielerreichung: Sind die Ziele der Datenanalysen erreicht worden? Können damit auch die Geschäftsziele erreicht werden? Wie?

1. Ziele der Datenanalysen

Die Datenanalyse hatte das Ziel, einen Proof of Concept (PoC) für die Bilderkennung von Galaxien zu entwickeln. Dabei lag der Fokus auf:

- Identifikation und Durchführung relevanter Schritte der Datenvorbereitung, wie Normalisierung, Größenanpassung und Datenaugmentation.

Evaluierung eines maschinellen Lernmodells (DenseNet121 mit Transfer Learning) zur Klassifikation der Galaxien in 10 Klassen, basierend auf visuellem Erscheinungsbild.

- Identifikation von Herausforderungen, die während der Datenaufbereitung und Modellentwicklung aufraten, ohne die Ergebnisse zu optimieren.

Diese Schritte sollten aufzeigen, wie eine grundlegende Pipeline für die Galaxienklassifikation erstellt werden kann, und mögliche Hindernisse bei der Implementierung dokumentieren.

2. Erfüllung der Ziele der Datenanalysen

Die Ziele der Datenanalysen wurden wie folgt erfüllt:

- Datenvorbereitung und Preprocessing:
 - Es wurden Methoden zur Normalisierung, Größenanpassung (Center Cropping) und Rauschunterdrückung eingesetzt, um die Daten für das Modell vorzubereiten.

- Herausforderungen wie Klassenungleichgewichte und das Entfernen irrelevanter Objekte (z. B. Sterne) wurden identifiziert und adressiert.
- Modellentwicklung und Evaluierung:
 - Ein vortrainiertes DenseNet121-Modell wurde erfolgreich für die Galaxienklassifikation verwendet.
 - Es wurde ein Training mit Klassengewichten durchgeführt, um die Klassenungleichheit zu kompensieren.
 - Die Modellgenauigkeit wurde durch regelmäßige Überwachung von Loss, Accuracy, Validation Loss und Validation Accuracy ausgewertet.
 - Eine Confusion Matrix wurde genutzt, um die Klassifikationsleistung pro Klasse zu analysieren und Fehlklassifikationen zu identifizieren.
- Keine Ergebnisoptimierung:
 - Es wurde bewusst darauf verzichtet, das Modell bis zur optimalen Genauigkeit zu verbessern, da der Fokus auf der Erstellung einer prototypischen Pipeline lag.
- Vergleich mit Literatur:
 - Die durchgeführten Schritte orientierten sich an einem wissenschaftlichen Paper von Hui et al. (2022), das bewährte Verfahren und gute Ergebnisse für den gleichen Datensatz aufzeigt.

3. Erfüllung der Geschäftsziele

Die im Rahmen des PoC erzielten Ergebnisse tragen dazu bei, die Geschäftsziele wie folgt zu unterstützen:

- Schaffung einer Grundlage für zukünftige Anwendungen:
 - Der PoC zeigt, wie eine Bilderkennungspipeline für Galaxien aufgebaut werden kann, und legt den Grundstein für weitere Entwicklungen.
- Bewertung der notwendigen Schritte und Herausforderungen:
 - Die Analyse dokumentiert, welche Schritte bei der Datenaufbereitung und Modellentwicklung erforderlich sind, und identifiziert Hindernisse wie Klassenungleichgewichte und die Wahl geeigneter Datenvorbereitungsmethoden.
- Relevanz für Geschäftsziele:
 - Die Ergebnisse zeigen, dass maschinelle Lernmodelle wie DenseNet121 geeignet sind, um Galaxien basierend auf visuellen Merkmalen zu klassifizieren.
 - Durch die klare Strukturierung und Evaluierung der Schritte können diese Erkenntnisse in zukünftige Produkte oder Projekte einfließen, wie z. B. automatisierte Klassifikationssysteme für astronomische Daten.

b) Monitoring und Wartung: Wie sollen zukünftig die Daten aktuell gehalten und die Auswertungen aktualisiert werden?

Um die kontinuierliche Aktualität der Daten und die Aktualisierung der Auswertungen sicherzustellen, wird ein MLDevOps-Ansatz verfolgt, der die einzelnen Phasen des Machine Learning und der Softwareentwicklung in einer integrierten Umgebung vereint. Der MLDevOps-Prozess sorgt dafür, dass die Modelle regelmäßig überwacht und bei Bedarf aktualisiert werden. Die wichtigsten Maßnahmen sind:

- Datenaktualisierung: Die Datenpipeline wird so konfiguriert, dass neue Galaxienbilder automatisch integriert werden. Dabei werden die Schritte zur Datenbereinigung und -vorbereitung wiederholt, um sicherzustellen, dass die Daten immer in einem für das Modell geeigneten Zustand sind.
- Modellüberwachung: Nach der Bereitstellung in der Produktionsumgebung wird die Leistung der Modelle kontinuierlich überwacht. Es wird auf Metriken wie Modellgenauigkeit und Modelldrift geachtet, um sicherzustellen, dass die Modelle auch mit neuen Daten weiterhin zuverlässige Ergebnisse liefern.
- Retraining und Anpassung: Bei festgestellter Modelldrift oder veränderten Datenverteilungen wird das Modell regelmäßig neu trainiert. Dafür werden die aktuellen Daten verwendet, um sicherzustellen, dass das Modell sich an Veränderungen im Datenverhalten anpassen kann.
- Regelmäßige Auswertungsaktualisierung: Die Ergebnisse der Auswertungen werden regelmäßig aktualisiert, basierend auf neuen Daten und verbesserten Modellversionen. Durch die Nutzung von MLDevOps werden die Modelle und Auswertungen automatisch aktualisiert, was sicherstellt, dass die bereitgestellten Informationen immer auf dem neuesten Stand sind.

Dieser kontinuierliche Monitoring- und Wartungsprozess stellt sicher, dass die Modelle langfristig stabil bleiben und die Auswertungen immer aktuelle und verlässliche Informationen zur Verfügung stellen.

c) Bericht: Schreiben Sie Ihren Bericht. Fassen Sie die wichtigsten Ergebnisse am Ende des Berichts zusammen und erstellen Sie einen halbseitigen Management Summary.

Einleitung

In diesem Projekt wurde ein Proof of Concept (PoC) zur Bilderkennung von Galaxien durchgeführt. Ziel war es, relevante Schritte der Datenvorbereitung sowie der Modellentwicklung zu evaluieren und die damit verbundenen Herausforderungen zu dokumentieren. Der Schwerpunkt lag nicht auf der Optimierung der Ergebnisse, sondern auf der Erstellung einer nachvollziehbaren Pipeline zur Klassifikation von Galaxienbildern in zehn vorgegebene Klassen.

Methodik

- Datenvorbereitung:
 - Die Bilder wurden auf die Größe von 224x224 Pixel zugeschnitten und normalisiert.
 - Eine morphologische Öffnung wurde angewendet, um Rauschen und irrelevante Strukturen wie Sterne zu entfernen, während die Galaxien selbst erhalten blieben.

- Datenaugmentation wurde genutzt, um die Vielfalt des Trainingsdatensatzes zu erhöhen und Klassenungleichgewichte auszugleichen.
 - Modellentwicklung:
 - Ein vortrainiertes DenseNet121-Modell wurde verwendet, um Transfer Learning zu ermöglichen. Dies beschleunigte das Training und verbesserte die Klassifikationsgenauigkeit.
 - Der Kopf des Modells wurde angepasst, um die spezifische Klassifikationsaufgabe für zehn Galaxieklassen zu lösen.
 - Training und Evaluierung:
 - Das Modell wurde unter Verwendung von Klassengewichten trainiert, um die Ungleichverteilung der Klassen auszugleichen.
 - Ein Learning Rate Scheduler wurde eingesetzt, um die Lernrate während des Trainings schrittweise anzupassen.
 - Die Ergebnisse wurden durch eine Confusion Matrix analysiert, um die Leistung für jede Klasse individuell zu bewerten.
-

Ergebnisse

- Die Confusion Matrix zeigt, dass visuell ähnliche Klassen häufiger miteinander verwechselt wurden. Dies betrifft beispielsweise die Klassen "Smooth, Cigar Shaped" und "Cigar Shaped Smooth" oder "Unbarred Tight Spiral" und "Unbarred Loose Spiral".
 - Der Chi-Quadrat-Unabhängigkeitstest bestätigte eine signifikante Abhängigkeit zwischen den tatsächlichen und vorhergesagten Klassen ($p < 0.05$), was bedeutet, dass das Modell besser als Zufallsvorhersagen arbeitet.
 - Die Gesamtgenauigkeit des Modells blieb trotz der verwendeten Methoden und Datenvorbereitung im mittleren Bereich, was auf die Komplexität der Klassenunterscheidung und die Herausforderung ungleicher Klassenverteilung zurückzuführen ist.
-

Herausforderungen und Verbesserungspotenziale

- Die visuelle Ähnlichkeit zwischen bestimmten Klassen erschwert die Unterscheidung für das Modell. Weitere Datenaugmentation oder spezialisierte Modelle könnten hier Verbesserungen bringen.
- Ein effizienteres Datenmanagement könnte helfen, die Speichernutzung zu optimieren und größere Trainingsmengen zu verarbeiten.

Aufgrund der Größe der Bilddaten ist es erforderlich, geeignete Hardware zu verwenden:

- Genügend RAM (Arbeitsspeicher), um die großen Datenmengen zu laden und zu verarbeiten.
- Leistungsstarke CPUs und GPUs mit hoher Taktrate und vielen Kernen, um Trainingsprozesse zu beschleunigen und Optimierungen effizient durchführen zu können.

- Ohne passende Hardware kann der Trainingsprozess extrem zeitaufwendig werden, was den gesamten Workflow ineffizient und frustrierend macht.
-

Management Summary

Im Rahmen dieses Projekts wurde ein Proof of Concept zur Klassifikation von Galaxienbildern entwickelt. Dabei wurden relevante Schritte der Datenvorbereitung, Modellentwicklung und Evaluierung untersucht. Es konnte gezeigt werden, dass ein vortrainiertes DenseNet121-Modell geeignet ist, um Galaxien in zehn Klassen zu unterscheiden. Die statistische Analyse der Ergebnisse bestätigte die Signifikanz der Modellleistung.

Wesentliche Erkenntnisse:

- Datenvorbereitung, einschließlich Normalisierung, Augmentation und Rauschunterdrückung, ist entscheidend für die Klassifikationspipeline.
- Transfer Learning mit DenseNet121 ermöglichte es, auch mit begrenzten Daten eine akzeptable Leistung zu erzielen.
- Herausforderungen traten insbesondere bei visuell ähnlichen Klassen und unterrepräsentierten Kategorien auf.

Obwohl keine Optimierung der Modellleistung durchgeführt wurde, bietet der PoC eine solide Grundlage für zukünftige Arbeiten. Weiterführende Maßnahmen wie erweiterte Augmentation oder spezialisierte Modelle könnten die Klassifikationsgenauigkeit verbessern.

d) Kritische Reflexion / Lessons Learned: Was hätte man besser machen können? Was werden Sie das nächste Mal besser machen? Was haben Sie dazu gelernt?

Im Verlauf des Projekts sind mehrere Herausforderungen und Optimierungspotenziale identifiziert worden. Die wichtigsten Lessons Learned und Verbesserungspunkte sind wie folgt:

1. Effizientes Datenmanagement und Speicherverbrauch

- Bei großen Bilddatensätzen ist es essenziell, den Arbeitsspeicher (RAM) effizient zu nutzen.
- Die Daten sollten frühzeitig in speichereffizienten Formaten gespeichert und bei Bedarf schrittweise geladen werden, anstatt alles gleichzeitig in den Speicher zu laden.
- Lösung: Nutzung von `.npy`-Dateien oder speicheroptimierten Dataloadern, um Teile der Daten dynamisch zu verarbeiten. Außerdem hilft der gezielte Einsatz von Kopien (`copy()`) und der Garbage Collector, um unnötige Speichernutzung zu vermeiden.

2. Mehr Zeit für die Datenvorbereitung (Data Preparation)

- Die morphologische Öffnung zur Entfernung von Sternen und Rauschen war ein guter Ansatz. Dennoch hätte dieser Schritt genauer abgestimmt werden können, um eine optimale Balance zwischen Rauschreduktion und Erhalt der Galaxien zu erreichen.
 - Es muss sichergestellt werden, dass durch die Vorverarbeitung keine relevanten Bildinformationen zerstört werden.
 - Lösung: Umfangreichere Tests mit verschiedenen Parametern für die Öffnung (z. B. Kernel-Größe, Filtermethoden) und visuelle Inspektion der Ergebnisse auf Qualität und Brauchbarkeit.
-

3. Visuell ähnliche Klassen besser berücksichtigen

- Die Confusion Matrix zeigte, dass visuell ähnliche Klassen (z. B. "Smooth, Cigar Shaped" und "Cigar Shaped Smooth") häufiger falsch klassifiziert wurden.
 - Lösung: Eine feinere Unterscheidung dieser Klassen könnte durch spezialisierte Modelle oder zusätzliche Features erzielt werden, die charakteristische Unterschiede besser hervorheben.
-

4. Optimierung der Modellarchitektur und des Trainingsprozesses*

- Die Nutzung von Transfer Learning mit DenseNet121 war hilfreich, aber die Trainingszeit und Genauigkeit könnten weiter optimiert werden.
 - Lösung:
 - Testen zusätzlicher Modelle (z. B. ResNet, EfficientNet) und Architekturen, die möglicherweise besser mit den spezifischen Galaxiedaten umgehen.
 - Nutzung von Learning Rate Warmup oder adaptiven Lernraten wie `ReduceLROnPlateau`.
 - Weitere Feineinstellung der Batchgröße und der Epochenzahl für eine optimale Balance zwischen Rechenzeit und Modellleistung.
-

5. Hardwareanforderungen besser planen*

- Für die Verarbeitung großer Bilddaten und das Training tiefer Modelle ist leistungsstarke Hardware notwendig:
 - Ausreichend RAM, um die Bilddaten effizient zu speichern und zu laden.
 - Starke GPUs zur Beschleunigung des Modelltrainings.
 - CPUs mit hoher Taktrate für die Vorverarbeitungsschritte.
 - Lösung: Frühzeitige Planung der Hardware-Ressourcen, um Verzögerungen und ineffiziente Arbeitsabläufe zu vermeiden.
-

6. Reflexion der Modellperformance

- Die Modellgenauigkeit war insgesamt zufriedenstellend für einen PoC, aber durch die Klassenungleichheit und die Komplexität der Daten begrenzt.
- Lösung: Weitere Datenaugmentation für unterrepräsentierte Klassen und gezieltes Feintuning des Modells, um die Genauigkeit weiter zu verbessern.

e) Ausblick: Was wären die nächsten Schritte?

Im Rahmen dieses Projekts wurde ein Proof of Concept zur Klassifikation von Galaxienbildern erfolgreich umgesetzt. Für zukünftige Arbeiten bieten sich folgende nächste Schritte an, um die Ergebnisse weiter zu verbessern und das Modell zu optimieren:

1. Verbesserung der Datenvorbereitung

- Präzisere Rausch- und Sternentfernung: Durch eine feinere Abstimmung der Parameter für die morphologische Öffnung oder durch den Einsatz moderner Filtertechniken wie Median Filtering oder Fourier-Transformationen könnte die Bildqualität weiter verbessert werden.
 - Datenqualität überprüfen: Eine manuelle oder automatisierte Validierung der aufbereiteten Bilder stellt sicher, dass keine relevanten Informationen verloren gehen.
-

2. Optimierung des Modells

- Alternative Modelle testen: Neben DenseNet121 könnten andere vortrainierte Modelle wie EfficientNet, ResNet oder spezialisierte Vision Transformer (ViT) getestet werden, um die Modellleistung zu vergleichen.
 - Hyperparameter-Tuning: Eine systematische Optimierung der Lernrate, Batchgröße, Epochenzahl und des Optimizers könnte die Performance des Modells weiter steigern.
 - Ensemble-Modelle: Kombination mehrerer Modelle zur Verbesserung der Vorhersagegenauigkeit und zur Reduktion von Fehlklassifikationen, insbesondere bei visuell ähnlichen Klassen.
-

3. Umgang mit Klassenungleichgewicht

- Gezielte Datenaugmentation: Erweiterung der Daten für unterrepräsentierte Klassen durch spezifische Augmentationen, die charakteristische Merkmale der jeweiligen Klasse beibehalten.
 - Fokus auf schwer zu unterscheidende Klassen: Einführung von Class-Specific Loss Functions oder Focal Loss, um das Modell auf schwierige und häufig falsch klassifizierte Klassen zu sensibilisieren.
-

4. Leistungsanalyse und Erklärbarkeit des Modells

- Interpretation der Ergebnisse: Nutzung von Grad-CAM oder SHAP-Werten, um zu verstehen, welche Bildbereiche für die Klassifizierung relevant sind. Dies ermöglicht eine gezielte Verbesserung der Modellarchitektur.
 - Detaillierte Fehleranalys*: Analyse der Confusion Matrix für alle Klassen, um die Gründe für Fehlklassifikationen besser zu verstehen und die Pipeline gezielt zu verbessern.
-

5. Skalierung und Hardwareoptimierung*

- Für größere Datensätze oder höhere Bildauflösungen sollte die Nutzung von Cloud Computing (z. B. AWS, Google Cloud) oder spezialisierter Hardware (z. B. GPUs mit höherem VRAM) in Betracht gezogen werden.
 - Implementierung von Dataloadern mit effizientem Batch-Processing zur Optimierung des Speicherverbrauchs während des Trainings.
-

6. Validierung und Generalisierung

- Cross-Dataset-Validierung: Testen des Modells auf anderen astronomischen Datensätzen (z. B. SDSS, Hubble-Daten), um die Generalisierbarkeit der Ergebnisse sicherzustellen.
 - Evaluierung im realen Umfeld: Einsetzen des Modells zur automatischen Klassifikation von Galaxienbildern in astronomischen Forschungsprojekten.
-

7. Erweiterung des Anwendungsbereichs

- Die entwickelte Pipeline könnte für ähnliche Aufgaben erweitert werden, wie die Erkennung und Klassifikation anderer astronomischer Objekte (z. B. Sterne, Quasare, Nebel).
- Integration in bestehende astronomische Tools oder Datenbanken zur Unterstützung von Forschern bei der automatisierten Analyse großer Bilddatensätze.

Literaturverzeichnis

Sandage, A., (1975): Classification and Stellar Content of Galaxies Obtained from Direct Photography. Online verfügbar unter <https://ned.ipac.caltech.edu/level5/Sandage/paper.pdf>.

Dey, Arjun; Schlegel, David J.; Lang, Dustin; Blum, Robert; Burleigh, Kaylan; Fan, Xiaohui et al. (2019): Overview of the DESI Legacy Imaging Surveys. In: AJ 157 (5), S. 168. DOI: 10.3847/1538-3881/ab089d.

Haralick, R. M.; Sternberg, S. R.; Zhuang, X. (1987): Image analysis using mathematical morphology. In: IEEE transactions on pattern analysis and machine intelligence 9 (4), S. 532–550. DOI: 10.1109/tpami.1987.4767941.

Huang, G.; Liu, Z.; van der Maaten, L.; Weinberger, K. Q. (2016): Densely Connected Convolutional Networks. Online verfügbar unter <http://arxiv.org/pdf/1608.06993>.

Hui, W.; Robert Jia, Z.; Li, H.; Wang, Z. (2022): Galaxy Morphology Classification with DenseNet. In: J. Phys.: Conf. Ser. 2402 (1), S. 12009. DOI: 10.1088/1742-6596/2402/1/012009.

Leung, H.; Bovy, J. (2024): Galaxy10 DECaLS Dataset — astroNN 1.2.dev0 documentation. Online verfügbar unter <https://astronn.readthedocs.io/en/latest/galaxy10.html>, zuletzt aktualisiert am 18.11.2024, zuletzt geprüft am 19.11.2024.

Lintott, C.; Schawinski, K.; Bamford, S.; Slosar, A.; Land, K.; Thomas, D. et al. (2011): Galaxy Zoo 1: data release of morphological classifications for nearly 900 000 galaxies★. In: Monthly Notices of the Royal Astronomical Society 410 (1), S. 166–178. DOI: 10.1111/j.1365-2966.2010.17432.x.

Lintott, C.; Schawinski, K.; Bamford, S.; Slosar, A.; Land, K.; Thomas, D. et al. (2008): Galaxy Zoo: morphologies derived from visual inspection of galaxies from the Sloan Digital Sky Survey ★. In: Monthly Notices of the Royal Astronomical Society 389 (3), S. 1179–1189. DOI: 10.1111/j.1365-2966.2008.13689.x.

Liu, F. T.; Ting, K. M.; Zhou, Z. (2009): Isolation Forest. In: Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on, S. 413–422. DOI: 10.1109/ICDM.2008.17.

Oehmcke, S.; Gieseke, F. (2019): Input Selection for Bandwidth-Limited Neural Network Inference. Online verfügbar unter <http://arxiv.org/pdf/1906.04673>.

Walmsley, M.; Lintott, C.; Géron, T.; Kruk, S.; Krawczyk, C.; Willett, K. W. et al. (2021): Galaxy Zoo DECaLS: Detailed visual morphology measurements from volunteers and deep learning for 314 000 galaxies. In: Monthly Notices of the Royal Astronomical Society 509 (3), S. 3966–3988. DOI: 10.1093/mnras/stab2093.

Xu, Z.; Dan, C.; Khim, J.; Ravikumar, P. (2020): Class-Weighted Classification: Trade-offs and Robust Approaches. Online verfügbar unter <http://arxiv.org/pdf/2005.12914>.