

Goethe-Universität Frankfurt am Main

Fachbereich 12: Institut für Informatik

**Generative Adversarial Networks und
Ensemble Learning
zur Generierung antiker Münzbilder und
Klassifizierung von Münztypen**

Abschlussarbeit zur Erlangung des akademischen Grades

Master of Science

eingereicht bei:

Dr. Karsten Tolle und Sebastian Gampe

eingereicht von:

Emmanuela Georgoula und Robin Krause

Abgabedatum: 17. Juni 2024

Inhaltsverzeichnis

Abbildungsverzeichnis	III
Tabellenverzeichnis	VI
1 Einleitung	1
2 Grundlagen	4
2.1 Machine Learning	4
2.1.1 Arten von Anwendungsbereichen	4
2.1.2 Typen von Machine Learning Algorithmen	5
2.1.3 Convolutional Neural Networks (CNNs)	6
2.1.4 Evaluationsmetriken	8
2.1.5 Gradient-weighted Class Activation Mapping (Grad-CAM)	10
2.2 Generative Adversarial Networks (GANs)	12
2.2.1 Allgemeine Funktionsweise und Komponenten	12
2.2.2 Evaluationsmethoden für die Bildqualität	15
2.2.3 Beispielarchitektur: Few-Shot GAN (FSGAN)	16
2.3 Ensemble Learning	17
2.3.1 Voraussetzungen und Vorteile von Ensemble Learning Methoden	19
2.3.2 Voting	21
2.3.2.1 Hard Voting	22
2.3.2.2 Soft Voting	23
2.3.2.3 Beispiel: Hard vs. Soft Voting	24
2.3.3 Stacking	24
3 Daten	26
4 Generierung von antiken Münzbildern mittels GANs	28
4.1 Frameworks und Bibliotheken	28
4.2 Vorbereitung der Trainingsdatensätze	29
4.3 Methodik des FSGAN-Trainings	29
4.4 Far-Domain Adaptation	31
4.5 Near-Domain Adaptation	32
4.5.1 Training eines StyleGAN2	33
4.5.2 Training und Auswertung der FSGANs	34
4.5.3 Weitere Erkenntnisse	37
4.5.3.1 Goldmünzen	37
4.5.3.2 Training mit Gipsabdrücken und Metallmünzen	41
4.5.3.3 Generierte Schriften auf den Münzmotiven	43
4.6 Wahl und Training der FSGANs für das Ensemble Learning	45
4.7 Fazit	46

5 Klassifizierung von Münztypen mittels Ensemble Learning	51
5.1 Training der Base Learner	51
5.1.1 Erstellung der Datensätze	51
5.1.2 Implementation des Training der CNN	52
5.1.3 Durchgeführte Trainings	53
5.1.4 Evaluation der Base Learner	55
5.1.4.1 Evaluation der CNNs trainiert mit TrainWithGenerated	56
5.1.4.2 Evaluation der CNNs trainiert mit TrainSubGenerated	62
5.1.5 Training von Base Learner für die Vorder- und Rückseite	68
5.1.6 Fazit	69
5.2 Bilden eines Ensembles	70
5.2.1 Implementierung des Ensemble Learnings	70
5.2.1.1 Bilden einer Ensemble-Klasse	70
5.2.1.2 Kombination der Base Learner Ausgaben mittels Voting	74
5.2.1.3 Kombination der Base Learner Ausgaben mittels Stacking	78
5.2.2 Evaluation des Ensemble Learnings	80
5.2.2.1 Evaluation der Ensembles und Kombinationsmöglichkeiten mittels der Fehlerrate	82
5.2.2.2 Evaluation ausgewählter Ensembles und Kombinations- möglichkeiten für die 13 ausgewählten Typen mittels der Sensitivität	90
5.2.2.3 Fazit	98
6 Vergleich von Bildgenerierungsmethoden: GAN vs. Stable Diffusion	100
6.1 Vorbereitung der Trainingsdaten	100
6.2 Training und Evaluation neuer Base Learner	101
6.3 Evaluation neuer Ensembles	104
7 Zusammenfassung und Ausblick	109
Anhang	112
Literatur	119
Erklärungen zur Abschlussarbeit	124

Abbildungsverzeichnis

1	Grundlegender Aufbau eines künstlichen neuronalen Netzes.	6
2	Schematische Darstellung der Fehlerrückführung.	7
3	Grundlegender Aufbau eines Convolutional Neural Networks [14].	8
4	Beispiel der Grad-CAM Visualisierungsmethode.	11
5	Allgemeine Funktionsweise eines GANs in Anlehnung an [13] und [20].	13
6	Basisarchitektur eines Ensemble Learning Modells in Anlehnung an [27].	18
7	Drei wesentliche Gründe, warum ein Ensemble besser funktionieren kann als ein einzelner Klassifikator [30].	21
8	Grundgedanke von Stacking in Anlehnung an [28].	25
9	Beispiel Münzbild mit beiden Seiten.	30
10	Beispielbilder diverser trainierter FSGANs mit dem FFHQ StyleGAN als Basis.	32
11	Trainingsverlauf des StyleGAN2.	34
12	Beispiel Near-Domain Adaptation.	35
13	Beispiel: Fehlende Feinheiten bei generierten Bildern.	36
14	Ergebnisse des Typ 20746 Vorderseite mit unterschiedlich großen Trai- ningsdatensätzen.	37
15	Beispiel: Generierte Goldmünzen.	38
16	Beispiel: Generierte Goldmünzen in schwarz-weiß.	39
17	Generierte Goldmünzen mit schwarzem oder grauem Hintergrund.	40
18	Beispiele für die Entfernung des Glanzes durch ein Python-Skript.	41
19	Beispiele von generierten Münzen, die mit dem Hintergrund verschmel- zen.	41
20	Beispiel generierte Münzen des Typs 6057. Hintergrund der Gipsab- drücke im Trainingsdatensatz weiß eingefärbt.	42
21	Generierte Bilder des Typs 6057. Trainingsdatensätze bestehen rein aus Metallmünzen bzw. Gipsabdrücken.	42
22	Beispiele generierter Münzen mit Schriften.	43

23	Ergebnisse verschiedener Trainings der Rückseite der Münze 294.	44
24	Beispiele der 13 Typen.	48
24	Beispiele der 13 Typen.	49
24	Beispiele der 13 Typen.	50
25	Vergleich Münzen des Typs 285 und 294.	57
26	Originalmünze cn_coin_8184_p_rez Typ 285.	58
27	Heatmaps der Münze cn_coin_8219_p_rez des Typs. Jedes ResNet hat die Münze dem Typen 285 zugeordnet.	59
28	Vergleich der Münzen der Typen 940, 945 und 946.	59
29	Vergleich der Münzen der Typen 21027, 12884, 20806 und 20912.	60
30	Heatmaps des ResNet50TrainWithGenerated. Richtiger Typ der Münzen 21027. Falsche Zuteilung zum Münztypen 12884.	60
31	Heatmaps des ResNet50TrainWithGenerated. Richtiger Typ der Münzen 21027. Falsche Zuteilung zum Münztypen 12884.	61
32	Beispilmünzen vom Typ 7697.	63
33	Beispilmünzen von den Typen 7686, 7907, 7910, 7696 und 7803.	63
34	Beispiele von generierten Münzen vom Typ 7697.	64
35	Heatmaps falsch zugeordneter Münzen des 769. ResNet trainiert auf dem <i>TrainSubGenerated</i> Datensatz.	64
36	Beispiele von Originalmünzen vom Typ 20230 welche falsch zugeteilt wurden.	65
37	Bilder des Typs 20230 aus dem Testdatensatzes.	65
38	Generierte Bilder des Typs 20230 mit schlechter Qualität.	65
39	Heatmaps für die Münze cn_coin_30918_p_rez des Typs 20230. Richtig zugeordnet zu Typ 20230. Trainiert auf den Datensatz <i>TrainSubGenerated</i> .	66
40	Heatmaps falsch zugeordneter Münzen des Typs 12783.	67
41	Beispilmünze (cn_coin_50460) vom Typ 21027.	76
42	Beispilmünzen der falsch zugeordneten Typen.	77
43	Konfusionsmatrizen der betrachteten Modelle.	94

44 Beispilmünzen der Typen 12752, 7025, 11128 und 12124 mit guter Qualität.	96
45 Beispilmünzen der Typen 12752, 7025, 11128 und 12124 mit schlechter Qualität.	96
46 Beispiele der generierten Münzbilder des Typen 12752.	97
47 Generierte Bilder des Typs 294 mit Stable Diffusion.	102
48 Generierte Bilder der Typs 5135 und 20809 mit Stable Diffusion.	103
49 Generierte Bilder des Typs 7697 mit Stable Diffusion.	103
50 Generierte Bilder des Typs 940 und 946 mit Stable Diffusion.	104
51 Generierte Münzen des Traininerten Stygan2 nach 2800kimg.	113
52 Beispiel Münzen des Trainingdatensatzes der Rückseite vom Typ 294.	113
53 Beispiel Münzen des Trainingdatensatzes der Vorderseite vom Typ 1286.	113
54 Konfusionsmatrizen der Base Learner ausgewertet auf dem Testdatensatz der 13 Typen.	114
54 Konfusionsmatrizen der Base Learner ausgewertet auf dem Testdatensatz der 13 Typen.	115
54 Konfusionsmatrizen der Base Learner ausgewertet auf dem Testdatensatz der 13 Typen.	116
54 Konfusionsmatrizen der Base Learner ausgewertet auf dem Testdatensatz der 13 Typen.	117
54 Konfusionsmatrizen der Base Learner ausgewertet auf dem Testdatensatz der 13 Typen.	118

Tabellenverzeichnis

1	Konfusionsmatrix für ein Zwei-Klassen-Problem.	9
2	Beispielhafte Klassenwahrscheinlichkeiten h_i^j der Base Learner und Voting Ergebnis.	24
3	Übersicht des Trainings der 13 Typen.	46
4	Übersicht der Anzahl an originalen und generierten Münzbildern im Trainingsdatensatz <i>TrainWithGenerated</i> .	52
5	Übersicht der Anzahl an originalen und generierten Münzbildern im Trainingsdatensatz <i>TrainSubGenerated</i> .	52
6	Training der Base Learner auf dem Trainingsdatensatz ohne generierte Daten.	54
7	Training des ResNet50 auf dem Trainingsdatensatz ohne generierter Daten.	54
8	Evaluation der Base Learner auf den gesamten Testdatensatz.	55
9	Evaluation der Base Learner auf dem Datensatz mit weniger als 100 Bildern pro Typ.	56
10	Evaluation der Base Learner auf den Testdatensatz der 13 Typen, für die generierte Daten in den Trainingsdatensätze hinzugefügt wurden.	56
11	Sensitivität der 13 Typen auf dem Base und <i>TrainWithGenerated</i> Datensatz.	57
12	Sensitivität der einzelnen ResNet trainiert auf dem <i>TrainSubGenerated</i> Datensatz.	62
13	Sensitivität der einzelnen ResNet trainiert auf dem <i>TrainSubGenerated</i> Datensatz.	63
14	Fehlerrate der CNNs trainiert auf der Vorderseite der Münzen.	68
15	Fehlerrate der CNNs trainiert auf der Rückseite der Münzen.	68
16	Sensitivität der CNNs der Vorderseite.	69
17	Sensitivität der CNNs der Rückseite.	69
18	Benötigte Python-Pakete für die Umsetzung des Ensemble Learnings.	71

19	Verwendete JSON-Dateien mit den eingesetzten Base Learnern für das Ensemble	72
20	Beispielhafte Klassenwahrscheinlichkeiten für die Top-1-Vorhersagen der Base Learner.	77
21	Ensemble Ausgabe für die verschiedenen Voting Funktionen.	77
22	Anzahl richtig klassifizierter Bilder auf dem <i>Test13</i> Datensatz für eine multinomiale logistische Regression mit unterschiedlichen Parametern. .	79
23	Fehlerrate der <i>Base</i> Ensembles.	83
24	Fehlerrate der <i>Generated</i> Ensembles.	83
25	Fehlerrate der <i>All</i> Ensembles.	84
26	Beste und schlechteste Fehlerrate jedes Ensembles sortiert nach der Anzahl der Base Learner für beide Testdatensätze.	84
27	Erweiterung der JSON-Dateien für das <i>Generated</i> Ensemble.	87
28	Fehlerrate des <i>GeneratedAll</i> und <i>GeneratedSub</i> Ensembles.	87
29	Fehlerrate des <i>All</i> Ensembles mit allen Base Learnern für das Top-3 bzw. Top-5 Voting.	88
30	Sensitivität des <i>Res50_Both_Base_NoAug_FreezeConv4</i> auf dem <i>Test13</i> Datensatz.	91
31	Sensitivität des <i>weighted_soft_voting</i> mit dem <i>Base</i> Ensemble auf dem <i>Test13</i> Datensatz.	92
32	Sensitivität des <i>hard_voting</i> mit dem <i>All</i> Ensemble auf dem <i>Test13</i> Datensatz.	92
33	Sensitivität des <i>stacking_with_LR</i> mit dem <i>All</i> Ensemble auf dem <i>Test13</i> Datensatz.	93
34	Verkleinerte Konfusionsmatrizen der betrachteten Modelle für die Typen 294, 7697, 12752 und 21027.	95
35	Evaluation auf dem Testdatensatz. Werte der Base CNNs als Vergleichswert angegeben.	101
36	Fehlerrate der CNNs auf dem <i>Test13</i> Datensatz.	102
37	Sensitivität der 13 Typen auf dem <i>Base</i> und <i>TrainWithGen</i> Datensatz.	102

38	Sensitivität der einzelnen ResNet. Ausgewertet auf dem <i>Test13</i> Daten- satz erweitert mit den entfernten Münzbildern.	103
39	Fehlerrate des <i>GeneratedBoth</i> und des <i>GeneratedSDBoth</i> Ensembles. . .	106
40	Sensitivität des <i>stacking with LR</i> mit dem <i>GeneratedBoth</i> Ensemble. .	106
41	Sensitivität des <i>stacking with LR</i> mit dem <i>GeneratedSDBoth</i> Ensemble. .	107
42	Fehlerrate des <i>GeneratedSDAllBoth</i> und <i>GeneratedSDSubBoth</i> Ensembles.	107
43	Fehlerrate des <i>All</i> und des <i>AllNew</i> Ensembles.	108
44	Fehlerrate des <i>AllNoSub</i> Ensembles.	108
45	Aufteilung der Kapitel.	112

1 Einleitung

Künstliche Intelligenz ist heutzutage ein großes Thema, das immer weiter wächst und nicht mehr wegzudenken ist. Dies wird durch den Erfolg von Technologien wie Siri oder ChatGPT deutlich, die täglich von Menschen verwendet werden. Machine Learning ist hierbei ein Teil dessen, was ein System benötigt, um eine künstliche Intelligenz zu werden [1]. Es ist ein sehr breites Themengebiet und wird für viele unterschiedliche Probleme in vielen verschiedenen Bereichen eingesetzt, wie beispielsweise Klassifizierung von Tumorzellen (vgl. [2]), Einsortieren von Kunden in Zielgruppen für gezieltere Werbestrategien (vgl. [3]), Erstellung kreativer Cartoons (vgl. [4]), Nutzung von Chatbots (vgl. [5]) und viele mehr.

Auch in der Numismatik werden Methoden des maschinellen Lernens verwendet, um händische Arbeit zu automatisieren und zu beschleunigen. Beispielsweise können antike Münzen mit Hilfe von neuronalen Netzen klassifiziert werden. Bei der Klassifizierung von Münzen kann zum einen die Münzstätte und zum anderen der Münztyp betrachtet werden.

Mit dieser Thematik beschäftigen sich seit 2021 Dr. Karsten Tolle und Sebastian Gampe im Rahmen des D4N4 (*Data quality for Numismatics based on Natural language processing and Neural Networks*) Projekts [6]. Probleme, die in dieser Zeit aufgekommen sind, haben im Sommersemester 2023 zu einer Data Challenge an der Goethe-Universität Frankfurt am Main geführt, bei der sich vier Gruppen von Studierenden ebenfalls mit dieser Thematik beschäftigt haben. Die meisten Gruppen haben dabei den Fokus auf die Klassifizierung der Münzstätten gelegt, da es dafür zum einen weniger Klassen gab und zum anderen mehr Klassen mit mindestens 20 Bildern. Von den 116 verschiedenen Münzstätten erfüllen 98 Münzstätten diese Bedingung, was ca. 84,5% des Datensatzes entspricht.

Bei den Münztypen erfüllen lediglich 207 von 12087 Typen diese Bedingung. Das entspricht lediglich ca. 1,7% des Datensatzes. Hinzu kommt, dass auch diese 207 Typen unbalanciert sind. Die einzelnen Typen enthalten zwischen 20 und 549 Bilder. Das erste Problem führt dazu, dass viele Münzen gar nicht nach ihrem Typ klassifiziert werden

können. Das liegt daran, dass der Machine Learning Algorithmus die Münztypen nie als Eingabe bekommt. Das zweite Problem führt dazu, dass ein Machine Learning Algorithmus für eine Klassifizierung eine schlechte Vorhersageleistung insbesondere für die kleinen verwendeten Klassen hat.

Diese Probleme werfen die Frage auf, ob es eine Möglichkeit gibt, den Datensatz künstlich zu erweitern und die Münztypen mit wenig Daten besser zu lernen.

Generative künstliche Intelligenz wurde erstmals in den 1960er in Form eines Chatbots namens ELIZA vorgestellt [7]. Allerdings konnte erst mit der Einführung von Generative Adversarial Networks (GANs) im Jahr 2014 generative künstliche Intelligenz überzeugend Bilder, Videos und Audios erstellen.

GANs sollen im Rahmen dieser Arbeit eingesetzt werden, um Bilder von antiken Münzen zu erstellen. In der Regel liefern diese Netzwerke sehr gute Ergebnisse, wenn sie einen großen Datensatz als Eingabe bekommen. Da in dieser Arbeit das Ziel eine Erweiterung des Datensatzes bei den kleinen Münztypen ist, stehen nicht so viele Bilder zur Verfügung. Daher wird für die Generierung der Münzbilder das im Jahr 2020 vorgestellte Few-Shot GAN von Robb et al. [8] verwendet. Anschließend soll eine Klassifizierung der Münztypen mit Hilfe von Ensemble Learning umgesetzt werden.

Als Erstes werden in Kapitel 2 einige Grundlagen erläutert. In Abschnitt 2.1 werden kurz ein paar relevante Begriffe des maschinellen Lernens thematisiert. Die Abschnitte 2.2 und 2.3 befassen sich mit den zwei in dieser Arbeit wesentlichen Machine Learning Teilbereichen. In Abschnitt 2.2 werden die Komponenten und die allgemeine Funktionsweise der GANs behandelt. Um die Bildqualität der generierten Bilder zu bewerten, wird anschließend auf Evaluationsmethoden der Bildgenerierung eingegangen. Abschließend wird die Architektur und Funktionsweise des Few-Shot GANs dargestellt. In Abschnitt 2.3 geht es um das Ensemble Learning, wobei zunächst die Voraussetzungen und die Vorteile benannt werden. Für die Klassifizierung der Münztypen wird einmal Voting und einmal Stacking verwendet. Daher behandelt Unterabschnitt 2.3.2 die Funktionsweise von Voting und welche verschiedenen Umsetzungsmöglichkeiten es hierfür gibt und in Unterabschnitt 2.3.3 wird das Prinzip des Stackings skizziert.

Anschließend thematisiert Kapitel 3 den dieser Arbeit zur Verfügung gestellten Daten-

satz der antiken Münzen.

Kapitel 4 und 5 gehen auf die Implementierung der einzelnen Teile ein. In Kapitel 4 wird erläutert, wie mit Hilfe des Few-Shot GANs antike Münzbilder generiert werden. Verschiedene Versuche und die daraus resultierenden Ergebnisse bzw. Beobachtungen, die beim Generieren der Bilder aufgefallen sind, werden aufgezeigt und evaluiert. Anschließend werden 13 Münztypen für die Generierung neuer Bilder ausgewählt, um die Trainingsdatensätze der Base Learner, die im nächsten Abschnitt thematisiert werden, zu erweitern.

Kapitel 5 behandelt die Implementierung der Klassifizierung der Münztypen mit Hilfe von Ensemble Learning. Für ein Ensemble werden mehrere Base Learner benötigt, weshalb zunächst dargelegt wird, wie die einzelnen Base Learner trainiert werden und wie deren Vorhersageleistung auf ungewohnte Daten ist. Die Base Learner werden auf drei unterschiedliche Trainingsdatensätze trainiert. Ein Trainingsdatensatz besteht nur aus Originaldaten und die zwei anderen Trainingsdatensätze erhalten auch generierte Bilder für die 13 ausgewählten Typen. Diese Base Learner können dann verschiedene Ensembles bilden, die einmal Voting und einmal Stacking anwenden. Sowohl beim Voting als auch beim Stacking werden verschiedene Implementierungen verwendet und bewertet. Weiterhin wird in diesem Kapitel auf die Frage eingegangen, ob generierte Bilder einen Trainingsdatensatz erweitern können, um die Klassifizierung zu verbessern.

Da sich neben dieser Arbeit eine weitere Arbeit mit der Generierung von antiken Münzbildern beschäftigt, werden in Kapitel 6 die Ergebnisse der Bildgenerierung mittels GANs und mittels Stable Diffusion miteinander verglichen. Für diesen Vergleich werden erneut Base Learner trainiert, wobei zwei neue Trainingsdatensätze erstellt werden, die Bilder enthalten, die mittels Stable Diffusion erzeugt wurden.

Abschließend werden die Ergebnisse dieser Arbeit in Kapitel 7 zusammengefasst und ein Ausblick auf weitere Möglichkeiten, die Problematik des unbalancierten Datensatzes und der Klassifizierung bei Münztypen zu verbessern, gegeben.

2 Grundlagen

Das folgende Kapitel soll einen grundlegenden Überblick über die relevanten Themen dieser Arbeit geben. Der Fokus liegt zum einen auf den Generative Adversarial Networks ([Abschnitt 2.2](#)), die zur Generierung neuer Münzbilder verwendet werden sollen, und zum anderen auf dem Ensemble Learning ([Abschnitt 2.3](#)), mit dem die Klassifizierung der Münztypen durchgeführt werden soll. Beide Themen sind Teilbereiche des maschinellen Lernens, weswegen kurz einige Grundlagen in [Abschnitt 2.1](#) behandelt werden. Wer in Machine Learning bereits Kenntnisse hat, kann dieses Kapitel überspringen und direkt zu [Abschnitt 2.2](#) übergehen.

2.1 Machine Learning

Im Folgenden sollen einige Grundbegriffe des maschinellen Lernens erläutert werden. Dieses Kapitel gibt eine kurze Zusammenfassung über Anwendungsbereiche ([Unterabschnitt 2.1.1](#)), Typen von Lernalgorithmen ([Unterabschnitt 2.1.2](#)), Convolutional Neural Networks ([Unterabschnitt 2.1.3](#)), Evaluationsmetriken ([Unterabschnitt 2.1.4](#)) und Grad-CAM ([Unterabschnitt 2.1.5](#)). Für den interessierten Leser bieten [\[1\]](#), [\[9\]](#) und [\[10\]](#) einen guten Überblick über den Bereich Machine Learning.

2.1.1 Arten von Anwendungsbereichen

Machine Learning ist eine Sammlung von Algorithmen und Techniken, die verwendet werden, um Systeme zu entwickeln, die aus Daten lernen können. Mit Hilfe der Algorithmen sollen verschiedene Probleme gelöst werden. Dabei geht es im Wesentlichen darum, Muster in Daten zu erkennen [\[11\]](#). Es können vier große Arten von Anwendungen unterschieden werden:

1. Klassifizierung (*classification*): Neue ungesehene Daten werden einer Kategorie bzw. Klasse c , zugeordnet. Werden nur zwei Klassen unterschieden, spricht man von einer Zwei-Klassen-Klassifizierung (*two-class classification*), bei mehreren Klassen von einer Multi-Klassen-Klassifizierung (*multi-class classification*).

2. Regression (*regression*): Die Beziehung zwischen verschiedenen Variablen aus den Daten wird geschätzt, um eine numerische Vorhersage zu machen.
3. Clusteranalyse (*clustering*): Erkennen von ähnlichen Strukturen in den Daten zur Bildung von Gruppen, welche auch Cluster genannt werden, und Zuordnung von Datenpunkten in diese Gruppen.
4. Synthetische Datengenerierung (*synthetic data generation*)¹: Generierung von künstlichen Daten aus echten Daten durch die Erstellung eines Modells, welches die Verteilungen und die Struktur der realen Daten erfasst.

Im Rahmen dieser Arbeit werden sowohl die Klassifizierung (1.) als auch die synthetische Datengenerierung (4.) auf antike Münzen angewendet.

2.1.2 Typen von Machine Learning Algorithmen

Abhängig vom zu lösenden Problem können unterschiedliche Daten vorliegen, die unterschiedliche Informationen enthalten und somit unterschiedliche Lernprozesse benötigen. Es können vier Typen von Lernalgorithmen unterschieden werden:

1. Überwachtes Lernen (*supervised learning*): Der Datensatz ist gelabelt und jeder Datenpunkt aus dem Datensatz kann einem festen Label zugeordnet werden. Während des Lernprozesses werden Muster in den Daten gelernt, die abhängig vom Label sind. Am Ende soll der Algorithmus in der Lage sein neue ungelabelte Daten einem Label zuzuordnen.
2. Unüberwachtes Lernen (*unsupervised learning*): Der Datensatz ist nicht gelabelt. Während des Lernprozesses werden Muster und Zusammenhänge in den Daten gelernt. Das Ziel ist es, Beziehungen zwischen den Daten zu finden.
3. Teilüberwachtes Lernen (*semi-supervised learning*): Es liegen sowohl gelabelte, als auch ungelabelte Daten vor. Für die gelabelten Daten wird überwachtes Lernen angewendet und für die ungelabelten Daten unbewachtes Lernen.

¹Es werden drei Typen von synthetischen Daten unterschieden: 1. Generierung aus echten Daten, 2. Generierung ohne echte Daten und 3. Mischung aus 1. und 2. Hier liegt der Fokus auf der Generierung aus echten Daten. Für den interessierten Leser bietet [12] einen guten Überblick.

4. Bestärkendes Lernen (*reinforcement learning*): Zusatzdaten werden durch ein Belohnungssystem erzeugt. Während des Lernprozesses gibt es eine direkte Interaktion mit der Umwelt, die von einem Agenten beobachtet wird. Auf Grundlage seiner Beobachtungen führt er eine Aktion aus, die er für angemessen hält. Die Umwelt schickt dem Agenten ein Feedbacksignal, die als Belohnung dient und dem Agenten hilft zu erkennen, wie gut seine Entscheidung war.

Für Klassifizierungsprobleme werden Algorithmen des überwachten Lernens eingesetzt und für die generative künstliche Intelligenz können sowohl Algorithmen des teilüberwachten als auch des unüberwachten Lernens angewendet werden [13].

2.1.3 Convolutional Neural Networks (CNNs)

Eine häufig verwendete Methode, um Probleme mit Hilfe von Machine Learning zu lösen sind künstliche neuronale Netze. Diese sind aus der Biologie inspirierte Netze aus künstlichen Neuronen. In der Regel sind künstliche neuronale Netze in Schichten organisiert, welche aus einer Reihe miteinander verbundener Knoten, die eine Aktivierungsfunktion enthalten, bestehen. Der grundlegende Aufbau eines künstlichen neuronalen Netzes ist in Abbildung 1 dargestellt.

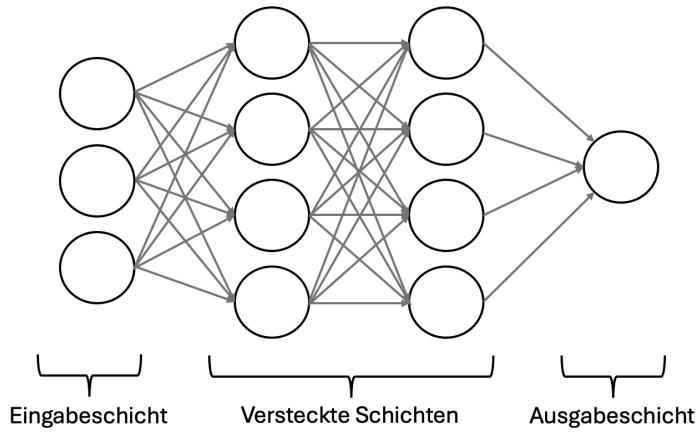


Abbildung 1: Grundlegender Aufbau eines künstlichen neuronalen Netzes.

Die Muster werden dem Netz über die Eingabeschicht präsentiert, die mit einer oder mehreren versteckten Schichten kommuniziert, wo die eigentliche Verarbeitung über ein System gewichteter Verbindungen erfolgt. Die versteckten Schichten sind dann mit

einer Ausgabeschicht verbunden, auf der die Antwort ausgegeben wird. Das Ziel dieser Netzwerke ist es den Verlust (*loss*) zu minimieren bzw. die Genauigkeit (*accuracy*) der Modellvorhersagen zu verbessern. Dies ist schematisch in Abbildung 2 dargestellt. Gegeben eine Funktion f mit Eingabe x , einem Label y und den Gewichten w , wird nach jeder Ausgabe der Gradient des Verlusts abhängig von den Gewichten berechnet. Durch die Fehlerrückführung (*backpropagation*) werden die Gewichte w angepasst, um den Verlust zu optimieren.

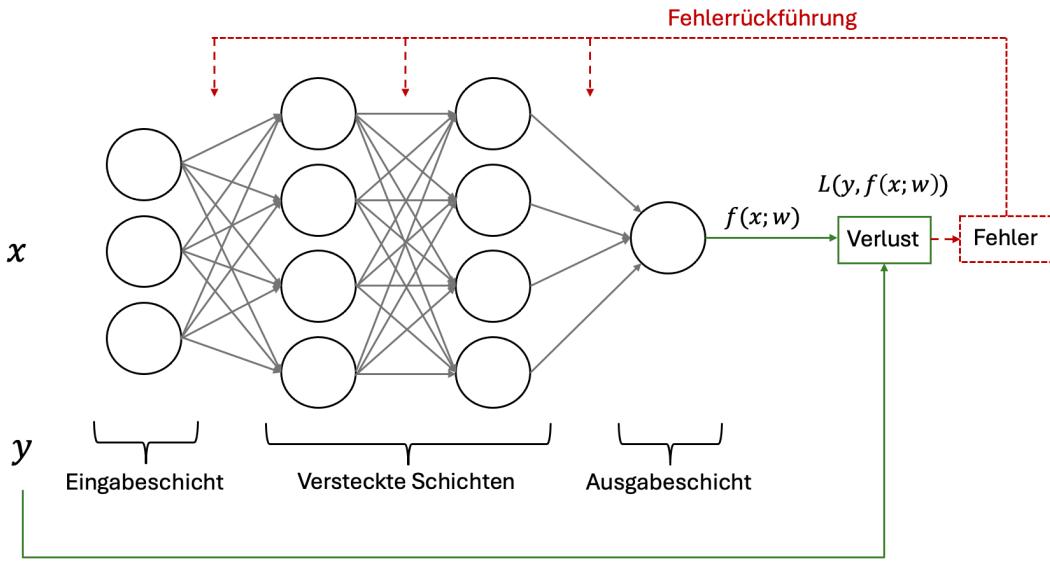


Abbildung 2: Schematische Darstellung der Fehlerrückführung.

Es können verschiedene Arten von künstlichen neuronalen Netzen unterschieden werden. Für die Klassifizierung der Münztypen werden die Convolutional Neural Networks, welche im Folgenden kurz erläutert werden, und für die Bildgenerierung die Generative Adversarial Networks, welche im Abschnitt 2.2 thematisiert werden, eingesetzt.

Die Convolutional Neural Networks sind mehrschichtige neuronale Netze, die aus den in Abbildung 1 bestehenden Schichten aufgebaut sind. Der Unterschied liegt in der versteckten Schicht, in der drei Arten von Schichten (*layer*) vorkommen, die sich unterschiedlich oft wiederholen (s. Abbildung 3):

1. Convolutional Layer: Der Convolutional Layer gibt dem Netzwerk seinen Namen. Hier werden die Eingaben gefaltet. Das heißt, dass zunächst ein kleiner Teil des Bildes gefiltert wird, dann ein überlappender Teil und so weiter, bis das gesamte

Bild erfasst ist. Alle Werte, die sich aus der Faltung des Filters über das Bild ergeben, werden in einem Array abgelegt, das als Feature Map bezeichnet wird und die Ausgabe dieser Schicht darstellt.

2. Pooling (oder Subsampling) Layer: Die Ergebnisse aus dem Convolutional Layer werden an ein Pooling Layer weitergegeben, der unnötige Ergebnisse verwirft und nur die notwendigen Informationen an die nächste Schicht weitergibt.
3. Fully-Connected Layer: Alle Neuronen werden mit allen Ein- und Ausgängen verbunden.

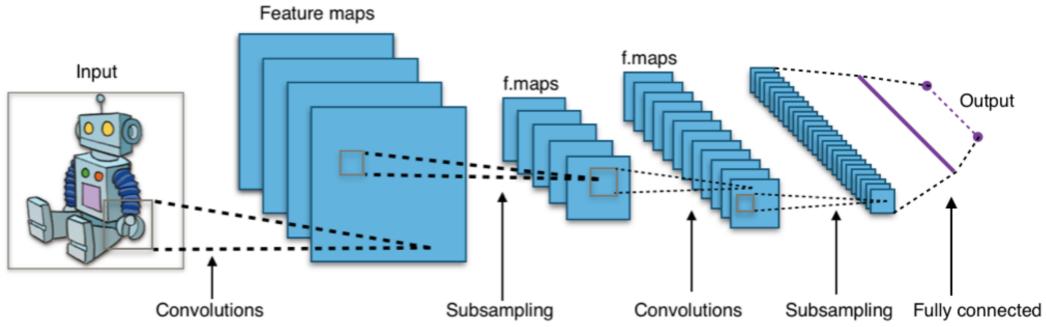


Abbildung 3: Grundlegender Aufbau eines Convolutional Neural Networks [14].

Loyal [15] bietet dem interessierten Leser einen guten Überblick über den genauen Aufbau und die (mathematische) Funktionsweise dieser Netze.

2.1.4 Evaluationsmetriken

Um die Eignung eines Machine Learning Algorithmus zu evaluieren, werden verschiedene Metriken verwendet, die abhängig von der Anwendung sind. Es ist von Interesse, die Fähigkeit des Algorithmus auf ungesiehene Daten zu bewerten. Dafür wird die Evaluation auf einem Testdatensatz durchgeführt, der sich von dem Datensatz, der für das Training verwendet wird, unterscheidet. Für eine Klassifizierung beispielsweise wird häufig die Genauigkeit (*accuracy*) verwendet. Die Genauigkeit gibt das Verhältnis von den richtigen Vorhersagen zur Gesamtanzahl aller Vorhersagen an, formal heißt das:

$$\text{Genauigkeit} = \frac{RP + RN}{RP + RN + FP + FN}, \text{ mit}$$

$RP = \text{Richtig Positiv}$, $RN = \text{Richtig Negativ}$,

$FP = \text{Falsch Positiv}$ und $FN = \text{Falsch Negativ}$.

Ähnliche Informationen können mit der Fehlerrate (*error rate*) gewonnen werden. Diese gibt das Verhältnis der falschen Vorhersagen zur Gesamtanzahl aller Vorhersagen an, das heißt formal:

$$\text{Fehlerrate} = \frac{FP + FN}{RP + RN + FP + FN} = 1 - \text{Genauigkeit}.$$

Eine Visualisierung dieser Ergebnisse kann mit Hilfe einer Konfusionsmatrix (*confusion matrix* oder *error matrix*) erstellt werden. Es handelt sich hierbei um eine quadratische Matrix, die in den Zeilen und Spalten alle Klassen enthält. Auf der Diagonalen sind alle richtigen Vorhersagen abgebildet, während die anderen Einträge zeigen, wie oft ein Exemplar aus der Klasse c_i fälschlicherweise zur Klasse c_j zugeordnet wurde. In Tabelle 1 ist die Konfusionsmatrix für ein Zwei-Klassen-Problem beispielhaft dargestellt.

		Vorhergesagte Klasse	
		c_1	c_2
Tatsächliche Klasse	c_1	RP	FN
	c_2	FP	RN

Tabelle 1: Konfusionsmatrix für ein Zwei-Klassen-Problem.

Die Genauigkeit kann bei Klassifizierungsproblemen mit einem unbalancierten Datensatz irreführend sein. Sei ein Zwei-Klassen-Problem wie in Tabelle 1 mit den Klassen c_1 und c_2 gegeben. Der Datensatz enthalte insgesamt 1000 Datenpunkte, wobei 990 zu c_1 und 10 zu c_2 gehören. Würde der Algorithmus alle Daten zu c_1 klassifizieren ($RP = 990$, $RN = 0$, $FP = 10$, $FN = 0$), hätte er eine Genauigkeit von $\frac{990}{1000} = 0,99 = 99\%$, allerdings würde kein Datenpunkt aus c_2 erkannt werden. Auf Grund dieser Problematik sollten bei unbalancierten Daten auch weitere Metriken betrachtet werden. Vier weitere geläufige Metriken sind:

1. Positiver Vorhersagewert (*precision* oder *positive predictive value*): Gibt das Verhältnis der richtigen positiven Vorhersagen zu allen positiven Vorhersagen an.

$$\text{Positiver Vorhersagewert} = \frac{RP}{RP + FP}$$

2. Sensitivität (*recall* oder *sensitivity* oder *true positive rate*): Gibt das Verhältnis der positiven Exemplare an, die richtig klassifiziert wurden.

$$\text{Sensitivität} = \frac{RP}{RP + FN}$$

3. Spezifität (*specificity* oder *true negative rate*): Gibt das Verhältnis der negativen Exemplare an, die richtig klassifiziert wurden.

$$\text{Spezifität} = \frac{RN}{RN + FP}$$

4. F-Maß (*F-measure* oder *F1-score*): Kombination vom positiven Vorhersagewert und der Sensitivität.

$$F - \text{Maß} = \frac{2 \cdot \text{Positiver Vorhersagewert} \cdot \text{Sensitivität}}{\text{Positiver Vorhersagewert} + \text{Sensitivität}} = \frac{2 \cdot RP}{2 \cdot RP + FP + FN}$$

2.1.5 Gradient-weighted Class Activation Mapping (Grad-CAM)

Bei der Klassifizierung von Bildern mit Hilfe eines CNN ist dem Benutzer oft unklar, weshalb das CNN die Entscheidung getroffen hat. Der Entscheidungsprozess ähnelt einer Black Box. Die Gradient-weighted Class Activation Mapping (Grad-CAM) ist eine häufig verwendete Methode, um Einblicke in diesen Entscheidungsprozess zu bekommen. Diese hat sich bereits in Vorarbeiten von Loyal [15] oder Gampe [16] bewährt, um den Entscheidungsprozess transparenter zu machen. Die Grad-CAM, ist eine Visualisierungsmethode, die Bildregionen hervorhebt, welche die Entscheidung des CNN stark beeinflusst haben.

Die Methode berechnet mit Hilfe der Gradienten der letzten Convolutional Schicht ei-

ne Heatmap, indem die Gradienten mittels Backpropagation bis zur Eingabeschicht zurückverfolgt werden. Durch die Rückverfolgung können die Bildregionen ermittelt werden, welche die Zuteilung zu der bestimmten Klasse besonders stark beeinflusst haben. Diese Bildregionen werden letztendlich in einer Heatmap ausgeben und über das Originalbild gelegt. Für eine ausführliche mathematische Beschreibung der Berechnung der Heatmaps wird auf den wissenschaftlichen Artikel [17] verwiesen. Abbildung 4 zeigt ein Bild, über das eine Heatmap gelegt wurde. Dabei ist am Farbverlauf zu erkennen, welche Regionen des Bildes besonders entscheidend waren. Der Farbverlauf verläuft von Blau bis Rot. Die besonders wichtigen Regionen sind rot und weniger wichtigen Regionen sind blau eingefärbt.



Abbildung 4: Beispiel der Grad-CAM Visualisierungsmethode.

Diese Methode eignet sich für die vorliegende Arbeit, da so die Unterschiede der Zuteilung der verschiedenen CNNs ermittelt werden können. Durch die Hervorhebung kann herausgefunden werden, ob die CNNs mit verschiedenen Datensätzen andere Merkmale für die Klassifizierung als besonders wichtig erachten. Grad-CAM kann für jede Vorhersage eines CNN benutzt werden. In dieser Arbeit werden nur die Heatmaps der Top-1 Vorhersagen betrachtet. Die verwendete Implementation stammt von Tensorflow [18]. Die Heatmaps dieser Arbeit werden von der letzten Convolutional Schicht conv5_block3_3_conv der ResNet errechnet.

2.2 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) wurden erstmals 2014 von Goodfellow et al. [19] vorgestellt. Sie können in einer Vielzahl von Anwendungen eingesetzt werden, wie beispielsweise Bildsynthese, semantische Bildbearbeitung, Stilübertragung usw. [13]. Im Folgenden liegt der Fokus der GAN-Anwendung auf Bilddaten. Es handelt sich um einen generativen Modellansatz, der zwei gegnerische (*adversarial*) Netzwerke verwendet. Beim einen Netzwerk handelt es sich um einen Generator \mathcal{G} , der Fälschungen mit dem Ziel, realistische Bilder zu erzeugen, generiert, und beim anderen Netzwerk handelt es sich um einen Diskriminator \mathcal{D} , der erkennen soll, welche Bilder echt sind und welche Fälschungen. Wie diese zwei Netzwerke miteinander interagieren, wird in [Unterabschnitt 2.2.1](#) erläutert.

Um die Qualität der generierten Bilder zu bewerten gibt es verschiedene Methoden. Im Rahmen dieser Arbeit werden zwei Methoden eingesetzt, die in [Unterabschnitt 2.2.2](#) vorgestellt werden.

Es gibt verschiedene GAN-Architekturen, wobei eine die Convolutional GANs sind. Diese setzen für die Netzwerke Convolutional Neural Networks (CNNs) ein. Das in dieser Arbeit verwendete Few-Shot GAN von Robb et al. [8], welches in [Unterabschnitt 2.2.3](#) thematisiert wird, gehört zu diesen Convolutional GANs.

2.2.1 Allgemeine Funktionsweise und Komponenten

Die allgemeine Funktionsweise von GANs kann anhand [Abbildung 5](#) erläutert werden. GANs bestehen aus zwei miteinander konkurrierenden Komponenten, dem Generator \mathcal{G} und dem Diskriminator \mathcal{D} . Bei diesen Komponenten handelt es sich in der Regel um CNNs, sie können aber auch durch andere differenzierbare Systeme implementiert werden, die Daten von einem Raum auf einen anderen abbilden [13].

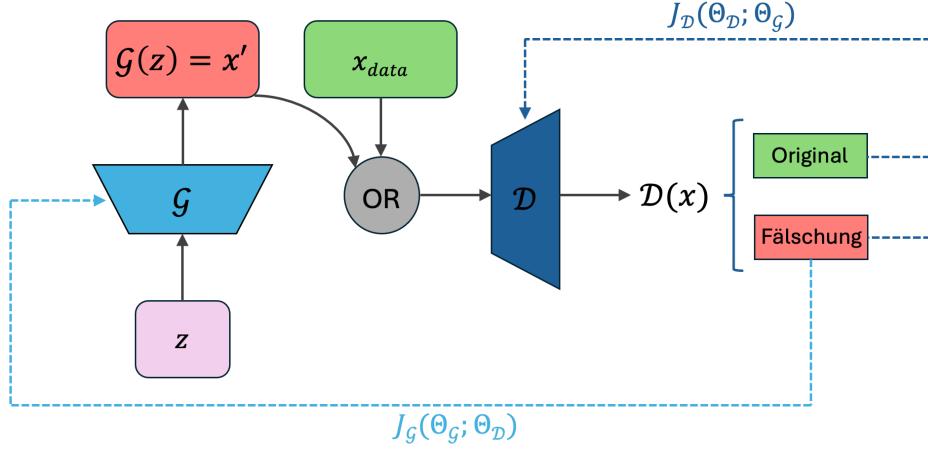


Abbildung 5: Allgemeine Funktionsweise eines GANs in Anlehnung an [13] und [20].

Der Generator \mathcal{G} erhält einen Rauschvektor z als Eingabe, der ein Zufallsvektor mit Gleichverteilung oder Gaußscher Verteilung ist. Gegeben dieser Eingabe erzeugt \mathcal{G} künstliche Bilder x' , die den Diskriminatoren \mathcal{D} täuschen sollen. Das Generatornetzwerk lässt sich formal als eine Abbildung von einem Repräsentationsraum, der als latenter Raum bezeichnet wird, auf den Raum der Daten ausdrücken [13]:

$$\mathcal{G} : \mathcal{G}(z) \rightarrow R^{|x|}, \text{ mit } z \in R^{|z|} \text{ und } x \in R^{|x|}.$$

Die Eingabe z ist ein Vektor aus dem latenten Raum, x ist ein Bild aus dem Raum der Daten und $|\cdot|$ gibt die Dimension an [13].

Der Diskriminatator \mathcal{D} erhält eine Eingabe x , die einem Bild entspricht. Er muss entscheiden, ob es sich bei diesem Bild um ein echtes Bild (Original) aus dem Datensatz handelt, oder um ein generiertes Bild (Fälschung) vom Generator. Das Diskriminatorennetzwerk kann als eine Funktion $\mathcal{D} : \mathcal{D}(x) \rightarrow (0,1)$ charakterisiert werden, bei der Bilddaten auf einen Wert zwischen 0 und 1 abgebildet werden [13]. Dieser Wert gibt die Wahrscheinlichkeit an, mit der das Bild x vom Diskriminatator \mathcal{D} als echt eingeschätzt wird. Ein hoher Wert bedeutet, dass das Bild laut \mathcal{D} aus den (realen) Trainingsdaten stammt, während ein niedriger Wert bedeutet, dass \mathcal{D} das Bild für eine Fälschung hält, die vom Generator synthetisch erzeugt wurde.

Im GAN-Training geht es sowohl um das Finden der geeigneten Parameter (Gewichte) für den Generator \mathcal{G} , das heißt Bilder generieren, die den Diskriminatator täuschen, als auch um das Finden der geeigneten Parameter (Gewichte) für den Diskriminatator \mathcal{D} , also das Maximieren der Klassifizierungsgenauigkeit. Diese Gewichte werden durch die Fehlerrückführung gelernt. $J_{\mathcal{G}}(\Theta_{\mathcal{G}}; \Theta_{\mathcal{D}})$ ist die Verlustfunktion vom Generator und $J_{\mathcal{D}}(\Theta_{\mathcal{D}}; \Theta_{\mathcal{G}})$ ist die Verlustfunktion vom Diskriminatator. Die Parametermengen $\Theta_{\mathcal{G}}$ und $\Theta_{\mathcal{D}}$ der entsprechenden Netzwerke kommen in beiden Verlustfunktionen vor, da diese durch das iterative Aktualisieren voneinander co-abhängig sind [13].

Das GAN-Training kann als Nullsummenspiel zwischen dem Generator \mathcal{G} und dem Diskriminatator \mathcal{D} gesehen werden. Die Kosten des Trainings werden mit einer Wertfunktion (*value function*) $V(\mathcal{G}, \mathcal{D})$, die sowohl vom Generator, als auch vom Diskriminatator abhängt, bewertet. Während \mathcal{G} diese Funktion minimieren möchte, hat \mathcal{D} die Maximierung zum Ziel. Formal lässt sich das wie Folgt ausdrücken [13]:

$$\max_{\mathcal{D}} \min_{\mathcal{G}} V(\mathcal{G}, \mathcal{D}),$$

mit

$$V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{p_{data}(x)} \log \mathcal{D}(x) + \mathbb{E}_{p_g(x)} \log(1 - \mathcal{D}(x)).$$

\mathbb{E} ist der Erwartungswert, $p_{data}(x)$ stellt die Wahrscheinlichkeitsdichtefunktion über einen Zufallsvektor x dar, der in $R^{|x|}$ liegt, und $p_g(x)$ bezeichnet die Verteilung der Vektoren, die vom Generatornetzwerk des GANs erzeugt werden [13].

Während des Trainings werden die Parameter des einen Netzwerks fix gehalten, während die anderen aktualisiert werden. Goodfellow et al. [19] zeigen, dass es für einen fixen Generator \mathcal{G} einen einzigen optimalen Diskriminatator $\mathcal{D}_{\mathcal{G}}^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$ gibt. Der Generator \mathcal{G} ist optimal, wenn $p_{data}(x) = p_g(x)$. In diesem Fall würde der Diskriminatator immer 0,5 vorhersagen, da er nicht zwischen den echten und den realen Bildern unterscheiden kann.

2.2.2 Evaluationsmethoden für die Bildqualität

Es gibt verschiedene Methoden, um GANs zu evaluieren. Einen guten Überblick über die verschiedenen Metriken bieten [20] und [21]. Insgesamt können drei Kategorien von Evaluationsmetriken unterschieden werden [20]:

1. Manuelle Evaluation: Bei der manuellen Evaluation der GANs werden die generierten Bildern mit den echten Bildern von Forschern oder Personen mit entsprechender Erfahrung verglichen. Während des Trainingsprozesses wird ein Modell nach unterschiedlich vielen Epochen zwischengespeichert. Anschließend kann aus diesen Zwischenergebnissen das Modell mit den besten generierten Bildern gewählt werden. Dies ist auch die einfachste, gängigste und intuitivste Methode zur Bewertung der Bildqualität. Das Problem dieser Methode ist allerdings, dass manuelle Bewertungen zum einen subjektiv und willkürlich sind, aber auch, dass sie sehr zeitaufwendig sind [20].
2. Qualitative Evaluation: Bei der qualitativen Evaluation von GANs handelt es sich um nicht-numerischer Messgrößen. Diese beinhalten häufig eine vergleichende oder subjektive Bewertung. Die bekannteste qualitative Bewertungsmethode ist die Nächste-Nachbarn (*Nearest Neighbors*) Methode. Diese wendet Abstandsmetriken wie den euklidische Abstand zwischen Pixelbildinformationen an, um den Grad der Ähnlichkeit der generierten Bilder mit den realen Bildern zu bewerten [20].
3. Quantitative Evaluation: Bei der quantitativen Evaluation der GANs werden numerische Werte berechnet, um die Qualität der generierten Bilder zu vergleichen. Eine häufig verwendete Metrik ist die *Fréchet Inception Distance* (FID), bei der der Abstand zwischen den Merkmalsvektoren der realen und der generierten Bilder berechnet wird [20].

In dieser Arbeit wird hauptsächlich die manuelle Evaluation der generierten Bildern angewendet, worauf in [Kapitel 4](#) eingegangen wird. Allerdings wird auch der FID-Wert beim StyleGAN2 (vgl. [Unterabschnitt 4.5.1](#)) berechnet, weswegen dieser nun kurz erläutert wird.

Bei der *Fréchet Inception Distance*, welche 2017 von Heusel et al. [22] erstmals eingeführt wurde, wird die Verteilung der erzeugten Bilder mit der Verteilung einer Reihe von echten Bildern verglichen.

Gegeben einer geeigneten Feature-Funktion ϕ , welche standardmäßig das convolutional Feature des Inception-Netzes ist, modelliert FID $\phi(\mathbb{P}_r)$ und $\phi(\mathbb{P}_g)$ als Gaußsche Zufallsvariablen mit den statistischen Erwartungswerten μ_r , μ_g und statistischer Kovarianzmatrix \mathbf{C}_r , \mathbf{C}_g [21]. Der Index r gibt an, dass es die reale Verteilung ist und der Index g gibt an, dass es die generierte Verteilung ist, die die reale Verteilung approximieren soll. Die Fréchet-Distanz (oder äquivalent dazu die Wasserstein-2-Distanz) zwischen den beiden Gauß-Verteilungen wird dann wie folgt berechnet [21]:

$$\text{FID}(\mathbb{P}_r, \mathbb{P}_g) = \|\mu_r - \mu_g\| + \text{tr}(\mathbf{C}_r + \mathbf{C}_g - 2(\mathbf{C}_r \mathbf{C}_g)^{\frac{1}{2}}).$$

Mit tr ist die Spur (*trace*) einer quadratischen Matrix gemeint, also die Summe der Hauptdiagonalelemente der Matrix.

Das Ziel eines GANs ist ein kleiner FID-Wert, da dann die generierte Verteilung \mathbb{P}_g am besten die reale Verteilung \mathbb{P}_r approximiert.

2.2.3 Beispielarchitekturen: Few-Shot GAN (FSGAN)

GANs benötigen in der Regel eine Vielzahl an Eingabedaten, um gute generierte Daten zu erzeugen. Im Rahmen dieser Arbeit ist das Ziel, generierte Münzbilder für Münztypen zu erzeugen, für die nur wenige Daten vorliegen. Daher wird das von Robb et al. entwickelte Few-Shot GAN (FSGAN) [8], das weniger als 100 Bilder benötigt und das StyleGAN2 [23] Training-Framework nutzt, eingesetzt.

Während bei GANs der Parameterraum innerhalb der Modellarchitektur identifiziert wird, wird beim FSGAN einen Parameterraum auf der Grundlage der vortrainierten Gewichte entdeckt [8]. Die Singulärwertzerlegung (*Singular Value Decomposition*, SVD) wird auf die vortrainierten Gewichte angewendet und eine Basis, die die orthogonale Richtungen mit maximaler Varianz im Gewichtsraum darstellt, wird entdeckt [8]. Unterschiedliche Singulärwerte führen zu natürlichen und semantisch bedeutsamen

Veränderungen in dem ausgegebenen Bild. Eine Änderung der Singulärwerte bietet eine aussagekräftige Parametrisierung der vortrainierten Gewichte [8]. Eine allgemeine mathematische Definition von SVD ist beispielsweise in [24] gegeben. Allgemein charakterisieren Singulärwerte Eigenschaften einer Matrix.

Beim FSGAN wird SVD sowohl auf jeder Schicht des Generatornetzes $\mathcal{G}^{(l)}$ als auch auf jeder Schicht des Diskriminatornetzes $\mathcal{D}^{(l)}$ eines vortrainierten GANs angewendet. Die Schichten können entweder aus 2D ($c_{in} \times c_{out}$) Fully-Connected oder 4D ($k \times k \times c_{in} \times c_{out}$) Convolutional-Filter Gewichten bestehen [8]. SVD kann nur auf 2D-Matrizen durchgeführt werden, weswegen der 4D-Tensor zunächst in einen 2D-Tensor umgewandelt werden muss. Für eine einzelne Schicht von vortrainierten Gewichten $W_0^{(l)} \in \mathbb{R}^{k^2 c_{in} \times c_{out}}$ wird dann SVD angewendet, um die Zerlegung zu erhalten [8]:

$$W_0^{(l)} = (U_0 \Sigma_0 V_0^\top)^{(l)}$$

Anschließend wird die Domänenanpassung durchgeführt, indem die linken/rechten Singulärwerte in $(U_0 V_0)^{(l)}$ eingefroren werden und die Singulärwerte $\Sigma = \lambda \Sigma_0$ unter Verwendung eines Standard-GAN-Ziels optimiert werden, um die übertragene Gewichte zu erhalten [8]:

$$W_\Sigma^{(l)} = (U_0 \Sigma V_0^\top)^{(l)}$$

Das Ziel der FSGAN Domänenanpassung ist das Finden einer neuen Menge an Singulärwerten für jede Schicht des vortrainierten GANs, sodass die generierten Bilder zu der Verteilung der Zieldomäne passen [8].

2.3 Ensemble Learning

Ensemblemethoden werden bereits in vielen verschiedenen Bereichen wie Versicherungen, Finanzen, Gesundheitsfürsorge oder Empfehlungssysteme eingesetzt. Die einfache Erklärung dieser Methoden stammt aus der menschlichen Natur. Menschen tendieren dazu, verschiedene Meinungen zu sammeln, diese zu kombinieren und schließlich eine finale Entscheidung zu treffen. Es ist ein Sammelbegriff für das Zusammenführen verschiedener Antworten einer Gruppe, um dann eine bessere finale Antwort zu geben.

Für den Erfolg dieser Methoden werden zum einen vielfältige Meinungen im Ensemble benötigt, und zum anderen eine geeignete Aggregation dieser Meinungen [25]. Der Grundgedanke dieser Methoden ist, dass eine kombinierte Antwort von verschiedenen Meinungen besser ist als eine einzelne [26].

Dieser Ansatz wird auch im maschinellen Lernen sowohl für strukturierte Daten, die beispielsweise in Tabellen oder relationalen Datenbanken vorliegen, als auch für unstrukturierte Daten, wie beispielsweise Bilder oder Videos, genutzt [25]. Alle möglichen Algorithmen des maschinellen Lernen, wie beispielsweise Neuronale Netze, Entscheidungsbäume oder auch lineare Regressionsmodelle, können als Einzelmodelle eines Ensembles, das eine Gruppe von maschinellen Lernverfahren bildet, verwendet werden [27]. Diese Einzelmodelle werden oft als *Weak Learner* oder auch *Base Learner* bezeichnet. In Abbildung 6 ist der Grundgedanke beispielhaft dargestellt. Die Base Learner bekommen alle eine Eingabe x , welche zum Beispiel ein Bild einer Münze sein kann, deren Typ klassifiziert werden soll. Jeder Base Learner macht eine Vorhersage, wie dieses Bild zu klassifizieren ist. Dabei ist jede Vorhersage unabhängig von den anderen. Die Vorhersagen der verschiedenen Base Learner werden schließlich kombiniert und eine Ausgabe y , welche die finale Klassifizierung ist, wird zurückgegeben. Insgesamt soll dadurch die Vorhersageleistung einer Aufgabe verbessert werden [27].

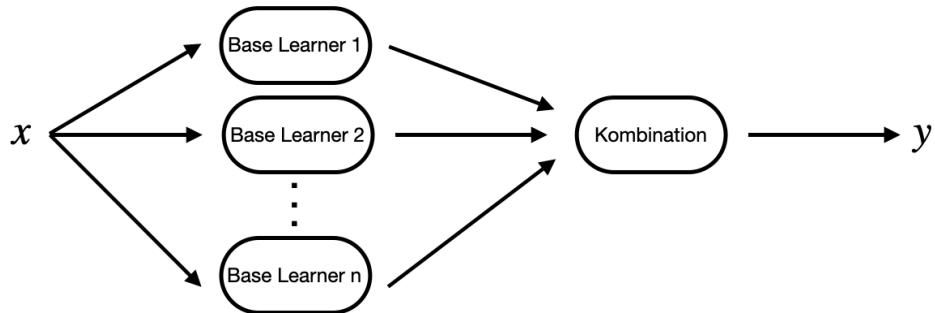


Abbildung 6: Basisarchitektur eines Ensemble Learning Modells in Anlehnung an [27].

Beim Ensemble Learning geht es um die Herausforderung, solche Ensembles aus verschiedenen Einzelmodellen zu trainieren und zu aggregieren [25]. Für die Aggregation der Base Learner gibt es zwei wesentliche Ansätze. Zum einen gibt es Gewichtungsmethoden und zum anderen gibt es Meta Learning Methoden [26]. Bei den Gewichtungsme-

thoden werden die Ausgaben der Base Learner gewichtet. Bei einem Klassifizierungsproblem ist der einfachste Ansatz das Majority Voting, bei dem die finale Ausgabe gewählt wird, die am häufigsten von den Einzelmodellen gewählt wurde. In Regressionsproblemen dagegen kann beispielsweise der Durchschnitt der einzelnen Ausgaben der Base Learner als endgültige Ausgabe berechnet werden [26]. Bei den Meta Learning Methoden werden Metadaten genutzt, um ein bestimmtes Problem zu lösen. Diese Metadaten werden zuerst von den Base Learnern generiert und dann im sogenannten *Meta Learner* verarbeitet [28]. Beim Meta Learning kann es mehrere Lernstufen geben. Das heißt, die Ausgabe aus einer Lernstufe wird als Eingabe für die nächste Lernstufe verwendet. In der letzten Stufe erzeugt dann der Meta Learner die finale Ausgabe [26]. Es gibt viele verschiedene Implementierungen von Ensemble Learning Methoden, die in eine dieser zwei wesentlichen Ansätze eingegliedert werden können.

Im Rahmen dieser Arbeit werden zwei Strategien, nämlich Voting und Stacking, für die Klassifizierung von Münztypen eingesetzt und miteinander verglichen. Hierbei fällt Voting, welches in [Unterabschnitt 2.3.2](#) näher erläutert wird, in die Gewichtungsmethoden, und Stacking, welches in [Unterabschnitt 2.3.3](#) behandelt wird, unter die Meta Learning Methoden. Bevor diese zwei Strategien erläutert werden, werden in [Unterabschnitt 2.3.1](#) die Voraussetzungen und Vorteile von Ensemble Learning thematisiert. [26] und [27] liefern dem interessierten Leser einen guten Überblick über Ensemble Learning und über weitere Strategien in diesem Bereich.

2.3.1 Voraussetzungen und Vorteile von Ensemble Learning Methoden

Damit ein Ensemble von Klassifikatoren präziser als ein Einzelmodell ist, nennen Hansen und Salamon [29] zwei notwendige und hinreichende Bedingungen für die einzelnen Klassifikatoren eines Ensembles:

1. Genauigkeit:

Ein genauer Klassifikator sollte eine Fehlerrate (*error rate*) habe, die besser ist, als das zufällige Raten auf einer neuen Eingabe x . Konkreter soll die Fehlerrate eines einzelnen Klassifikators des Ensembles sogar unter 50% liegen [30]. Das heißt, jeder Klassifikator sollte eine Genauigkeit (*accuracy*) von über 50% haben.

2. Diversität:

Zwei Klassifikatoren werden als divers bezeichnet, wenn sie unterschiedliche Fehler auf neuen Eingaben x machen. Das heißt, die Fehler der Einzelmodelle sollten unkorreliert sein [30].

Diese zwei Voraussetzungen beantworten allerdings noch nicht die Frage, wieso ein Ensemble besser klassifizieren könnte als ein einzelner Klassifizierungsalgorithmus. Dietterich [30] nennt hierfür drei Gründe:

1. Statistisches Problem:

Oben links in Abbildung 7 ist das statistische (*statistical*) Problem dargestellt. Ein Lernalgorithmus kann wie eine Suche in einem Hypothesenraum H angesehen werden, bei der es darum geht, die beste Hypothese in diesem Raum zu finden. Sobald die Menge der verfügbaren Trainingsdaten im Vergleich zur Größe des Hypothesenraums zu klein ist, kommt es zum statistischen Problem. Lediglich die Hypothesen im inneren blauen Bereich liefern eine gute Genauigkeit, gegeben der vorliegenden Daten. In der Abbildung ist der rote Punkt f die wahre unbekannte Hypothese. Die blauen Punkte h_1, h_2, h_3 und h_4 sind die von den verschiedenen Klassifikatoren gefundenen Hypothesen mit gleicher Genauigkeit. Mit Hilfe eines Ensembles können nun diese Ausgaben kombiniert werden und so das Risiko einer falschen Klassifizierung verringern.

2. Rechnerisches Problem:

Oben rechts in Abbildung 7 ist das rechnerische (*computational*) Problem dargestellt. Da Lernalgorithmen oft eine Art lokale Suche im Hypothesenraum H durchführen, besteht die Gefahr, in lokalen Optima stecken zu bleiben. Auch wenn genug Trainingsdaten vorhanden sind, kann es sehr schwierig sein, die beste Hypothese zu finden. Daher können durch die Wahl verschiedener Ausgangspunkte verschiedene individuelle Hypothesen bzw. lokale Optima h_1, h_2 und h_3 gefunden werden, welche durch ihre Kombination eine bessere Annäherung an die wahre unbekannte Hypothese f liefern.

3. Repräsentatives Problem:

Unten in Abbildung 7 ist das repräsentative (*representational*) Problem dargestellt. Bei vielen Aufgaben des maschinellen Lernens kann die wahre unbekannte Hypothese f durch keine Hypothese im Hypothesenraum H repräsentiert werden. Durch die Kombination verschiedener individueller Hypothesen h_1, h_2 und h_3 , kann der Hypothesenraum H erweitert werden, sodass der Lernalgorithmus in der Lage ist, eine genauere Annäherung an die wahre unbekannte Hypothese f zu finden.

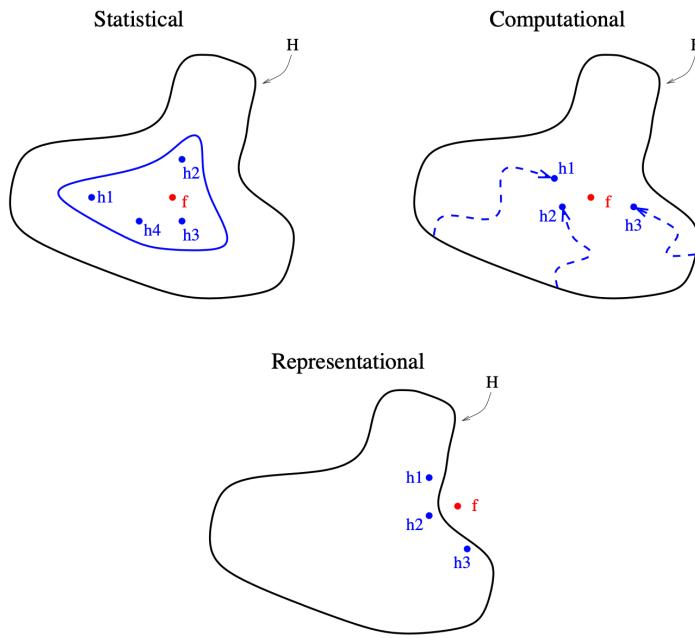


Abbildung 7: Drei wesentliche Gründe, warum ein Ensemble besser funktionieren kann als ein einzelner Klassifikator [30].

Diese drei genannten Probleme sind die wesentlichen Gründe, wieso einzelne Lernalgorithmen versagen. Ensemblemethoden sollen dabei helfen, diese Probleme zu reduzieren oder eventuell auch zu eliminieren [30].

2.3.2 Voting

Die beliebteste Kombinationsmethode für nominale Ausgaben ist das Voting [27]. Da Klassifizierung Teil dieser Arbeit ist, wird im Folgenden anhand eines Klassifizierungsbeispiels erklärt, wie Voting funktioniert.

Sei $\{h_1, \dots, h_T\}$ die Menge von T individuellen Klassifikatoren. Die Aufgabe ist, die einzelnen Klassifikatoren h_i zu kombinieren, um das Klassenlabel aus einer Menge von l möglichen Klassenlabels $\{c_1, \dots, c_l\}$ zu bestimmen. Die Ausgabe eines Klassifikators h_i für eine Eingabe x ist ein l -dimensionaler Label-Vektor $(h_i^1(x), \dots, h_i^l(x))^\top$, wobei $h_i^j(x)$ die Ausgabe von h_i für das Klassenlabel c_j ist. Das $h_i^j(x)$ kann dabei zwei Arten von Werten annehmen [27]:

- Eindeutiges Label: $h_i^j(x) \in \{0,1\}$ mit $h_i^j(x) = 1$, wenn h_i die Klasse c_j vorhersagt und $h_i^j(x) = 0$ sonst.
- Klassenwahrscheinlichkeit: $h_i^j(x) \in [0,1]$, was als Schätzung der A-posteriori-Wahrscheinlichkeit $P(c_j|x)$ angesehen werden kann.

Für die Kombination der Ausgaben gibt es verschiedene Voting Strategien, die in [27] näher erläutert werden. Im Folgenden werden lediglich zwei Voting Ansätzen näher thematisiert: Hard Voting und Soft Voting.

2.3.2.1 Hard Voting

Hard Voting ist auch als Majority Voting bekannt und ist die beliebteste Voting Strategie [27], [28]. Jeder Klassifikator h_i des Ensembles stimmt unabhängig von den anderen für ein konkretes Klassenlabel c_j . Die finale Ausgabe des Ensembles $H(x)$ ist dann die Klasse c_j , die mehr als die Hälfte der Stimmen bekommt. Wenn keine Klasse die absolute Mehrheit erlangt, kommt es zur Verwerfung und das Ensemble macht keine Ausgabe. Die finale Ausgabe des Klassenlabels lässt sich wie folgt berechnen [27]:

$$H(x) = \begin{cases} c_j & \text{wenn } \sum_{i=1}^T h_i^j(x) > \frac{1}{2} \sum_{k=1}^l \sum_{i=1}^T h_i^k(x), \\ \text{verwerfen} & \text{sonst} \end{cases}$$

Hard Voting kann auch dann angewendet werden, wenn Klassenwahrscheinlichkeiten, statt einem eindeutigen Klassenlabel, ausgegeben werden. Hierbei wird dann für die Klasse c_j gestimmt, die die höchste A-posteriori-Wahrscheinlichkeit $P(c_j|x)$ für eine Eingabe x hat. Das heißt $h_i^j(x) = 1$ wenn $h_i^j(x) = \max_j \{h_i^j(x)\}$ [27].

2.3.2.2 Soft Voting

Werden von einem Klassifikator Klassenwahrscheinlichkeiten statt einem eindeutigen Klassenlabel ausgegeben, wird üblicherweise Soft Voting verwendet. Jeder einzelne Klassifikator gibt hier einen l -dimensionalen Vektor $(h_i^1(x), \dots, h_i^l(x))^{\top}$ für eine Eingabe x aus [27].

Es können zwei Soft Voting Ansätze unterschieden werden: Simple Soft Voting und Weighted Soft Voting.

Beim Simple Soft Voting werden alle Klassifikatoren h_i des Ensembles gleich behandelt. Dadurch ergibt sich die kombinierte Ausgabe als der Durchschnitt über jede einzelne Ausgabe. Formal ist die finale Ausgabe für eine Klasse c_j [27]:

$$H^j(x) = \frac{1}{T} \sum_{i=1}^T h_i^j(x).$$

Statt jeden Klassifikator gleich zu behandeln, können die einzelnen Ausgaben auch gewichtet werden. In diesem Fall handelt es sich dann um Weighted Soft Voting. Hier können zwei häufig in der Praxis verwendeten Fälle unterschieden werden [27]:

1. Klassifikatorspezifische Gewichte: Jeder Klassifikator h_i erhält ein Gewicht w_i . Die kombinierte Ausgabe für eine Klasse c_j ist

$$H^j(x) = \sum_{i=1}^T w_i h_i^j(x).$$

2. Klassenspezifische Gewichte: Jeder Klassifikator h_i erhält für jede Klasse c_j ein Gewicht w_i^j . Die kombinierte Ausgabe für eine Klasse c_j ist

$$H^j(x) = \sum_{i=1}^T w_i^j h_i^j(x).$$

Auch beim Soft Voting kann am Ende ein konkretes Klassenlabel c_j ausgegeben werden. Dies wäre die Klasse c_j mit der größten durchschnittlichen Wahrscheinlichkeit $H^j(x)$ [28]. Das heißt $H(x) = \max_j\{H^j(x)\}$.

2.3.2.3 Beispiel: Hard vs. Soft Voting

Das folgende Klassifizierungsbeispiel soll die Unterschiede zwischen Hard und Soft Voting verdeutlichen, wobei lediglich auf Simple Soft Voting eingegangen wird. Gegeben seien drei verschiedene Klassifikatoren h_1, h_2 und h_3 und drei verschiedene Klassenlabels c_1, c_2 und c_3 . In [Tabelle 2](#) stehen für jeden Klassifikator h_i und jede Klasse c_j die Klassenwahrscheinlichkeiten $h_i^j(x) \in [0,1]$.

	c_1	c_2	c_3	Hard Voting
h_1	0,2	0,3	0,5	c_3
h_2	0,4	0,6	0	c_2
h_3	0,3	0,3	0,4	c_3
Simple Soft Voting	0,3	0,4	0,3	

Tabelle 2: Beispielhafte Klassenwahrscheinlichkeiten h_i^j der Base Learner und Voting Ergebnis. Die finale Ausgabe von Hard Voting ist c_3 und von Simple Soft Voting c_2 .

Für die gegebenen Klassenwahrscheinlichkeiten wird einmal Hard Voting und einmal Simple Soft Voting angewendet. Hier muss für das Hard Voting eine kleine Anpassung gemacht werden, wobei $h_i^j(x) = 1$ wenn $h_i^j(x) = \max_j\{h_i^j(x)\}$. Dadurch ergibt sich für $h_1^3(x) = h_2^2(x) = h_3^3 = 1$ und für den Rest 0. Mit Hard Voting ist die finale Ausgabe c_3 , da zwei von drei Klassifikatoren die Klasse c_3 vorhersagen. Mit Soft Voting ist die finale Vorhersage die Klasse c_2 .

2.3.3 Stacking

Beim Stacking geht es darum, einen Meta Learner für das Kombinieren der Base Learner zu trainieren [\[27\]](#). Dies kann in mehreren Lernstufen erfolgen, wobei der Fokus dieser Arbeit lediglich auf zwei Lernstufen liegt. Die Grundidee des Stackings mit zwei Lernstufen und drei Base Learnern ist in [Abbildung 8](#) abgebildet und soll anhand dieser erläutert werden. Damit das Meta Modell am Ende auf ungesiehene Daten evaluiert werden kann, werden die Originaldaten in drei Datensätze aufgeteilt [\[28\]](#), [\[31\]](#):

1. Trainingsdaten (*train set*)
2. Validierungsdaten (*validation set*)
3. Testdaten (*test set*)

Jeder Base Learner (1-3), oder auch *First-Level Learner* [27], wird auf den Trainingsdaten trainiert [28]. Mit Hilfe der Validierungsdaten sollen die Metadaten, welche die Trainingsdaten für den Meta Learner sind, erstellt werden. Hierfür machen die Base Learner 1 bis 3 unabhängig voneinander ihre Vorhersagen P1, P2 und P3 für die Validierungsdaten. Die Vorhersagen P1, P2 und P3 werden anschließend gestapelt (*stacked*) und bilden die Metadaten [31]. Diese Daten werden schließlich verwendet um den Meta Learner, auch *Second-Level Learner* [27], zu trainieren [28]. In der Regel wird für den Meta Learner ein einfacher Algorithmus des maschinellen Lernens verwendet, um eine Überanpassung (*overfitting*) zu vermeiden. Für Klassifizierungsprobleme können hierfür beispielsweise die logistische Regression (*logistic regression*) oder ein Random Forest verwendet werden. Um das Meta Modell abschließend zu evaluieren, werden die bisher ungesehenen Testdaten verwendet. Hierfür macht der Meta Learner seine Vorhersagen P für die Testdaten. Da der Meta Learner auf die Base Learner gestapelt (*stacked*) wird, wird dieses Verfahren Stacking genannt.

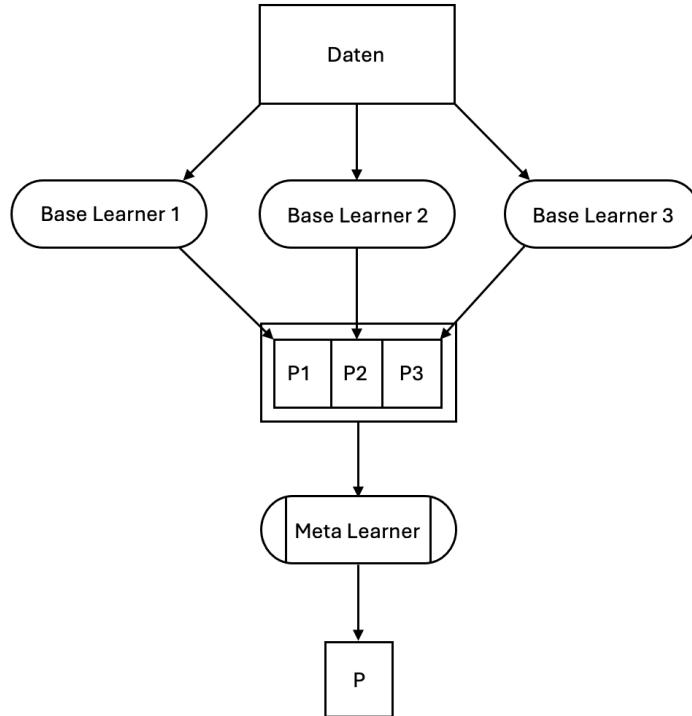


Abbildung 8: Grundgedanke von Stacking in Anlehnung an [28].

3 Daten

In dieser Arbeit geht es allgemein zum einen um die Generierung und zum anderen um die Klassifizierung von Bildern. Für beide Aufgaben werden Daten benötigt, die die entsprechenden Modelle (Generative Adversarial Networks und Convolutional Neural Networks) des maschinellen Lernens als Eingabe erhalten.

Bei dem Datensatz, der in dieser Arbeit verwendet wird, handelt es sich um Bilder von Münzen aus der griechischen Antike, die sich von der Archaik (800 v. Chr. bis 500 v. Chr.), über die Klassik (480 v. Chr. bis 336 v. Chr.), bis zum Hellenismus (336 v. Chr. bis 30 v. Chr.) erstreckt, und der römischen Kaiserzeit (27 v. Chr. bis 284 n. Chr.).

Die Daten stammen aus der Corpus Nummorum (CN) Datenbank. CN ist ein Gemeinschaftsprojekt des Münzkabinetts der staatlichen Museen zu Berlin, der Berlin-Brandenburgischen Akademie der Wissenschaften und der Goethe-Universität Frankfurt am Main [32]. Das D4N4 (*Data quality for Numismatics based on Natural language processing and Neural Networks*) Projekt, mit dem sich Sebastian Gampe und Dr. Karsten Tolle seit 2021 beschäftigen, ist das Nachfolgeprojekt des CN Projekts [6]. Im Rahmen des D4N4 Projektes hat es im Sommersemester 2023 eine Data Challenge an der Goethe-Universität Frankfurt am Main gegeben. Bei dieser wurden den Studierenden die Schwierigkeiten bzw. Herausforderungen der Klassifizierung von Münzstätten und Münztypen gezeigt.

Im Rahmen dieser Arbeit liegt der Fokus auf den Münztypen. Ein Münztyp ist durch die Kombination der Motive auf der Vorder- und Rückseite definiert. Mehrere Typen können auch dieselbe Kombination an Motiven besitzen, welche sich nur in kleinen Feinheiten auf den Motiven unterscheiden. Insgesamt besteht dieser Datensatz, der auf dem zur Verfügung gestellten Rechner bereits vorliegt, aus 12087 verschiedenen Typen. Jeder dieser Typen enthält zwischen einer und 549 Münzen, wodurch der Datensatz sehr unbalanciert ist. Für die Münztypen gibt es einen Datensatz (Ordner: *all_coins*), der für jede Münze ein konkateniertes Bild aus Vorder- und Rückseite enthält. Dieser Datensatz wird im weiteren Verlauf dieser Arbeit kopiert und als *Both* bezeichnet. Weiterhin gibt es für die Münztypen einen Datensatz (Ordner: *all_coins_no_concat*), der

für die Vorder- und Rückseite jeder Münze separate Bilder hat. Dieser Datensatz wird kopiert und in zwei Ordner aufgeteilt. Der eine Ordner erhält nur die Vorderseite und wird im Weiteren *Obv* genannt und der zweite Ordner besteht nur aus der Rückseite und wird als *Rev* angegeben. Um einen Machine Learning Algorithmus auf die Klassifizierung von Münztypen zu trainieren und evaluieren zu können, werden mehrere Daten für einen Typen benötigt. Wird die Grenze auf nur 20 Münzen pro Typ gesetzt, erfüllen lediglich 207 Typen diese Bedingung. Das heißt, dass 11880 Typen, was ca. 98,3% des Datensatzes entspricht, nicht für das maschinelle Lernen geeignet sind. Daher soll im Rahmen dieser Arbeit getestet werden, ob mit Hilfe von generativer künstlicher Intelligenz der Datensatz erweitert werden kann und dadurch die Vorhersageleistung für die Klassifizierung verbessert werden kann.

4 Generierung von antiken Münzbildern mittels GANs

Der Datensatz der griechischen und römischen Münzen besitzt über 12000 Typen, von diesen Typen haben ca. 98,3% unter 20 Bilder. Ein Ziel dieser Arbeit ist die Erweiterung des Datensatzes mit generierten Bildern und die Überprüfung dessen Auswirkungen auf die Klassifizierung der Münztypen durch Machine Learning Algorithmen. Der Fokus liegt auf den Typen, die zwischen 5 und 20 Bilder besitzen. Für die Generierung der Bilder soll das Few-Shot GAN (FSGAN) von Robb et. al [8] verwendet werden, welches sich dadurch auszeichnet, mit einer geringen Anzahl an Trainingsbildern (in der Regel weniger als 100) hochwertige Bilder zu generieren. Dies wird durch das Anpassen der Gewichte eines vortrainierten StyleGAN2 mit Hilfe der Singulärwertzerlegung erreicht (vgl. [Unterabschnitt 2.2.3](#)). Da es sich beim FSGAN um unüberwachtes Lernen handelt (keine Label), können nur Bilder für die einzelnen Typen (Label) generiert werden, wenn für jeden Typen ein eigenes Netzwerk trainiert wird. Die Arbeit von Jannick Hock [33] hatte sich bereits mit der Generierung von Münzdaten mittels verschiedener GANs beschäftigt. In seiner Arbeit wurde auch das FSGAN getestet, jedoch wurde es nur sehr kurz behandelt und nicht ausführlich getestet. Deshalb beschäftigt sich diese Arbeit nochmals ausführlicher mit dem FSGAN.

4.1 Frameworks und Bibliotheken

Für die Arbeit wurde ein Rechner mit Ubuntu 22.10 (GNU/Linux 5.19.0-46-generic x86_64) und einer RTX 3090 mit 24GB Arbeitsspeicher zur Verfügung gestellt. Alle nötigen Komponenten des Few-Shot GANs können aus der dazugehörigen GitHub-Bibliothek [34] heruntergeladen werden. Der FSGAN benutzt den StyleGAN2 von NvidiaLabs als Grundlage, sodass nur die Python-Bibliotheken benötigt werden, welche in der GitHub-Bibliothek des StyleGAN2 [35] unter *Requirements* zu finden sind. Mehrere Versuche eine Anaconda-Umgebung für den FSGAN einzurichten sind fehlgeschlagen. Der StyleGAN2 erfordert die Tensorflow Version 1.14 oder 1.15, welche zum Zeitpunkt

der Arbeit, aufgrund der neueren CUDA Versionen der Grafikkarte, nicht mit einer Nvidia RTX 3090 kompatibel waren. Die Einrichtung einer Anaconda-Umgebung mit einer älteren CUDA-Version hat wegen den abhängigen GCC-Bibliotheken nicht funktioniert. Mehrere Versuche eine ältere Version der GCC-Bibliothek zu installieren und die Abhängigkeiten zu lösen, waren ebenfalls erfolglos.

Nach ausgiebiger Recherche wurde eine Lösung mit Hilfe eines Docker-Containers gefunden. Für die Arbeit wird das Docker-Image `nvidia-tf-20.11-tf1` [36] verwendet, welches von Nvidia entwickelt wurde, um die Kompatibilität zwischen den Grafikkarten der Ampere-Architektur und den älteren Tensorflow-Versionen sicherzustellen.

4.2 Vorbereitung der Trainingsdatensätze

Die Trainingsdatensätze wurden manuell erstellt, um sicherzustellen, dass möglichst gute Bilder verwendet werden. Die Bilder enthalten teilweise in der unteren rechten Ecke ein Wasserzeichen, welches vor dem Training entfernt wurde. Dieses Wasserzeichen ist für die spätere Klassifizierung nicht relevant und sollte nicht von den FSGANs gelernt werden. Weitere Anpassungen an den Bildern waren vorerst nicht nötig. Die erstellten Trainingsdatensätze mussten als letzten Schritt in `tf-records` umgewandelt werden. Die Umwandlung erfolgt durch das Skript `dataset_tool.py` des FSGAN.

4.3 Methodik des FSGAN-Trainings

Für das Training können verschiedene Konfigurationen gewählt werden, die unterschiedliche GAN-Modelle repräsentieren. Da in der Arbeit nur der FSGAN im Fokus liegt, wurde für alle Trainings die Konfiguration `config-ada-sv-flat` festgelegt, welche die Konfiguration des FSGANs ist [34]. Für jede Seite der Münze wurde ein separater FSGAN trainiert. Dies lag daran, dass auf den Bildern mit beiden Seiten sowohl oben als auch unten größere schwarze Abschnitte sind.



Abbildung 9: Beispiel Münzbild mit beiden Seiten.

Diese könnten das Ergebnis verfälschen und haben keine Relevanz für die Generierung der Münzbilder. Des Weiteren sind auf den Einzelbildern die Merkmale größer, was die Qualität der generierten Bildern verbessern könnte.

Das FSGAN kann auf verschiedenen Bildauflösungen trainiert werden. Die Trainingsdauer für die Auflösung 1024x1024 beträgt etwa 17 bis 19 Stunden pro Typ pro Seite. Durch die Verkleinerung der Auflösung kann die Trainingsdauer deutlich verkürzt werden, wodurch mehrere Trainings in kürzerer Zeit ermöglicht werden. Aufgrund der zeitlichen Begrenzung dieser Arbeit wurde die Auflösung 256x256 gewählt, da hier die Trainingsdauer ca. drei bis vier Stunden beträgt. So konnte sichergestellt werden, dass möglichst viele FSGANs mit unterschiedlichen Motiven trainiert werden können. Ein weiter Grund ist, dass die später verwendeten Base Learner für das Training Bilder mit einer Auflösung von 224x224 benötigen und somit bietet die höhere Auflösung keinen großen Vorteil in Bezug auf die finale Anwendung.

Da das FSGAN keine Bilder mit Label generieren kann, muss für jeden Münztyp und jede Seite der Münze ein eigenes FSGAN trainiert werden. Für die Verlustfunktion und die Lernrate wurden die Standardwerte gewählt, da das Fine-Tuning dieser Parameter viele Trainingsvorgänge erfordert und nicht sicher ist, ob alle trainierten FSGANs mit denselben Hyperparametern gute Ergebnisse erzielen. Der Fokus dieser Arbeit liegt darin, das FSGAN auf möglichst viele verschiedene Münztypen zu testen und nicht ein optimales FSGAN für einen Münztyp zu finden.

Für die Auswertung des FSGANs wird die manuelle Evaluation genutzt. Diese ist zwar zeitaufwendiger als beispielsweise eine quantitative Evaluation mittels der Fréchet In-

ception Distance (FID), allerdings ist sie auch die gängigste Methode zur Beurteilung der Bildqualität. Die manuelle Evaluation bietet den Vorteil, dass sie sehr einfach und intuitiv ist. Da außerdem für jeden Typ und jede Seite ein eigenes Netzwerk trainiert wird, können die generierten Bilder für die einzelnen Münztypen und Seiten einfach mit dem entsprechenden Original verglichen werden. So können Details, wie beispielsweise Schriftzüge, feine Linien etc., genauer untersucht werden. Des Weiteren werden Unstimmigkeiten oder Artefakte in den generierten Bildern einfacher erfasst und potenzielle Fehlerquellen identifiziert und behoben. Insgesamt bietet die manuelle Evaluation einen detaillierteren Einblick in die Qualität und Generierungsleistung des Modells.

4.4 Far-Domain Adaptation

Das Few-Shot GAN benötigt für das Training ein vortrainiertes StyleGAN2. Da jedoch kein solches Modell auf einem Münzdatensatz zur Verfügung stand, wurde zunächst die Far-Domain Adaptation als Lösungsmöglichkeit in Betracht gezogen. Bei der Far-Domain Adaptation stammen die Bilder der Ursprungsdomäne nicht aus demselben Spektrum wie die Bilder der Zieldomäne. Die Far-Domain Adaptation zielt darauf ab, die Merkmale der Ursprungsdomäne so anzupassen, dass sie der Zieldomäne möglichst nahekommen.

Beim Durchgehen des Datensatzes ist aufgefallen, dass ein Großteil der Münzen Porträts als Motiv besitzen. Der FFHQ-StyleGAN2 ist auf Gesichter von bekannten Personen trainiert. Da diese der Zieldomäne am nächsten sind, wurde dieser als Ursprungsdomäne gewählt. Um ein breites Spektrum abzudecken, wurden für die ersten Tests Münzen mit unterschiedlichen Motiven gewählt. Für jeden Münztyp und jede Seite des Typs wurde ein eigenes FSGAN trainiert, wobei die Trainingsdauer von 100kimg festgelegt wurde. Wie in Abbildung 10 zu sehen ist, waren die Ergebnisse nicht zufriedenstellend. Teilweise sind Ansätze der Münzmotive erkennbar, wie auf der Vorder- und Rückseite der Münze des Typs 19775, jedoch wurden beispielsweise die Motive der Münze des Typs 946 gar nicht dargestellt, obwohl die Vorderseite ein weniger komplexeres Motiv aufweist und auch deutlich mehr Bilder im Trainingsdatensatz hatte.



Abbildung 10: Beispielbilder diverser trainierter FSGANs mit dem FFHQ StyleGAN als Basis.
Jeweils 100kimg Trainingszeit.

Eine Verlängerung der Trainingsdauer hat ebenfalls kein besseres Ergebnis herbeigeführt, sondern verschlechterte das Ergebnis teilweise sogar. Die Wahl einer anderen Ursprungsdomäne, wie z.B. des Church-StyleGANs, welcher auf Bildern von Kirchen und Gebäude trainiert ist, hat ebenfalls keine Verbesserung erbracht. Daher wurde der Ansatz der Far-Domain Adaptation verworfen und sich mit der Near-Domain Adaptation befasst.

4.5 Near-Domain Adaptation

Im Gegensatz zur Far-Domain Adaptation stammen bei der Near-Domain Adaptation die Bilder der Ursprungs- und Zieldomäne aus demselben Spektrum, wobei sich die Domänen hauptsächlich in feineren Merkmalen unterscheiden. Daher wurden als Ursprungsdomäne alle zur Verfügung stehenden Bilder der Vorder- und Rückseite der

Münzen gewählt. Ein gesonderter GAN für die Vorder- und Rückseite wäre zu zeit- aufwendig gewesen und durch das Zusammenführen der beiden Datensätze konnte die Menge an Trainingsdaten maximiert werden, welches für das Training von dem StyleGAN2 einen großen Vorteil erbringt, da dieser auf großen Datensätzen trainiert werden sollte. Ziel war es, ein StyleGAN2 als Grundlage zu haben, welcher Bilder von Münzen mit plausiblen Motiven generieren kann, ohne den Anspruch auf Perfektion.

4.5.1 Training eines StyleGAN2

Das Training des StyleGAN2 konnte im gleichen Docker-Image, wie das Training der Few-Shot-GANs durchgeführt werden. Es musste nur die dazugehörige Github-Bibliothek [35] heruntergeladen werden. Eine weitere Installation von Bibliotheken oder Pakten war nicht nötig. Die ca. 85000 Trainingsbilder wurden nicht vorverarbeitet und mussten wie beim FSGAN nur in `tf-records` umgewandelt werden. Bei der Wahl der Hyperparameter für das Training wurden alle Standardwerte benutzt und die Trainingsdauer auf 2800kimg festgelegt. Aufgrund der langen Trainingsdauer von ca. drei Tagen wurde kein Fine-Tuning der Hyperparameter durchgeführt. Dies war aufgrund der begrenzten Zeit nicht möglich und da das StyleGAN2 nur als Basis für die FSGANs dient, musste es somit auch nicht perfektioniert werden. Im Gegensatz zu den FSGANs wurde bei dem StyleGAN2 die quantitative Evaluationsmethode mittels des FID-Werts gewählt, da die generierten Bilder keine Labels besitzen und sich die ursprünglichen Münzentypen bei weit über 10000 Typen schwer ermitteln lassen. In Abbildung 11 ist der FID-Wert während des Trainings abgebildet. Es wird deutlich, dass dieser sich gegen Ende des Trainings auf ca. zehn einpendelt. In der Arbeit von Jannick Hock [33] wurde ebenfalls ein StyleGAN2 trainiert, welcher einen FID-Wert von 8,2 erreichte. Daher wird das Ergebnis des trainierten StyleGAN2 als ausreichend gut angesehen.

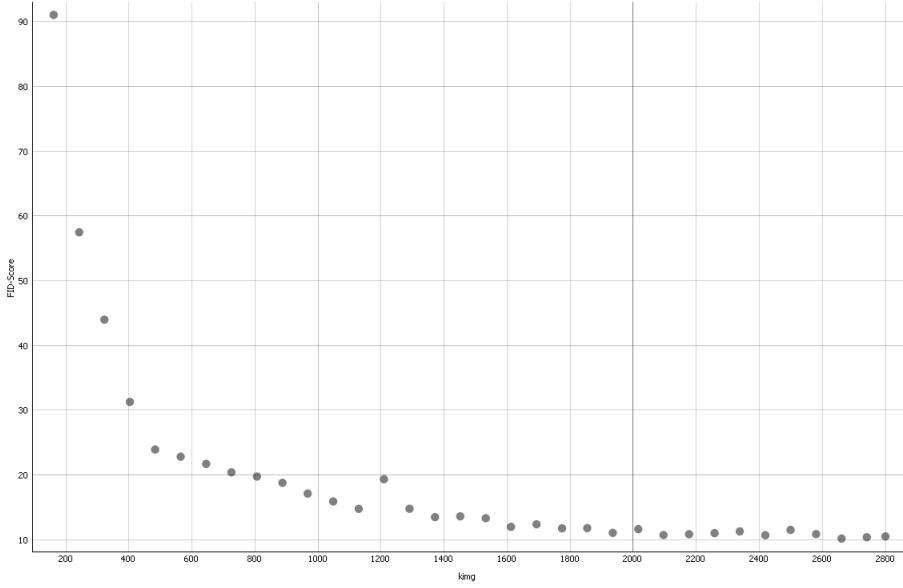


Abbildung 11: Trainingsverlauf des StyleGAN2. Kleine Werte sind besser.

Die generierten Bilder des StyleGAN2 nach 2800kimg wurden noch einmal überprüft, ob diese auch Münzen darstellen und Motive auf den Münzen zu erkennen sind. Es sind erkennbare Münzmotive vorhanden. Allerdings gibt es auch viele generierte Münzen auf denen kein richtiges Motiv zu erkennen ist. In Abbildung 51 im Anhang sind ein paar Beispiele zu sehen.

4.5.2 Training und Auswertung der FSGANs

Für die Vergleichbarkeit der Far- und Near-Domain Adaptation wurden die selben Trainings mit dem neu trainierten StyleGAN2 als Basis durchgeführt. Es ist eine deutliche Steigerung bei der Bildqualität zu sehen. Die Vorderseite des Münztyp 19775 (Abbildung 12a) entspricht der Orginalmünze, sodass diese von einem Laien als echte antike Münze angesehen werden könnte. Bei der Rückseite sind jedoch nur Ansätze erkennbar und viele Feinheiten im Motiv fehlen. Die größte Qualitätssteigerung hatte der Münztyp 946, bei der auf der Vorderseite nun das Motiv erkennbar ist und die Rückseite ebenfalls gute Ansätze zeigt.



(a) Typ 19775 Vorderseite Generiertes Bild. 5 Bilder im Trainingsdatensatz. Ergebnis nach 30 kimg	(b) Typ 19775 Rückseite generiertes Bild. 5 Bilder im Trai- ningsdatensatz. Ergebnis nach 50 kimg	(c) Typ 946 Vorderseite generiertes Bild. 5 Bilder im Trainingsdatensatz. Ergebnis nach 60 kimg	(d) Typ 946 Rückseite generiertes Bild. 5 Bilder im Trai- ningsdatensatz. Ergebnis nach 40 kimg
---	---	---	--

Abbildung 12: Beispiel Near-Domain Adaptation. Gleiche Münztypen wie in Abbildung [10](#).

Bei der Auswertung weiterer Ergebnisse wurde schnell deutlich, dass die Porträts am besten dargestellt werden konnten. Dies lässt sich sehr wahrscheinlich darauf zurückführen, dass diese den Großteil der Motive ausmachen und diese Domäne auch einfacher gelernt bzw. adaptiert werden konnte. Dieser Bias im StyleGAN2 hat wahrscheinlich auch dazu geführt, dass die komplexeren Motive nur ansatzweise dargestellt wurden. Es könnte jedoch auch sein, dass die Resolution zu klein ist, sodass die Feinheiten nicht richtig erkannt werden. Ein gutes Beispiel ist die Münze des Typs 13321 in [Abbildung 13](#), welche die groben Strukturen des Motivs gut darstellen kann, jedoch feinere Strukturen nicht darstellt. Auf der Vorderseite sind die Blätter des Lorbeerkränzes nur teilweise dargestellt und der Zweig in der Mitte der Rückseite wurde gar nicht dargestellt.



Abbildung 13: Beispiel: Fehlende Feinheiten bei generierten Bildern.

Der Einfluss der Größe der Trainingsdatenätze wurde ebenfalls getestet und es stellte sich heraus, dass die Vergrößerung des Trainingsdatensatzes nicht unbedingt eine Verbesserung der Bildqualität hervorruft. Es wurde beispielsweise jeweils ein FSGAN mit 5, 10 und 15 Bildern für die Vorderseite der Münze vom Typ 20746 trainiert. In [Abbildung 14](#) ist zu sehen, dass die Vergrößerung auf 15 Münzen ein schlechteres Ergebnis liefert, als die FSGANs mit weniger Münzen im Trainingsdatensatz. Der Grund für dieses Ergebnis könnte sein, dass im Datensatz mit nur 5 Bildern die Münzen untereinander ähnlicher sind als in den beiden anderen Datensätzen. Ebenfalls sind durch die Erweiterung auch Münzen in schlechterer Qualität dazugekommen. Daher wird angenommen, dass es für das Training besser ist, weniger Bilder zu wählen, die dafür eine gute Qualität aufweisen und sich nicht zu sehr unterscheiden. Jedoch ist anzumerken, dass sich diese Erkenntnis nicht auf alle Motive beziehungsweise Typen übertragen lässt, da beispielsweise die Porträtmotive mit größeren Datensätzen kleine Verbesserungen aufwiesen.



Abbildung 14: Ergebnisse des Typ 20746 Vorderseite mit unterschiedlich großen Trainingsdatensätzen.

Des Weiteren fiel bei der Auswertung der Ergebnisse auf, dass es je nach Motiv einen deutlichen Unterschied für eine optimale Trainingsdauer gibt. Die Motive mit Porträts haben schon nach 20-50kimg gute Ergebnisse erzielt, hingegen brauchen komplexere Motive meist 30-80 kimg, um die Ansätze der Motive gut darzustellen. Die Trainingsdauer wurde auch stark vom Zustand der Münzen beeinflusst. Gut erhaltene Münzen erzielten meistens schneller gute Ergebnisse. Insgesamt lässt sich keine optimale Trainingsdauer festlegen, da das Ergebnis von vielen verschiedenen Faktoren abhängt.

4.5.3 Weitere Erkenntnisse

Im Zuge dieser Arbeit wurden über 100 verschiedene FSGANs trainiert. Dabei wurden viele Typen mit unterschiedlichen Münzmotiven verwendet und auch unterschiedliche Zusammenstellungen der Trainingsdatensätze in Bezug auf die Größe und den Anteil an Gipsabdrücken und Metallmünzen betrachtet. In diesem Abschnitt werden die größten Auffälligkeiten, die in den Ergebnissen zu erkennen waren, genauer evaluiert.

4.5.3.1 Goldmünzen

Das Generieren von Goldmünzen mit Hilfe des FSGANs erwies sich als besonders schwierig. Die generierten Bilder hatten oft einen starken Glanz auf den Motiven, welcher sehr wahrscheinlich durch Reflexion des Fotoblitzes auf den originalen Bildern entsteht. Auffällig ist, dass der Schimmer nur sehr deutlich bei generierten Bildern von

Typen mit komplexeren Motiven zu sehen ist. Das Motiv aus Abbildung 15h zeigt, dass der Glanz bei diesem Motiv kein großes Problem darstellt. Dies kann aber auch von der Zusammensetzung des Trainingsdatensatzes abhängen, da hier mehr Gips- als Goldmünzen im Datensatz vorhanden sind. Die Mehrzahl an Gipsabdrücken könnte eine Verbesserung hervorrufen und auch den geringeren Schimmer auf der Vorderseite der Münze im Vergleich zu den Vorderseiten 11286 und 11254 erklären.



Abbildung 15: Beispiel: Generierte Goldmünzen.

Ein Versuch, den Glanz auf den generierten Bildern zu reduzieren, war ein Training mit schwarz-weiß Bildern. Durch die Reduktion der Farbkanäle sollte das Rauschen durch die Reflexion des Fotoblitzes verringert werden. Jedoch hat dies keine Verbesserung hervorgerufen wie in Abbildung 16 zu sehen ist.

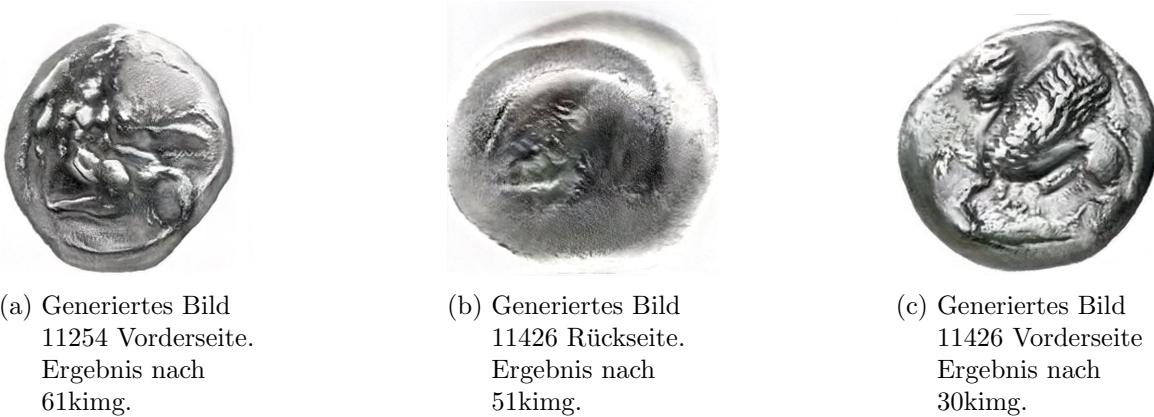


Abbildung 16: Beispiel: Generierte Goldmünzen in schwarz-weiß.

Des Weiteren wurde getestet, wie sich die Farbe des Hintergrunds auf die Qualität des Bildes und den Glanzeffekt auswirkt. Da die Farbe der Reflexionen sehr nah am weißen Hintergrund sind, wurde angenommen, dass Änderungen des Hintergrunds zu Schwarz oder Grau eine Verbesserung hervorrufen könnten. Der Hintergrund könnte deshalb nicht mit den hellen Stellen auf der Münze vermischt werden, sodass ein Effekt wie beim generierten Bild in [Abbildung 15f](#) verhindert wird. Die Resultate sind bei beiden Fällen deutlich schlechter. Zwar ist der Schimmer bei den Bildern mit schwarzem Hintergrund ein wenig zurückgegangen, jedoch hat die Qualität des Motivs deutlich abgenommen. Der graue Hintergrund konnte im Vergleich zum weißen auch keine deutliche Verbesserung hervorrufen. Zusätzlich wurden Münzbilder generiert, bei denen der Hintergrund mit der Münze verschmilzt oder das Motiv auf der Münze verschwimmt. Anders als zuvor angenommen, hat diese Änderung keine Verbesserung hervorgerufen und es wird empfohlen, Goldmünzen nicht mit einem schwarzen oder grauen Hintergrund zu generieren.



Abbildung 17: Generierte Goldmünzen mit schwarzem oder grauem Hintergrund.

Als letztes wurde versucht den Schimmer in den Trainingsbildern zu entfernen. Dafür wurde ein Python-Skript genutzt, welches die Bibliotheken OpenCV und scikit-image nutzt. Das Skript erkennt die hellen Stellen auf der Münze und ersetzt diese mit Stellen in der Farbe der umliegenden Pixel. Dies hat leider auch keine brauchbaren Ergebnisse erzielt, da bei der Entfernung des Glanzes zum einen die Stellen nur mattiert wurden, was zur Folge hatte, dass feine Strukturen auf den Münzen nicht mehr erkennbar waren und zum anderen ist der Hintergrund mit der Münze verschmolzen, weshalb dieser Ansatz verworfen wurde.



(a) Typ 1186 Vorderseite



(b) Typ 11254
Vorderseite



(c) Typ 11426 Rückseite

Abbildung 18: Beispiele für die Entfernung des Glanzes durch ein Python-Skript.

4.5.3.2 Training mit Gipsabdrücken und Metallmünzen

In einigen Trainings ist ein ähnliches Verschmelzen mit dem Hintergrund beziehungsweise ein Verschwimmen der generierten Münzmotive wie in Abbildung 16 aufgefallen. Bei näherer Betrachtung der Trainingsdatensätze der FSGANs fiel auf, dass diese oft sowohl Gipsabdrücke als auch Metallmünzen enthielten, wobei die Gipsabdrücke immer einen grauen Hintergrund haben. In Abbildung 19 sind einige Beispiele dieser Effekte zu sehen.



(a) Typ 6057
Vorderseite



(b) Typ 6057
Vorderseite



(c) Typ 6057 Rückseite



(d) Typ 6057
Rückseite

Abbildung 19: Beispiele von generierten Münzen, die mit dem Hintergrund verschmelzen.

Aus den Ergebnissen der generierten Goldmünzen liegt die Vermutung nahe, dass dies mit den verschiedenen Hintergrundfarben der Gipsabdrücke zusammenhängt. Daher wurden die Hintergründe der Gipsabdrücke weiß eingefärbt und eine neues Training gestartet. Dies führte zu einer Verbesserung bei den heller generierten Metallmünzen, jedoch haben sich die Ergebnisse der generierten Gipsabdrücke und der dunkleren Metallmünzen immer noch nicht verbessert.



(a) Typ 6057
Vorderseite

(b) Typ 6057
Vorderseite

(c) Typ 6057 Rückseite (d) Typ 6057
Rückseite

Abbildung 20: Beispiel generierte Münzen des Typs 6057. Hintergrund der Gipsabdrücke im Trainingsdatensatz weiß eingefärbt.

Aus diesem Grund wurden die Gipsabdrücke und Metallmünzen in zwei verschiedene Trainingsdatensätze geteilt, wobei der Hintergrund der Gipsabdrücke nicht zu Weiß geändert und zwei neue FSGANs trainiert wurden. Dies hatte eine deutliche Steigerung der Bildqualität zu Folge, wie in [Abbildung 21](#) sichtbar wird.



(a) Typ 6057
Vorderseite

(b) Typ 6057
Vorderseite

(c) Typ 6057 Rückseite (d) Typ 6057
Rückseite

Abbildung 21: Generierte Bilder des Typs 6057. Trainingsdatensätze bestehen rein aus Metallmünzen bzw. Gipsabdrücken.

Das Verschwimmen des Motivs der dunkleren Metallmünzen hat sich hingegen nur minimal verbessert. Bei der Auswertung weiterer FSGANs auf anderen Münztypen ist dieser Fehler immer wieder sichtbar geworden. Dunklere Münzen scheinen nicht optimal für das Training zu sein, da es scheint, als sei der Kontrast bei diesen Münzen nicht groß genug, sodass die Motive der Münze zusammenfließen. Es wurde ebenfalls festgestellt, dass die FSGANs bessere Ergebnisse erzielen, wenn die Farben der Münzen ähnlicher sind. Es gab bei Trainingsdatensätzen mit helleren Metallmünzen und Gipsabdrücken des Öfteren keine großen Probleme.

4.5.3.3 Generierte Schriften auf den Münzmotiven

Bei der Auswertung der Ergebnisse der FSGANs der Typen, welche Schriften auf den Motiven enthalten, fiel besonders auf, dass die Schriften auf den generierten Münzen nicht adäquat reproduziert werden können. Es sind teilweise schriftartige Artefakte auf den generierten Münzen vorhanden, jedoch entsprechen diese nie der Originalschrift wie in [Abbildung 22](#) zu sehen ist. Beim Trainingsdatensatz von Typ 11028 waren 3 Münzen bei denen die Schrift nicht lesbar beziehungsweise sehr abgetragen war. Man sieht, dass das FSGAN solche Eigenschaften auch lernt und diese gut darstellen kann. Die Qualität der Schrift konnte auch nicht durch die Anpassung der Trainingsdatensätze, wie z.B. die Veränderung der Anzahl an Münzen, Anpassung der Anzahl der Gipsabdrücke und Metallmünzen oder das Hinzufügen von Bildern in besserer oder schlechterer Qualität, beeinflusst werden.



Abbildung 22: Beispiele generierter Münzen mit Schriften.

Besonders auffällig ist das Ergebnis der Rückseite des Münztyps 294. Obwohl diese nur vier Schriftzeichen enthält, welche im Vergleich zu den anderen Münzmotiven relativ groß dargestellt sind, konnte der FSGAN diese überhaupt nicht darstellen, wie in Abbildung 23b zu sehen ist. Bei näherer Betrachtung der Bilder, welche im Trainingsdatensatz enthalten sind (s. Abbildung 52 im Anhang), ist aufgefallen, dass die Motive sehr unterschiedlich auf den Münzen platziert sind und die Ausrichtung der Schriftzeichen auf den Münzen nicht einheitlich ist. Um dies als Fehlerursache auszuschließen wurden die Bilder des Trainingsdatensatz rotiert, sodass alle Bilder die selbe Ausrichtung aufweisen. Dies führte zu einem besseren Ergebnis, da die Unterteilung des Motivs in vier Teile nun erkennbar ist, jedoch konnte die Schrift weiterhin nicht vollständig generiert werden (Abbildung 23b). Ein weiterer Ansatz zur Verbesserung der Bildqualität war das Ausschneiden der Münze, sodass so wenig Hintergrund wie möglich auf den Bildern zu sehen ist. Dies wurde sowohl für die rotierten als auch die Ursprungsbilder umgesetzt. Diese Anpassungen führten ebenfalls nicht zu einer Verbesserung der Schriftzeichen. Weitere Versuche, wie die Anpassung der Hintergrundfarbe zu Grau oder Schwarz, schlugen ebenfalls fehl.



Abbildung 23: Ergebnisse verschiedener Trainings der Rückseite der Münze 294.

Diese Ergebnisse sind sehr verwunderlich, da die kleinen Schriften auf den anderen Münzen wenigstens ansatzweise erkennbar sind und die Schriften auf der Münze vom Typ 294, im Vergleich zu den anderen Schriften, für den menschlichen Betrachter ein-

facher darstellbar erscheinen aufgrund ihrer Größe und Klarheit.

Es liegt nahe anzunehmen, dass die schlechten Ergebnisse auf die Motivpositionierung auf den Originalmünzen zurückzuführen sind. Bei der näheren Betrachtung der Originalmünzen des Typs 11286 (s. Abbildung 53 im Anhang) sind ebenfalls sehr verschobene Motivpositionierungen zu sehen. In beiden Fällen versucht das FSGAN die Münzmotive zentral zu platzieren und lernt nicht, die verschiedenen Motivpositionen zu berücksichtigen. Diese Beobachtung war bei weiteren FSGANs auffällig, was darauf hinweist, dass eine zu große Variation der Motivpositionierung sich negativ auf das Training der FSGANs auswirken kann.

4.6 Wahl und Training der FSGANs für das Ensemble Learning

In Kapitel 5 sollen die Auswirkungen von generierten Münzen auf das Training und die Performance eines Klassifizierers getestet und erörtert werden. Dafür sollten einmal kleine Typen mit weniger als 30 Bildern mit generierten Daten erweitert werden, um so den Datensatz mehr auszubalancieren. Als zweites sollte für die Typen, für die ein FSGAN trainiert wurde, die Datensätze auf die Anzahl der verwendeten Münzen während des Trainings der FSGANs reduziert und mit generierten Münzen aufgefüllt werden. Für die Wahl der Typen wurden die Typen, die mehr als 20 Bilder besitzen, betrachtet. Ein Münztyp wurde ausgewählt, wenn dieser genug gute Bilder für das Training eines FSGANs besitzt. Es wurde darauf geachtet, auch Typen mit verschiedenen Münzmotiven zu wählen. Für Goldmünzen konnte kein Typ gefunden werden, welcher mehr als 20 Bilder besitzt, daher wurden diese nicht in Betracht gezogen. In Tabelle 3 sind die ausgesuchten Typen mit der Anzahl der Münzen und das Verhältnis von Gipsabdrücken zu Metallmünzen im Trainingsdatensatz angegeben. Es wurde jeweils ein FSGAN für das Motiv der Vorder- und Rückseite trainiert. Die Trainingszeit aller FSGANs für diese Münztypen betrug etwa 100 Stunden.

Typ	294	940	946	5135	5944	6057	7697	12752	12783	20230	20545	20809	21027
Anzahl Trainingsbilder	10	10	10	5	10	10	5	5	5	5	15	20	30
Verteilung Metall/Gips	8/2	8/2	8/2	5/0	10/0	10/0 und 0/10	3/2	3/2	3/2	3/2	5/10	20/0	26/4
FSGAN Vorderseite	41kimg	61kimg	61kimg	41kimg	20kimg	Metall: 41kimg Gips: 61kimg	41kimg	61kimg	41kimg	61kimg	61kimg	80kimg	80kimg
FSGAN Rückseite	41kimg	61kimg	41kimg	81kimg	61kimg	Metall: 80kimg Gips 61kimg	80kimg	61kimg	81kimg	81kimg	61kimg	100kimg	41kimg

Tabelle 3: Übersicht des Trainings der 13 Typen. Die letzten beiden Zeilen geben an, welcher FSGAN für die Generierung der Bilder genutzt wurde. Für Münztyp 6057 wurden jeweils separater FSGANs für Gipsabdrücke und Metallmünzen trainiert.

Ein Vergleich zwischen den generierten Bildern und Originalbildern ist in [Abbildung 24](#) zu sehen. Der Typ 294 ist trotz der nicht allzu guten Qualität der generierten Bilder ausgewählt worden, da der Einfluss schlechter Bilder getestet werden sollte. Die Typen 940 und 946 wurden aufgrund ihrer Ähnlichkeit gewählt. Hier sollte ermittelt werden, ob durch die generierten Daten der Unterschied zwischen den Münzen gut dargestellt werden kann. Für Münztyp 6057 wurden getrennt FSGANs für Metallmünzen und Gipsabdrücke trainiert. Auf Wunsch von Sebastian Gampe sollten die Münztypen 21027 und 7697 in die Auswahl miteinbezogen werden. Der Münztyp 7697 besitzt hauptsächlich Bilder in schlechter Qualität. Bei den Münztypen soll festgestellt werden, ob es möglich ist, auch eine gute Performance mit schlechten Bildern zu erreichen. Die restlichen Münztypen wurden ausgewählt, da diese alle unterschiedliche Motive und auch genügend gute Bilder für das Training eines FSGANs besitzen.

4.7 Fazit

Die Ergebnisse der Arbeit haben gezeigt, dass es durchaus möglich ist, mit dem FSGAN gute Münzbilder zu generieren. Dies ist auch mit einer geringen Anzahl an Trainingsdaten möglich. Die Ergebnisse hängen dabei von mehreren Faktoren ab:

- Qualität und Zustand der Münzen: Es konnten bessere und schnellere Ergebnisse mit gut erhaltenen Münzen erzielt werden. Es sind auch gute Ergebnisse mit nicht optimalen Bildern möglich, allerdings fehlen diesen oft Feinheiten oder die Motive sind nur teilweise dargestellt.

- Motivpositionierungen: Zu unterschiedliche Positionen auf den Motiven oder nur halb vorhandene Motive, können das Ergebnis deutlich verschlechtern. Dies führt öfter zu einem verschwommenen Motiv auf den generierten Bildern.
- Material der Münzen: Das Material hat einen großen Einfluss auf das Training, genau wie das Verhältnis der verschiedenen Materialien im Trainingsdatensatz. Hier sollte darauf geachtet werden, möglichst nur Münzen desselben Materials zu verwenden. Jedoch können hellere Metallmünzen und Gipsabdrücke auch zusammen benutzt werden.
- Motivkomplexität: Einfache Motive werden durch das FSGAN viel besser dargestellt. Bei komplexeren Motiven fehlen oft die Feinheiten, welche das FSGAN nicht abbilden kann.
- Hintergrundfarbe: Die Hintergrundfarbe der Münzen sollte einheitlich sein. Eine Mischung aus verschiedenen Farben führt zu verschwommenen Münzmotiven oder einer Verschmelzung des Hintergrunds mit der Münze. Für Münzen aus Metall ist ein weißer Hintergrund am besten. Für Gipsabdrücke können sowohl weiße als auch graue Hintergründe verwendet werden.

Aus diesen Gründen lässt sich schwer eine Entscheidung treffen, ob sich das FSGAN für alle Münztypen verwenden lässt, welche weniger als 20 Bilder besitzen. Das Training ist auch sehr aufwendig, da geeignete Bilder für das Training ausgesucht werden müssen und für einen Münztypen die Mindesttrainingsdauer ca. 6 Stunden beträgt. Dies ist ein enormer Aufwand für die über 10000 Münztypen, welche weniger als 20 Bilder besitzen.



(a) Original 294



(b) Generiertes Bild 294.



(c) Original 940



(d) Generiertes Bild 940



(e) Original 946



(f) Generiertes Bild 946



(g) Original 5135



(h) Generiertes Bild 5135



(i) Original 5944



(j) Generiertes Bild 5944

Abbildung 24: Beispiele der 13 Typen. Links Originalbilder. Rechts generierte Bilder. Die Bilder rechts sind zusammengesetzt aus einem generierten Bild der Vorderseite und einem generierten Bild der Rückseite.



(k) Original 6057



(l) Generiertes Bild 6057



(m) Original 7697



(n) Generiertes Bild 7697



(o) Original 12752



(p) Generiertes Bild 12752



(q) Original 12783



(r) Generiertes Bild 12783



(s) Original 20230



(t) Generiertes Bild 20230

Abbildung 24: Beispiele der 13 Typen. Links Originalbilder. Rechts generierte Bilder. Die Bilder rechts sind zusammengesetzt aus einem generierten Bild der Vorderseite und einem generierten Bild der Rückseite.



(u) Original 20545



(v) Generiertes Bild 20545



(w) Original 20809



(x) Generiertes Bild 20809



(y) Original 21207



(z) Generiertes Bild 21207

Abbildung 24: Beispiele der 13 Typen. Links Originalbilder. Rechts generierte Bilder. Die Bilder rechts sind zusammengesetzt aus einem generierten Bild der Vorderseite und einem generierten Bild der Rückseite.

5 Klassifizierung von Münztypen mittels Ensemble Learning

5.1 Training der Base Learner

In diesem Abschnitt wird sich mit dem Training der Base Learner befasst. Diese werden für das Ensemble Learning benötigt. Als Base Learner werden in dieser Arbeit CNNs verwendet. Unterabschnitt 5.1.1 befasst sich mit der Erstellung der Datensätze für das Training der CNNs. In Unterabschnitt 5.1.2 wird auf die Trainingsdurchführung eingegangen und in Unterabschnitt 5.1.4 werden die Trainingsergebnisse evaluiert.

5.1.1 Erstellung der Datensätze

Aus dem ganzen Münzdatensatz wurden zunächst alle Typen entfernt, welche weniger als 20 Bilder besitzen. Anschließend wurden drei neue Datensätze erstellt: ein Trainingsdatensatz, ein Validationsdatensatz und ein Testdatensatz. Diese wurden mit Hilfe eines Python-Skripts generiert, dass die vorhandenen Münzen wie folgt aufteilte: Der Testdatensatz und Validationsdatensatz erhalten pro Typ 10% der vorhandenen Bilder, wobei jedoch mindestens 3 Bilder pro Typ enthalten sein müssen. Die verbleibenden 80% der Bilder sind dem Trainingsdatensatz zugeordnet. Für die 13 Münztypen aus Abschnitt 4.6 sind die Bilder für den Testdatensatz per Hand ausgewählt worden. Dadurch wird sichergestellt, dass keine Bilder, welche auch für das Training eines FSGANs verwendet wurden, im Testdatensatz vorhanden sind. Dies muss sichergestellt werden, um einen möglichen Bias zu verhindern.

Um den Vergleich mit und ohne generierten Daten zu ermöglichen, wurden zwei weitere Trainingsdatensätze erstellt. Diese unterscheiden sich vom ursprünglichen Trainingsdatensatz, der nun als *Base* bezeichnet wird, wie folgt: Ein Trainingsdatensatz, nun als *TrainWithGenerated* bezeichnet, wurde mit generierten Daten für die 13 Typen erweitert, sodass für jeden Typ mindestens 50 Bilder vorhanden sind. Der dritte Trainingsdatensatz, nun als *TrainSubGenerated* bezeichnet, enthält lediglich die Originalbilder für die 13 Typen, welche im Trainingsdatensatz des entsprechenden FSGANs

vorhanden waren, und wurde zusätzlich mit generierten Daten aufgefüllt. Dabei wurde eine unterschiedliche Anzahl bei einigen Typen gewählt. Hierdurch soll eine mögliche Mindestanzahl an generierten Bildern ermittelt werden, welche für die Klassifizierung notwendig ist. Eine genaue Übersicht der zwei Trainingsdatensätze ist in [Tabelle 4](#) und in [Tabelle 5](#) zu sehen. Im *Base* und *TrainWithGenerated* ist eine Münze cn coin 41841 fälschlicherweise im Münztypen 5135 enthalten. Dieser Fehler ist erst sehr spät gegen Ende der Arbeit aufgefallen. Ein erneutes Training aller Netze wäre zu aufwendig gewesen. Diese eine falsche Münze sollte aber das Gesamtergebnis nicht stark verfälschen.

Typ	294	940	946	5135	5944	6057	7696	12752	12783	20230	20545	20809	21027
#Orginal	15	15	14	17	14	17	99	16	17	14	14	16	31
#Generiert	35	35	36	33	36	33	20	34	33	36	36	34	19
Summe	50	50	50	50	50	50	119	50	50	50	50	50	51

Tabelle 4: Übersicht der Anzahl an originalen und generierten Münzbildern im Trainingsdatensatz *TrainWithGenerated*.

Typ	294	940	946	5135	5944	6057	7696	12752	12783	20230	20545	20809	21027
#Orginal	10	10	10	5	5	10	5	5	5	5	14	16	30
#Generiert	40	30	20	40	40	40	35	30	25	20	36	34	20
Summe	50	40	30	50	45	50	40	35	30	25	50	50	50

Tabelle 5: Übersicht der Anzahl an originalen und generierten Münzbildern im Trainingsdatensatz *TrainSubGenerated*.

5.1.2 Implementation des Training der CNN

Für das Training der CNNs wurde wieder ein Docker verwendet. Das verwendete Docker Image ist `tensorflow:2.10.0-gpu` von Tensorflow[\[37\]](#). Dieses benutzt die Tensorflow Version 2.10. Aus vorangegangen Arbeiten wurden gute Erfahrungen mit der Fine-Tuning Trainingsmethode gemacht, daher wurde diese auch in dieser Arbeit verwendet. Beim Fine-Tuning wird ein vortrainiertes CNNs benutzt und die Gewichte des Netzes bis zu einem bestimmten Convolutional Layer eingefroren. Dadurch macht man sich die bereits gelernten Merkmale, welche der vortrainierte CNN erkennt, zu Nutze und lernt nur auf den nicht eingefrorenen convolutional Schichten neue Merkmale. Da in den er-

sten Schichten meist einfache Merkmale, wie Kanten und Ecken, erkannt werden, kann durch diese Methode die Trainingsdauer deutlich verkürzt werden, da diese Merkmale nicht neu gelernt werden müssen. Die vortrainierten Gewichte aller durchgeführten Trainings stammen von einem Training auf dem ImageNet Datensatz, welcher über 1000 verschiedene Klassen besitzt, mit über 14 Millionen Bildern. Die Gewichte werden von Tensorflow zur Verfügung gestellt. Als Optimizer wird ADAM genutzt. Die Trainingsdurchläufe werden zweistufig ausgeführt. In der ersten Stufe beträgt die Lernrate 0,001 und die Trainingsdauer beträgt 5 Epochen. In der zweiten Stufe wird die Lernrate auf 0,000001 gesenkt und für weitere 30 Epochen trainiert. Bei der zweiten Stufe wird außerdem die Callback Funktion von Tensorflow genutzt, welche während des Trainings einen vorgegebenen Wert überwacht und wenn dieser sich nach einer gewissen Anzahl an Epochen nicht um ein $\delta > 0$ ändert, wird das Training beendet. Bei allen Trainings wurde der Loss auf dem Validationdatensatz betrachtet und wenn dieser sich nicht um mindestens $\delta = 0,001$ innerhalb 5 Epochen ändert wird das Training beendet.

Die Hyperparameter wurden so gewählt, da im Rahmen der Data Challenge, in welcher Münzen nach Münzstätten klassifiziert wurden, diese auch genutzt wurden und sich sehr geeignet haben für das Training. Die ganze Implementation des Trainings ist im beigefügten USB-Stick enthalten.

5.1.3 Durchgeführte Trainings

Pro Trainingdatensatz sollen drei verschiedene Base Learner zur Verfügung stehen. Deshalb wurde als erster Schritt die verschiedenen CNN Architekturen auf dem Base Trainingsdatensatz getestet und ihre Fehlerrate ausgewertet. Die vier Architekturen, die getestet wurden, sind die ResNet50, ResNet101, ResNet152 und die VGG16 Architektur. Die VGG16 Architektur hat deutlich schlechtere Ergebnisse als die ResNet Architekturen erzielt und daher wurde diese Architektur nicht weiter als Base Learner in Betracht gezogen. Als nächstes wurde getestet wie sich die Augmentation der Trainingsdaten auf die Performance auswirkt. Da das Training eines CNNs nicht eindeutig ist und es immer einen gewissen Zufallsfaktor gibt, wurden für jede der drei ausgewählten Architekturen 10 Trainingsvorgänge durchgeführt und die Fehlerrate sowie das beste

Netz abgespeichert. Danach wurde die durchschnittliche Fehlerrate ermittelt, welche in [Tabelle 6](#) zu sehen ist. Dies wurde zur besseren Vergleichbarkeit der Performance gemacht. Es ist deutlich zu sehen, dass die Augmentation der Daten keinen Vorteil erbringt. Die maximale Fehlerrate der CNNs ohne Augmentation liegt bereits deutlich unter der minimalen Fehlerrate der CNNs mit Augmentation.

Model	Ohne Augmentation			Mit Augmentation		
	ResNet50	ResNet101	ResNet152	ResNet50	ResNet101	ResNet152
Fehlerrate - Mean	20,90%	21,35%	21,49%	33,23%	37,34%	35,77%
Fehlerrate - Min. Wert	18,85%	19,90%	20,00%	30,58%	35,38%	32,69%
Fehlerrate - Max. Wert	22,60%	22,98%	22,88%	36,63%	39,13%	40,19%
Fehlerrate - Standardabweichung	1,02%	0,86%	0,88%	1,74%	1,15%	1,96%

Tabelle 6: Training der Base Learner auf dem Trainingsdatensatz ohne generierte Daten. Evaluation auf dem Testdatensatz. Werte auf zwei Nachkommastellen gerundet. Kleinere Werte sind besser.

Beim Fine-Tuning kann ausgewählt werden, bis zu welcher Convolutional Schicht die Gewichte des CNNs eingefroren werden und somit nicht neu gelernt werden. Es wurden das Einfrieren der Gewichte bis zum vierten und fünften Convolutional Block der ResNet Architektur getestet. Aufgrund der beschränkten Zeit wurde dies nur für ResNet50 und ResNet101 getestet. Die Ergebnisse des ResNet50 in [Tabelle 7](#) zeigen eine bessere Fehlerrate beim einfrieren bis zur vierten Schicht, dies deckt sich auch mit den Ergebnissen des ResNet101.

Gewichte eingefroren bis Layer	Conv 4		Conv 5	
	Ohne Augmentation	Mit Augmentation	Ohne Augmentation	Mit Augmentation
Fehlerrate - Mean	20,90%	33,26%	25,05%	32,14%
Top1Error - Min. value	18,85%	30,58%	23,65%	30,58%
Top1Error - Max. value	22,60%	36,63%	26,83%	33,75%
Top1Error - Standard deviation	1,02%	1,74%	0,94%	0,94%

Tabelle 7: Training des ResNet50 auf dem Trainingsdatensatz ohne generierter Daten. Evaluation auf dem Testdatensatz. Werte auf zwei Nachkommastellen gerundet. Kleinere Werte sind besser.

Weitere Versuche die Fehlerrate zu verbessern, wurden nicht durchgeführt, da das Fine-Tuning von Hyperparameter viel Zeit in Anspruch nimmt und eine Fehlerrate von etwa

20% für die Base Learner als ausreichend angesehen wird. Als nächstes wurden weitere Base Learner für die Trainingsdatensätze *TrainWithGenerated* und *TrainSubGenerated* trainiert. Dabei wurden alle ohne Augmentation trainiert und die Gewichte wurden bis zum vierten Convolutional Block eingefroren. Die Trainings wurden wieder pro Architektur zehnmal wiederholt und jeweils das CNN gespeichert, welches auf dem gesamten Testdatensatz die kleinste Fehlerrate besitzt.

5.1.4 Evaluation der Base Learner

In diesem Abschnitt wird zuerst die Gesamtperformance der trainierten CNNs ausgewertet. [Absatz 5.1.4.1](#) und [Absatz 5.1.4.2](#) befassen sich detailliert mit den Ergebnissen der 13 Typen, für die Bilder generiert wurden.

Die Fehlerrate der verschiedenen Base Learner auf den gesamten Testdatensatz ist in [Tabelle 8](#) zu sehen. Die Unterschiede der Fehlerrate sind dabei sehr gering. Dies kann auf den sehr unausgeglichenen Testdatensatz zurückgeführt werden, da pro Typ 10% der vorhandenen Gesamtbilder für den Datensatz gewählt wurden. Der Typ 12559 hat dadurch 50 Bilder im Testdatensatz hingegen, besitzen die kleineren Typen teilweise nur 3 Bilder. Dadurch wird das Gesamtergebnis sehr von den größeren Typen verzerrt.

Model	ResNet50			ResNet101			ResNet152		
	Base	TrainSubGenerated	TrainWithGenerated	Base	TrainSubGenerated	TrainWithGenerated	Base	TrainSubGenerated	TrainWithGenerated
Fehlerrate - Mean	20,90%	22,05%	21,44%	21,35%	22,11%	21,82%	21,49%	22,56%	22,36%
Fehlerrate - Min.	18,85%	20,29%	19,81%	19,90%	20,48%	20,67%	20,00%	21,15%	21,35%
Fehlerrate - Max.	22,60%	25,10%	22,60%	22,98%	23,17%	22,98%	22,88%	23,56%	24,23%
Fehlerrate - σ	1,02%	1,58%	1,03%	0,86%	0,92%	0,77%	0,88%	0,79%	0,84%

Tabelle 8: Evaluation der Base Learner auf den gesamten Testdatensatz. Kleinere Werte sind besser.

Aus diesem Grund wurde die Fehlerrate auf einem weiteren Testdatensatz ermittelt, welcher nur Typen mit weniger als 100 Bildern enthält. Wie aus [Tabelle 9](#) erkenntlich wird, sind die Unterschiede ebenfalls noch nicht sehr groß. Die Performance scheint sich bei diesem Testdatensatz für die CNNs, welche mit generierten Daten erweitert wurden, nicht sonderlich verschlechtert zu haben. Beim Resnet101 hat sich die Performance sogar erhöht.

Model	ResNet50			ResNet101			ResNet152		
Trainingsdatensatz	Base	TrainSubGenerated	TrainWithGenerated	Base	TrainSubGenerated	TrainWithGenerated	Base	TrainSubGenerated	TrainWithGenerated
Fehlerrate	20,36%	21,27%	20,70%	21,38%	20,58%	22,17%	20,81%	21,95%	23,08%

Tabelle 9: Evaluation der Base Learner auf dem Datensatz mit weniger als 100 Bildern pro Typ. Es wurden nur noch die CNNs verwendet, welche die besten Ergebnisse in Tabelle 8 erzielt haben, da während des Trainings die Evaluation auf diesem Testdatensatz nicht durchgeführt wurde.

Als letztes wurden die CNNs auf einem Testdatensatz evaluiert, welcher nur die 13 Typen enthält. Hier ist ein deutlicher Unterschied in der Fehlerrate erkennbar. Die CNNs, welche mit dem Trainingsdatensatz *TrainWithGenerated* trainiert wurden, sind gleich auf mit den CNNs des *Base* Trainingsdatensatz. Das ResNet101, trainiert mit *TrainWithGenerated*, hat sogar die geringste Fehlerrate erzielt, was darauf hindeutet, dass die Erweiterung der Orginaldaten und damit das Ausgleichen der Anzahl an Bildern pro Typ im Trainingsdatensatz einen kleinen Vorteil erbringen kann. Man muss jedoch beachten, dass der Testdatensatz nur 53 Bilder enthält und ein Unterschied in der Fehlerrate von 1,8 Prozentpunkten nur bedeutet, dass ein Bild mehr richtig zugeordnet wurde.

Model	ResNet50			ResNet101			ResNet152		
TrainSet - Mode	Base	TrainSubGenerated	TrainWithGenerated	Base	TrainSubGenerated	TrainWithGenerated	Base	TrainSubGenerated	TrainWithGenerated
Fehlerrate - Mean	31,82%	57,14%	31,98%	34,42%	58,28%	30,19%	31,01%	52,76%	33,12%
Fehlerrate - Min.	26,79%	51,79%	26,79%	30,36%	48,21%	25,00%	26,79%	46,43%	28,57%
Fehlerrate - Max.	37,50%	64,29%	39,29%	39,29%	71,43%	33,93%	33,93%	58,93%	41,07%
Fehlerrate - σ	3,18%	3,91%	3,61%	2,66%	6,20%	3,43%	2,56%	4,10%	3,51%

Tabelle 10: Evaluation der Base Learner auf den Testdatensatz der 13 Typen, für die generierte Daten in den Trainingsdatensätze hinzugefügt wurden.

Die CNNs des Trainingsdatensatz *TrainSubGenerated* haben deutlich schlechter abgeschnitten. Daraus lässt sich vermuten, dass die Erweiterung von weniger Originalbildern mit vielen generierten Bildern, keinen Vorteil erbringt. Es könnte aber auch an der zum Teil schlechten Qualität der generierten Daten liegen.

5.1.4.1 Evaluation der CNNs trainiert mit TrainWithGenerated

In diesem Abschnitt wird genauer auf die Performance der CNNs eingegangen, welche mit generierten Daten erweitert wurden. Für die Auswertung wird die Sensitivität (*recall*) auf den einzelnen Münztypen errechnet. Diese gibt an, wie viele Bilder des Typen

richtig zugeordnet wurden. In Tabelle 11 ist die Sensitivität für jeden Base Learner und für jeden Typen zu sehen.

Typ	294	940	946	5135	5944	6057	7697	12752	12783	20230	20545	20809	21027
ResNet50 Base	0,0	0,6667	1,0	0,6667	1,0	1,0	1,0	0,3333	1,0	1,0	0,6667	0,75	0,4
ResNet101 Base	0,6667	0,6667	0,6667	0,3333	1,0	1,0	0,9	0,6667	0,6	0,6667	0,3333	1,0	0,4
ResNet152 Base	0,0	1,0	1,0	0,6667	1,0	0,3333	0,9	0,6667	1,0	0,6667	0,6667	0,5	0,3
ResNet50 TrainWithGenerated	0,3333	0,3333	1,0	0,6667	1,0	0,6667	0,9	0,3333	1,0	0,6667	0,6667	1,0	0,2
ResNet101 TrainWithGenerated	0,0	0,3333	1,0	1,0	1,0	1,0	0,9	0,3333	1,0	1,0	0,6667	1,0	0,3
ResNet152 TrainWithGenerated	0,0	1,0	0,6667	1,0	1,0	0,6667	1,0	0,3333	0,8	1,0	0,6667	0,75	0,2
#Bilder im Testdatensatz	3	3	3	3	3	3	10	3	5	3	3	4	10

Tabelle 11: Sensitivität der 13 Typen auf dem Base und *TrainWithGenerated* Datensatz. Höhere Werte sind besser.

Als nächstes wurden die falschen Zuteilungen der Münzen der einzelnen CNNs genauer betrachtet, sodass die Fehler von den CNNs deutlich werden. Der Hauptfokus wird auf die Münzytypen gelegt, welche viele Fehler in der Zuteilung haben. In Abbildung 54 im Anhang sind alle Konfusionsmatrizen auf dem *Test13* Datensatz abgebildet.

Die Münzen des Typs 294 wurden am häufigsten dem Typen 285 falsch zugeordnet. Vergleicht man zwei Münzen der Typen nebeneinander, sieht man, dass diese fast identisch aussehen. Die einzigen Unterschiede sind auf der Vorderseite zu sehen, da der Helm bei der Münze des Typs 294 unten ein bisschen breiter ist. Dieser Unterschied kann durch die CNNs anscheinend nicht richtig erkannt werden.



Abbildung 25: Vergleich Münzen des Typs 285 und 294.

Beim Vergleich der Trainingsdaten für diese beiden Münztypen fällt auf, dass beispielsweise für den Trainingsdatensatz *Base* viel mehr Bilder für den Münztyp 285 als für 295 vorhanden sind. Der Münztyp 285 hat insgesamt 47 Bilder im Trainingsdatensatz und

der Münztyp 294 nur 15. Des Weiteren ist der Anteil an Gipsabdrücken im Datensatz von 285 deutlich höher. Dieser hat insgesamt 18 Stück, hingegen hat der Typ 294 nur eine Einzige. Dies könnte ein Grund sein, weshalb die Gipsabdrücke im Testdatensatz am häufigsten von allen CNNs falsch zugeordnet wurden. Die Gipsabdrücke der Typen 285 und 294 zeichnen sich außerdem durch ihren schwarzen Hintergrund aus.



Abbildung 26: Originalmünze cn_coin_8184_p_rez Typ 285.

Daher könnte eine falsche Zuteilung auch aufgrund des schwarzen Hintergrunds erfolgt sein. Obwohl im Trainingsdatensatz *TrainWithGenerated* die Anzahl der Münzen zwischen diesen zwei Typen ausgeglichen wurde, konnte auch hier keine Steigerung der Sensitivität erfolgen. Dies kann zum einen an der schlechten Qualität der generierten Bilder liegen, aber auch daran, dass die generierten Münzen alle einen weißen Hintergrund haben. Die generierten Münzen ähneln zudem mehr den Metallmünzen als den Gipsabdrücken. Dadurch konnte die Ungleichheit an Gipsabdrücken zwischen den beiden Datensätzen nicht behoben werden und dieser Bias durch die unterschiedlichen Hintergründe bleibt bestehen. Um die falschen Zuordnungen besser zu verstehen, wurden mit Hilfe der *Grad-Cam* Visualisierungsmethode die Heatmaps erstellt. Diese Methode hebt die entscheidenden Regionen hervor, welche ausschlaggebend für die Klassifizierung sind. Die Betrachtung der Heatmaps für eine falsch zugeordnete Münze von den CNNs, die mit *TrainWithGenerated* trainiert wurden, zeigt, dass diese einen sehr starken Fokus auf die Rückseite setzten. Der Fokus ist sehr wahrscheinlich durch die schlechte Qualität der Rückseite der generierten Münzen zustande gekommen. Der Hintergrund wird nur teilweise hervorgehoben. Daher kann nicht gesagt werden, ob der Hintergrund einen wirklichen Nachteil für die Klassifizierung darstellt.

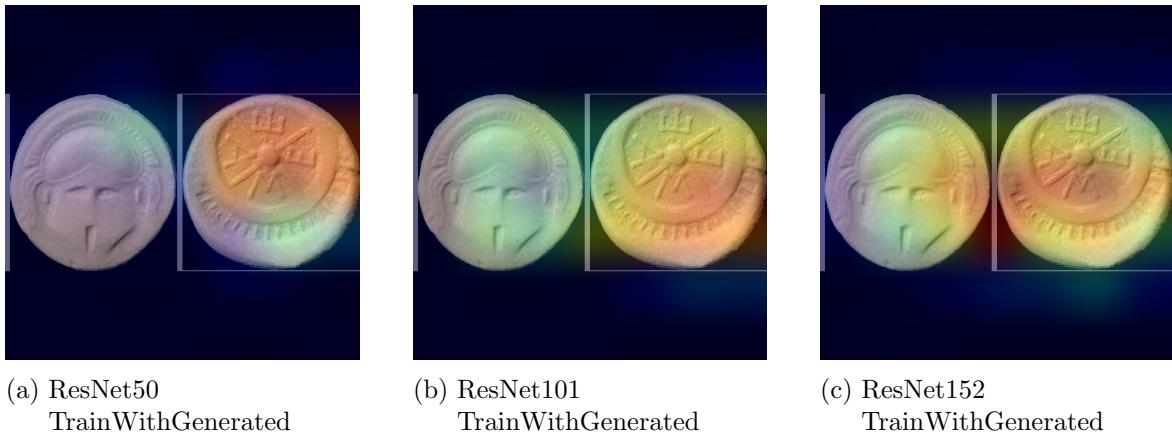


Abbildung 27: Heatmaps der Münze cn_coin_8219_p_rez des Typs. Jedes ResNet hat die Münze dem Typen 285 zugeordnet. Farbverlauf von Blau (wenig wichtig) zu Rot (sehr wichtig).

Die Münzen der Typen 940 und 946 wurden ebenfalls größtenteils einem Münztyp zugeordnet, welcher diesen beiden sehr ähnlich sieht. Die drei Münztypen unterscheiden sich am meisten durch die Schriften, welche auf der Rückseite der Münze um die Weinrebe zu sehen sind.



Abbildung 28: Vergleich der Münzen der Typen 940, 945 und 946.

Auch wenn keine wirkliche Verbesserung durch die generierten Daten auf den ersten Blick ersichtlich ist, haben diese dennoch einen Vorteil für die Klassifizierung erbracht. Beispielsweise im CNN *ResNet50Base* wurden drei Münzen des Typs 945 falsch zugeordnet. Dieser Fehler konnte bei den anderen *ResNets50*, welche mit generierten Daten trainiert wurden, deutlich reduziert werden. Im CNN *ResNet50TrainWithGenerated* war eine falsche Zuteilung für den Typen vorhanden und im *ResNet50TrainSubGenerated* sogar gar keine mehr. Hier haben die generierten Daten einen indirekten Vorteil für die Klassifizierung erbracht. Dies ist wahrscheinlich auch im Gegensatz zum Münztyp 294 der deutlichen besseren Bildqualität zu verdanken.

Der Münztyp 21027 hat eher eine überwiegend schlechte Performance. Für den Münztypen 21027 gibt es drei weitere Typen 20806, 20912 und 12884, welche diesem zum Verwechseln ähnlich sehen. Die meisten falschen Zuteilungen fanden auch zu diesen drei Typen statt. Beim Münztyp 21027 konnte durch die generierten Daten gar keine Verbesserung erzielt werden, obwohl dieser die meisten Originalmünzen in dem Datensatz *TrainWithGenerated* besitzt. Hier sind die generierten Bilder definitiv nicht ausreichend, um die feinen Unterschiede zwischen diesen 4 Typen herzustellen, da die generierte Schrift nur Ansätze zeigt und weit von der originalen Schrift entfernt ist.



Abbildung 29: Vergleich der Münzen der Typen 21027, 12884, 20806 und 20912.

In Abbildung 30 sind die Heatmaps für Münzen dargestellt, welche dem Münztypen 12884 von dem *ResNet50TrainWithGenerated* falsch zugeordnet wurden. Hier wird deutlich, dass die falsche Zuteilung aufgrund der Schriften erfolgt. Die generierten Schriften auf der Rückseite haben hier einen großen Nachteil für die Klassifizierung der Münzen erbracht.



Abbildung 30: Heatmaps des ResNet50TrainWithGenerated. Richtiger Typ der Münzen 21027.
Falsche Zuteilung zum Münztypen 12884.

Interessanterweise trifft dies aber nicht auf die falsch zugeordneten Gipsabdrücke zu.

Wie in Abbildung 31 zu sehen ist, liegt der Fokus mehr auf der Vorderseite. Es wird aber auch deutlich der graue Hintergrund hervorgehoben, was eine falsche Zuordnung aufgrund der Hintergrundfarbe vermuten lässt. Die generierten Bilder für diesen Typen haben alle einen weißen Hintergrund und ähneln eher den Metallmünzen. Hier könnte entweder ein Bias durch den Hintergrund entstanden sein oder durch die wenigen Gipsabdrücke im Trainingsdatensatz. Eine weitere Vermutung ist, dass die CNNs die Abstände zwischen den Münzen als Entscheidungsmerkmal gelernt haben. Die Gipsabdrücke haben im Datensatz öfters einen größeren Abstand zwischen den Münzen. Es lässt sich daher nicht genau sagen, ob dies am Hintergrund oder am Abstand zwischen den Münzen liegt.



(a) cn_coin_6119_o_rez

(b) cn_coin_6208_o_rez

(c) cn_coin_6227_o_rez

Abbildung 31: Heatmaps des ResNet50TrainWithGenerated. Richtiger Typ der Münzen 21027.
Falsche Zuteilung zum Münztypen 12884.

Bei den restlichen Münztypen ist zwischen den *Base* und den *TrainWithGenerated* Modellen kein großer Unterschied aufgefallen. Die Erweiterung mit generierten Bildern hat hier keinen großen Vorteil oder Nachteil erbracht. Die Münzen, welche falsch zugeordnet wurden, weisen in diesen Klassen oft eine schlechte Qualität auf. Die Vorder- oder Rückseite sind auf diesen Münzen oft nicht erkennbar. Dies kann auch dazu beitragen, dass die Performance sich kaum unterscheidet, da diese Münzen schwerer zuzuordnen sind. Aufgrund der kleinen Testmenge kann leider keine endgültige Entscheidung getroffen werden, ob die Erweiterung mit generierten Daten und somit das Ausbalancieren der Datenmenge einen wirklichen Vorteil erbringt. Hierfür benötigt man einen deutlich größeren Testdatensatz.

5.1.4.2 Evaluation der CNNs trainiert mit TrainSubGenerated

In diesem Abschnitt werden die Ergebnisse der CNNs näher betrachtet, welche mit dem Trainingsdatensatz *TrainSubGenerated* trainiert wurden. Da für die Münztypen 20545, 20809 und 21027 keine weiteren Münzen entfernt wurden, werden diese in der Analyse ausgelassen, da diese Typen etwa genauso gut abgeschnitten haben wie in den CNNs des Trainingdatensatzes *TrainWithGenerated*. Betrachtet man zunächst die Sensitivität auf dem Testdatensatz in [Tabelle 12](#) sind die Unterschiede teilweise relativ groß, was dadurch zustande kommt, dass für die meisten Typen nur drei Testbilder vorhanden sind und somit jedes richtig klassifizierte Bild den Wert um 0,3333 erhöht.

Typ	294	940	946	5135	5944	6057	7697	12752	12783	20230
Res50	0,0	1,0	0,6667	0,6667	0,6667	0,6667	0,0	0,3333	0,8	0,6667
Res101	0,0	0,6667	0,6667	0,3333	0,3333	0,6667	0,0	0,3333	0,8	1,0
Res152	0,0	1,0	1,0	0,6667	1,0	0,6667	0,3	0,3333	0,4	0,6667
#Bilder im Testdatensatz	3	3	3	3	3	3	10	3	5	3
#Originalbilder im Trainingsdatensatz	10	10	10	5	5	10	5	5	5	5
#Generierte Bilder im Trainingsdatensatz	40	30	20	40	40	40	35	30	25	20

Tabelle 12: Sensitivität der einzelnen ResNet trainiert auf dem *TrainSubGenerated* Datensatz.
Ausgewertet auf dem Testdatensatz. Höhere Werte sind besser

Da der Testdatensatz sehr klein ist, wurde, um die Performance der CNNs *TrainSubGenerated* besser bewerten zu können, der Testdatensatz mit den Münzen erweitert, welche im ursprünglichen Base Trainingsdatensatz entfernt wurden. Durch die vergrößerte Testmenge, lässt sich die Performance besser bewerten, auch wenn der Testdatensatz immer noch ziemlich klein ist. Es ist zu beachten, dass Münzen aus dem Trainingsdatensätzen der FSGANs in diesem erweiterten Testdatensatz enthalten sein können. Diese können die Ergebnisse leicht beeinflussen. Die Ergebnisse sind [Tabelle 13](#) zu sehen.

Typ	294	940	946	5135	5944	6057	7697	12752	12783	20230
ResNet 50 Sensitivität	0,25	1,0	0,8571	0,7333	0,5833	0,8	0,00962	0,42857	0,6470	0,5834
ResNet 101 Sensitivität	0,125	0,625	0,8571	0,6667	0,4167	0,9	0,0288	0,4286	0,529	0,5834
ResNet 152 Sensitivität	0,125	1,0	1,0	0,8	1,0	0,8	0,0385	0,3571	0,4117	0,5834
#Bilder im Testdatensatz	8	8	7	15	12	10	104	14	17	12
#Originalbilder im Trainingsdatensatz	10	10	10	5	5	10	5	5	5	5
#Generierte Bilder im Trainingsdatensatz	40	30	20	40	40	40	35	30	25	20

Tabelle 13: Sensitivität der einzelnen ResNet trainiert auf dem *TrainSubGenerated* Datensatz.
Ausgewertet auf dem Testdatensatz erweitert mit den entfernten Münzbildern. Höhere Werte sind besser.

Die Sensitivität des Typen 7697 sticht bei den Ergebnissen am meisten heraus. Bei diesem Typen hat sich die Performance im Vergleich zu den *Base* Modellen am stärksten verschlechtert. Der Typ 7967 beinhaltet hauptsächlich Münzbilder von schlechter Qualität. Auf diesen ist des Öfteren nicht nur die Motivpositionierungen sehr unterschiedlich, sondern auch die Motive selbst sind häufig nicht richtig erkennbar.



Abbildung 32: Beispilmünzen vom Typ 7697.

Bei genauerer Prüfung der Zuordnung der Münzen im Testdatensatz fällt erneut auf, dass es weitere Münztypen gibt, welche sich in ihren Motiven sehr ähnlich sind. Wie in Abbildung 33 zu erkennen ist, sind die Vorderseiten der Münzen meist identisch. Der Hauptunterschied liegt in der stehenden Person auf der Rückseite der Münztypen.



Abbildung 33: Beispilmünzen von den Typen 7686, 7907, 7910, 7696 und 7803.

Eine genauere Untersuchung der einzelnen Bilder der fünf Typen im Trainingsdatensatz *TrainSubGenerated* zeigt auf, dass diese hauptsächlich Gipsabdrücke und oft auch Bilder in schlechter Qualität sind. Im Trainingdatensatz des Typen 7697 hingegen sind nur

zwei Originalmünzen, die Gipsabdrücke sind. Die generierten Daten sind häufiger Metallmünzen. Des Weiteren gibt es eine Mischung aus Gipsabdrücken und Metallmünzen bei den generierten Münzbildern. All diese Faktoren tragen sehr wahrscheinlich zur schlechten Zuteilung bei. Die überwiegende Anzahl an Metallmünzen im Trainingsdatensatz des Typs führt dazu, dass eine Vielzahl an Gipsabdrücken eher den anderen Typen zugeordnet wird. Dies wird auch höchstwahrscheinlich durch die hohe Anzahl an Münzbildern in schlechter Qualität bei den anderen Typen begünstigt. Das CNN lernt nicht Münzbilder in schlechter Qualität dem Typen 7697 zuzuordnen, da es nicht genügend Beispiele im Trainingsdatensatz hat.



Abbildung 34: Beispiele von generierten Münzen vom Typ 7697.

Ein Blick auf die Heatmaps für falsch zugeordnete Münzen in [Abbildung 35](#) zeigt, dass der Fokus überwiegend auf der Rückseite und teilweise auch auf den Schriften liegt. Die nicht vorhandenen Schriften auf den generierteren Bilder könnten auch ein Grund für die schlechte Performance sein.



Abbildung 35: Heatmaps falsch zugeordneter Münzen des 769. ResNet trainiert auf dem *TrainSubGenerated* Datensatz.

Eine ähnlich fehlerhafte Zuteilung von schlecht erhaltenen Münzen ist auch beim Ty-

pen 20230 zu erkennen. Im Gegensatz zum Typen 7697 werden diese Münzen jedoch keinem Typen mit ähnlichen Motiven zugeteilt, sondern Typen, bei denen auf der Vorderseite ebenfalls ein Porträt und auf der Rückseite ein anderes Motiv zu sehen ist. Allerdings weisen diese Typen viele Bilder auf, auf denen auf der Rückseite gar kein Motiv erkennbar ist.



Abbildung 36: Beispiele von Originalmünzen vom Typ 20230 welche falsch zugeteilt wurden. Bilder aus dem erweiterten Testdatensatz mit entfernten Münzen.

Vergleicht man die Performance dieses Typen ([Tabelle 12](#)) auf dem ursprünglichen Testdatensatz mit den CNNs, die auf dem *Base* Datensatz ([Tabelle 11](#)) trainiert sind, sind Unterschiede in der Sensitivität kaum auszumachen. Diese Bilder weisen insgesamt keine gute Qualität auf. Der Unterschied zu Typ 7697 ist, dass beim Typen 20230 Bilder generiert wurden, welche die schlechte Qualität gut abbilden. Dies hat wahrscheinlich einen deutlichen Vorteil für die richtige Zuteilung der Münzen erbracht.



Abbildung 37: Bilder des Typs 20230 aus dem Testdatensatzes.



Abbildung 38: Generierte Bilder des Typs 20230 mit schlechter Qualität.

Betrachtet man die Heatmaps für Münzen mit korrekter Zuteilung, erkennt man, dass

diese auch aufgrund der Rückseite richtig erkannt wurden. Der Fokus bei den CNNs lag sogar ausschließlich auf der Rückseite.



Abbildung 39: Heatmaps für die Münze cn_coin_30918_p_rez des Typs 20230. Richtig zugeordnet zu Typ 20230. Trainiert auf den Datensatz *TrainSubGenerated*.

Die Ergebnisse der Typen 7697 und 20230 zeigen, dass es nicht unbedingt von Vorteil ist, nur generierte Daten in guter Qualität zu haben. Vielmehr braucht man auch generierte Daten, welche die schlecht erhaltenen Münzen darstellen, sodass schlecht erhaltene Originale auch richtig zugeordnet werden können.

Bei den generierten Münzen des Typen 12783 ist die Rückseite nicht optimal dargestellt. Dies hat zur Folge, dass viele Münzen falsch zugeordnet werden und dies betrifft Typen, die ebenfalls ein Porträt auf der Vorderseite haben und ein Tier auf der Rückseite. Zusätzlich ist über dem Tier auf der Rückseite der Originalmünzen oft ein Schriftzug, welcher auch über den Tieren der anderen Münztypen zu finden ist. Dieser Schriftzug fehlt auf den generierten Münzen. Hier ist der fehlende Schriftzug jedoch nicht ausschlaggebend für die falschen Zuteilungen, wie in den Heatmaps in Abbildung 40 zu sehen ist.



(a) Münze cn coin 19261 o rez.
Falsch zugeordnet zu
Münztyp 20652 vom
ResNet152.

(b) Münze cn coin 40586. Falsch
zugeordnet zu Münztyp
20652 vom ResNet152.

(c) Münze cn coin 40586. Falsch
zugeordnet zu Münztyp
19756 vom ResNet152.

Abbildung 40: Heatmaps falsch zugeordneter Münzen des Typs 12783.

Die Sensitivitäten der Typen 5135 und 5944 waren im Vergleich zu den anderen Typen, welche nur fünf Originalbilder enthalten, deutlich besser. Die meisten Testbilder dieser zwei Typen haben alle eine sehr gute Qualität und eine einheitliche Motivpositionierung. Bei den Ergebnissen lässt sich daher schwer sagen, ob die Mehrzahl an generierten Bildern im Trainingsdatensatz oder die Qualität der Testbilder einen Vorteil erbracht hat im Vergleich zu den anderen Typen, welche nur fünf Originalbilder besitzen.

Die Typen mit mindestens zehn Originalbildern im Trainingsdatensatz haben alle ähnlich gute Ergebnisse, wie die CNNs der *Base* und *TrainWithGenerated* Trainingdsdatensätze. Für den Typen 6057 wurden jeweils für die Vorder- und Rückseite zwei verschiedene FS-GANs trainiert. Der eine wurde auf Gipsabdrücken und der andere auf Metallmünzen trainiert. Aus den Ergebnissen lässt sich leider nicht eindeutig ablesen, ob dies einen großen Vorteil erbracht hat. Das gute Abschneiden der CNNs, trainiert auf dem Trainingdatensatz *TrainSubGenerated*, kann ein Indiz dafür sein, dass die Gleichverteilung von Gipsabdrücken und Metallmünzen einen Vorteil erbringt. Jedoch kann dies durch die Ergebnisse nicht sicher belegt werden.

Der Typ 294, welcher auch auf dem Datensatz *Base* und *TrainWithGenerated* schwach abgeschnitten hat, erzielte auch hier keine guten Ergebnisse. Durch die Verringerung der Anzahl an Originalmünzen im Trainingsdatensatz war zudem keine Steigerung der Performance zu erwarten, weshalb das mangelhafte Abschneiden wenig überrascht. Ins-

gesamt lässt sich keine endgültige Entscheidung treffen, wie das Verhältnis von Originalmünzen zu generierten Daten sein soll und wie viele Originale mindestens benötigt werden. Die Ergebnisse geben keinen Aufschluss über eine genaue Anzahl.

5.1.5 Training von Base Learner für die Vorder- und Rückseite

Um Münztypen genau zu klassifizieren, werden die Bilder der Vorder- und Rückseite benötigt, da viele Münztypen die gleichen Motive auf einer Seite haben. Für das Ensemble Learning sind jedoch Base Learner von Vorteil, welche unterschiedliche Fehler machen. Um die Auswirkungen von Base Learnern zu testen, welche nur auf einer Seite der Münze trainiert sind, wurden für die drei Trainingsdatensätze jeweils ein ResNet50 und ResNet101 für die Vorder- und Rückseite der Münzen trainiert. Es wird sich durch das Hinzufügen dieser Base Learner ein besseres Ergebnis bei den verschiedenen Ensemble Learning Methoden erhofft. Diese CNNs wurden mit den gleichen Hyperparametern trainiert wie in [Unterabschnitt 5.1.2](#) beschrieben. Die Fehlerrate der jeweiligen CNNs sind in [Tabelle 14](#) und [Tabelle 15](#) zu sehen.

Model	ResNet50			ResNet101		
	Trainingsdatensatz	Base	TrainSubGenerated	TrainWithGenerated	Base	TrainSubGenerated
Testdatensatz	24,42%	24,81%	26,67%	24,13%	25,67%	26,15%
Testdatensatz 13 Typen	39,29%	51,79%	37,50%	32,14%	53,17%	37,50%

Tabelle 14: Fehlerrate der CNNs trainiert auf der Vorderseite der Münzen. Kleinere Werte sind besser.

Die Fehlerrate von dem Modell der Rückseite ist auf dem gesamten Testdatensatz nicht sehr viel schlechter als die der CNNs in [Tabelle 8](#). Beim Modell der Vorderseite ist ein starker Anstieg der Fehlerrate zu sehen. Auf den Testdatensatz der 13 Münztypen ist bei den CNNs ebenfalls ein deutlicher Anstieg der Fehlerrate zu sehen. Der Anstieg der Fehlerrate ist bei den CNNs des Trainingsdatensatzes *TrainSubGenerated* im Vergleich zu den CNNs, welche auf beiden Seiten trainiert wurden, nicht so stark.

Model	ResNet50			ResNet101		
	Trainingsdatensatz	Base	TrainSubGenerated	TrainWithGenerated	Base	TrainSubGenerated
Testdatensatz	20,48%	22,21%	20,96%	20,48%	22,11%	21,25%
Testdatensatz 13 Typen	35,71%	53,57%	39,29%	37,50%	55,36%	39,29%

Tabelle 15: Fehlerrate der CNNs trainiert auf der Rückseite der Münzen. Kleinere Werte sind besser.

In Tabelle 16 und Tabelle 17 sind die Sensitivitäten für die 13 Typen abgebildet. Aufgrund der begrenzten Ausarbeitungszeit wird auf eine weitere Evaluation der CNNs verzichtet.

Typ	294	940	946	5135	5944	6057	7697	12752	12783	20230	20545	20809	21027
ResNet50 Base	0,6667	1,0	1,0	1,0	0,6667	1,0	0,4	0,3333	1,0	1,0	0,6667	0,5	0,1
ResNet101 Base	0,6667	1,0	1,0	0,6667	0,6667	0,6667	0,8	0,6667	0,6	1,0	1,0	0,5	0,3
ResNet50 TrainWithGenerated	0,3333	1,0	1,0	0,3333	0,6667	1,0	0,6	0,6667	0,8	1,0	0,6667	0,75	0,2
ResNet101 TrainWithGenerated	0,0	1,0	1,0	0,6667	0,6667	1,0	0,7	0,3333	0,4	1,0	0,6667	1,0	0,3
ResNet 50 TrainSubGenerated	0,0	1,0	1,0	0,3333	0,6667	0,6667	0,2	0,0	0,4	0,6667	1,0	0,75	0,4
ResNet101 TrainSubGenerated	0,0	0,6667	1	0,6667	0,6667	0,6667	0,2	0,0	0,0	1,0	1,0	1,0	0,3
#Bilder im Testdatensatz	3	3	3	3	3	3	10	3	5	3	3	4	10

Tabelle 16: Sensitivität der CNNs der Vorderseite. Höhere Werte sind besser.

Typ	294	940	946	5135	5944	6057	7697	12752	12783	20230	20545	20809	21027
ResNet50 Base	0,0	1,0	0,6667	0,6667	1,0	0,6667	1,0	0,6667	1,0	0,6667	0,3333	0,25	0,3
ResNet101 Base	0,0	0,6667	0,6667	0,6667	1,0	0,6667	0,8	1,0	1,0	0,3333	1,0	0,25	0,3
ResNet50 TrainWithGenerated	0,3333	0,3333	1,0	1,0	1,0	0,3333	0,8	0,3333	1,0	0,3333	0,6667	0,25	0,4
ResNet101 TrainWithGenerated	0,0	1,0	0,6667	1,0	1,0	1,0	0,8	0,66667	0,6	0,6667	1,0	0,25	0,2
ResNet 50 TrainSubGenerated	0,0	0,6667	1,0	1,0	1,0	0,6667	0,1	0,0	1,0	0,0	1,0	0,25	0,3
ResNet101 TrainSubGenerated	0,3333	0,6667	1,0	0,6667	1,0	0,6667	0,2	0,0	0,8	0,0	0,6667	0,25	0,3
#Bilder im Testdatensatz	3	3	3	3	3	3	10	3	5	3	3	4	10

Tabelle 17: Sensitivität der CNNs der Rückseite. Höhere Werte sind besser.

5.1.6 Fazit

Die Klassifizierung der Münzen zu ihren Münztypen ist eine sehr schwierige Aufgabe. Es gibt viele Typen, die sich nur in kleinen Details unterscheiden. Dies macht die Varianz zwischen den Typen teilweise sehr gering. Gleichzeitig kann die Varianz innerhalb eines Typs sehr groß sein, durch beispielsweise unterschiedliche Motivpositionierungen, das Material der Münzen oder den Erhaltungsgrad der Münzen. All diese Faktoren können das Training stark beeinflussen und die Klassifizierungsaufgabe erschweren. Im Absatz 5.1.4.1 wurde deutlich, dass die Erweiterung der Münztypen mit generierten Bildern und somit das Ausbalancieren der Münztypen höchstens kleine Verbesserungen hervorruft. Alle Münztypen mit generierten Daten auszugleichen wäre für diesen geringen Vorteil sehr zeitaufwändig. Absatz 5.1.4.2 hat gezeigt, dass es durchaus möglich ist, gute Ergebnisse mit wenig Originalmünzen und generierten Münzen zu erreichen.

Die guten Ergebnisse sind jedoch nicht nur von der Qualität der generierten Daten abhängig, sondern auch von der Anzahl an ähnlichen Münztypen, Qualität der Münzen des Testdatensatzes und vieles mehr. Daher lässt sich aus den Ergebnissen keine generalisierte Entscheidung treffen, ob diese Methode sinnvoll und zielführend ist.

5.2 Bilden eines Ensembles

In diesem Abschnitt wird die Umsetzung des Ensemble Learnings behandelt. Hierfür wird Python in der Version 3.11.5 verwendet. Zunächst wird die konkrete Implementierung in [Unterabschnitt 5.2.1](#) dargelegt und anschließend werden in [Unterabschnitt 5.2.2](#) die einzelnen umgesetzten Kombinationsmöglichkeiten evaluiert.

5.2.1 Implementierung des Ensemble Learnings

Die trainierten Base Learner werden genutzt, um ein Ensemble zu bilden, das eine finale Ausgabe für die Klassifizierung der Münztypen macht. In [Abschnitt 2.3](#) wurden zwei Kombinationsmethoden vorgestellt, die im Rahmen dieser Arbeit realisiert werden. Die Implementierung des Ensemble Learnings ist in der `Ensemble.py` Datei, in Form einer Klasse, umgesetzt. Der Aufbau der Klasse und ihre Verwendung werden in Unterabschnitt [5.2.1.1](#) erläutert. Innerhalb dieser Klasse wird Voting und Stacking implementiert. Die konkrete Umsetzung dieser zwei Ensemblemethoden und die verschiedenen Funktionen werden in den Unterabschnitten [5.2.1.2](#) und [5.2.1.3](#) erläutert. In [5.2.1.2](#) wird die Implementierung von fünf verschiedenen Voting Funktionen erklärt. [5.2.1.3](#) thematisiert die Implementierung des Stackings, wobei für den Meta Learner zwei unterschiedliche Modelle verwendet werden.

5.2.1.1 Bilden einer Ensemble-Klasse

Um verschiedene Ensemble Learning Umsetzungen zu testen und zu vergleichen, wurde eine Ensemble-Klasse (`Ensemble.py`) erstellt. In dieser Klasse werden viele verschiedene Python-Bibliotheken zur Hilfe verwendet. In [Tabelle 18](#) sind alle nötigen Bibliotheken, die installiert wurden, mit ihrer Versionsnummer und dem Grund ihrer Verwendung gelistet:

Paket	Version	Verwendung	Referenz
argparse	1.1	Zum Starten der Ensemble-Klasse mit den gewünschten Parametern im Terminal.	[38]
joblib	1.2.0	Zum Speichern der Meta Modelle beim Stacking mit <code>dump</code> .	[39]
json	2.0.9	Zum Öffnen der JSON-Dateien, mit den gewünschten Netzen für das Ensemble und ihren jeweiligen Pfaden zu den Vorhersagen (<i>predictions</i>) für die Validierungs- und Testdaten.	[40]
matplotlib	3.7.2	Zum Visualisieren und Speichern der Ergebnisse.	[41]
numpy	1.24.3	Verschiedene Hilfsfunktionen für Arrays zum Laden der Vorhersagen, Erstellen von Arrays, Sortieren von Arrays, Stapeln (<i>stack</i>) der Vorhersagen etc.	[42]
pandas	2.0.3	Zum Speichern der Ergebnisse in einer CSV-Datei.	[43]
pickle	4.0	Zum Laden der Informationen über die Codierung der Münztypen.	[44]
pathlib	1.0.1	Zum Erhalten des Namens der JSON-Dateien, ohne den Pfad und der Dateiendung.	[45]
sklearn	1.3.0	Zum Bilden des Meta Modells fürs Stacking, wobei einmal die <code>LogisticRegression</code> und einmal der <code>RandomForestClassifiers</code> verwendet wird. Zum Evaluieren der Modelle mittels verschiedener Metriken, wie z.B. der Genauigkeit (<i>accuracy</i>) oder der Sensitivität (<i>recall</i>).	[46], [47], [48]

Tabelle 18: Benötigte Python-Pakete für die Umsetzung des Ensemble Learnings.

Die Ensemble-Klasse soll das Testen verschiedener Ensembles für die Klassifizierung von Münztypen ermöglichen. Im Rahmen dieser Arbeit soll auch die Frage, ob das Generieren von Münzbildern zur Balancierung des Datensatzes die Klassifizierung der Münztypen verbessert, beantwortet werden. Dadurch sollen die Netzwerke, die lediglich auf den Originaldaten trainiert wurden, von den Netzen, die beim Training auch generierte Bilder erhalten haben, getrennt betrachtet werden können. Mit Hilfe von JSON-Dateien kann der Nutzer dieser Ensemble-Klasse entscheiden, welche Netzwerke ein Ensemble bilden sollen. In Tabelle 19 sind die JSON-Dateien, die im Rahmen dieser Arbeit erstellt und verwendet wurden, mit den jeweils verwendeten Base Learnern aufgelistet:

JSON-Datei	Base Learner	Test-daten
AllBothTest13Preds (9 Base Learner)	Alle Netze, die auf den <i>Both</i> Datensatz mit und ohne generierten Bildern trainiert wurden.	Test13
AllBothTestPreds (9 Base Learner)	Alle Netze, die auf den <i>Both</i> Datensatz mit und ohne generierten Bildern trainiert wurden.	Test
AllTest13Preds (21 Base Learner)	Alle Netze, die auf den <i>Both</i> , <i>Obv</i> oder <i>Rev</i> Datensatz mit und ohne generierten Bildern trainiert wurden.	Test13
AllTestPreds (21 Base Learner)	Alle Netze, die auf den <i>Both</i> , <i>Obv</i> oder <i>Rev</i> Datensatz mit und ohne generierten Bildern trainiert wurden.	Test
BaseBothTest13Preds (3 Base Learner)	Alle Netze, die auf den <i>Both</i> Datensatz ohne generierte Bilder (nur Originaldaten) trainiert wurden.	Test13
BaseBothTestPreds (3 Base Learner)	Alle Netze, die auf den <i>Both</i> Datensatz ohne generierte Bilder (nur Originaldaten) trainiert wurden.	Test
BaseTest13Preds (7 Base Learner)	Alle Netze, die auf den <i>Both</i> , <i>Obv</i> oder <i>Rev</i> Datensatz ohne generierte Bilder (nur Originaldaten) trainiert wurden.	Test13
BaseTestPreds (7 Base Learner)	Alle Netze, die auf den <i>Both</i> , <i>Obv</i> oder <i>Rev</i> Datensatz ohne generierte Bilder (nur Originaldaten) trainiert wurden.	Test
GeneratedBothTest13Preds (6 Base Learner)	Alle Netze, die auf den <i>Both</i> Datensatz mit generierten Bildern trainiert wurden.	Test13
GeneratedBothTestPreds (6 Base Learner)	Alle Netze, die auf den <i>Both</i> Datensatz mit generierten Bildern trainiert wurden.	Test
GeneratedTest13Preds (14 Base Learner)	Alle Netze, die auf den <i>Both</i> , <i>Obv</i> oder <i>Rev</i> Datensatz mit generierten Bildern trainiert wurden.	Test13
GeneratedTestPreds (14 Base Learner)	Alle Netze, die auf den <i>Both</i> , <i>Obv</i> oder <i>Rev</i> Datensatz mit generierten Bildern trainiert wurden.	Test

Tabelle 19: Verwendete JSON-Dateien mit den eingesetzten Base Learner für das Ensemble.

In diesen Dateien sind für jedes Netzwerk die Pfade zu den Vorhersagen zu den Test- und Validierungsdaten enthalten. Jeder verwendete Base Learner macht seine Vorhersagen für diese Datensätze mit Hilfe der `ModelEvaluator.py` Datei bevor er ins Ensemble kommt. Dies vermeidet das Laden der einzelnen Netze im Ensemble und das Warten der Vorhersagen für jedes Netzwerk, wodurch eine schnelle Auswertung des Ensembles möglich ist. Bei den Testdaten wird zwischen *Test* und *Test13* unterschieden. In *Test*

sind von jedem Typ der auch für das Training der Base Learner verwendet wurde, mindestens drei Bilder für die endgültige Evaluation enthalten. Dagegen beinhaltet *Test13* nur die Bilder von den 13 Typen, die auch zum generieren von weiteren Münzbildern verwendet wurde. Die Testdaten werden für die Evaluation des Ensembles benötigt. Der Nutzer kann hierbei entscheiden, ob die Evaluationsergebnisse auch in einer Konfusionsmatrix und einer CSV-Datei gespeichert werden sollen (`--eval`), oder nicht. Die Validierungsdaten sind lediglich für das Stacking von Interesse. Diese werden nämlich benötigt, um das Meta Modell (`LogisticRegression` oder `RandomForestClassifier`) zu trainieren.

Ob beim Ensemble eine Kombination der Base Learner Ausgaben mittels Voting (vgl. Unterabschnitt 5.2.1.2) oder mittels Stacking (vgl. Unterabschnitt 5.2.1.3) erfolgt, kann ebenfalls der Nutzer entscheiden. Der Nutzer kann hier zwischen fünf verschiedenen Voting und zwei verschiedenen Stacking Implementierungen wählen:

- Voting: `hard_voting`, `simple_soft_voting`, `weighted_soft_voting`,
`weighted_pred_soft_voting`, `weighted_rev_soft_voting`
- Stacking: `stacking_with_RF`, `stacking_with_LR`

Allgemein wird die Ensemble-Klasse mit folgendem Befehl im Terminal ausgeführt:

```
python3 Ensemble.py \
Pfad_zur_JSON \
Pfad_zum_Dictionary \
Kombinationsmethode \
Top-X-Wert \
--eval
```

Das Dictionary liegt in Form einer Pickle-Datei vor. Es ist für eine korrekte Übersetzung der Münztypen nötig. Beim Training der Base Learner werden die Label der 207 Typen mit den Zahlen 0 bis 206 codiert. Im Dictionary ist dann für jede Zahl zwischen 0 und 206 das tatsächliche Label der verwendeten Münztypen gespeichert. Das Dictionary wird für jeden Base Learner erstellt, allerdings ist das für alle Base Learner gleich,

sodass der Nutzer ein beliebiges Dictionary für das Ausführen des Ensembles verwenden kann.

5.2.1.2 Kombination der Base Learner Ausgaben mittels Voting

Wie bereits erwähnt kann der Nutzer beim Voting zwischen fünf verschiedenen Implementierungen wählen. Weiterhin kann der Nutzer einen Wert X angeben, der für jedes Netz die Top- X -Vorhersagen betrachtet. Die genaue Umsetzung der verschiedenen Voting Kombinationsmöglichkeiten wird im Folgenden erläutert.

hard_voting:

Bei der Wahl von `hard_voting` wird eine Abwandlung des in Unterabschnitt 2.3.2.1 erklärten Hard votings, oder auch Majority votings, umgesetzt. Beim klassischen Hard Voting wird immer nur dann eine Ausgabe erzeugt, wenn eine absolute Mehrheit zwischen den Base Learnern vorliegt. Sind nicht mindesten 50% der verwendeten Netze der gleichen Meinung, wird keine Ausgabe erzeugt. Damit das verwendete Ensemble in dieser Arbeit aber immer eine Vorhersage macht, wird hier die Vorhersage als finale Ausgabe gewählt, die am häufigsten von den Base Learnern ausgegeben wurde. Durch die Angabe der zu betrachtenden Top- X -Vorhersagen werden dann alle X Vorhersagen der Base Learner berücksichtigt. Sollten mehrere Münztypen gleich oft vorhergesagt werden, sodass keine Entscheidung getroffen werden kann, wird `simple_soft_voting` verwendet.

In Unterabschnitt 2.3.2.2 wurde das Soft Voting erklärt, wobei zwischen Simple Soft Voting und Weighted Soft Voting unterschieden wurde.

simple_soft_voting:

Beim `simple_soft_voting` wird von den übergebenen Top- X -Vorhersagen die Summe der Wahrscheinlichkeiten für jede Klasse bzw. jeden Münztyp berechnet. Der Unterschied zum allgemeinen Simple Soft Voting liegt hierin, dass nicht nochmal über die Anzahl der verwendete Netze im Ensemble geteilt wird. Die finale Ausgabe ist dann der Typ mit der höchsten Summe.

Alternativ kann das Weighted Soft Voting verwendet werden, was allgemein klassifikatorspezifische oder klassenspezifische Gewichte haben kann.

weighted_soft_voting:

Beim `weighted_soft_voting` wurden die klassifikatorspezifischen Gewichte gewählt. Da sich ein Münztyp allgemein aus der Vorder- und Rückseite zusammensetzt, werden die Ausgaben der neuronalen Netze, die auf den beidseitigen Bildern (*Both*) trainiert wurden, stärker gewichtet (Faktor: 1,5). Es wurden verschiedene Gewichte zwischen 1 und 3 auf dem *Test13* Datensatz mit allen trainierten Base Learnern im Ensemble getestet. Die Unterschiede bei der Genauigkeit lagen lediglich um die 3 Prozentpunkte. Die Genauigkeit mit dem Faktor 1,5 gehörte zu den höchsten beobachteten Genauigkeiten, weswegen dieser Wert gewählt wurde. Wird ein Ensemble verwendet, das nur CNNs enthält, die auf dem *Both* Datensatz trainiert wurden, entspricht das `weighted_soft_voting` dem `simple_soft_voting`.

weighted_rev_voting:

Alternativ werden beim `weighted_rev_voting`, welches nur für die Top-1-Vorhersagen implementiert wurde, die Netze der Vorder- und Rückseite (*Obv, Rev*) stärker gewichtet (Faktor: 1,5), wenn die gleiche Architektur die gleiche Vorhersage macht.

weighted_pred_soft_voting:

Beim `weighted_pred_soft_voting` werden die Top-X Vorhersagewahrscheinlichkeiten jedes Netzes gewichtet, was klassenspezifische Gewichte sind. Die höchste Vorhersagewahrscheinlichkeit eines Netzes wird mit dem Wert X gewichtet, die zweithöchste mit dem Wert X-1 usw. bis die niedrigste das Gewicht 1 erhält. Wird `weighted_pred_soft_voting` nur auf die Top-1-Wahrscheinlichkeiten angewendet, entspricht es dem `simple_soft_voting`.

Beispiel - Ausgabe der implementierten Voting Funktionen:

Ein einfaches Klassifizierungsbeispiel soll die Unterschiede dieser fünf Voting Funktionen verdeutlichen. Im Folgenden wird ein Ensemble mit sechs Base Learnern und zwei unterschiedlichen Architekturen (A1 bzw. A2) betrachtet:

1. BaseLearnerA1Both: Ein Base Learner, der auf den beidseitigen Datensatz (*Both*) der Münzbilder auf der A1 Architektur trainiert wurde.
2. BaseLearnerA1Obv: Ein Base Learner, der nur auf den Datensatz der Vorderseite (*Obv*) der Münzbilder auf der A1 Architektur trainiert wurde.
3. BaseLearnerA1Rev: Ein Base Learner, der nur auf den Datensatz der Rückseite (*Rev*) der Münzbilder auf der A1 Architektur trainiert wurde.
4. BaseLearnerA2Both: Ein Base Learner, der auf den beidseitigen Datensatz (*Both*) der Münzbilder auf der A2 Architektur trainiert wurde.
5. BaseLearnerA2Obv: Ein Base Learner, der nur auf den Datensatz der Vorderseite (*Obv*) der Münzbilder auf der A2 Architektur trainiert wurde.
6. BaseLearnerA2Rev: Ein Base Learner, der nur auf den Datensatz der Rückseite (*Rev*) der Münzbilder auf der A2 Architektur trainiert wurde.

Jeder Base Learner bekommt als Eingabe ein Bild einer antiken Münzen, die vom Typ 21027 (s. Abbildung 41) ist, und soll den Münztyp bestimmen.



Abbildung 41: Beispielmünze (cn_coin_50460) vom Typ 21027.

Die beispielhaften Wahrscheinlichkeiten der sechs Base Learner für die Top-1-Vorhersagen sind in Tabelle 20 zu sehen:

	12884	20806	21027	Top-1
BaseLearnerA1Both			0,9	21027
BaseLearnerA1Obv			0,8	21027
BaseLearnerA1Rev		0,7		20806
BaseLearnerA2Both			0,9	21027
BaseLearnerA2Obv	0,8			12884
BaseLearnerA2Rev	0,7			12884

Tabelle 20: Beispielhafte Klassenwahrscheinlichkeiten für die Top-1-Vorhersagen der Base Learner.

Wie in [Tabelle 20](#) zu sehen ist, werden neben dem richtigen Typ 21027 auch noch die Typen 12884 und 20806 vorhergesagt. In [Abbildung 42](#) sind für diese zwei Typen beispielhaft Münzbilder zu sehen, die für das ungeübte menschliche Auge auch starke Ähnlichkeiten mit dem tatsächlichen Typ aufweisen.



Abbildung 42: Beispilmünzen der falsch zugeordneten Typen.

In [Tabelle 21](#) sind für die verschiedenen Voting Funktionen die Ausgaben zu sehen:

	12884	20806	21027	Ausgabe
hard_voting	2	1	3	21027
simple_soft_voting	1,5	0,7	2,6	21027
weighted_soft_voting	1,5	0,7	3,5	21027
weighted_rev_voting	2,25	0,7	2,6	21027

Tabelle 21: Ensemble Ausgabe für die verschiedenen Voting Funktionen.

Beim `weighted_pred_soft_voting` werden die Top-X-Vorhersagen mit X gewichtet. Da hier nur die Top-1-Vorhersage betrachtet wird, ist das Gewicht 1 und die Ausgabe entspricht dem `simple_soft_voting`. Für alle Voting Funktionen würde das Ensemble bei den gegebenen Base Learnern immer den richtigen Typ vorhersagen.

5.2.1.3 Kombination der Base Learner Ausgaben mittels Stacking

Eine weitere Kombinationsmöglichkeit von Base Learnern Ausgaben ist das in [Unterabschnitt 2.3.3](#) vorgestellte Stacking. Der Nutzer der Ensemble-Klasse kann zwischen zwei verschiedene Implementierungen wählen. Der Unterschied liegt beim verwendeten Meta Modell, welches auf den Validierungsdaten trainiert wird und anschließend gespeichert wird. Für das Stacking ist der Top-X-Wert irrelevant, weswegen der Nutzer da eine beliebige Eingabe machen kann.

stacking_with_RF:

Beim `stacking_with_RF` ist das Meta Modell ein Random Forest. Bei einem Random Forest handelt es sich um einen Wald von Entscheidungsbäumen. Für die Implementierung wurde der `RandomForestClassifier` von scikit-learn[47] mit 1000 Entscheidungsbäumen verwendet. Die Variation der Anzahl der Entscheidungsbäume hatte lediglich einen geringen Einfluss auf die Vorhersageleistung des *Test13* Datensatzes mit allen Base Learnern im Ensemble. Bei 1000 Entscheidungsbäumen wurden 44 von 56 Bilder richtig klassifiziert, wohingegen beispielsweise bei 1200 Entscheidungsbäume 43 und bei 1800 Bäumen 42 Bilder richtig klassifiziert wurden. Diese Werte waren allerdings auch nicht bei jedem Training mit der identischen Anzahl an Bäumen gleich, da unterschiedliche Entscheidungsbäume entstehen können. Für die Wahl der 1000 Entscheidungsbäume wurden die drei möglichen Qualitätsmaße für einen Split (`gini`, `entropy`, `log_loss`) geprüft, wobei `gini`, was auch der Standardwert ist, für das Ensemble mit allen Base Learnern auf dem *Test13* Datensatz das beste Ergebnis erzielt hat. Von den insgesamt 56 Bildern in diesem Datensatz wurden mit `gini` 44 richtig erkannt, wohingegen `entropy` nur 32 und `log_loss` nur 30 Bilder richtig erkannt haben. Ein weiterer Parameter, der getestet wurde, ist die maximale Tiefe. Hier wurden Tiefen zwischen 25 und 225 getestet. Je kleiner die Tiefe war, desto weniger Bilder wurden richtig klassifiziert. Ab einer Tiefe von 150 wurden ungefähr 42 Bilder richtig klassifiziert. Wird der Wert der maximalen Tiefe auf `None` gesetzt, was auch dem Standardwert entspricht, wächst der Baum, bis alle Blätter leer sind oder weniger als zwei Exemplare enthalten. In diesem Fall wurden 44 von 56 Bildern richtig klassifiziert.

Auf Grund der minimalen Verbesserung bei `None` wurde dies für den Parameter der maximalen Tiefe gewählt. In [47] sind noch weitere Parameter angegeben, für die jeweils der Standardwert genommen wurde. Weiterhin können bei jedem Training auch mit gleichen Parametern unterschiedliche Entscheidungsbäume entstehen und bei den durchgeführten Tests wurden keine nennenswerten Verbesserungen erzielt.

stacking_with_LR:

Beim `stacking_with_LR` ist das Meta Modell eine logistische Regression. Da die Klassifizierungsaufgabe dieser Arbeit zwischen 207 Münztypen unterscheidet, wird eine multinomiale logistische Regression durchgeführt. Für die Implementierung wurde die `LogisticRegression` von scikit-learn[46] genommen. Da es sich bei der Klassifizierungsaufgabe um ein Mehrklassenproblem handelt, wird der Parameter `multi_class` auf `multinomial` gesetzt. Die restlichen Parameter werden mit den Standardwerten genutzt. Auch bei diesem Meta Modell wurden einige Parameter geändert, um zu überprüfen, ob eine Verbesserung erzielt werden kann. Für die Parameter `penalty` und `solver` (für Mehrklassenprobleme) wurden die verschiedenen Möglichkeiten auf dem *Test13* Datensatz für das Ensemble mit allen trainierten Base Learnern getestet, allerdings konnten keine besseren Ergebnisse erzielt werden. In Tabelle 22 ist angegeben, wie viele von den 56 Bildern des Datensatzes für die verschiedenen Parameter richtig klassifiziert wurden:

		LogisticRegression(penalty= , solver= , multi_class="multinomial")			
		penalty			
		None	12	11	elasticnet
solver	<code>lbfgs</code>	44	44	-	-
	<code>sag</code>	44	44	-	-
	<code>saga</code>	44	44	41	42
	<code>newton-cg</code>	42	44	-	-

Tabelle 22: Anzahl richtig klassifizierter Bilder auf dem *Test13* Datensatz (insgesamt 56 Bilder) für eine multinomiale logistische Regression mit unterschiedlichen `penalty` und `solver`. Bei der Verwendung von `solver="elasticnet"` muss auch der Parameter `l1_ratio` gesetzt werden, für den 0,5 gewählt wurde.

Werden bei der `LogisticRegression` außer beim `multi_class` Parameter die Standardwerte genommen, werden 44 von 56 Bildern richtig erkannt, weswegen dies als Meta Modell genutzt wird.

Auch bei der logistischen Regression konnte die Abweichung von den Standardwerten der Parameter keine nennenswerten Verbesserungen hervorrufen.

5.2.2 Evaluation des Ensemble Learnings

Für die Evaluation des Ensembles werden verschiedene Zusammensetzungen von Base Learnern verwendet und für alle Kombinationsmöglichkeiten auf zwei unterschiedlichen Testdatensätzen ausgewertet. In [Tabelle 19](#) sind alle zwölf JSON-Dateien, die für die Evaluation verwendet wurden, gelistet. Um die Evaluationsergebnisse zu speichern, muss der Nutzer bei der Ausführung der Ensemble-Klasse im Terminal lediglich noch `--eval` am Ende hinzufügen. Allgemein werden folgende drei Ensemble Konstellationen differenziert:

1. Ensemble mit CNNs, die nur auf Originaldaten trainiert wurde. Diese Ensembles werden als *Base* bezeichnet.
2. Ensemble mit CNNs, die neben den Originaldaten auch auf generierte Daten für die in [Abschnitt 4.6](#) genannten 13 Typen trainiert wurde. Diese Ensembles werden als *Generated* bezeichnet.
3. Ensemble mit allen trainierten CNNs. Diese Ensembles werden als *All* bezeichnet.

Innerhalb dieser Ensembles werden noch zwei Unterscheidungen gemacht, die ebenfalls den Datensatz, auf dem die Base Learner trainiert wurden betrifft:

1. Ensemble mit CNNs, die nur beide Seiten (*Both*) der Münze beim Training gesehen haben. Diese Ensembles erhalten den Zusatz *Both*.
2. Ensemble mit CNNs, die beide Seiten (*Both*), nur die Vorderseite (*Obv*) oder nur die Rückseite (*Rev*) der Münze beim Training gesehen haben. Diese Ensembles erhalten keinen Zusatz.

Eine weitere Unterscheidung wird mit dem verwendeten Testdatensatz gemacht. Es werden die folgenden zwei Datensätze verwendet:

1. *Test*: Testdatensatz, der für jeden der 207 Typen mindestens 3 Bilder enthält. Insgesamt sind 1040 Bilder in diesem Datensatz enthalten.
2. *Test13*: Testdatensatz, der nur für die in [Abschnitt 4.6](#) genannten 13 Typen mindestens 3 Bilder enthält. Insgesamt sind 56 Bilder in diesem Datensatz enthalten. Die Daten dieser 13 Typen entsprechen denen aus dem *Test* Datensatz.

Zunächst werden im Unterabschnitt [5.2.2.1](#) für alle sieben implementierten Kombinationsmöglichkeiten für die Top-1-Wahrscheinlichkeiten die $3 \cdot 2 \cdot 2 \cdot 7 = 84$ Fehlerraten gegenübergestellt, um herauszufinden, welches Ensemble die geringsten Fehler macht und welches Voting bzw. welches Stacking bei der Klassifizierung besser ist. Da für einige Voting Funktionen (alle außer `weighted_obv_rev_voting`) auch die Top-X Wahrscheinlichkeiten berücksichtigt werden können, wird auch der Einfluss der Top-3- bzw. Top-5-Wahrscheinlichkeiten für ein ausgewähltes Ensemble betrachtet.

Anschließend werden das beste Voting und das beste Stacking für die 13 Typen, die generierte Bilder erhalten haben, in Unterabschnitt [5.2.2.2](#) hervorgehoben. In diesem Fall wird die Sensitivität (*recall*) betrachtet. Dieser Wert zeigt für jeden der 13 Münztypen an, wie viele der Bilder im *Test13* Datensatz richtig klassifiziert wurden. Diese Evaluationsergebnisse werden anschließend mit dem besten Base Learner verglichen.

Alle Evaluationsergebnisse (CSV-Dateien und Konfusionsmatrizen) sind in dem beigefügten USB-Stick enthalten. Im Fokus dieser Evaluation liegt die Beantwortung folgender Fragen:

1. Welchen Einfluss hat die Anzahl der Base Learner auf ein Ensemble? Wie beeinflusst das Hinzufügen der Netze, die nur auf der Vorder- oder Rückseite (*Obv*, *Rev*) der Münzen trainiert wurden, das Ensemble?
2. Wie schneidet ein Ensemble, welches nur Originaldaten erhalten hat, gegenüber einem Ensemble, das für die 13 Typen auch generierte Daten erhalten hat, ab? Kann ein Ensemble, das neuronale Netze enthält, die auf einem balancierteren Datensatz trainiert wurden, eine geringere Fehlerrate erzielen?

3. Wie performen die verschiedenen Voting Funktionen? Kann das Voting verbessert werden, wenn die Top-3- oder Top-5-Wahrscheinlichkeiten berücksichtigt werden? Welche Voting Konstellation hat die geringste Fehlerrate?
4. Wie gut schneidet Stacking bei der Klassifizierung ab? Welches Meta Modell ist besser?
5. Wie schneidet das Ensemble Learning im Vergleich zu dem besten Base Learner ab?

5.2.2.1 Evaluation der Ensembles und Kombinationsmöglichkeiten mittels der Fehlerrate

In diesem Abschnitt wird die Fehlerrate der Ensembles für die unterschiedlichen Kombinationsmöglichkeiten auf dem *Test* und dem *Test13* Datensatz gegenübergestellt, um einen ersten Überblick zu erhalten, welche Konstellation die beste Klassifizierung macht. Je geringer die Fehlerrate ist, desto weniger Fehler macht ein Ensemble bei der Klassifizierung.

Als erstes wird die Evaluation auf den zwei *Base* Ensembles durchgeführt. Es wird zwischen dem Ensemble, das nur Netze enthält, das die Vorder- und Rückseite jeder Münze gesehen hat (vgl. *BaseBothTestPreds* und *BaseBothTest13Preds* in Tabelle 19), und dem Ensemble, das zusätzlich noch Netze enthält, das nur die Vorder- oder nur die Rückseite von den Münzen gesehen hat (vgl. *BaseTestPreds* und *BaseTest13Preds* in Tabelle 19), unterschieden. In Tabelle 23 sind die Fehlerraten für alle sieben Kombinationsmöglichkeiten für die Top-1-Wahrscheinlichkeiten gelistet.

	BaseBoth		Base	
	Test	Test13	Test	Test13
hard_voting	14,90%	26,79%	11,54%	23,21%
simple_soft_voting	15,10%	26,79%	11,92%	23,21%
weighted_soft_voting	15,10%	26,79%	11,63%	19,64%
weighted_pred_soft_voting	15,10%	26,79%	11,92%	23,21%
weighted_obv_rev_voting	15,10%	26,79%	11,54%	23,21%
stacking_with_RF	19,04%	28,57%	13,08%	28,57%
stacking_with_LR	16,35%	35,71%	10,19%	23,21%

Tabelle 23: Fehlerrate der *Base* Ensembles. Die beste Fehlerrate einmal für Voting und einmal für Stacking für jeden Datensatz ist grün markiert.

In Tabelle 24 ist die Fehlerrate für jede Kombinationsmöglichkeit für die Top-1-Wahrscheinlichkeiten und beide Testdatensätze für die *Generated* Ensembles zu sehen. Auch hier werden zwei Ensembles betrachtet. Das erste besteht aus Base Learnern, die auf dem beidseitigen Datensatz trainiert wurden (vgl. GeneratedBothTestPreds und GeneratedBothTest13Preds in Tabelle 19), und das zweite enthält CNNs, die zusätzlich nur auf der Vorder- oder Rückseite trainiert wurden (vgl. GeneratedTestPreds und GeneratedTest23Preds in Tabelle 19).

	GeneratedBoth		Generated	
	Test	Test13	Test	Test13
hard_voting	14,13%	26,79%	09,71%	21,43%
simple_soft_voting	14,23%	26,79%	09,81%	23,21%
weighted_soft_voting	14,23%	26,79%	10,38%	23,21%
weighted_pred_soft_voting	14,23%	26,79%	09,81%	23,21%
weighted_obv_rev_voting	14,23%	26,79%	09,90%	23,21%
stacking_with_RF	16,73%	25,00%	11,63%	25,00%
stacking_with_LR	14,81%	28,57%	10,87%	28,57%

Tabelle 24: Fehlerrate der *Generated* Ensembles. Die beste Fehlerrate einmal für Voting und einmal für Stacking für jeden Datensatz ist grün markiert.

Abschließend werden die *All* Ensembles ausgewertet (vgl. Tabelle 29), wobei auch hier zwischen den Base Learnern unterschieden wird, die nur auf dem *Both* Datensatz trainiert wurden (vgl. AllBothTestPreds und AllBothTest13Preds in Tabelle 19), und allen trainierten Base Learner (vgl. AllTestPreds und AllTest13Preds in Tabelle 19).

	AllBoth		All	
	Test	Test13	Test	Test13
hard_voting	13,56%	25,00%	09,52%	21,43%
simple_soft_voting	13,94%	23,21%	09,71%	23,21%
weighted_soft_voting	13,94%	23,21%	10,19%	21,43%
weighted_pred_soft_voting	13,94%	23,21%	09,71%	23,21%
weighted_obv_rev_voting	13,94%	23,21%	09,52%	23,21%
stacking_with_RF	15,77%	26,79%	10,87%	21,43%
stacking_with_LR	14,04%	25,00%	08,94%	21,43%

Tabelle 25: Fehlerrate der *All* Ensembles. Die beste Fehlerrate einmal für Voting und einmal für Stacking für jeden Datensatz ist grün markiert.

Hier sei noch anzumerken, dass das `weighted_pred_soft_voting` nur der Vollständigkeit halber in den Tabellen steht, da es bei den betrachteten Top-1-Wahrscheinlichkeiten dem `simple_soft_voting` entspricht. Außerdem entspricht bei den Ensembles mit dem *Both* Zusatz das `weighted_soft_voting` ebenfalls dem `simple_soft_voting`.

Beantwortung der 1. Frage:

Für jedes Ensemble (*Base*, *Generated*, *All*) wird ein besseres Ergebnis erzielt, wenn neben den *Both* Base Learner auch noch die Base Learner, die auf dem *Obv* oder *Rev* Datensatz trainiert wurden, hinzugefügt werden.

Ein möglicher Grund für dieses Ergebnis kann die steigende Anzahl der Base Learner im Ensemble sein. Ein weiterer Grund könnte das Hinzufügen der CNNs sein, die auf dem *Obv* und *Rev* Datensatz trainiert wurden. Um das zu untersuchen, sind in Tabelle 26 die Ensembles nach ihrer Größe sortiert. Für jedes Ensemble ist einmal die beste (+) und einmal die schlechteste (-) Fehlerrate, unabhängig der verwendeten Kombinationsmöglichkeit, gelistet.

	BaseBoth	Generated-Both	Base	AllBoth	Generated	All
Anzahl der Base Learner	3	6	7	9	14	21
Test	+	14,90%	14,13%	10,19%	13,56%	09,71%
	-	19,04%	16,73%	13,08%	15,77%	11,63%
Test13	+	26,79%	25,00%	19,64%	23,21%	21,43%
	-	35,71%	28,57%	28,57%	26,79%	23,21%

Tabelle 26: Beste (+) und schlechteste (-) Fehlerrate jedes Ensembles sortiert nach der Anzahl der Base Learner für beide Testdatensätze.

Auf dem gesamten Testdatensatz (*Test*) ist fast eine kontinuierliche Verbesserung der besten und schlechtesten Fehlerrate mit steigender Anzahl der Base Learner im Ensemble zu erkennen. Die einzige Ausnahme bildet das *AllBoth* Ensemble. Obwohl es zwei Base Learner mehr enthält als das *Base* Ensemble, ist es ca. 3 Prozentpunkte schlechter bei der Klassifizierung. Dies lässt vermuten, dass das Hinzufügen von Base Learnern, die nur auf dem *Obv* oder *Rev* Datensatz trainiert wurden, auch eine Verbesserung des Ensembles auf Grund der steigenden Diversität bedeutet.

Bei der Betrachtung des Testdatensatzes der 13 Typen (*Test13*) ist bei der besten Fehlerrate ebenfalls fast eine kontinuierliche Verbesserung bei steigender Base Learner Anzahl zu erkennen. Die einzige Ausnahme bildet das *Base* Ensemble. Dieses Ensemble hat auf dem *Test13* Datensatz die geringste Fehlerrate. Dieses Ergebnis bekräftigt die Vermutung, dass Ensembles mit Base Learnern, die auch auf dem *Obv* oder *Rev* Datensatz trainiert wurden, auf Grund der höhere Diversität die Münzbilder besser klassifizieren können.

Bei der Betrachtung des kleinsten Ensembles und des größten Ensembles fällt auf, dass die beste (+) Fehlerrate beim größeren Ensemble auf beiden Testdatensätzen um mehr als 5 Prozentpunkte besser ist. Bei der schlechtesten (–) Fehlerrate ist das größere Ensemble auf dem *Test* Datensatz mehr als 8 Prozentpunkte besser und auf dem *Test13* Datensatz sogar mehr als 12 Prozentpunkte. Dies zeigt deutlich, dass eine steigende Anzahl der Base Learner im Ensemble nicht nur die gesamte Vorhersageleistung verbessert, sondern auch die der kleinen Münztypen.

Beantwortung der 2. Frage:

Die *BaseBoth* und *GeneratedBoth* Ensembles haben auf beiden Testdatensätzen eine sehr ähnliche Fehlerrate. Das Voting erzielt hier bei beiden Ensembles auf dem *Test* Datensatz eine Fehlerrate von ca. 14 – 15% und auf dem *Test* Datensatz beträgt diese überall 26,79%. Beim Stacking sind die Unterschiede auf dem *Test* Datensatz ebenfalls minimal. Die Fehlerrate ist beim *GeneratedBoth* Ensemble um ca. 2 – 3 Prozentpunkte besser.

Auf dem *Test13* Datensatz hingegen sind die Unterschiede größer. Da erzielt das Stacking eine Verbesserung um ca. 3 – 7 Prozentpunkte. Dies ist auch der Datensatz, für den Bilder generiert wurden, weswegen die Balancierung der Daten eine Verbesserung erzielen könnte. Allerdings könnte auch die höhere Anzahl der Base Learner (drei CNNs mehr) im *GeneratedBoth* Ensembles diese minimal geringere Fehlerrate erklären.

Die *Base* und *Generated* Ensembles, bei denen CNNs hinzugefügt werden, die auf dem *Obv* und *Rev* Datensatz trainiert wurden, haben eine geringere Fehlerrate. Allerdings erzielen auch hier das Voting und Stacking auf dem gesamten Testdatensatz für beide Ensembles eine ähnliche Fehlerrate zwischen 9 – 13%, wobei das *Generated* Ensemble fast immer minimal besser ist (Außnahme: `stackin_with_LR`). Auch auf dem *Test13* Datensatz sind die Unterschiede in der Fehlerrate beim Voting nur zwischen 0 – 3 Prozentpunkte und beim Stacking zwischen 3 – 5. Das `weighted_soft_voting` auf dem *Base* Ensemble auf dem *Test13* Datensatz erzielt die insgesamt geringste Fehlerrate. Bei dieser Betrachtung kann keine Aussage darüber getroffen werden, ob ein Ensemble, das auf balancierteren Datensätzen trainierte Base Learner enthält, eine geringere Fehlerrate hat oder die teilweise minimalen Verbesserungen auf Grund der steigenden Anzahl der Base Learner ist.

Aus diesem Grund wird das gesamte *Generated* Ensemble in zwei Ensembles *GeneratedAll* und *GeneratedSub* aufgeteilt. Diese sind dann genau so groß wie das gesamte *Base* Ensemble. Das führt zu einer Ergänzung der in [Tabelle 19](#) aufgeführten JSON-Dateien. Die hinzugefügten Dateien sind in [Tabelle 27](#) gelistet:

JSON-Datei	Base Learner	Test-daten
GeneratedAllTest13Preds (7 Base Learner)	Alle Netze, die auf den <i>Both</i> , <i>Obv</i> oder <i>Rev</i> Datensatz mit generierten Bildern trainiert wurden. Der originale Trainingsdatensatz wurde hierbei beibehalten und um generierte Bilder erweitert.	Test13
GeneratedAllTestPreds (7 Base Learner)	Alle Netze, die auf den <i>Both</i> , <i>Obv</i> oder <i>Rev</i> Datensatz mit generierten Bildern trainiert wurden. Der originale Trainingsdatensatz wurde hierbei beibehalten und um generierte Bilder erweitert.	Test
GeneratedSubTest13Preds (7 Base Learner)	Alle Netze, die auf den <i>Both</i> , <i>Obv</i> oder <i>Rev</i> Datensatz mit generierten Bildern trainiert wurden. Der originale Trainingsdatensatz wurde hierbei verkleinert und um generierte Bilder erweitert.	Test13
GeneratedSubTestPreds (7 Base Learner)	Alle Netze, die auf den <i>Both</i> , <i>Obv</i> oder <i>Rev</i> Datensatz mit generierten Bildern trainiert wurden. Der originale Trainingsdatensatz wurde hierbei verkleinert und um generierte Bilder erweitert.	Test

Tabelle 27: Erweiterung der JSON-Dateien für das *Generated Ensemble*.

Für diese vier weiteren JSON-Dateien, ist in [Tabelle 28](#) für das `hard_voting`, das `weighted_soft_voting` und das `stacking_with_LR` die Fehlerrate zu sehen.

	GeneratedAll		GeneratedSub	
	Test	Test13	Test	Test13
<code>hard_voting</code>	11,73%	21,43%	12,21%	41,08%
<code>weighted_soft_voting</code>	11,63%	23,21%	12,79%	39,29%
<code>stacking_with_LR</code>	10,29%	25,00%	12,31%	33,93%

Tabelle 28: Fehlerrate des *GeneratedAll* und *GeneratedSub* Ensembles für zwei Voting Funktionen und ein Stacking Meta Modell.

Das *GeneratedAll* Ensemble unterscheidet sich vom *Base Ensemble* dahingehend, dass bei den CNNs die Trainingsdatensätze identisch geblieben sind und nur um generierte Bilder erweitert wurden. Diese Erweiterung führt zu einem balancierteren Datensatz, der eine bessere Klassifizierung hervorbringen könnte. Allerdings ist die Fehlerrate dieser zwei Ensembles auf beiden Testdatensätzen sehr ähnlich. Das heißt, die Balancierung hat zu keiner Verbesserung beigetragen. Bei der Betrachtung der Fehlerrate des

GeneratedSub Ensembles, das CNNs enthält, die mit weniger Originaldaten und mehr generierten Daten trainiert wurden, ist sogar eine deutliche Verschlechterung auf dem *Test13* Datensatz zu erkennen. Dies bestätigt, dass eine Balancierung des Datensatzes durch generierte Bilder keine Verbesserung der Gesamtleistung erzielt, und zeigt, dass die Klassifizierung für Münztypen mit wenig Originaldaten sehr fehlerhaft ist. Das lässt vermuten, dass Münztypen aus dem gesamten Datensatz, die weniger als 20 Bilder besitzen, nicht durch generierte Bilder in das Training aufgenommen werden können.

Beantwortung der 3. Frage:

Für jedes Ensemble und auf beiden Testdatensätzen unterscheiden sich die Fehlerraten für die verschiedenen Voting Funktionen gar nicht, oder lediglich um ca. 0,2 – 3,5 Prozentpunkte. Auf Grund dieser kaum nennenswerten Unterschiede soll im Folgenden der Einfluss der Top-3- bzw. Top-5-Wahrscheinlichkeiten untersucht werden. Hierfür werden für das *All* Ensemble mit den 21 Base Learnern alle Voting Funktionen (außer `weighted.obv.rev_voting`) auf beiden Testdatensätzen für die Top-3- und Top-5-Wahrscheinlichkeiten ausgewertet.

	All bei Top-1		All bei Top-3		All bei Top-5	
	Test	Test13	Test	Test13	Test	Test13
<code>hard_voting</code>	09,51%	21,43%	09,71%	28,57%	10,67%	33,93%
<code>simple_soft_voting</code>	09,71%	23,21%	10,00%	23,21%	10,10%	23,21%
<code>weighted_soft_voting</code>	10,19%	21,43%	10,48%	23,21%	09,62%	23,21%
<code>weighted_pred_soft_voting</code>	09,71%	23,21%	09,71%	23,21%	10,48%	23,21%

Tabelle 29: Fehlerrate des *All* Ensembles mit allen Base Learnern für das Top-1, Top-3 und Top-5 Voting.

Beim Vergleich der Fehlerraten des Votings für die Top-1-, Top-3- und Top-5-Wahrscheinlichkeiten fallen kaum Unterschiede auf. Auf dem *Test13* Datensatz wird das `hard_voting` zunehmend schlechter. Eine minimale Verschlechterung gibt es hier auch beim `weighted_soft_voting`, aber die anderen Voting Funktionen bleiben unverändert. Auch auf dem *Test* Datensatz unterscheiden sich die Fehlerraten kaum. Insgesamt ist das Voting mit den Top-1-Wahrscheinlichkeiten immer noch am besten.

Das beste Voting ist `hard_voting` oder `weighted.obv.rev_voting` mit dem *All* En-

semble, wenn der gesamte Testdatensatz betrachtet wird. Auf dem *Test13* Datensatz hingegen ist das `weighted_soft_voting` mit dem *Base* Ensemble am besten.

Auf Grund der sehr ähnlichen Fehlerraten der Voting Funktionen, lohnt es sich nicht so viele verschiedene Funktionen zu implementieren. Daher genügt es, als Kombinationsmöglichkeit einmal das `hard_voting` und einmal das `weighted_soft_voting` zu verwenden.

Beantwortung der 4. Frage:

Für das *Base* und das *All* Ensemble wird beim Stacking mit der logistischen Regression auf beiden Testdatensätzen ein besseres (oder gleiches) Ergebnis erzielt. Beim *Generated* Ensemble auf dem gesamten *Test* Datensatz erzielt ebenfalls die logistische Regression eine geringere Fehlerrate. Auf dem *Test13* Datensatz erzielt allerdings das Random Forest Meta Modell eine um mehr als 3 Prozentpunkte bessere Klassifizierung.

Das Training des Meta Modells wird auf der CPU des bereitgestellten Rechners durchgeführt. Hierbei handelt es sich um einen AMD Ryzen 9 5900X 12-Core Processor. Während die logistische Regression für das Training und die Evaluation nicht mal 10 Sekunden benötigt, braucht der Random Forest, je nach Ensemble, 10 bis 20 Minuten für das Training und nochmal 1 bis 5 Minuten für die Evaluation.

Da die logistische Regression so schnell ist und auf den meisten Ensembles und Testdatensätzen auch besser (oder gleich gut) performt, empfiehlt es sich, `stacking_with_LR` zu nutzen.

Beantwortung der 5. Frage:

In **Unterabschnitt 5.1.4** wurden die einzelnen Base Learner evaluiert. Auf dem gesamten Testdatensatz hat der beste Base Learner (`Res50_Both_Base_NoAug_FreezeConv4`) eine Fehlerrate von 18,85%. Mit dem Ensemble, das alle trainierten CNNs enthält, beträgt die Fehlerrate beim Stacking mit einer logistischen Regression lediglich 8,94%. Das ist eine Verbesserung der Fehlerrate um fast 10 Prozentpunkte. Auch für das beste Voting (`hard_voting` und `weighted_pred_soft_voting`) wird eine Verbesserung um fast 10 Prozentpunkte erzielt.

Auf dem Testdatensatz, der lediglich die 13 Typen enthält, beträgt die Fehlerrate des `Res50_Both_Base_NoAug_FreezeConv4` 26,79%. Stacking erzielt auf dem Ensemble, das alle trainierten CNNs enthält, für beide Meta Modelle eine Verbesserung der Fehlerrate um etwas mehr als 5 Prozentpunkte. Sowohl für die logistische Regression, als auch für den Random Forest beträgt die Fehlerrate auf dem Ensemble mit allen Base Learnern 21,43%. Das beste Voting (`weighted_soft_voting`) erzielt das *Base Ensemble* (*Both*, *Obv*, *Rev*) mit einer Fehlerrate von 19,64%. Dies ist eine Verbesserung um mehr als 7 Prozentpunkte. Die besseren Fehlerraten zeigen, dass mit Ensemble Learning die Klassifizierung von Münztypen verbessert werden kann.

5.2.2.2 Evaluation ausgewählter Ensembles und Kombinationsmöglichkeiten für die 13 ausgewählten Typen mittels der Sensitivität

Der vorherige Abschnitt hat einen Überblick über die besten Ensembles und Kombinationsmöglichkeiten gegeben. In diesem Abschnitt sollen die 13 Typen, für die Bilder generiert wurden, genauer betrachtet werden. Da die Betrachtung von 13 Typen für alle Ensembles und alle Kombinationsmöglichkeiten sehr zeitaufwendig und wenig zielführend wäre, werden in diesem Abschnitt folgende Ensembles, Kombinationsmöglichkeiten und Base Learner auf dem *Test13* Datensatz miteinander verglichen:

- Bester Base Learner: Das `Res50_Both_Base_NoAug_FreezeConv4` hat die beste Leistung von allen Base Learnern.
- Beste Kombinationsmöglichkeit beim *Base Ensemble*: Das `weighted_soft_voting` auf dem *Base Ensemble* (*Both*, *Obv*, *Rev*) erreicht auf dem *Test13* Datensatz die geringste Fehlerrate.
- Größtes Ensemble: Das *All Ensemble* mit allen trainierten Base Learnern ist das größte Ensemble. Hierfür werden sein bestes Voting (`hard_voting` bei Top-1) und sein bestes Stacking (`stacking_with_LR`) betrachtet.

Die Fehlerrate ist ein gutes Maß, um einen ersten Eindruck über ein Modell zu erhalten. Je kleiner ihr Wert ist, desto weniger Fehler macht dieses Modell. Allerdings kann die Fehlerrate bei unbalancierten Datensätzen irreführend sein. Daher wurde neben

der Fehlerrate auf dem gesamten Testdatensatz auch die Fehlerrate auf dem *Test13* Datensatz betrachtet. Es fällt direkt auf, dass die Modelle auf diesem kleinen Datensatz viel mehr Fehler machen (höhere Fehlerrate). Allerdings ist nicht ersichtlich, wo genau die Fehler passieren. Daher wird im Folgenden für die vier oben genannten Modelle die Sensitivität dieser 13 Typen untersucht. Dieser Wert gibt an, wie viele Bilder dieses Typs tatsächlich richtig erkannt wurden. Beträgt dieser Wert bei einem Typ 1 sind alle Bilder des Types im *Test13* Datensatz richtig klassifiziert worden. In den Tabellen [\[30\]](#), [\[31\]](#), [\[32\]](#) und [\[33\]](#) sind die Typen, auf die das zutrifft, grün markiert.

In [Tabelle 30](#) ist die Sensitivität für jeden der 13 Münztypen für den besten Base Learner zu sehen. Das `Res50_Both_Base_NoAug_FreezeConv4` klassifiziert alle Bilder des Trainingsdatensatzes für die Münztypen 946, 5944, 6057, 7967, 12783 und 20230 richtig.

Bester Base Learner: <code>Res50_Both_Base_NoAug_FreezeConv4</code> (Fehlerrate: 26,79%)													
Typ	294	940	946	5135	5944	6057	7697	12752	12783	20230	20545	20809	21027
Sensitivität	0.0	0.6667	1.0	0.6667	1.0	1.0	1.0	0.3333	1.0	1.0	0.6667	0.75	0.4
# Bilder	3	3	3	3	3	3	10	3	5	3	3	4	10

Tabelle 30: Sensitivität des `Res50_Both_Base_NoAug_FreezeConv4` auf dem *Test13* Datensatz. Werte sind auf vier Nachkommastellen gerundet. Grün markierte Typen wurden komplett richtig klassifiziert.

In [Tabelle 31](#) ist zu sehen, dass das `weighted_soft_voting` mit dem *Base Ensemble* ebenfalls die Bilder dieser sechs Münztypen richtig zuordnet. Weiterhin werden die Münztypen 940, 12752 und 20545 komplett richtig klassifiziert. Das Ensemble trägt hier zu einer besseren Klassifizierung und einer geringeren Fehlerrate bei. In diesem Ensemble sind neben dem besten Base Learner (`Res50_Both_Base_NoAug_FreezeConv4`) noch sechs weitere CNNs enthalten, die nur auf den Originaldaten trainiert wurden. Außerdem sind dort auch Netze enthalten, die nur auf dem *Obv* oder *Rev* Datensatz trainiert wurden. Dadurch könnte sowohl die Nutzung eines Ensembles, als auch die Diversität im Ensemble, zu einer verbesserten Leistung beitragen.

Beste Kombinationsmöglichkeit: <code>weighted_soft_voting</code> mit <i>Base Ensemble</i> (Fehlerrate: 19,64%)													
Typ	294	940	946	5135	5944	6057	7697	12752	12783	20230	20545	20809	21027
Sensitivität	0.3333	1.0	1.0	0.6667	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.75	0.3
# Bilder	3	3	3	3	3	3	10	3	5	3	3	4	10

Tabelle 31: Sensitivität des `weighted_soft_voting` mit dem *Base Ensemble* auf dem *Test13* Datensatz. Werte sind auf vier Nachkommastellen gerundet. Grün markierte Typen wurden komplett richtig klassifiziert.

In [Tabelle 32](#) ist die Sensitivität jedes Münztyps für das `hard_voting` mit dem *All Ensemble* aufgeführt. Hier werden ebenfalls Bilder von neun verschiedenen Münztypen vollständig richtig zugeordnet. Im Vergleich zum `Res50_Both_Base_NoAug_FreezeConv4` werden zusätzlich die Münztypen 940, 5135, 20545 und 20809 richtig klassifiziert. Allerdings wird vom Münztyp 7697 eine Münze ($Sensitivität = 0,9 = \frac{RP}{RP+FN} = \frac{9}{10}$) falsch zugeordnet. Dieser Münztyp wird im Anschluss noch genauer betrachtet. Insgesamt erzielt auch hier das Ensemble und die Diversität der Base Learner eine Verbesserung in der Klassifizierung.

Bestes Ensemble mit Voting: <code>hard_voting</code> mit <i>All Ensemble</i> (Fehlerrate: 21,43%)													
Typ	294	940	946	5135	5944	6057	7697	12752	12783	20230	20545	20809	21027
Sensitivität	0.0	1.0	1.0	1.0	1.0	1.0	0.9	0.6667	1.0	1.0	1.0	1.0	0.3
# Bilder	3	3	3	3	3	3	10	3	5	3	3	4	10

Tabelle 32: Sensitivität des `hard_voting` mit dem *All Ensemble* auf dem *Test13* Datensatz. Werte sind auf vier Nachkommastellen gerundet. Grün markierte Typen wurden komplett richtig klassifiziert.

In [Tabelle 33](#) sind die Ergebnisse für das `stacking_with_LR` mit dem *All Ensemble* zu sehen. Es werden Bilder von elf Münztypen vollständig richtig zugeordnet. Hierbei handelt es sich um alle Münztypen außer 12752 und 21027. Diese Münztypen werden im Anschluss weiter untersucht. Weiterhin fällt auf, dass der Münztyp 294 bei der logistischen Regression am besten erkannt wird. Auch dieser Münztyp wird im Anschluss noch genauer betrachtet. Insgesamt führt das Nutzen des Ensembles und die Diversität zu einer Verringerung der Fehlerrate um mehr als 5 Prozentpunkte.

Bestes Ensemble mit Stacking: <code>stacking_with_LR</code> mit <i>All</i> Ensemble (Fehlerrate: 21,43%)													
Typ	294	940	946	5135	5944	6057	7697	12752	12783	20230	20545	20809	21027
Sensitivität	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.3333	1.0	1.0	1.0	1.0	0.0
# Bilder	3	3	3	3	3	3	10	3	5	3	3	4	10

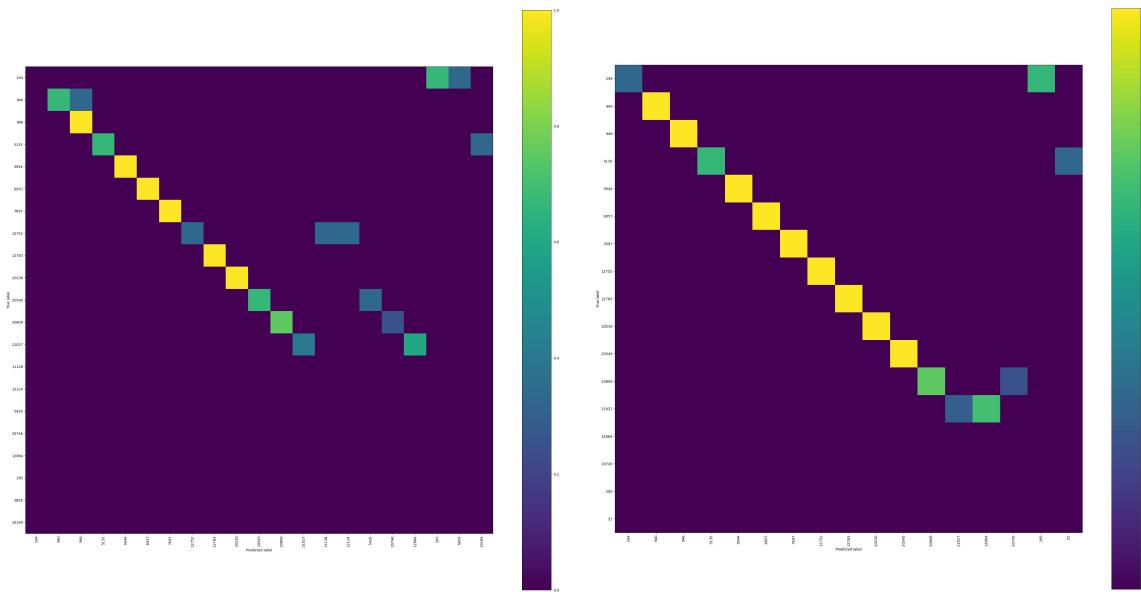
Tabelle 33: Sensitivität des `stacking_with_LR` mit dem *All* Ensemble auf dem *Test13* Datensatz.
Werte sind auf vier Nachkommastellen gerundet. Grün markierte Typen wurden komplett richtig klassifiziert.

Was bisher in diesen Vergleichen nicht beachtet wurde, sind die Münztypen, bei denen die Bilder nicht alle richtig klassifiziert wurden. Hier fällt vor allem auf, dass die Münztypen 294 und 21027, bei (fast) allen Konstellationen sehr schlecht erkannt wurden.

Weiterhin klassifiziert jede der hier betrachteten Konstellationen die Bilder des Münztyps 7697 richtig, außer das `hard_voting` mit dem *All* Ensemble. Daher wird auch dieser Münztyp kurz betrachtet. Der Münztyp 12752 wird nur von `weighted_soft_voting` mit dem *Base* Ensemble komplett richtig erkannt, weswegen auch dieser Typ weiter untersucht wird.

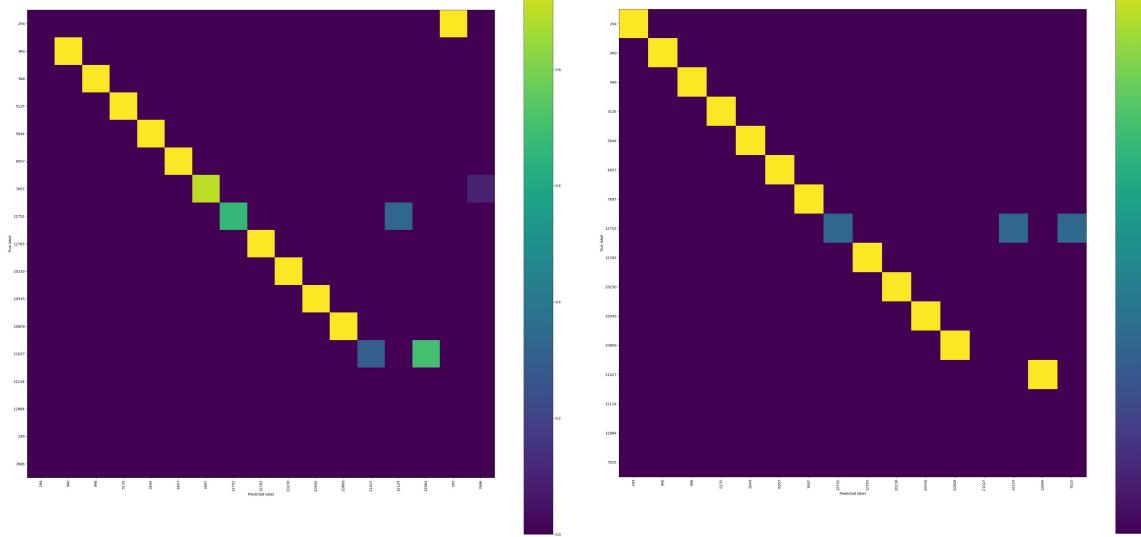
Insgesamt werden im Folgenden diese vier Münztypen (294, 7697, 12752 und 21027) mit Hilfe der Konfusionsmatrizen genauer analysiert.

In Abbildung 43 sind die Konfusionsmatrizen für diese vier betrachteten Konstellationen zu sehen. Auf der x-Achse der Matrix befinden sich die vorhergesagten Typen (*Predicted label*) und auf der y-Achse befinden sich die tatsächlichen Typen (*True label*). Die grün markierten Typen aus den Tabellen 30, 31, 32 und 33 sind in der Matrix durch die gelben Felder auf der Diagonalen zu erkennen. Die ersten 13 Typen sind die betrachteten 13 Typen in aufsteigender Sortierung. Anschließend folgen alle Typen, die fälschlicherweise vorhergesagt wurden. Dadurch kann dieser Matrix die Information entnommen werden, zu welchem Typ ein Bild fälschlicherweise klassifiziert wurde.



(a) Konfusionsmatrix vom
Res50_Both_Base_NoAug_FreezeConv4

(b) Konfusionsmatrix vom
weighted_soft_voting



(c) Konfusionsmatrix vom **hard_voting**

(d) Konfusionsmatrix vom **stacking_with_LR**

Abbildung 43: Konfusionsmatrizen der betrachteten Modelle.

Die Typen sind in Abbildung 43 nicht gut erkennbar, daher ist in Tabelle 34 für jede Matrix eine verkleinerte Matrix angegeben, die nur die tatsächlichen Typen 294, 7697, 12752 und 21027 hervorhebt.

		Vorhergesagte Klasse								
		294	7697	12752	21027	285	5859	11128	12124	12884
Tatsächliche Klasse	294	0				2	1			
	7697		10							
	12752			1				1	1	
	21027				4					6

(a) Konfusionsmatrix vom `Res50_Both_Base_NoAug_FreezeConv4`

		Vorhergesagte Klasse					
		294	7697	12752	21027	285	12884
Tatsächliche Klasse	294	1				2	
	7697		10				
	12752			3			
	21027				3		7

(b) Konfusionsmatrix vom `weighted_soft_voting` mit dem *Base* Ensemble

		Vorhergesagte Klasse						
		294	7697	12752	21027	285	7686	12124
Tatsächliche Klasse	294	0			3			
	7697		9			1		
	12752			2			1	
	21027				3			7

(c) Konfusionsmatrix vom `hard_voting` mit dem *All* Ensemble

		Vorhergesagte Klasse						
		294	7697	12752	21027	7025	12124	12884
Tatsächliche Klasse	294	3						
	7697		10					
	12752			1		1	1	
	21027				0			10

(d) Konfusionsmatrix vom `stacking_with_LR` mit dem *All* Ensemble

Tabelle 34: Verkleinerte Konfusionsmatrizen der betrachteten Modelle für die Typen 294, 7697, 12752 und 21027. Alle leeren Zellen entsprechen dem Wert 0.

Der häufigste Fehler beim Münztyp 294 ist die fälschliche Vorhersage zum Münztyp 285. In [Abbildung 25](#) ist einmal eine Beispielmünze des Typs 294 und einmal des Typs 285 zu sehen. Bei der Betrachtung dieser zwei Beispielmünzen sind viele Ähnlichkeiten zu erkennen, weswegen diese falsche Klassifizierung nicht sehr überraschend ist und bereits in Unterabschnitt [5.1.4.1](#) evaluiert wurde. In den Tabellen [11](#), [12](#), [16](#) und [17](#) ist die Sensitivität dieses Münztyps für die verschiedenen Base Learner zu sehen. Die meisten Netze haben eine Sensitivität von 0 für den Typ 294. Von den insgesamt 21 CNNs klassifizieren sieben CNNs einen Teil der Bilder dieses Münztyps richtig. Dieser Fehler bleibt auch beim Ensemble Learning bestehen, wobei dieser auf Grund der Di-

versität der Base Learner vor allem beim Stacking und bei einigen Voting Funktionen verkleinert wird.

Auch die fehlerhafte Klassifizierung beim Münztyp 7697 (vgl. Abbildungen 32 und 33) wurde für die Base Learner bereits in 5.1.4.2 thematisiert. Während bei den Base Learnern die Sensitivität für diesen Typ zwischen 0 (z.B. bei ResNet50 auf dem Train-SubGenerated) und 1 (z.B bei ResNet50 auf den Originaldaten) schwankt, erreicht das Ensemble Learning mindestens eine Sensitivität von 0,8.

Beim Vergleich des Münztyps 12752 in den Tabellen 30, 31, 32 und 33 fällt auf, dass nur das `weighted_soft_voting` mit dem *Base* Ensemble alle Bilder des Testdatensatzes richtig klassifiziert. Die hier betrachteten Konstellationen ordnen Bilder dieses Typs fälschlicherweise den Typen 7025, 11128 und 12124 zu. In Abbildung 44 sind Beispilmünzen der Typen 12752, 7025, 11128 und 12124 zu sehen.



Abbildung 44: Beispilmünzen der Typen 12752, 7025, 11128 und 12124 mit guter Qualität.

Die einzige Gemeinsamkeit, die diese Münztypen haben, ist das nach rechts gewandte Profil einer Person, das sich auf der Vorderseite befindet. Auf der Rückseite ist für jede dieser Münztypen ein anderes Motiv zu sehen. Was diese Datensätze aber alle gemeinsam haben, ist die teilweise schlechte Qualität der Münzbilder, die in Abbildung 45 beispielhaft zu sehen ist.



Abbildung 45: Beispilmünzen der Typen 12752, 7025, 11128 und 12124 mit schlechter Qualität.

Der originale Trainingsdatensatz des Typs 12752 besteht nur aus 16 Bildern, wobei die Hälfte Gipsabdrücke sind. Der Typ 7025 hat 25 Münzen, von denen 13 Gipsabdrücke sind. Der Typ 11128 hat 34 Bilder, von denen 14 Gipsabdrücke sind. Und beim Typen 12124 sind nur acht der insgesamt 58 Bilder Gipsabdrücke.

Die generierten Münzen des Typs 12752 enthalten sowohl gipsähnliche, als auch metallähnliche Münzen. Einige Beispiele sind in Abbildung 46 zu sehen.



Abbildung 46: Beispiele der generierten Münzbilder des Typen 12752.

Bei den generierten Bildern sind die Ansätze des Motivs der Vorder- und Rückseite des Typs 12752 zu erkennen. Es fällt aber auf, dass die verschiedenen Materialien bei der Bildgenerierung teilweise verschmelzen.

Der Testdatensatz für diesen Typ besteht aus drei Bildern, von denen nur eine Münze ein Gipsabdruck ist. In Tabelle 34 sind drei der fünf falschen Vorhersagen bei diesem Typen zum Typ 12124 klassifiziert worden, was am hohen Metallanteil der Münzen im Trainingsdatensatz liegen könnte.

Beim Durchgehen der Sensitivität dieses Münztyps auf den anderen Kostellationen, die sich auf dem beigefügten USB-Stick befinden, fällt auf, dass auch für andere Kombinationsmöglichkeiten mit dem *Base Ensemble* als einziges die Sensitivität von 1 erreicht wird. Bei der Betrachtung der Sensitivität der einzelnen Base Learner für den Typ 12752 (vgl. Tabellen 11, 12, 16 und 17), fällt auf, dass die meisten Modelle eine niedrige Sensitivität haben. Lediglich einige Modelle, die ausschließlich auf den Originaldaten trainiert wurden, erreichen eine höhere Sensitivität. Diese ganzen Modelle sind im *Base Ensemble* enthalten, wodurch hier manchmal eine Sensitivität von 1 erreicht wird. Ins- gesamt scheinen hier die generierten Bilder sogar zu einer Verschlechterung zu führen.

Bilder des Münztyps 21027 werden häufig fälschlicherweise dem Typ 12884 zugeordnet.

Die möglichen Gründe hierfür wurden bereits in Unterabschnitt 5.1.4.1 thematisiert.

Beim Vergleich der Sensitivität des Münztyps 21027 bei den verschiedenen Modellen fällt auf, dass diese im Ensemble sogar schlechter wird. Während der beste Base

Learner noch eine Sensitivität von 0,4 erreicht (vgl. Tabelle 11), erreichen die meisten Ensembles schlechtere Werte. Nur beim *Base* Ensemble erreichen einige Kombinationsmöglichkeiten auch den Wert von 0,4. Die anderen beiden Ensembles enthalten auch CNNs, die mit generierten Bildern trainiert wurden. Bei der Betrachtung der Sensitivität dieser CNNs in [11, 12, 16] und [17] fällt auf, dass diese häufig einen niedrigen Wert von 0,2 haben. Insgesamt scheinen auch hier die generierten Bilder zu einer Verschlechterung zu führen.

Insgesamt ist zu erkennen, dass das Ensemble Learning auf den meisten der 13 Münztypen eine Verbesserung der Sensitivität erzielt. Da es sich bei den Originaldaten hauptsächlich um kleine Typen handelt, ist das Ensemble hilfreich um mit unbalancierten Datensätzen umzugehen.

5.2.2.3 Fazit

Zusammenfassend kann gesagt werden, dass mehr Base Learner im Ensemble die Vorhersagelsetzung eines Modells verbessern. Weiterhin erhöhen CNNs, die nur auf der Vorder- oder Rückseite trainiert wurden, die Diversität des Ensembles, was zu einer besseren Gesamtperformance beiträgt.

Die Balancierung der Datensätze durch generierte Bilder im Ensemble hat wahrscheinlich keinen Einfluss auf die Vorhersageleistung. Das Ensemble, das mit generierten Bildern trainierte CNNs enthält, ist auf dem gesamten Testdatensatz bis auf eine Ausnahme immer minimal besser als das *Base* Ensemble, allerdings sind die Ergebnisse auf dem *Test13* Datensatz bei beiden Ensembles sehr ähnlich. Die teilweise minimale Verbesserung kommt wahrscheinlich von der steigenden Base Learner Anzahl beim gesamten *Generated* Ensemble. Bei der Aufteilung dieses Ensembles in zwei gleich Große Ensembles, *GeneratedAll* und *GeneratedSub*, hat das *GeneratedAll* eine ähnliche Fehlerrate wie das *Base* Ensemble und das *GeneratedSub* hat eine deutlich höhere Fehlerrate. Das heißt, das Hinzufügen generierter Bilder führt zu keiner Verbesserung der Klassifizierung.

Die Fehlerrate der fünf verschiedenen Voting Funktionen ist auf jedem Ensemble sehr ähnlich. Auch die Betrachtung verschiedener Top-X Wahrscheinlichkeiten hat keine Ver-

besserung gezeigt. Daher folgt aus unseren Resultaten die Empfehlung, das Voting einfach zu halten und lediglich `hard_voting` und `weighted_soft_voting` zu nutzen.

Beim Stacking ist das Meta Modell mit der logistischen Regression nicht nur deutlich schneller trainiert, es hat in den meisten Fällen auch eine geringere Fehlerrate, weswegen sich diese Implementierung empfiehlt.

Insgesamt erzielt das Ensemble Learning eine Verbesserung der Klassifizierung, was auch bei der Evaluation auf den 13 Typen zu sehen ist.

6 Vergleich von Bildgenerierungsmethoden: GAN vs. Stable Diffusion

Parallel zu dieser Arbeit hat sich Cahide Heidemann [49] ebenfalls mit der Generierung von antiken Münzbildern beschäftigt. Der Unterschied lag bei der verwendeten Methode. Während die Autoren dieser Arbeit, Emmanuela Georgoula und Robin Krause, Bilder mit Hilfe von Few-Shot GANs generiert haben, hat Cahide Heidemann Bilder mit Hilfe von Stable Diffusion erzeugt. In diesem Abschnitt soll kurz verglichen werden, ob eine Methode der Bildgenerierung besser funktioniert als die andere, bzw. welche Gemeinsamkeiten oder Unterschiede bei der Klassifizierung auffallen.

6.1 Vorbereitung der Trainingsdaten

Sowohl Cahide Heidemann als auch die Autoren dieser Arbeit haben für die in [Abschnitt 4.6](#) genannten 13 Typen Bilder generiert. Für die Klassifizierung der Münztypen werden in dieser Arbeit CNNs verwendet, die ein Ensemble bilden sollen. Hierfür wurden Netze ohne generierte Bilder (nur Originaldaten) sowie mit generierten Bildern trainiert (vgl. [Abschnitt 5.1](#)). Es wurden zwei Trainingsdatensätze mit generierten Bildern erstellt, das *TrainWithGenerated* (vgl. [Tabelle 4](#)) und das *TrainSubGenerated* (vgl. [Tabelle 5](#)). Diese zwei Ansätze den Trainingsdatensatz mit generierten Bildern zu ergänzen wurden auch für die von Cahide Heidemann generierten Bilder angewendet. Das *TrainAllWithCahides* entspricht dem originalen Trainingsdatensatz, wobei für die 13 Typen so viele Bilder ergänzt werden, bis alle Typen 50 Bilder enthalten. Zwei Münztypen (7697 und 21027) bilden hier eine Ausnahme, da weiterhin mindestens 20 generierte Bilder hinzugefügt werden sollten. In dem *TrainSubWithCahides* wurden die Originaldaten in den 13 Münztypen minimiert und anschließend mit 20 bis 45 Bildern aufgefüllt.

Bevor dies getan werden konnte, musste der Datensatz mit den generierten Bildern angepasst werden. Im bereitgestellten Datensatz waren Bilder enthalten, die im Rahmen dieser Arbeit im Validations- oder Testdatensatz genutzt wurden. Damit für das Meta

Modell im Stacking weiterhin die Validationsdaten verwendet und damit alle Modelle am Ende auf dem gleichen Testdatensatz evaluiert und verglichen werden können, wurden alle Bilder gelöscht, die aus Münzen aus diesen zwei Datensätzen generiert wurden. Weitere generierte Münzbilder, die gelöscht wurden, waren zum einen Bilder mit der Münze cn_coin_41841, die fälschlicherweise in dem Datensatz des Typs 5135 enthalten ist, und zum anderen schwarze Bilder.

6.2 Training und Evaluation neuer Base Learner

Im Folgenden werden die drei verwendeten ResNet-Architekturen (vgl. Unterabschnitt 5.1.3) mit den zwei erstellten Trainingsdatensätzen auf dem *Both* Datensatz trainiert. Die CNNs auf den zwei Trainingsdatensätze wurden wie in [Unterabschnitt 5.1.2](#) beschrieben trainiert. Dadurch ist eine Vergleichbarkeit mit den CNNs der Autoren gegeben. Es wurden jeweils wieder zehn Trainingsvorgänge durchgeführt und das CNN gespeichert, welches auf dem Testdatensatz die geringste Fehlerrate hat.

Auf dem Testdatensatz wurden gleich gute Ergebnisse wie in [\[8\]](#) erzielt. Deutliche Vorteile und Nachteile in der Performance lassen sich hier nicht erkennen.

Model	ResNet50			ResNet101			ResNet152		
	Base	TrainSubWithCahides	TrainAllWithCahides	Base	TrainSubWithCahides	TrainAllWithCahides	Base	TrainSubWithCahides	TrainAllWithCahides
Fehlerrate - Mean	20,90%	21,52%	21,26%	21,35%	22,01%	24,72%	21,49%	22,00%	23,08%
Fehlerrate - Min.	18,85%	20,48%	20,39%	19,90%	20,19%	21,87%	20,00%	20,38%	20,48%
Fehlerrate - Max.	22,60%	22,78%	23,17%	22,98%	24,52%	27,56%	22,88%	23,65%	24,90%
Fehlerrate - σ	1,02%	0,7%	0,93%	0,86%	1,2%	1,74%	0,77%	0,97%	1,12%

Tabelle 35: Evaluation auf dem Testdatensatz. Werte der Base CNNs als Vergleichswert angegeben.

Die größten Unterschiede bei der Performance auf den 13 Typen ist die zwischen CNNs, welche auf *TrainSubWithCahides* und *TrainSubGenerated* trainiert wurden (Vgl. [Tabelle 10](#)). Die Unterschiede in der durchschnittlichen Fehlerrate betragen über 10 Prozentpunkte und bei der ResNet152 Architektur ist sogar ein Unterschied von 16 Prozentpunkten in der minimalen Fehlerrate auszumachen. Die generierten Bilder, die mittels Stable Diffusion erzeugt wurden, scheinen deutlich besser für das Training geeignet zu sein.

Model	ResNet50			ResNet101			ResNet152		
	Base	TrainSubWithCahides	TrainAllWithCahides	Base	TrainSubWithCahides	TrainSubWithCahides	Base	TrainSubWithCahides	TrainAllWithCahides
Fehlerrate - Mean	31,82%	43,83%	31,49%	34,42%	46,10%	31,33%	31,01%	41,40%	32,14%
Fehlerrate - Min.	26,79%	37,50%	28,57%	30,36%	39,29%	25,00%	26,79%	30,36%	28,57%
Fehlerrate - Max.	37,50%	48,83%	37,50%	39,29%	55,36%	35,71%	33,93%	46,43%	37,50%
Fehlerrate - σ	3,18%	3,77%	2,56%	2,66%	5,4%	3,69%	2,56%	4,57%	3,39%

Tabelle 36: Fehlerrate der CNNs auf dem *Test13* Datensatz. Werte der Base CNNs als Vergleichswert angegeben.

Zwischen der Sensitivität der CNNs, die auf dem Trainingsdatensatz *TrainWithGenerated* und *TrainAllWithCahides* trainiert wurden, sind ebenfalls keine großen Unterschiede zu erkennen.

Typ	294	940	946	5135	5944	6057	7697	12752	12783	20230	20545	20809	21027
ResNet50 TrainAllWithCahides	0,0	0,6667	1,0	0,6667	1,0	0,6667	0,9	0,6667	0,8	1,0	0,6667	1,0	0,3
ResNet101 TrainAllWithCahides	0,0	0,6667	0,6667	1,0	1,0	0,6667	0,9	0,3333	1,0	0,3333	1,0	1,0	0,3
ResNet152 TrainAllWithCahides	0,0	0,6667	1,0	1,0	1,0	1,0	0,9	0,6667	0,4	0,6667	0,3333	1,0	0,2
ResNet50 TrainSubWithCahides	0,0	0,6667	0,6667	0,6667	1,0	0,6667	0,4	0,3333	1,0	0,6667	1,0	1,0	0,3
ResNet101 TrainSubWithCahides	0,0	0,6667	0,6667	0,6667	1,0	0,6667	0,4	0,6667	0,4	0,3333	1,0	1,0	0,3
ResNet152 TrainSubWithCahides	0,0	0,6667	0,3333	1,0	1,0	0,333	0,4	0,3333	0,8	0,6667	0,6667	1,0	0,4
#Bilder im Testdatensatz	3	3	3	3	3	3	10	3	5	3	3	4	10

Tabelle 37: Sensitivität der 13 Typen auf dem *Base* und *TrainWithGen* Datensatz.

Die Performance beim Typen 294 war in allen CNNs sehr schlecht. Dies ist verwunderlich, da die generierten Bilder eine deutlich bessere Qualität aufweisen. Die Rückseite ist viel besser erkennbar, auch wenn es Münzen ohne gute Rückseite gibt. Hier konnte durch eine bessere Bildqualität keine Steigerung erreicht werden.



Abbildung 47: Generierte Bilder des Typs 294 mit Stable Diffusion.

Bei den Münztypen 5135 und 20809 ist die Performance sehr gut sowohl in *TrainSubWithCahides* und *TrainAllWithCahides*. Dies ist vermutlich auf die gute Bildqualität zurückzuführen. Die Bilder haben eine deutlich höhere Qualität als die der FSGANs.



(a) Typ 5135

(b) Typ 5135

(c) Typ 20809

(d) Typ 20809

Abbildung 48: Generierte Bilder der Typs 5135 und 20809 mit Stable Diffusion.

Die CNNs des Trainingdatensatz *TrainAllWithCahides* haben bei den restlichen Typen eine ähnliche Performance wie die CNNs der Trainingsdatensätze *Base* und *TrainWithGenerated*.

Die CNNs des Trainingdatensatz *TrainSubWithCahides* wurden ebenfalls auf dem Testdatensatz evaluiert, zu dem die entfernten Münzen hinzugefügt wurden. Hier ist ebenfalls zu beachten, dass Münzen aus diesem Testdatensatz im Trainingdatensatz der Stable Diffusion enthalten sein können.

Typ	294	940	946	5135	5944	6057	7697	12752	12783	20230
ResNet50	0,25	0,75	0,4286	0,8667	1,0	0,9	0,4615	0,7143	0,8235	0,75
ResNet101	0,25	0,75	0,7143	0,8667	0,9167	0,9	0,4231	0,5714	0,7059	0,6667
ResNet152	0,25	0,87	0,5714	0,9334	1,0	0,8	0,4231	0,6429	0,7647	0,6667
#Bilder im Testdatensatz	8	8	7	15	12	10	104	14	17	12

Tabelle 38: Sensitivität der einzelnen ResNet. Ausgewertet auf dem *Test13* Datensatz erweitert mit den entfernten Münzbildern. Höhere Werte sind besser.

Im Vergleich zu [Tabelle 13](#) ist der größte Unterschied bei der Performance der Münze 7697. Hier konnte eine höhere Sensitivität erreicht werden. Bei der Betrachtung der generierten Bilder fällt auf, dass auch viele in schlechter Qualität darstellen (Vgl. [Abbildung 49](#)). Dies könnte ein entscheidender Vorteil für die richtige Klassifizierung sein. Es zeigt auf, dass die CNNs Beispiele von schlecht erhaltenen Münzen benötigen, um solche dem richtigen Typen zuzuordnen.

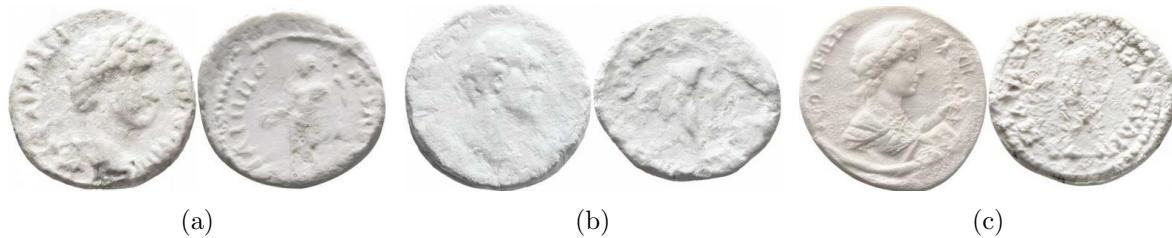


Abbildung 49: Generierte Bilder des Typs 7697 mit Stable Diffusion.

Die Bilder der FSGANs haben nur einen Vorteil bei den Typen 940 und 946 erbracht (Vgl. Tabelle 13). Dies könnte an teils schlecht generierten Bildern von der Stable Diffusion liegen.



(a) Typ 940

(b) Typo 946

Abbildung 50: Generierte Bilder des Typs 940 und 946 mit Stable Diffusion.

Insgesamt können die Bilder der Stable Diffusion eine deutliche höhere Qualität aufweisen als die Bilder der FSGANs. Die Bilder sind deutlich detaillierter und bilden auch die Feinheiten der Münzen besser ab. Dafür werden im Gegensatz zum FSGAN aber auch Bilder erstellt, die nicht dem ursprünglichen Motiv entsprechen. Die Bilder des FSGAN sind in der Qualität deutlich einheitlicher.

Die Stable Diffusion scheint sich durch das leicht verbesserte Ergebnis auch eher für die Bildgenerierung zu eignen. Jedoch müssen die generierten Bilder nochmals sortiert werden, sodass nur Bilder von guter Qualität für das Training genutzt werden.

6.3 Evaluation neuer Ensembles

Im Folgenden werden mit den in Abschnitt 6.2 neu trainierten Base Learner fünf neue Ensembles gebildet:

1. *GeneratedSDBoth* : Enthält alle neu trainierten Base Learner. Insgesamt sind 6 Base Learner enthalten.
2. *GeneratedSDAllBoth* : Enthält alle neu trainierten Base Learner, die auf dem *TrainAllWithCahides* Datensatz trainiert wurden. Insgesamt sind 3 Base Learner enthalten.
3. *GeneratedSDSubBoth* : Enthält alle neu trainierten Base Learner, die auf dem *TrainSubWithCahides* Datensatz trainiert wurden. Insgesamt sind 3 Base Learner enthalten.

4. *AllNew*: Das *All* Ensemble wird um alle neu trainierten Base Learner erweitert.
Insgesamt sind 27 Base Learner enthalten.
5. *AllNoSub*: Das *AllNew* Ensemble wird um alle Base Learner minimiert, die nur auf dem *TrainSubWithGenerated* oder dem *TrainSubWithCahides* trainiert wurden.
Insgesamt sind 17 Base Learner enthalten.

Diese neuen Ensembles sollen mit den bestehenden Ensembles auf beiden Testdatensätzen (*Test* und *Test13*) verglichen werden². Das *GeneratedSDBoth* und das *GeneratedBoth* Ensemble unterscheiden sich lediglich darin, mit welcher Methode die Daten für die Trainingsdatensätze generiert wurden. Dieser Vergleich soll zeigen, ob die Bildgenerierungsmethode einen Einfluss auf das Ensemble hat. In Unterabschnitt 5.2.2.1 wurde zusätzlich für das *Generated* Ensemble die Trennung zwischen *GeneratedAll* und *GeneratedSub* untersucht, wobei die schlechte Leistung des *GeneratedSub* Ensembles aufgefallen ist. Daher sollen auch hier die Ensembles, die CNNs enthalten, die auf dem *TrainAllWithCahides* Datensatz trainiert wurden, von denen, die auf dem *TrainSubWithCahides* Datensatz trainiert wurden, unterschieden und verglichen werden. Anschließend soll das *All* Ensemble um alle weiteren neu trainierten Base Learner zum *AllNew* Ensemble erweitert und miteinander verglichen werden. Dies soll zeigen, ob es eine Grenze an Base Learnern in einem Ensemble gibt. Abschließend sollen die Ergebnisse des *All* und des *AllNew* Ensembles mit dem *AllNoSub* Ensemble verglichen werden, da dieses zwar weniger Base Learner enthält, aber dafür welche mit einer geringeren Fehlerrate. Dies soll zeigen, welchen Einfluss die Zusammensetzung des Ensembles hat, bei der die schlechteren Base Learner entfernt werden.

Das Fazit zum implementierten Ensemble Learning in Unterabschnitt 5.2.2.3 hat gezeigt, dass es reicht lediglich `hard_voting`, `weighted_soft_voting` und `stacking_with_LR` als Kombinationsmöglichkeit der Base Learner im Ensemble zu nutzen, weswegen diese im Folgenden verwendet werden. Das `weighted_soft_voting` entspricht bei den neuen *GeneratedSD* Ensembles dem `simple_soft_voting`, da die neuen Base Learner nur auf dem *Both* Datensatz trainiert wurden.

²Für jedes neue Ensemble und jeden Trainingsdatensatz wird eine neue JSON Datei erstellt. Es kommen $5 \cdot 2 = 10$ neue Dateien hinzu, die sich auf dem USB-Stick befinden.

Als erstes wird das *GeneratedBoth* und das *GeneratedSDBoth* Ensemble miteinander verglichen. Für einen übersichtlichen Vergleich sind in [Tabelle 39](#) nochmals die Werte für das *GeneratedBoth* Ensemble aus [Tabelle 24](#) angegeben.

	GeneratedBoth		GeneratedSDBoth	
	Test	Test13	Test	Test13
hard_voting	14,13%	26,79%	14,42%	26,79%
weighted_soft_voting	14,23%	26,79%	14,52%	26,79%
stacking_with_LR	14,81%	28,57%	15,19%	32,14%

Tabelle 39: Fehlerrate des *GeneratedBoth* und des *GeneratedSDBoth* Ensembles für zwei Voting Funktionen und ein Stacking Meta Modell.

Die Unterschiede zwischen diesen zwei Ensembles sind minimal. Beim Voting erreichen beide auf dem *Test13* Datensatz eine Fehlerrate von 26,79%. Auf dem *Test* Datensatz ist das *GeneratedSDBoth* um 0,29 Prozentpunkte besser, was kein nennenswerter Unterschied ist. Dies lässt vermuten, dass die Bilder beider Bildgenerierungsmethoden (GANs und Stable Diffusion) einen ähnlichen Einfluss auf die neuronalen Netze und somit auf das Ensemble haben. Beim Stacking ist der Unterschied auf dem *Test13* Datensatz mit 3,57 Prozentpunkten etwas höher, weswegen hier kurz die Sensitivität der 13 Typen auf diesen zwei Modellen gegenübergestellt wird. In [Tabelle 40](#) ist die Sensitivität der 13 Typen für das **stacking_with_LR** auf dem *GeneratedBoth* Ensemble zu sehen und in [Tabelle 41](#) ist die Sensitivität der 13 Typen für das **stacking_with_LR** auf dem *GeneratedSDBoth* Ensemble zu sehen.

stacking_with_LR mit <i>GeneratedBoth</i> Ensemble (Fehlerrate: 28,57%)													
Typ	294	940	946	5135	5944	6057	7697	12752	12783	20230	20545	20809	21027
Sensitivität	0.0	0.6667	1.0	1.0	1.0	1.0	0.3333	1.0	1.0	0.6667	1.0	0.1	
# Richtige	0	2	3	3	3	3	10	1	5	3	2	4	1
# Bilder	3	3	3	3	3	3	10	3	5	3	3	4	10

Tabelle 40: Sensitivität des **stacking_with_LR** mit dem *GeneratedBoth* Ensemble. Werte sind auf vier Nachkommastellen gerundet. Grün markierte Typen wurden vollständig korrekt klassifiziert. Unterschied zum *GeneratedSDBoth* Ensemble ist rot markiert.

stacking_with_LR mit GeneratedSDBoth Ensemble (Fehlerrate: 32,14%)													
Typ	294	940	946	5135	5944	6057	7697	12752	12783	20230	20545	20809	21027
Sensitivität	0.3333	0.6667	0.6667	1.0	1.0	1.0	0.9	0.3333	1.0	0.6667	1.0	1.0	0.0
# Richtig	1	2	2	3	3	3	9	1	5	2	3	4	0
# Bilder	3	3	3	3	3	3	10	3	5	3	3	4	10

Tabelle 41: Sensitivität des `stacking_with_LR` mit dem *GeneratedSDBoth* Ensemble. Werte sind auf vier Nachkommastellen gerundet. Grün markierte Typen wurden vollständig korrekt klassifiziert. Unterschied zum *GeneratedBoth* Ensemble ist rot markiert.

Um die Unterschiede zwischen diesen zwei Ensembles besser zu sehen, wurden die Tabellen um eine Reihe erweitert, die die Anzahl der richtig klassifizierten Bilder angibt (# Richtig). Es fällt direkt auf, dass es sich immer nur um ein Bild mehr oder weniger handelt, das richtig klassifiziert wurde. Das *GeneratedBoth* Ensemble klassifiziert 40 der 56 Bilder richtig und das *GeneratedSDBoth* Ensemble klassifiziert 38 Bilder richtig. Durch diesen kleinen Unterschied kann für die 13 Typen keine Aussage darüber getroffen werden, ob das eine Ensemble besser ist als das andere.

In [Tabelle 42](#) sind die Fehlerraten des *GeneratedSDAllBoth* und des *GeneratedSDSubBoth* Ensembles zu sehen.

	GeneratedSDAll- Both		GeneratedSDSub- Both	
	Test	Test13	Test	Test13
<code>hard_voting</code>	17,60%	25,00%	16,06%	44,64%
<code>weighted_soft_voting</code>	17,50%	25,00%	15,77%	42,86%
<code>stacking_with_LR</code>	17,79%	33,93%	17,79%	35,71%

Tabelle 42: Fehlerrate des *GeneratedSDAllBoth* und *GeneratedSDSubBoth* Ensembles für zwei Voting Funktionen und ein Stacking Meta Modell.

Wie bereits in [Tabelle 28](#) für das *Generated* Ensemble zu sehen war, ist auch hier zu erkennen, dass das Ensemble, dass nur CNNs enthält, die auf dem *TrainSubWithCahides* trainiert wurden, vor allem bei den 13 Münztypen (*Test13*), für die Bilder generiert wurden, eine sehr hohe Fehlerrate hat. Auch hier haben die generierten Bilder zu keiner Verbesserung der Klassifizierung beigetragen.

Anschließend wird das *All* Ensemble aus [Abschnitt 5.2](#) um die sechs neu trainierten CNNs zum *AllNew* Ensemble erweitert. Für einen übersichtlichen Vergleich sind in [Tabelle 43](#) nochmals die Fehlerraten für das *All* Ensemble gelistet.

	All		AllNew	
	Test	Test13	Test	Test13
hard_voting	09,52%	21,43%	09,81%	21,43%
weighted_soft_voting	10,19%	21,43%	10,67%	21,43%
stacking_with_LR	08,94%	21,43%	09,52%	25,00%

Tabelle 43: Fehlerrate des *All* und des *AllNew* Ensembles für zwei Voting Funktionen und ein Stacking Meta Modell.

Beim Vergleich der Fehlerraten dieser zwei Ensembles fallen kaum Unterschiede auf. Auf dem *Test13* Datensatz erzielt das Voting genau die gleiche Fehlerrate. Beim Stacking kommt es in dem größeren Ensemble sogar zu einer kleinen Verschlechterung um 3,57 Prozentpunkte. Auch auf dem *Test* Datensatz wird die Fehlerrate bei dem *AllNew* Ensemble minimal schlechter. Das zeigt, dass ein Ensemble in der Anzahl seiner Base Learner beschränkt ist.

Abschließend werden die zwei eben betrachteten Ensembles mit dem *AllNoSub* Ensemble verglichen. Die Fehlerraten dieses Ensembles sind in Tabelle 44 gelistet.

	AllNoSub	
	Test	Test13
hard_voting	11,45%	19,64%
weighted_soft_voting	11,25%	21,43%
stacking_with_LR	09,23%	25,00%

Tabelle 44: Fehlerrate des *AllNoSub* Ensembles für zwei Voting Funktionen und ein Stacking Meta Modell.

Auf dem gesamten Testdatensatz verschlechtert sich das Voting mit der geringeren Anzahl der Base Learner. Allerdings bleibt es auf dem *Test13* Datensatz für das *weighted_soft_voting* unverändert. Das *hard_voting* erreicht bei diesem Ensemble sogar die gleiche Fehlerrate (19,64%) wie das *weighted_soft_voting* auf dem *Base* Ensemble, welches die geringste Fehlerrate auf dem *Test13* Datensatz ist. Die Verschlechterung auf dem *Test* Datensatz, aber die gleichzeitig konstanten oder minimal besseren Ergebnisse auf dem *Test13* Datensatz zeigen, dass die Ensemble Zusammensetzung einen Einfluss auf die Klassifizierung hat und unterschiedliche Münztypen mal besser oder schlechter erkannt werden.

7 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde auf der einen Seite die Generierung antiker Münzbilder und auf der anderen Seite die Klassifizierung von Münztypen untersucht. Für die Bildgenerierung wurde der Few-Shot GAN (FSGAN) eingesetzt und für die Klassifizierung Ensemble Learning und CNN.

Das Ziel der Bildgenerierung war herauszufinden, ob mit wenig Bilddaten (5-20 Bilder) plausible Münzbilder generiert werden können. Die Ergebnisse in [Kapitel 4](#) haben gezeigt, dass es durchaus möglich ist, überzeugende Bilder zu generieren, wobei die Resultate von fünf Faktoren abhängen:

1. Qualität und Zustand der Münzen
2. Position der Motive
3. Material der Münze
4. Komplexität des Motivs
5. Hintergrundfarbe

Aufgrund der limitierten Daten vieler Münztypen können diese Faktoren oftmals nicht eingehalten werden und dadurch ist diese Methode auch nicht auf alle Münztypen anwendbar. Des Weiteren ist Bildgenerierung durch den Few-Shot GAN sehr zeitaufwendig, da für jeden Münztyp jeweils ein FSGAN für die Vorder- und Rückseite trainiert werden muss.

Bei der Klassifizierung der Münztypen wurden drei Ziele verfolgt:

Das erste Ziel war es zu ermitteln, ob das Ausbalancieren des Datensatzes mittels generierter Bilder die Gesamtperformance der Klassifizierung eines CNNs verbessern kann. Unterabschnitt [5.1.4.1](#) zeigt, dass dies keinen großen Einfluss auf die Klassifizierung hat, weswegen es sich aufgrund des Zeitaufwands nicht lohnt, Münztypen, die mehr als zwanzig Bilder enthalten, mit generierten Bildern zu erweitern.

Weiterhin war ein Ziel dieser Arbeit herauszufinden, ob Münztypen, die zwischen fünf

und zehn Bildern enthalten, mit Hilfe von generierten Bildern zum Training hinzugefügt werden können. Unterabschnitt 5.1.4.2 zeigt, dass dies für einzelne Münztypen möglich ist. Das Ergebnis wird zum einen von der Qualität der generierten Daten und zum anderen von der Zusammensetzung des Trainingsdatensatzes beeinflusst.

Das dritte Ziel war durch Ensemble Learning die Klassifizierung von Münztypen zu verbessern. Unterabschnitt 5.2.2 hat gezeigt, dass nicht nur die Gesamtperformance verbessert wird, sondern auch kleine Münztypen besser klassifiziert werden. Der beste Base Learner hat eine Genauigkeit (*accuracy*) von 81,15% auf dem gesamten Testdatensatz. Beim Ensemble Learning erreicht das *All* Ensemble mit dem Stacking mit der logistischen Regression eine Verbesserung von 9,91 Prozentpunkten. Auf dem Testdatensatz der kleinen Münztypen erreicht der beste Base Learner eine Genauigkeit von 75% und das *Base* Ensemble mit dem gewichteten Voting erzielt 80,36%. Die zwei entscheidenden Punkte für die Verbesserung sind die Anzahl der Base Learner im Ensemble und die Diversität dieser. Auch im Ensemble Learning hatten die generierten Bilder keine Auswirkungen auf die Klassifizierung. Hier war sogar bei dem Ensemble, das nur Base Learner enthalten hat, die mit weniger Originaldaten trainiert wurden, eine deutliche Verschlechterung bei genau diesen Münztypen zu sehen. Die höchste Genauigkeit lag lediglich bei 66,07% auf dem Testdatensatz der kleinen Münztypen.

Da sich zeitgleich eine weitere Arbeit mit der Generierung von Bildern beschäftigt hat, wurden zum Vergleich auch CNNs mit diesen Bildern trainiert. Die Bildgenerierungs-methode in der parallelen Arbeit war Stable Diffusion. Kapitel 6 zeigt, dass mit diesen Bildern vergleichbare Ergebnisse wie mit den FSGAN generierten Bildern erzielt werden. Die Qualität der Bilder, die durch die Stable Diffusion erstellt wurden, ist deutlich höher. Daher könnte sich diese Methode eher für das Generieren von Münzbildern eignen.

Ausblick: Die Generierung der Münzbilder könnte durch einen besseren StyleGAN2 verbessert werden. Auch besteht die Möglichkeit, auf Motive spezialisierte StyleGAN2 zu trainieren. Dadurch könnte die Qualität der Bilder des Few-Shot GANs gesteigert werden. Ebenso könnte ein StyleGAN2, welcher nicht nur auf römisch und griechische

antike Münzen spezialisiert ist, eventuell bessere Ergebnisse erzielen. Hier könnten beispielsweise in den Trainingsdatensatz auch keltische Münzen oder sogar Münzen aus der heutigen Zeit verwendet werden. Ein weiterer Ansatz wäre die Verwendung des Modells *DALLE*^[50] von OpenAI, dass die Funktion anbietet, Variationen eines gegebenen Bildes zu erstellen. Dieses Modell konnte jedoch nicht getestet werden, da die Nutzung nicht kostenlos ist.

Die Performance der Klassifizierung könnte durch die Vereinheitlichung des gesamten Datensatzes gesteigert werden, wie zum Beispiel eine einheitliche weiße Hintergrundfarbe. Aufgrund der vielen Münztypen, welche fast identische Vorder- und Rückseitenmotive haben, könnte man die Klassifizierung in zwei Schritte unterteilen. Im ersten Schritt werden Obertypen definiert und nach diesen klassifiziert. Ein Obertyp umfasst alle Typen mit sehr ähnlichen Münzmotiven, welche sich nur in kleinen Merkmalen abgrenzen. Im zweiten Schritt gibt es mehrere spezialisierte Klassifizierer, die die Untertypen eines Obertypen unterteilen. Durch diese Unterteilung könnten die feinen Merkmale zwischen ähnlichen Münzmotiven besser unterschieden werden. Das Ensemble Learning könnte durch andere Base Learner erweitert werden, welche die Münzen anhand anderer Merkmale klassifiziert, wie beispielsweise anhand der Beschreibung der Münzen. Ebenfalls könnten andere Bildklassifizierungsmodelle außer CNN einbezogen werden, wie zum Beispiel Transformer Modelle, welche Cahide Heidemann in ihrer Arbeit^[49] genutzt hat.

Anhang

A Aufteilung der Kapitel

Kapitel	Titel	Author
1	Einleitung	E.G.
2	Grundlagen	
2.1	Machine Learning	
2.1.1	Arten von Anwendungsbereichen	E.G.
2.1.2	Typen von Machine Learning Algorithmen	E.G.
2.1.3	Convolutional Neural Networks (CNNs)	E.G.
2.2.2	Evaluationsmetriken	E.G.
2.1.5	Grad CAM	R.K.
2.2	Generative Adversarial Networks	E.G.
2.3	Emsemble Learning	E.G.
3	Daten	E.G.
4	Generierung von antiken Münzbildern mittels GANs	R.K.
5	Klassifizierung von Münztypen mittels Ensemble Learning	
5.1	Training der Base Learner	R.K.
5.2	Bilden eines Ensembles	E.G.
6	Vergleich von Bildgenerierungsmethoden: GAN vs. Stable Diffusion	
6.1	Vorbereitung der Trainingsdaten	E.G.
6.2	Training und Evaluation neuer Base Learner	R.K.
6.3	Evaluation neuer Ensembles	E.G.
7	Zusammenfassung und Ausblick	R.K.

Tabelle 45: Aufteilung der Kapitel. E.G. ist Emmanuela Georgoula und R.K. ist Robin Krause.

B Beispielbilder generierter Münzen

B.1 Beispielbilder des trainierten StyleGAN2



Abbildung 51: Generierte Münzen des Traininerten Stylgan2 nach 2800kimg.

B.2 Beispielbilder von Münzen mit unterschiedlicher Motivpositionierung



Abbildung 52: Beispiel Münzen des Trainingdatensatzes der Rückseite vom Typ 294.



Abbildung 53: Beispiel Münzen des Trainingdatensatzes der Vorderseite vom Typ 1286.

C Konfusionsmatrizen der Base Learner

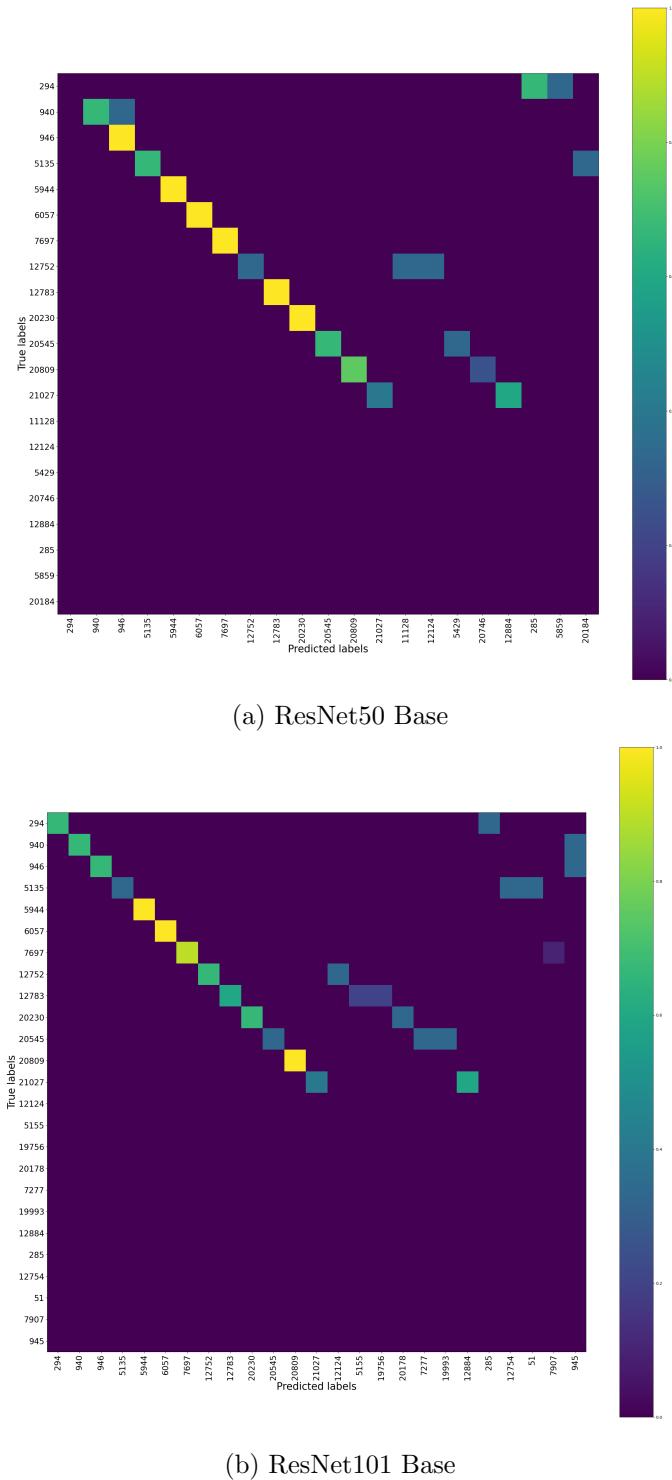
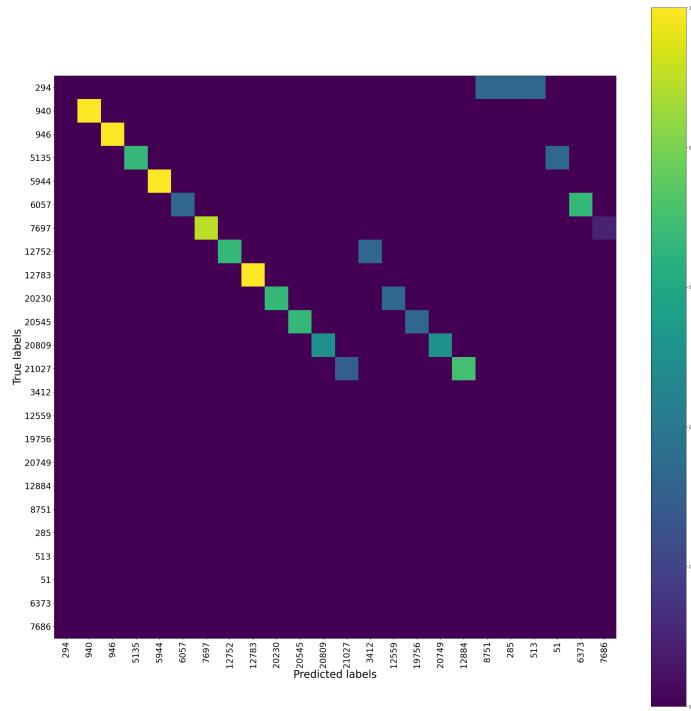
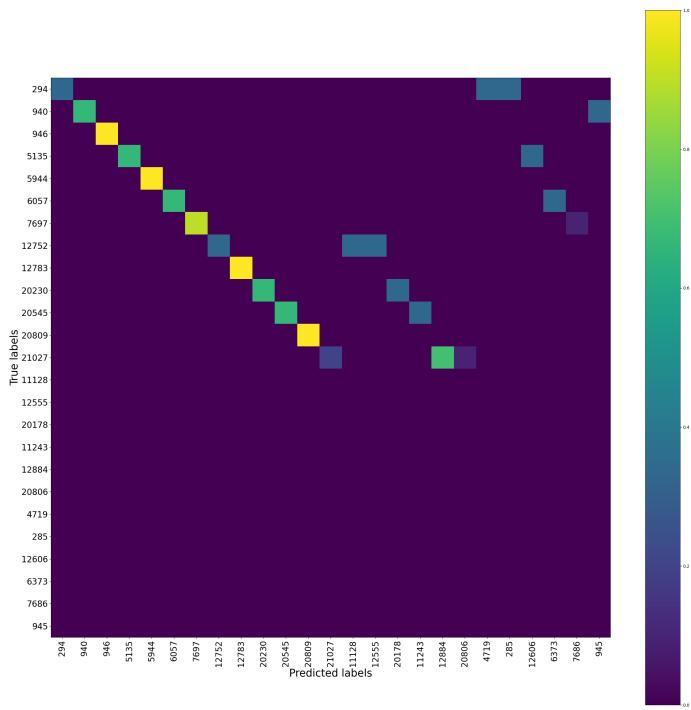


Abbildung 54: Konfusionsmatrizen der Base Learner ausgewertet auf dem Testdatensatz der 13 Typen.

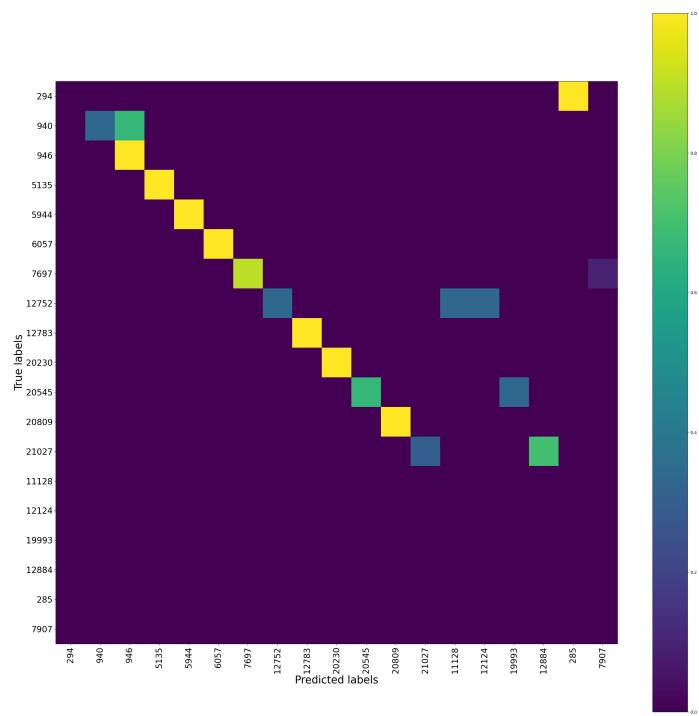


(c) ResNet152 Base

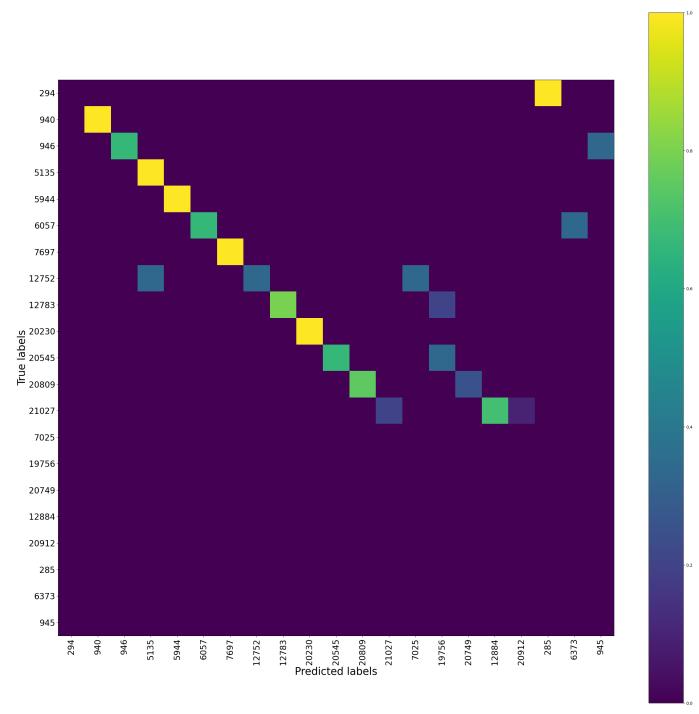


(d) ResNet50 TrainWithGenerated

Abbildung 54: Konfusionsmatrizen der Base Learner ausgewertet auf dem Testdatensatz der 13 Typen.

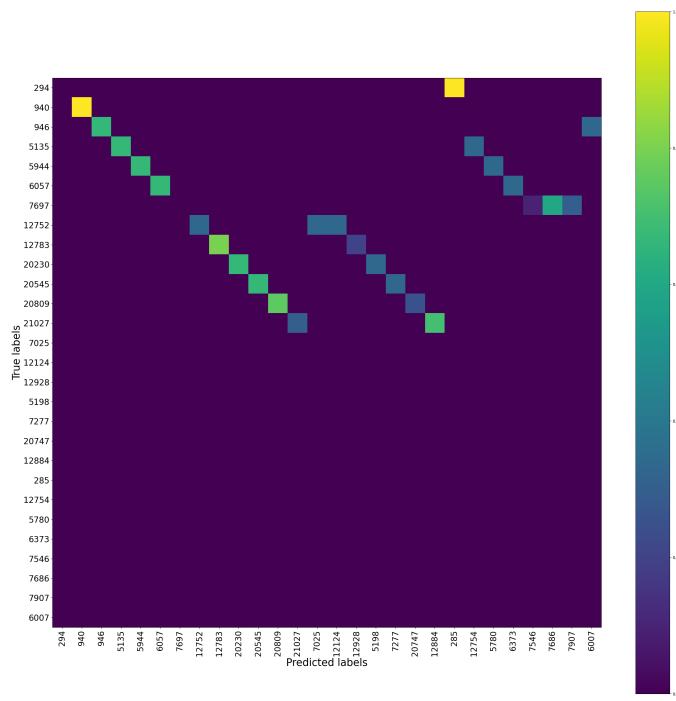


(e) ResNet101 TrainWithGenerated

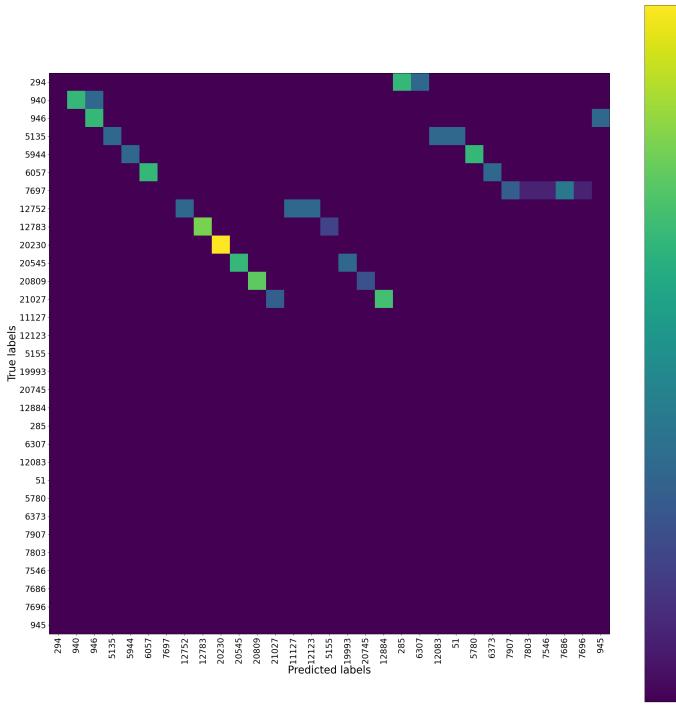


(f) ResNet152 TrainWithGenerated

Abbildung 54: Konfusionsmatrizen der Base Learner ausgewertet auf dem Testdatensatz der 13 Typen.

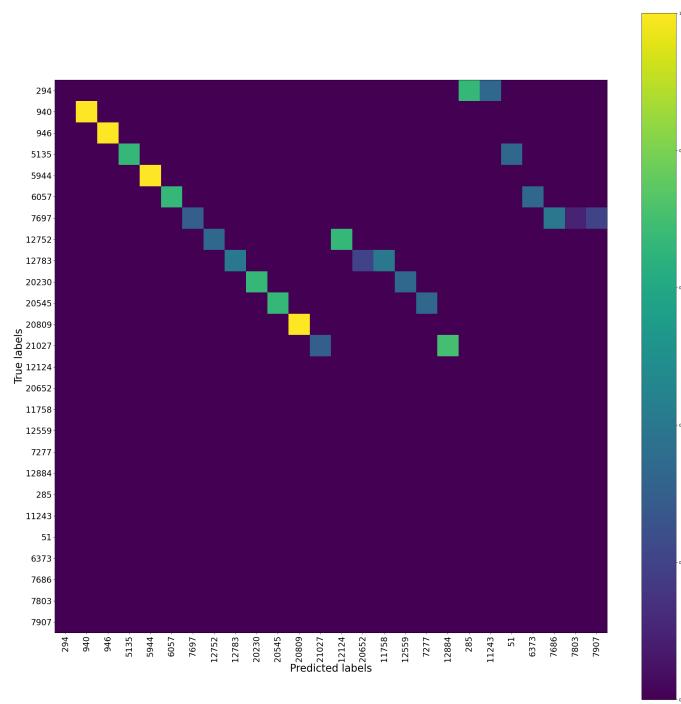


(g) ResNet50 TrainSubGenerated



(h) ResNet101 TrainSubGenerated

Abbildung 54: Konfusionsmatrizen der Base Learner ausgewertet auf dem Testdatensatz der 13 Typen.



(i) ResNet152 TrainSubGenerated

Abbildung 54: Konfusionsmatrizen der Base Learner ausgewertet auf dem Testdatensatz der 13 Typen.

Literatur

- [1] J. P. Mueller und L. Massaron, *Machine learning for dummies*. John Wiley & Sons, 2016.
- [2] A. S. Assiri, S. Nazir und S. A. Velastin, “Breast tumor classification using an ensemble machine learning method”, *Journal of Imaging*, Jg. 6, Nr. 6, S. 39, 2020. doi: [10.3390/jimaging6060039](https://doi.org/10.3390/jimaging6060039).
- [3] P. Monil, P. Darshan, R. Jecky, C. Vimarsh und B. Bhatt, “Customer segmentation using machine learning”, *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, Jg. 8, Nr. 6, S. 2104–2108, 2020. doi: [10.22214/ijraset.2020.6344](https://doi.org/10.22214/ijraset.2020.6344).
- [4] Y. Liu, Z. Qin, T. Wan und Z. Luo, “Auto-painter: Cartoon image generation from sketch by using conditional Wasserstein generative adversarial networks”, *Neurocomputing*, Jg. 311, S. 78–87, 2018. doi: [10.1016/j.neucom.2018.05.045](https://doi.org/10.1016/j.neucom.2018.05.045).
- [5] A. Braun, *Chatbots in der Kundenkommunikation*. Springer-Verlag, 2013.
- [6] Big Data Lab. “D4N4 - Datenqualität für Numismatik basierend auf Natural Language Processing und Neuronalen Netzen Data quality for Numismatics based on Natural language processing and Neural Networks”. (2024), Adresse: <http://www.bigdata.uni-frankfurt.de/d4n4/> (besucht am 23.05.2024).
- [7] M. T. Zemčík, “A brief history of chatbots”, *DEStech Transactions on Computer Science and Engineering*, Jg. 10, 2019. doi: [10.12783/dtcse/aicae2019/31439](https://doi.org/10.12783/dtcse/aicae2019/31439).
- [8] E. Robb, W.-S. Chu, A. Kumar und J.-B. Huang, “Few-shot adaptation of generative adversarial networks”, *arXiv preprint arXiv:2010.11943*, 2020. doi: [10.48550/arXiv.2010.11943](https://doi.org/10.48550/arXiv.2010.11943).
- [9] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.
- [10] S. W. Knox, *Machine learning: a concise introduction*. John Wiley & Sons, 2018.
- [11] W.-M. Lee, *Python Machine Learning*. John Wiley & Sons, 2019.
- [12] K. El Emam, L. Mosquera und R. Hopfroff, *Practical synthetic data generation: balancing privacy and the broad availability of data*. O'Reilly Media, 2020.

- [13] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta und A. A. Bharath, “Generative adversarial networks: An overview”, *IEEE signal processing magazine*, Jg. 35, Nr. 1, S. 53–65, 2018. DOI: [10.1109/MSP.2017.2765202](https://doi.org/10.1109/MSP.2017.2765202).
- [14] Aphex34. “typical CNN architecture”. (2015), Adresse: https://upload.wikimedia.org/wikipedia/commons/6/63/Typical_cnn.png (besucht am 23.05.2024).
- [15] A. Loyal, “Maschinelles Lernen angewendet auf Bilder antiker Münzen”, Masterarbeit, Goethe-Universität Frankfurt, 2018. Adresse: http://www.bigdata.uni-frankfurt.de/wp-content/uploads/2014/11/Masterarbeit_Antje_Loyal.pdf%7D.
- [16] S. Gampe, “Neuronale Netze zur Bestimmung römischer Kaiser auf Bildern antiker Münzen”, Masterarbeit, Goethe-Universität Frankfurt, 2021. Adresse: http://www.bigdata.uni-frankfurt.de/wp-content/uploads/2022/05/Masterarbeit_Sebastian_Gampe_online.pdf.
- [17] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh und D. Batra, “Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization”, *CoRR*, Jg. abs/1610.02391, 2016. arXiv: [1610 . 02391](https://arxiv.org/abs/1610.02391). Adresse: [http://arxiv.org/abs/1610.02391](https://arxiv.org/abs/1610.02391).
- [18] Tensorflow. “Grad-CAM Implementation”. (2021), Adresse: https://github.com/keras-team/keras-io/blob/master/examples/vision/grad_cam.py (besucht am 19.05.2024).
- [19] I. Goodfellow, J. Pouget-Abadie, M. Mirza u. a., “Generative Adversarial Nets”, *Advances in neural information processing systems*, Jg. 27, Nr. 11, 2014. DOI: [10.1145/3422622](https://doi.org/10.1145/3422622).
- [20] P. Salehi, A. Chalechale und M. Taghizadeh, “Generative adversarial networks (GANs): An overview of theoretical model, evaluation metrics, and recent developments”, *arXiv preprint arXiv:2005.13178*, 2020.
- [21] Q. Xu, G. Huang, Y. Yuan u. a., “An empirical study on evaluation metrics of generative adversarial networks”, *arXiv preprint arXiv:1806.07755*, 2018.
- [22] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler und S. Hochreiter, “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium”, *Advances in neural information processing systems*, Jg. 30, 2017.

- [23] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen und T. Aila, “Analyzing and Improving the Image Quality of StyleGAN”, in *Proc. CVPR*, 2020.
- [24] B. Lenze, “Spezielle Matrizen, Singulärwertzerlegung und Pseudoinverse”, in *Basiswissen Lineare Algebra: Eine Einführung mit Aufgaben, Lösungen, Selbsttests und interaktivem Online-Tool*. Springer Fachmedien Wiesbaden, 2020, S. 231–262. DOI: [10.1007/978-3-658-29969-9_11](https://doi.org/10.1007/978-3-658-29969-9_11).
- [25] G. Kunapuli, *Ensemble methods for machine learning*. Manning Publications, 2023.
- [26] O. Sagi und L. Rokach, “Ensemble learning: A survey”, *Wiley interdisciplinary reviews: data mining and knowledge discovery*, Jg. 8, Nr. 4, e1249, 2018. DOI: [10.1002/widm.1249](https://doi.org/10.1002/widm.1249).
- [27] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC, 2012.
- [28] G. Kyriakides und K. G. Margaritis, *Hands-On Ensemble Learning with Python: Build highly optimized ensemble machine learning models using scikit-learn and Keras*. Packt Publishing Ltd, 2019.
- [29] L. K. Hansen und P. Salamon, “Neural network ensembles”, *IEEE transactions on pattern analysis and machine intelligence*, Jg. 12, Nr. 10, S. 993–1001, 1990. DOI: [10.1109/34.58871](https://doi.org/10.1109/34.58871).
- [30] T. G. Dietterich, “Ensemble methods in machine learning”, in *1st International Workshop on Multiple Classifier Systems*, 2000.
- [31] D. Sarkar und V. Natarajan, *Ensemble Machine Learning Cookbook: Over 35 practical recipes to explore ensemble machine learning techniques using Python*. Packt Publishing Ltd, 2019.
- [32] Corpus Nummorum. “Corpus Nummorum Online”. (2024), Adresse: <https://www.corpus-nummorum.eu> (besucht am 23.05.2024).
- [33] J. Hock, “Generate coin images from ancient numismatics using Few-Shot Learning”, Masterarbeit, Goethe-Universität Frankfurt, 2023. Adresse: <https://zenodo.org/records/10016333>.
- [34] e-271. “FSGAN Official TensorFlow Code”. (2019), Adresse: <https://github.com/e-271/few-shot-gan> (besucht am 24.04.2024).

- [35] NVIDIA Corporation. “StyleGAN2 — Official TensorFlow Implementation”. (2019), Adresse: <https://github.com/NVlabs/stylegan2#requirements> (besucht am 12.01.2024).
- [36] NVIDIA Corporation. “TensorFlow Release 20.11”. (2024), Adresse: https://docs.nvidia.com/deeplearning/frameworks/tensorflow-release-notes/rel_20-11.html#rel_20-11 (besucht am 28.12.2023).
- [37] Tensorflow. “Docker Image tensorflow:2.10.0”. (2022), Adresse: <https://hub.docker.com/layers/tensorflow/tensorflow/2.10.0-gpu/images/%5Cnewline%20sha256-3aeb6a5489ad8221d79ab50ec09e0b09afc483dfdb4b868ea38cfb9335269049?context=explore> (besucht am 13.05.2024).
- [38] Python Software Foundation. “argparse — Parser for command-line options, arguments and sub-commands”. (2024), Adresse: <https://docs.python.org/3.11/library/argparse.html> (besucht am 23.05.2024).
- [39] Joblib developers. “joblib.dump”. (2021), Adresse: <https://joblib.readthedocs.io/en/stable/generated/joblib.dump.html> (besucht am 23.05.2024).
- [40] Python Software Foundation. “json — JSON encoder and decoder”. (2024), Adresse: <https://docs.python.org/3.11/library/json.html> (besucht am 23.05.2024).
- [41] The Matplotlib development team. “matplotlib.pyplot”. (2024), Adresse: https://matplotlib.org/3.7.5/api/pyplot_summary.html (besucht am 23.05.2024).
- [42] Python Software Foundation. “numpy 1.24.3”. (2023), Adresse: <https://pypi.org/project/numpy/1.24.3/> (besucht am 23.05.2024).
- [43] Python Software Foundation. “pandas 2.0.3”. (2023), Adresse: <https://pypi.org/project/pandas/2.0.3/> (besucht am 23.05.2024).
- [44] Python Software Foundation. “pickle — Python object serialization”. (2024), Adresse: <https://docs.python.org/3.11/library/pickle.html> (besucht am 23.05.2024).
- [45] Python Software Foundation. “pathlib — Object-oriented filesystem paths”. (2024), Adresse: <https://docs.python.org/3.11/library/pathlib.html> (besucht am 23.05.2024).

- [46] scikit-learn developers (BSD License). “sklearn.linear_model.LogisticRegression”. (2023), Adresse: https://scikit-learn.org/1.3/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression (besucht am 23.05.2024).
- [47] scikit-learn developers (BSD License). “sklearn.ensemble.RandomForestClassifier”. (2023), Adresse: <https://scikit-learn.org/1.3/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier> (besucht am 23.05.2024).
- [48] scikit-learn developers (BSD License). “Metrics and scoring: quantifying the quality of predictions”. (2023), Adresse: https://scikit-learn.org/1.3/modules/model_evaluation.html#classification-metrics (besucht am 23.05.2024).
- [49] C. Heidemann, “Transformer-basierte Methoden und Ensemble Learning zur Klassifikation von antiken Münzbildern”, Masterarbeit, Goethe-Universität Frankfurt, 2024.
- [50] TOpenAi. “Image Geneartion”. (2022), Adresse: <https://platform.openai.com/docs/guides/images/usage> (besucht am 11.06.2024).

Erklärungen zur Abschlussarbeit

Bitte dieses Formular zusammen mit der Abschlussarbeit abgeben!

Erklärung zur Abschlussarbeit

**gemäß § 34, Abs. 16 der Ordnung für den Masterstudiengang Wirtschaftsinformatik
vom 01. April 2019**

Hiermit erkläre ich

(Nachname, Vorname)

Meinen entsprechend gekennzeichneten Anteil der vorliegenden Arbeit habe ich selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst.

Ich bestätige außerdem, dass die vorliegende gemeinsame Arbeit nicht, auch nicht auszugsweise, für eine andere Prüfung oder Studienleistung verwendet wurde.

Zudem versichere ich, dass alle eingereichten schriftlichen gebundenen Versionen der gemeinsamen Masterarbeit mit der digital eingereichten elektronischen Version der gemeinsamen Masterarbeit übereinstimmen.

Frankfurt am Main, den

Unterschrift der/des Studierenden

Bitte dieses Formular zusammen mit der Abschlussarbeit abgeben!

Erklärung zur Abschlussarbeit

**gemäß § 34, Abs. 16 der Ordnung für den Masterstudiengang Informatik
vom 17. Juni 2019**

Hiermit erkläre ich

(Nachname, Vorname)

Meinen entsprechend gekennzeichneten Anteil der Arbeit habe ich
ich selbstständig und ohne Benutzung anderer als der angegebenen
Quellen und Hilfsmittel verfasst.

Ebenso bestätige ich, dass diese Arbeit nicht, auch nicht auszugsweise,
für eine andere Prüfung oder Studienleistung verwendet wurde.

Zudem versichere ich, dass alle eingereichten schriftlichen gebundenen
Versionen der gemeinsamen Masterarbeit mit der eingereichten
elektronischen Version der gemeinsamen Masterarbeit übereinstimmen.

Frankfurt am Main, den

Unterschrift der/des Studierenden