Systematische Evaluierung von Architekturen und Features für die Bestimmung von klassischen Chiffren-Typen mit neuronalen Netzen

Ernst Leierzopf

MASTERARBEIT

Nr. 1910304008

eingereicht am
Fachhochschul-Masterstudiengang

SICHERE INFORMATIONSSYSTEME

in Hagenberg

Juni 2021

Die Arbeit ist thematisch dem

Smart Security Lab (SSL)

zuzuordnen.

Betreuer:

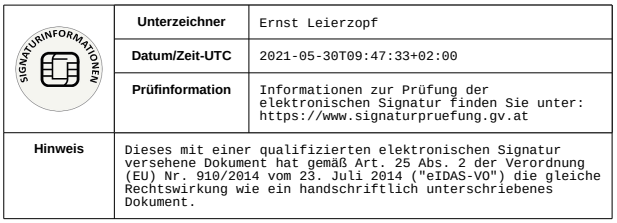
FH-Prof. Dr. Harald Lampesberger
FH-Prof. DI Dr. Eckehard Hermann
Prof. Bernhard Esslinger
Dr. Nils Kopal

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt. Die vorliegende, gedruckte Arbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Hagenberg, am 30. Mai 2021

Ernst Leierzopf



Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Vorwort

Eine Masterarbeit zu verfassen ist keine einfache Aufgabe und erfordert viel Geduld und Unterstützung. An dieser Stelle bedanke ich mich bei allen, die mich bei der Erstellung dieser Masterarbeit unterstützt haben.

Zuerst bedanke ich mich bei FH-Prof. Dr. Harald Lampesberger, ohne dessen Einführung in die Thematik des maschinellen Lernens, insbesondere den Erklärungen verschiedener Verfahren mit neuronalen Netzen, mein Verständnis und damit die Qualität dieser Arbeit nicht im vorliegenden Ausmaß gegeben wäre.

FH-Prof. DI Dr. Eckehard Hermann danke ich für seine konstruktiven, implementierungstechnischen Vorschläge. Sein Input hat geholfen die Parallelisierung der Software und die Anpassung an die DGX-1 zu optimieren.

Ebenfalls bedanke ich mich herzlich bei Prof. Bernhard Esslinger für seine unverzüglichen und sehr detaillierten Feedbacks, sowie seine hartnäckigen Nachfragen bei unklaren Formulierungen. Durch diese Feedbacks konnte der rote Faden dieser Arbeit gesichert und die Texte feingeschliffen werden.

Dr. Nils Kopal vom CrypTool-Projekt[1] verdanke ich die Übergabe der ersten Version für die Erkennung von Chiffren-Typen, welche ich weiter entwickeln durfte. Seine Auswahl des Feed-Forward Neural Networks mit 5 Hidden Layern war erstaunlicherweise präzise und konnte nur noch wenig optimiert werden. Außerdem war er meine Ansprechperson in Fragen zu klassischen Chiffren und der implementierten Features.

Ein besonderer Dank gilt auch meinen Eltern, Ernst und Ivona Leierzopf, sowie Goran Dujmovic, ohne deren Unterstützung mein Studium nicht möglich gewesen wäre und infolgedessen diese Arbeit auch nicht entstehen hätte können.

[1]CrypTool: https://www.cryptool.org/

Kurzfassung

Kryptoanalyse verschlüsselter Dokumente beginnt normalerweise mit der Identifikation des Chiffren-Typs. Es gibt eine große Anzahl verschlüsselter historischer Dokumente, deren Entschlüsselung möglicherweise das Wissen über historische Ereignisse verbessern kann. Diese Masterarbeit befasst sich mit der Klassifizierung von klassischen Chiffren, welche bis zum zweiten Weltkrieg eingesetzt wurden, mithilfe von verschiedenen maschinellen Lernverfahren. Konkret wurden 56, von der American Cryptogram Association definierte, klassische Chiffren für die Klassifizierung implementiert. Es werden die Feature-Engineering-Algorithmen Feed-Forward Neural Networks (FFNN), Random Forests (RF) und Naive Bayes Networks (NBN), sowie die Feature-Learning-Algorithmen Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) und Tranformer untersucht. Nach einer sorgfältigen Auswahl der Features sowie der Optimierung der Hyperparameter erreicht das FFNN eine Genauigkeit von 78,31% für 10 Millionen Testdaten mit einer fixen Geheimtextlänge von 100 Zeichen. Mithilfe eines Ensemble-Modells, das aus FFNN, RF, NBN, LSTM und Transformer Modellen besteht, konnte für denselben Datensatz eine Genauigkeit von 82,78% erreicht werden. Ein Ensemble-Modell mit variablen Textlängen von 51 bis maximal 428 Zeichen konnte eine Genauigkeit von 70,79% erreichen, was eine Verbesserung von 21% gegenüber dem derzeitigen Stand der Technik bedeutet. Transfer Learning wurde ebenfalls experimentell evaluiert und Catastrophic Forgetting wurde in den untersuchten Fällen nicht beobachtet. Durch die Anwendung von Transfer Learning konnten die Trainingszeiten um ca. 40-60% verringert werden. Die Software-Suite wird unter dem Namen „**Neural Cipher Identifier (NCID)**“ auf Github veröffentlicht.

Abstract

Cryptanalysis of encrypted documents usually begins with the identification of the cipher type. There are a large number of encrypted historical documents, the decryption of which can potentially improve knowledge of historical events. This master's thesis deals with the classification of classical ciphers using various machine learning methods. Specifically, it is about 56 classical ciphers defined by the American Cryptogram Association. The feature-engineering algorithms Feed-Forward Neural Networks (FFNN), Random Forests (RF) and Naive Bayes Networks (NBN), as well as the feature-learning algorithms Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) and Transformers were investigated. After carefully selecting the features and optimizing the hyperparameters, the FFNN achieved 78.31% accuracy for a test dataset consisting of 10 million records with a fixed ciphertext length of 100 characters. Using an ensemble model consisting of FFNN, RF, NBN, LSTM and Transformer models, an accuracy of 82.78% could be achieved for the same dataset. An ensemble model with variable text lengths in the range [51, 428] was able to achieve an accuracy of 70.79%, which means an improvement of 21% compared to the current state of the art. Furthermore, it was experimentally shown that catastrophic forgetting does not occur for this problem and that training times can be reduced by approx. 40-60% through the use of transfer learning, depending on how many classes are involved. The software suite is published on Github under the name "Neural Cipher Identifier (NCID)".

Abkürzungsverzeichnis

ACA *American Cryptogram Association*
CANN *CryptAnalysis with Neural Networks*
CNN *Convolutional Neural Network*
CT *Geheimtext*
DBN *Deep Belief Network*
EB *Entscheidungsbaum*
ELU *Exponential Linear Unit*
FFNN *Feed-Forward Neural Network*
HMM *Hidden Markov Model*
LSTM *Long Short-Term Memory*
NBN *Naive Bayes Network*
PT *Klartext*
RBF *Radial Basis Function*
RBM *Restricted Boltzmann Machine*
ReLU *Rectified Linear Unit*
RNN *Recurrent Neural Network*
SELU *Scaled Exponential Linear Unit*
SGD *Stochastic Gradient Descent*
SVM *Support-Vector-Machine*
tanh H*yperbeltangens*

Kapitel 1 **Einleitung**

1.1 Problemstellung und Motivation

Historische Aufzeichnungen belegen, dass Verschlüsselung in etwa so alt ist wie Schriften selbst. Der früheste dokumentierte Einsatz von Kryptographie lässt sich auf das Alte Ägyptische Reich im dritten Jahrtausend vor Christus zurückführen [1]. Kryptographie wurde in der älteren Geschichte vor allem in der Aristokratie und von Militärs genutzt. Grundsätzlich lassen sich klassische Chiffren, die vor dem zweiten Weltkrieg entwickelt und verwendet wurden, in Substitution und Transposition unterteilen. Bei der simplen Substitution wird jeder Buchstabe mit einem anderen aus dem Alphabet ersetzt. Homophone Substitution zeichnet sich dadurch aus, dass häufige Buchstaben durch mehrere Substitute ersetzt werden können. Transpositionschiffren verschieben lediglich die Buchstaben in eine quasi-zufällige Reihenfolge. Aufgrund der Einfachheit werden diese Chiffren als „klassische Chiffren“ bezeichnet.

DECRYPT ist ein vom Swedish Research Council gefördertes Projekt, welches als Ziel die Bereitstellung einer Infrastruktur für Forscher*innen hat, um die Sammlung, automatisierte Digitalisierung und Analyse von verschlüsselten Dokumenten zu ermöglichen. Wenn möglich sollen diese Dokumente anschließend entschlüsselt werden, jedoch sind zusätzliche Informationen, wie z.B. der Chiffren-Typ, für die Kryptoanalyse sehr hilfreich. Diese Arbeit befasst sich mit der Cipher Type Detection von klassischen Chiffren. Zum Zeitpunkt des Schreibens handelt es sich dabei um ungefähr 1.700 historische Dokumente aus dem 15. bis 19. Jahrhundert. Bekannte Ortungen der Dokumente sind unter anderem Brüssel, Wien, Venedig, Vatikan, Den Haag, Budapest, Madrid, Milano, Modena, Kew und Dresden. Die am häufigsten verwendeten Sprachen sind (Alt) Italienisch, Latein, Deutsch, Französisch, Ungarisch und Englisch. An dem Projekt arbeiten Wissenschaftler aus den Bereichen Informatik, Kryptographie, Philologie, Linguistik und Geschichte. Ein sehr großer Teil der Angriffe auf Chiffren wird nur anhand des Geheimtextes durchgeführt (Ciphertext-only-Szenario). Außerdem besteht die Wahrscheinlichkeit, dass bereits verwendete Schlüssel für weitere Dokumente auch weiterverwendet wurden. [2]

Die Problemstellung für die vorliegende Masterarbeit ist aus der Zusammenarbeit mit der Forschungsgruppe von Prof. Esslinger, welche wiederum an DECRYPT beteiligt [2] ist, entsprungen.

Die hohe Anzahl der Varianten von klassischen Chiffren und die Vielfalt der verwendeten Sprachen und ihre Varianten über Zeit und Regionen erschwert die Kryptoanalyse von Geheimtexten, wodurch die manuelle Kryptoanalyse für die große Anzahl an Dokumenten praktisch unmöglich ist. Daher ist es notwendig, Geheimtexte in einer angemessenen Laufzeit ihren Chiffren-Typen zuzuordnen. Hat man eine große Menge an Geheimtexten und sind die verwendeten Chiffren-Typen bekannt, können Chiffren-spezifische sowie heuristische Algorithmen die Klartexte mit vertretbarem Aufwand und schneller finden. Das System soll auch helfen, dass auch nicht-Kryptographie-Experten, wie Historiker, neu gefundenen Geheimtexten einfach einem Chiffren-Typ zuordnen können.

1.2 Ausgangsstellung

CANN – CryptAnalysis with Neural Networks war der Versuch, die Arbeit von Kopal [3] zu verbessern, welche wiederum ein neuronales Netzwerk verwendet. Da sich in CANN die Ergebnisse des neuronalen Netzwerks verbessern konnten, ist auch das Ziel für die Masterarbeit, neuronale Netzwerke als primäres Klassifikationsverfahren einzusetzen. Die Hauptaufgabe des CANN-Projektes war es, den bestehenden Code, welcher prototypisch entwickelt wurde, in eine verbesserte Projektinfrastruktur einzubetten sowie die Erkennungsrate zu erhöhen und die Trainingsdauer zu senken. Ein weiterer Teil dieser Aufgabe war die Parallelisierung des Trainingsprozesses für die bessere Ressourcennutzung auf High-Performance-Servern, wie in diesem Fall einer NVIDIA DGX-1[2], zu ermöglichen.

1.3 Erwartungen

Das Ziel der Arbeit ist die Entwicklung eines zuverlässigen Programms für die Klassifizierung von klassischen Chiffren mit unterschiedlichen Ansätzen, primär basierend auf neuronalen Netzwerken.

Im Rahmen der Evaluierung soll das entwickelte Programm mit dem Stand der Forschung verglichen werden. Beispielsweise haben Nuhn und Knight ein neuronales Netz für die Erkennung von 50 verschiedenen American-Cryptogram-Association-(ACA-)Chiffren [4] mit einer Genauigkeit von 58,5% trainiert [5]. Mit der Veröffentlichung dieser Masterarbeit werden auch ein realeasefähiges Programm und mehrere trainierte neuronale Netze veröffentlicht.

[2] NVIDIA DGX-1: https://www.nvidia.com/de-de/data-center/dgx-1/

1.4 Forschungsfrage

Das CANN-Projekt [6] bietet bereits eine Genauigkeit der Chiffren-Typisierung von ca. 90% für die Chiffren Columnar Transposition, Hill, Playfair, Simple Substitution und Vigenère. Daraus lässt sich die folgende Hauptforschungsfrage mit den dazugehörigen Unterfragen ableiten.

In welchem Ausmaß lässt sich die verwendete klassische Chiffre mittels statistischer Metriken (den sog. Features) und mit Feature-Learning-Algorithmen [7] in einem neuronalen Netz bestimmen, und wie wirken sich die Architektur und die Anzahl der zu erkennenden Chiffren auf Erkennungsrate und Laufzeit aus?
Unterforschungsfragen:

- Was ist der Stand der Technik im Bereich der Cipher Type Detection und welche Architekturkonzepte für die Textklassifikation mit neuronalen Netzen eignen sich für die Klassifikation von klassischen Chiffren?
- Welches Vorgehen eignet sich am besten zur Bestimmung der Qualität eines neuronalen Netzes mit verschiedenen Parametern und Features?
- Lässt sich ein bereits trainiertes neuronales Netz zur Erkennung von Chiffren-Typen mithilfe von Transfer Learning weiter trainieren oder tritt das sog. Catastrophic Forgetting [8] auf? Kann ein Backbone als Basis für den Trainingsprozess erstellt werden, um mithilfe von Transfer Learning neue Klassen hinzuzufügen?

1.5 Methodik der Arbeit

Aufgrund der praktischen Art der Problemlösung kommt die Design-Science-Methodik [9] zum Einsatz. Die Design-Science-Methodik zeichnet sich durch die Untersuchung einer Problemstellung anhand eines entwickelten IT-Artefakts, z.B. Software, aus, dessen Nützlichkeit in weiterer Folge systematisch evaluiert wird.

In dieser Arbeit werden unterschiedliche Architekturen für Klassifizierungsprobleme in neuronalen Netzen definiert und mittels automatisierter Eignungstests empirisch untersucht. Neben der manuellen Auswahl von Hyperparametern sind auch automatisierte Suchen wie die Rastersuche, Zufallssuche und Modellbasierte Hyperparameter Optimierung vorgesehen [10, pp. 416-437]. Die Auswahl der besten Hyperparameter erfolgt mittels einer semi-automatisierten Suche. Im Zuge dessen lassen sich folgende operative Problemstellungen formulieren:

- Eine Aussage über die Qualität der Architektur eines neuronalen Netzes lässt sich nach einer Anzahl von Trainingszyklen bestimmen. Dafür ist es notwendig, passende Abbruchkriterien zu finden.
- Hyperparameter-Anpassungsstrategien, wie z.B. ein Learning Rate Scheduler oder Dropout Regulierung, können verwendet werden. Ein Teil der Architektur- und Hyperparameterauswahl ist die gewählte Anpassungsstrategie.

Das in Abschnitt 3.3 CANN – CryptAnalysis with Neural Networks beschriebene Programm dient als Ausgangsbasis und soll mittels Parallelisierung optimiert und mit weiteren Metriken erweitert werden, um eine höhere Anzahl an Trainingsdurchläufen und höhere Genauigkeit des neuronalen Netzes zu ermöglichen. Alle 60 ACA-Chiffren [4], sowie aus dem Stand der Technik bekannte Features, werden im Zuge der Arbeit betrachtet und systematisch gewählt. Das Bion's Gadget [11] ist, unter anderem, eine Online-Version eines Cipher Type Klassifikators. Die verwendeten Metriken sind auf der Website beschrieben und sollen in dieser Arbeit implementiert werden.

Als Catastrophic Forgetting [8] wird in einem neuronalen Netz das Phänomen bezeichnet, bei dem durch neuartige Daten die bekannten Features verlernt werden. Ein Backbone eines neuronalen Netzes ist für die Extraktion und Speicherung von Features verantwortlich [12]. Die Fragestellung über das Catastrophic Forgetting wird mithilfe von mehreren Versuchen untersucht. Dafür werden ein Basismodell und mehrere Vergleichsmodelle trainiert, damit die Unterschiede zum Training mithilfe von Transfer Learning festgestellt werden können. Der erwartete Vorteil dieses Vorgehens ist eine niedrigere Trainingszeit und bessere Genauigkeit des neuronalen Netzes. Besonders viel profitieren würden, aufgrund der kürzeren Ladezeit von Input Daten, Feature-Learning-Algorithmen, die in Punkt 2.6.2 beschrieben sind.

Die Ergebnisse werden mithilfe von statistischen Metriken, wie z.B. Genauigkeit, Precision und Recall, sowie mit einem Testdatensatz evaluiert werden. Dieser Testdatensatz muss für alle neuronalen Netze gleich sein, damit die Ergebnisse vergleichbar bleiben. Um bessere Vergleichbarkeit zum derzeitigen Stand der Technik von Nuhn [5] zu schaffen, werden 319 Testdatensätze von der ACA [4] evaluiert.

1.6 Aufbau der Arbeit

Der derzeitige Stand der Technik in den Bereichen Cipher Type Detection und Architekturkonzepte für die Textklassifikation mit neuronalen Netzen, sowie alle dazu benötigten Vorkenntnisse, werden in Kapitel 2 erläutert. Im dritten Kapitel werden der Aufbau und die Durchführung der empirischen Messungen der Qualität von verschiedenen Features und Architekturen von neuronalen Netzen aufgezeigt. Die Ergebnisse dieser Messungen werden im vierten Kapitel zusammengefasst und begründet. Dabei sind auch die entscheidenden Merkmale für die Auswahl von Features und Architekturen beschrieben. Die wichtigsten Erkenntnisse und Ergebnisse werden im Fazit und Ausblick zusammengefasst. Der Ausblick zeigt weitere Problemfelder und Forschungsfragen auf, welche im Zuge der Arbeit aufgetreten sind.

1.7 Publikationen

Derzeit befinden sich zentrale Teile der Arbeit im wissenschaftlichen Publikationsprozess. Der folgende Beitrag ist im Rahmen der vorliegenden Arbeit entstanden:

- E. Leierzopf, N. Kopal, B. Esslinger, H. Lampesberger und E. Hermann, „A Massive Machine-Learning Approach For Classical Cipher Type Detection Using Feature Engineering“, in *Proceedings of the 4th International Conference on Historical Cryptology, HistoCrypt*, 2021. (akzeptiert)

Kapitel 2 Grundlagen

Dieses Kapitel beinhaltet die Grundlagen neuronaler Netzwerke. Die Funktionsweise neuronaler Netze ist in den Abschnitten 2.1 bis 2.4 beschrieben. Mechanismen zur Anpassung der Lernraten sind in Abschnitt 2.5 ausgeführt. Verschiedene Architekturkonzepte für die Textklassifikation mit neuronalen Netzen sind in Abschnitt 2.6 dargestellt.

2.1 Neuronale Netze

Klassische Wissensverarbeitungsansätze sind oft für die Auswertung von nicht-trivialen Datensätzen aufgrund der eingeschränkten Fähigkeit des Menschen Daten zu verknüpfen nicht geeignet. Neuronale Netze generieren mit bekannten Input-Variablen, den sog. Features, verwertbare Outputs, die sog. Prädiktionen. Das Ziel dieses Ansatzes ist es ein generalisiertes Verfahren zu trainieren, um aus noch nie gesehenen Daten richtige Prädiktionen generieren können. [13, pp. 14-15]

Neuronale Netze können auch nach dem Problemfeld unterschieden werden. Die gängigsten Varianten sind [13, pp. 16-17]:

- **Klassifikation:** Klassifikationsprobleme sind dadurch definiert, dass ein Input zu genau einer, aus einer fixen Menge von Klassen gewählten, Output-Klasse zugeordnet wird. Ein Klassifikationsproblem ist z.B. die Zuordnung einer Chiffre zu einem Geheimtext.
- **Regression:** Regressionsalgorithmen bieten eine allgemeinere Form von Vorhersagen gegenüber Klassifikationsalgorithmen. Regressionsprobleme werden durch eine Datenstruktur als Output des Modells gelöst. Ein Beispiel ist die Schätzung von Immobilienpreisen anhand äußerer Faktoren.
- **Clustering:** Clustering-Verfahren versuchen in einer Menge von Input-Daten, ohne die Kenntnis der dazugehörigen Output-Daten, sinnvolle Strukturen in Form von Clustern zu finden. Ein Beispiel für Clustering ist die automatische Bildung von Klassen, die später für Klassifizierungsaufgaben verwendet werden können.
- **Anomalieerkennung:** Bei der Anomalieerkennung wird mithilfe einer Menge von Input-Daten das Normalverhalten eines Systems abgebildet. Ein trainiertes Modell soll Input-Daten mit einem Grad der Normalität, die z.B. im Wertebereich [0..1] liegt, bewerten. Ein Beispiel für diese Algorithmen ist die Anomalieerkennung in Logdaten eines Systems.

Ein weiteres Entwurfskriterium für neuronale Netze ist die Art des Lernverfahrens. Folgende grundlegende Lernverfahren gibt es unter anderem [13, pp. 17-19]:

- **Supervised Learning:** In diesem Lernverfahren bestehen Trainingsdaten aus bekannten Input-Output-Paaren. Das Ziel ist es ein generalisiertes Modell zu trainieren, um damit die bestmöglichen Prädiktionen für die, dem Modell unbekannten, Daten zu bestimmen. Klassifikations- und Regressionsprobleme fallen typischerweise in diese Kategorie.
- **Unsupervised Learning:** Beim unüberwachten Lernen können, aufgrund der fehlenden Output-Daten, keine generellen Zusammenhänge gefunden werden. Das Ziel in diesem Szenario ist der Erkenntnisgewinn aus Daten. Clustering- und Anomalieerkennungsalgorithmen fallen in diese Kategorie.
- **Reinforcement Learning:** Beim bestärkenden Lernen gibt es keine Trainingsdaten. Das System soll mittels eines Belohnungssystems [14, pp. 960-981] Rückmeldung in Form von Belohnungen erhalten, um damit das intelligenteste Verhalten wählen zu können. Eines der bekannteren neuronalen Netze ist AlphaZero [15], welches in den Spielen Schach, Shogi und Go gegenüber klassischen Systemen bereits sehr gute Ergebnisse liefert.

Neuronen in neuronalen Netzen ahmen stark vereinfacht die Funktionsweise von Neuronen im menschlichen Gehirn nach, indem Verbindungen gelernt oder geschwächt werden [16]. Die Stärke der Verbindung wird auch **weight** oder **Gewicht** genannt. Eine Aktivierungsfunktion bestimmt mittels eines Schwellwerts und den weights welche Neuronen aktiviert werden. Grundsätzlich werden Neuronen in drei Gruppen unterteilt [14, pp. 845-856]:

- **Input-Neuronen:** Input-Neuronen enthalten Daten, wie z.B. Häufigkeitsverteilungen für Feature-Learning-Algorithmen (siehe Punkt 2.6.2) oder Unterscheidungsmerkmale für Feature-Engineering-Algorithmen (siehe Punkt 2.6.1).
- **Hidden-Neuronen:** Hidden-Neuronen sind in einem neuronalen Netzwerk für die Bildung komplexer Zusammenhänge verantwortlich. Die Stärke der Verbindungen, welche durch das weight beziffert werden, bildet das Entscheidungsgewicht. Neuronen werden in mehreren Layern organisiert. Input- und Output-Neuronen sind jeweils der erste bzw. letzte Layer eines neuronalen Netzwerks. Für die Auswahl der optimalen Anzahl von Layern und der optimalen Anzahl von Neuronen pro Layer gibt es keine standardisierte Methode, weshalb die Auswahl in der Praxis experimentell durchgeführt wird.

- **Output-Neuronen:** Output-Neuronen repräsentieren das Ergebnis des neuronalen Netzes. Bei Klassifikationsproblemen ist die Anzahl der Output-Neuronen üblicherweise die Anzahl der Klassen. Das Neuron mit dem höchsten Wert kann als die Prädiktion des Modells interpretiert werden.

Neuronale Netze stellen beim Training zwei Fehlerquellen gegenüber: den Abbildungsfehler und den Vorhersagefehler. Der Abbildungsfehler wird als der Informationsverlust der Trainingsdaten auf dem Weg zum Modell definiert. Praktisch alle Lernalgorithmen optieren zu einem möglichst niedrigen Abbildungsfehler. Der Vorhersagefehler wird mit separaten Testdaten evaluiert. [13, pp. 26-28]

Als Bias werden Fehler aufgrund nicht gelernter Beziehungen bezeichnet. Als Varianz wird die Empfindlichkeit gegenüber Trainingsdaten bezeichnet. Ein zu einfaches Modell ist durch ein hohes Bias und eine niedrige Varianz zu erkennen (Underfitting). Ein zu kompliziertes Modell bekommt aufgrund der irrelevanten Features der Daten ein niedriges Bias und eine hohe Varianz (Overfitting). [17]

2.2 Optimizer und Loss-Funktionen

Lernen in neuronalen Netzen basiert auf dem einfachen Konzept ein Minimum einer Funktion, der sog. Loss-Funktion, zu finden. Optimizer sind Algorithmen, die das Ziel haben den Loss (Abbildungsfehler) zu minimieren, indem sie die Eigenschaften eines neuronalen Netzes, wie z.B. die weights und Lernrate, anpassen. Gradient Descent ist ein Algorithmus zum Finden des lokalen Minimums einer Funktion erster Ordnung. Die Rückführung der Ergebnisse zur Anpassung der Eigenschaften eines neuronalen Netzwerks wird als Backpropagation bezeichnet.

Jeder Optimizer verwendet zum Trainieren eines neuronalen Netzes zwingend eine Loss-Funktion. Diese Fehlerfunktion wird benötigt, um Lösungen, oder auch Predictions genannt, zu beurteilen und die weights anzupassen. Beim Training eines neuronalen Netzwerks wird ein Durchlauf aller Trainingsdaten allgemein als Epoche bezeichnet. Oft ist es nicht möglich alle Daten in den Speicher zu laden, weshalb das Training in sog. Mini-Batches stattfindet. Wenn Daten nur einmal in Form von Mini-Batches trainiert werden, ist die Epoche immer 1. Dabei wird ein Teil der Daten trainiert, bevor er wieder aus dem Arbeitsspeicher verworfen wird. Ein einzelner Datensatz wird weiters als Iteration bezeichnet. Im Bereich der Klassifizierungsprobleme wird grundsätzlich zwischen binären und Multi-Klassen-Problemen unterschieden. Erreicht der Loss den Wert 0, bedeutet das, dass die Trainingsdaten perfekt im Modell abgebildet sind. Zur Zeit des Schreibens dieser Arbeit ist der Sparse Categorical Cross-

entropy Loss kombiniert mit der Maximum Likelihood (≙ Softmax Aktivierungsfunktion für die Entscheidung) die Standard-Loss-Funktion. Categorical Crossentropy ist eine Funktion, die Gleichheit zweier Matrizen in einem Zahlenwert darstellt [18]:

$$Loss = -\sum_{i=0}^{Anzahl\ der\ Klassen} y_i \cdot \log \hat{y}_i.$$

wobei $\hat{y}_i$der i-te Wert des realen Outputs und y_i der dazugehörige erwartete Output (Ground Truth) ist. Der Unterschied der Sparse Categorical Crossentropy zur normalen Categorical Crossentropy ist, dass die Categorical Crossentropy Werte One-Hot-Encoding benötigen, gegenüber der Verwendung von Integern bei der Sparse Variante. Beim One-Hot-Encoding werden die Label der Daten in einer binären Form dargestellt. Dieses Encoding hat den Vorteil, dass bei Problemstellungen bei denen Daten zu mehreren Klassen zugeordnet werden können, darstellbar sind. Die Sparse Categorical Crossentropy ermöglicht die Klassifizierung zu einer Klasse indem der Output der Funktion die Position der genannten Klasse ist. Bei der Cipher Type Detection soll ein Geheimtext nur zu einer Klasse zugeordnet werden, weshalb die Sparse Categorical Crossentropy verwendet wird.

2.3 Evaluierungsmetriken

Metriken sind zur Beurteilung eines neuronalen Netzes notwendig, jedoch verändern sie die Parameter eines neuronalen Netzes nicht. Die Konfusionsmatrix ist Grundlage vieler weiterführender Metriken, wie Precision und Recall. Sie besteht aus den vier statistischen Werten True Positives (TP), True Negatives (TN), False Positives (FP) und False Negatives (FN). Als TP wird eine Prädiktion gerechnet, die mit der echten Klasse übereinstimmt. In diesem Fall werden alle anderen Klassen als TN gewertet, weil sie wie erwartet nicht vorgeschlagen wurden. Falls eine falsche Auswahl stattfindet, wird die Prädiktion als FP und die richtige Klasse mit FN gewertet.

Die Genauigkeit und Top-K-kategorische-Genauigkeit Metriken [19] werden für die Beurteilung der Qualität des Modells verwendet. Die Genauigkeit wird berechnet, indem die Anzahl der gesichteten Datensätze durch die Anzahl der richtigen Klassifizierungen dividiert wird. Die Genauigkeit kann in diesem Kontext auch die Validierungsgenauigkeit genannt werden. Die Top-K-kategorische-Genauigkeit Metrik sagt gegenüber der normalen Genauigkeit aus wie gut das Modell die richtige Aussage in den Top K Ergebnissen, gegenüber nur dem ersten Ergebnis, liefert. Aufgrund der Aufgabenstellung wird K=3 verwendet, da das neuronale Netz als Hilfestellung für weitere Kryptoanalyse verwendet werden soll.

2.4 Aktivierungsfunktionen

Neuronale Netzwerken bestehen grundsätzlich aus mindestens einer oder mehreren Aktivierungsfunktionen. Eine Aktivierungsfunktion bestimmt mittels Schwellwerten welche Neuronen eines Layers aktiviert werden sollen und welche nicht. In diesem Abschnitt werden einige der existierenden Möglichkeiten beschrieben. Die Implementierungen basieren auf Tensorflow 2.3.0 [19] und Keras [18]. Aus diesem Grund werden bevorzugt existierende Aktivierungsfunktionen aus dem Keras-Modul vorgestellt. Abbildung 2.1 zeigt den Funktionsverlauf verschiedener Aktivierungsfunktionen. Aufgrund des lernenden Verhaltens der Parametric ReLU-Funktion kann diese nicht graphisch dargestellt werden.

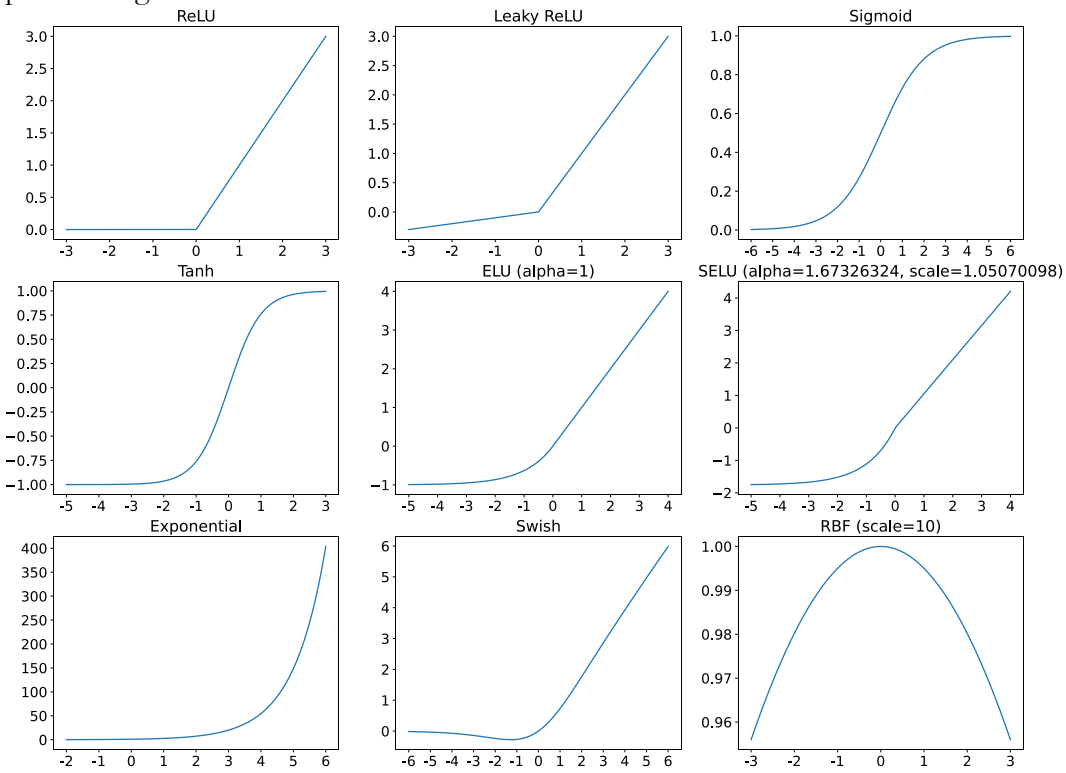


Abbildung 2.1: Grafische Darstellung der Aktivierungsfunktionen

2.4.1 Rectified Linear Unit

Die Rectified-Linear-Unit-(ReLU-)Funktion ist eine einfache, effiziente, nicht-lineare Funktion. Sie wird wie folgt für jeden Eingabewert definiert [18]:

$$relu(x) = \max(x, threshold),$$

wobei gilt, dass ein Wert, der den Threshold unterschreitet, entweder Null gesetzt oder mit einem Multiplikator < 1 verringert wird [18]. Neben den vielen Vorteilen dieser Funktion gibt es das „**Dying ReLU problem**" [20], wenn sich Eingabewerte Richtung Null entwickeln oder negativ sind. In diesem Fall wird der Gradient Null und das neuronale Netzwerk kann nicht lernen. **Leaky ReLU** und **Parametric ReLU** [20] sind zwei Ansätze, die helfen sollen, diesem Problem entgegenzuwirken. Die Leaky-ReLU-Funktion definiert einen Wert α, mit welchem der Gradient auf einen kleinen Wert gesetzt wird, damit das Lernen weiter funktionieren kann. Parametric ReLU basiert auf der gleichen Idee, jedoch wird bei diesem Ansatz der α Parameter gelernt, statt auf einen fixen Wert beschränkt.

2.4.2 Sigmoid

Die Sigmoid-Aktivierungsfunktion, auch logistische Funktion genannt, ist eine nicht-lineare Funktion, die Werte im Bereich [0..1] liefert. Vorteile dieser Funktion sind die automatische Normalisierung von Werten und die klaren Prädiktionen. Neben den Vorteilen dieser Aktivierungsfunktion gibt es auch Nachteile, wie z.B. der hohe Rechenaufwand, die Nicht-Nullzentrierung der Ausgaben und das „**Vanishing Gradient**"-Problem [20]. Letzteres bedeutet, dass sehr hohe oder sehr niedrige Werte fast keine numerische Änderungen des Gradienten bewirken und daher das Lernen sehr langsam wird oder stagniert. Die Funktion wird folgendermaßen definiert [18]:

$$sigmoid(x) = \frac{1}{1 + e^{-x}}.$$

2.4.3 Hyperbeltangens

Die Hyperbeltangens (tanh)-Funktion verläuft sehr ähnlich zur Sigmoid-Funktion und teilt auch die Vor- und Nachteile dieser Funktion. Der Wertebereich ist]-1..1], daher liegt das Zentrum der Funktion bei 0. Die Steigung der tanh-Funktion ist steiler als die der Sigmoid-Funktion. Die Funktion wird wie folgt definiert [18]:

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^{2x} - 1}{e^{2x} + 1}.$$

2.4.4 Softmax

Softmax ist eine Kombination aus mehreren Sigmoiden und ermöglicht dadurch eine Klassifizierung für mehrere Klassen durchzuführen, im Gegensatz zu binären Klassifizierungsproblemen, die mit der Sigmoid-Funktion gelöst werden. Die Softmax-Funktion berechnet Wahrscheinlichkeiten für die Zugehörigkeit eines Datenpunktes zu allen Klassen und wird daher meistens als letzter Layer eines neuronalen Netzes verwendet. Die Funktion wird folgendermaßen definiert, wobei K die Anzahl der Klassen ist [18]:

$$softmax(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \text{ für } j = 1, \dots, K.$$

2.4.5 Exponential Linear Unit

Die Exponential Linear Unit (ELU) ist der Leaky ReLU und Parametric ReLU sehr ähnlich, jedoch verwendet diese statt einer Geraden eine logarithmische Kurve für die negativen Werte. Eine Definition der ELU sieht folgendermaßen aus [21]:

$$elu(x) = f(x) = \begin{cases} x, & x > 0, \\ \alpha \cdot (e^x - 1), & x \leq 0. \end{cases}$$

2.4.6 Scaled Exponential Linear Unit

Die Scaled Exponential Linear Unit (SELU) ist nahezu gleich zur ELU, mit dem Unterschied, dass ein konstanter Wert für α und ein zweiter Parameter $scale$ mit konstantem Wert verwendet wird. Die Formel lässt sich wie folgt definieren [18]:

$$selu(x) = f(x) = \begin{cases} scale \cdot x, & x > 0, \\ scale \cdot \alpha \cdot (e^x - 1), & x \leq 0 \end{cases}$$

$$\text{für } \alpha = 1.67326324 \text{ und } scale = 1.05070098.$$

2.4.7 Exponential

Die Exponentialfunktion hat den Vorteil, dass alle negativen Werte zu sehr kleinen positiven Werten umgerechnet werden und alle positiven Werte hochskaliert werden. Sie wird folgendermaßen definiert [18]:

$$\exp(x) = e^x.$$

Aufgrund der hohen Skalierung dieser Funktion ist die Anwendbarkeit nur auf bestimmte Fälle eingeschränkt und sollte nur in Kombination mit anderen Funktionen verwendet werden.

2.4.8 Swish

Swish ist eine Aktivierungsfunktion, die 2017 von Ramachandran, Zoph und Le entwickelt wurde [22]. Swish ist ähnlich rechenaufwändig wie die Sigmoid-Funktion, jedoch können oft bessere Ergebnisse als mit der ReLU-Funktion erzielt werden. Die Funktion wird in der Literatur folgendermaßen definiert [22]:

$$swish(x) = x \cdot sigmoid(x) = \frac{x}{1 - e^{-x}}.$$

Der Wertebereich der Funktion liegt zwischen -∞ und ∞. Eine Eigenschaft der Swish-Funktion ist, dass sie nicht monoton ist. Das bedeutet, dass das Ergebnis kleiner werden kann, obwohl der Eingabewert steigt. Die nicht-monotone Eigenschaft ist vorteilhaft, da kleine negative Eingabewerte zu negativen Ausgabewerten führen und es dadurch möglich ist Schemen in kleinen negativen Wertebereichen zu erkennen bei denen z.B. die ReLU-Funktion Null-Werte liefern würde.

2.4.9 Radiale Basisfunktion

Die Radiale Basisfunktion (RBF) basiert auf der Gaußschen Glockenkurve. Sie verwendet einen Parameter für die Festlegung des Zentrums c (meistens der Mittelwert) von dem die Distanz der Punkte berechnet wird. Die Summe der mit dieser Funktion berechneten Werte kann als Annäherung einer Funktion und damit als simples neuronales Netz mit einem Layer interpretiert werden [23]. Die Annäherungsfunktion wird in der Literatur mit einer Gewichtung w_i, einem Zentrum c_i für jedes Features und einer Skalierung scale wie folgt definiert [23]:

$$y(x) = \sum_{i=1}^{N} w_i \cdot rbf(x, c_i) = \sum_{i=1}^{N} w_i * e^{\frac{-(x-y)^2}{2*scale^2}}.$$

2.5 Learning Rate Scheduler

Die Lernrate eines Optimizers für neuronale Netze kann als der wichtigste Hyperparameter bezeichnet werden. Eine zu hohe Lernrate bedeutet, dass ein Modell eine schlechte Genauigkeit liefert. Eine zu niedrige Lernrate bewirkt, dass ein Modell sehr lange mit vielen Daten trainieren muss, bis es konvergiert. Der Adam-Optimizer hat zwar einen internen Mechanismus für die Anpassung der Lernrate, jedoch funktioniert dieser stochastisch und berechnet eine effektive Lernrate zwischen 0 und Lernrate. Daher kann es sinnvoll sein, die maximale Lernrate anzupassen. In den folgenden Unterabschnitten werden unterschiedliche Algorithmen für die Anpassung der Lernrate vorgestellt. Diese ermöglichen es eine höhere initiale Lernrate zu verwenden und damit die benötigte Zeit zur Konvergenz eines Modells deutlich zu kürzen. Nachfolgend werden Schedulers betrachtet, die mit einer maximalen Lernrate starten und diese über den Trainingsverlauf verringern.

2.5.1 Time-Based Decay

Der einfachste Scheduler ist der Time-Based Decay und er wird in der Keras-Implementierung des SGD-Optimizers [18] verwendet. Die Lernrate wird nach jedem Batch (t) mit einem Faktor multipliziert, der von dem fixen Parameter **decay** und der Anzahl der bereits trainierten Iterationen abhängig ist. Mathematisch lässt sich der Prozess folgendermaßen darstellen [24]:

$$lr_{t+1} = lr_t \cdot \frac{1}{1 + decay \cdot iterations_{t+1}}.$$

2.5.2 Step- / Drop-Based Decay

Der Step oder Drop-Based Decay basiert auf einer stufenweisen Verringerung der Lernrate nach einer fixen Anzahl von Epochen (drop) um einen fixen Faktor (decay). Zum Beispiel kann die Lernrate alle 10 Epochen halbiert werden. Die Formel dafür sieht wie folgt aus und wird jede Epoche (t) aufgerufen [24]:

$$lr_t = lr_0 \cdot decay^{\left\lfloor \frac{t}{drop} \right\rfloor}.$$

2.5.3 Exponential Decay

Der exponentielle Decay hat den Vorteil, dass die Lernrate zu Beginn mit höheren Werteanpassungen reduziert wird und diese exponentiell kleiner werden und somit das Modell besser konvergieren kann. Die Lernrate wird nach jeder Epoche mit folgender Formel berechnet, wobei k ein Hyperparameter und t die Anzahl der Epochen ist [24]:

$$lr_t = lr_0 \cdot e^{-kt}.$$

2.6 Architekturkonzepte für die Textklassifikation

Die Problemstellung bei der Cipher Type Detection ist die Zuordnung einer Klasse zu einer gegebenen Sequenz. Dabei soll der Geheimtext selbst oder extrahierte Statistiken und Features, mithilfe eines neuronalen Netzes, genau einer Klasse zugeordnet werden. Nicht zu verwechseln sind diese Klassen mit Sequence-to-Sequence-Architekturen, die z.B. Spracherkennung oder Textvervollständigung zur Aufgabe haben [25]. Dieser Abschnitt soll eine grundlegende Beschreibung von möglichen Ansätzen zur Klassifizierung von Sequenzen vorstellen. Folgend wird ein guter, jedoch unvollständiger, Überblick aller wesentlichen Algorithmen geschaffen. Grundsätzlich lassen sich die Architekturen in zwei Hauptkategorien unterteilen: Feature-Engineering und Feature-Learning neuronale Netze.

Die folgenden Punkte beschreiben die wichtigsten Architekturen für die Klassifizierung von Textsequenzen. Unsupervised-Learning-Algorithmen, wie z.B. Self-Organizing Maps werden, aufgrund der nahezu unbegrenzt zur Verfügung stehenden gelabelten Daten, in dieser Arbeit nicht betrachtet.

2.6.1 Feature-Engineering-Algorithmen

Feature-Engineering-Verfahren haben die Eigenschaft, dass durch Expertenwissen ausgewählte Eigenschaften und Statistiken als Input für das Modell bereitgestellt werden. In Bezug auf Textklassifizierung sind die Häufigkeit der Zeichen und Entropie Beispiele für mögliche Features. Kryptoanalyse bietet sich, insbesondere bei klassischen Chiffren, für die Modellierung von Domänenwissen in Features an. Ein essenzieller Vorteil dieser Verfahren ist, dass bekannte Schwächen von Chiffren als Features modelliert und in das neuronale Netz trainiert werden können. Der größte Nachteil dieser Verfahren ist der initiale Aufwand zur Konstruktion und Implementierung der genannten Features. Folgende Verfahren werden in den nächsten Absätzen genauer erläutert:

- Feed-Forward Neural Network (FFNN)
- Support Vector Machine (SVM)
- Entscheidungsbäume (EB)
- Restricted Boltzmann Machine (RBM)
- Deep Belief Network (DBN)
- Naive Bayes Network (NBN)

Feed-Forward Neuronale Netze (FFNN) basieren auf differenzierbaren Aktivierungsfunktionen und dem Finden des lokalen Minimums für die Gradient-Descent-Loss-Funktion. Der Aufbau eines FFNN besteht aus einem oder mehreren Layern. Die Layer zwischen den Input- und Output-Layern werden Hidden Layer genannt. Ein Neuron hat niemals Verbindungen zu anderen Neuronen, jedoch wird der Output eines Layers als Input des nächsten Layers verwendet. Das Ziel der Trainingsphase ist die Berechnung der optimalen Gewichte für die Minimierung des Fehlers. Die Komplexität eines Modells wird von der Anzahl und Breite der Hidden Layer bestimmt und darf weder zu einfach noch zu kompliziert sein.

Support Vector Machines (SVM) können für eine Vielzahl von Klassifizierungs- und Regressionsproblemen eingesetzt werden. Die ursprüngliche SVM funktioniert mit zwei Klassen, die miteinander verglichen werden, und einer linearen Klassifizierungsfunktion. Viele Klassifizierungsprobleme sind in ihrem Vektorraum nicht linear separierbar und daher wurde das Konzept der Kernel-Funktionen entwickelt. Die Aufgabe

dieser Funktionen ist es, Features in einen höherdimensionalen Raum zu überführen, ohne diese Funktion tatsächlich rechnerisch ausführen zu müssen. [26]
Für die Klassifizierung von mehr als zwei Klassen wird eine Multi-Klassen-Methode benötigt. Dafür gibt es grundsätzlich zwei Ansätze. Der erste Ansatz trainiert für alle Klassen eine SVM und verwendet die maximale Spanne der Ergebnisse für die Klassifizierung. Dieser Ansatz wird auch One-vs-Rest-Klassifikator genannt. Ein zweiter Ansatz ist die Berechnung von SVMs für alle Klassenpaare. Dieser Ansatz verwendet sog. One-vs-One-SVM-Klassifikatoren. Dadurch ergeben sich n(n-1)/2 Klassifikatoren. Die Evaluierung der Testdaten mit den One-vs-One-SVMs ergibt jeweils eine Stimme für die gewinnende Klasse. Die Klasse mit den meisten Stimmen wird für die Klassifizierung verwendet. [27] Laut Louradour und Larochelle [28] ist der größte Nachteil von Kernel-basierten SVMs die schlechte Skalierbarkeit mit großen Datensätzen. Die Komplexität für die Optimierung einer SVM ist quadratisch zu der Anzahl der Trainingsdatensätze. Die Komplexität der Berechnung der Kernel zwischen den Mengen ist quadratisch zur Anzahl der Vektoren pro Menge.

Entscheidungsbaum-(EB-)Algorithmen treffen Entscheidungen durch den Aufbau von binären Bäumen. Jeder EB hat einen Wurzelknoten, interne Knoten und Blattknoten. Die Entscheidung findet schlussendlich immer in einem Blattknoten statt. EB sind anfällig für Overfitting, und daher Missklassifizierung, weshalb der Aufbau des EB in zwei Phasen unterteilt wird: Training und Pruning. Beim Training wird der Baum mit allen Knoten aufgebaut. Die Aufgabe der Pruning-Phase ist das Entfernen von selten verwendeten Knoten für die Verbesserung der Genauigkeit und Laufzeit des EB. EB können in serielle Algorithmen, wie z.B. C4.5 oder CART und parallele Algorithmen, wie z.B. Random Forests, unterteilt werden. [29]

Restricted Boltzmann Machines (RBM) sind neuronale Netze mit bidirektional verbundenen stochastischen Verarbeitungseinheiten. RBMs lernen mit dem Gibbs-Sampling-Algorithmus Wahrscheinlichkeitsverteilungen in Bezug auf die Input-Daten. RBMs bestehen immer aus zwei Layern, einem Visible und einem Hidden Layer. Jede Einheit aus einem Layer ist mit jeder Einheit des anderen Layers verbunden, es existieren jedoch keine Verbindungen zwischen Einheiten des gleichen Layers. Durch das Starten in einem zufälligen Zustand eines Layers und die Anwendung des Gibbs-Samplings können Daten aus einer RBM generiert werden. Diese Daten werden dann für den Update-Prozess verwendet. Dieser Prozess wird so lange durchgeführt, bis ein Gleichgewicht der Verteilung erreicht wird. Die **weights** können durch die Maximierung der Wahrscheinlichkeit der RBM errechnet werden. Eine RBM kann als eigenständiges neuronales Netz oder für die Extraktion von Features für andere Architekturen verwendet werden. [30]

Aufgrund der Bidirektionalität vom Visible und Hidden Layer einer RBM sind **Deep Belief Networks** (DBN) entstanden. DBNs bestehen aus mehreren gestapelten RBMs und sollen die Abhängigkeit mit mehreren stochastischen und verborgenen Layern untersuchen. Der Input eines Layers ist der Output des vorherigen. Das Training eines DBN besteht aus zwei Phasen: Der Pre-Training Phase und der Fine-Tuning Phase. In der Pre-Training Phase wird Unsupervised Learning durchgeführt, beginnend bei den untersten Layern. Die Fine-Tuning-Phase adjustiert das neuronale Netz mit Supervised Learning von den obersten Layern aus. [30]

Naive Bayes Networks (NBN) basieren auf der Annahme, dass alle Attribute oder Features von Daten vollständig unabhängig voneinander sind. NBN-Klassifikatoren treffen Entscheidungen durch Multiplikation der bedingten Wahrscheinlichkeiten [31]. Abhängig davon ob das Klassifizierungsproblem eine oder mehrere Klassen erfordert, muss eine Entscheidungsfunktion implementiert werden. Bei eindeutigen Entscheidungsproblemen wird in den meisten Fällen die Klasse mit der größten Wahrscheinlichkeit gewählt. Klassifizierungsprobleme mit mehreren Ergebnissen können mittels einer Schwellwertmethode klassifiziert werden. Grundsätzlich kann zwischen Bernoulli und Multinomialen NBN unterschieden werden. Bernoulli NBN können nur binäre Features verwenden. Im Gegensatz dazu sind Multinomiale NBN im Stande diskrete Daten für die Klassifizierung zu nutzen. [32]

2.6.2 Feature-Learning-Algorithmen

Im Gegensatz zu den im letzten Punkt beschriebenen Algorithmen haben Feature-Learning-Algorithmen die Fähigkeit, selbst aus den Daten wichtige Features zu extrahieren. Maschinelle Lernverfahren können in vielen Aufgabenstellungen besser Features entdecken als es ein Mensch kann [33]. In diesem Unterabschnitt ist der Stand der Technik von Feature-Learning-Architekturen für neuronale Netze aufgezeigt. Die folgenden Feature-Learning-Verfahren werden in den nächsten Abschnitten vorgestellt:

- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Transformer

Convolutional Neural Networks (CNN) basieren auf der mathematischen Operation Convolution. Im Bereich der neuronalen Netze ist Convolution eine Kombination von zwei Funktionen, die echte Werte verarbeiten. Ein Beispiel dafür wäre die Berechnung des Wertes zu einem bestimmten Zeitpunkt und die Berechnung eines gewichteten Durchschnittswertes. Die erste Funktion wird im allgemeinen als **Input** bezeichnet. Die gewichtete Funktion wird als **Kernel** bezeichnet. Der Output der Convolution kann als **Feature Map** bezeichnet werden und stellt die extrahierten Features dar. CNNs haben die Eigenschaft gegenüber anderen Architekturen für neuronale Netze, dass sie eine variable Anzahl an Inputs verarbeiten können. Das kann erreicht werden, wenn der Kernel kleiner dimensioniert ist als der Input. Ein typisches CNN besteht aus drei Layern. Im ersten Layer werden mehrere Convolutions durchgeführt, um Werte aus linearen Aktivierungsfunktionen zu berechnen. Die Werte werden als Input für eine nicht-lineare Aktivierungsfunktion verwendet. Diese Phase wird auch **Detection Stage** genannt. **Pooling**-Funktionen modifizieren den Output in die Richtung, dass sie mehrere Werte zusammenfassen und z.B. den maximalen Wert auswählen (Max Pooling). [10, pp. 326-366]

Ein **Recurrent Neural Network** (RNN) ist ein neuronales Netz, das den temporalen Kontext, in dem sich das Training befindet, berücksichtigt. Im Gegensatz zum FFNN haben RNNs die Eigenschaft, dass zyklische Verbindungen im Netz existieren und sie mappen nicht nur den Input zum Output, sondern auch eine bestimmte Menge des bisherigen Inputs. Eine spezielle RNN-Architektur ist Long Short-Term Memory (LSTM) [34]. Ein LSTM-Netz ist gleich wie ein RNN, außer dass die Summierungseinheiten der Hidden Layer mit Speicherblöcken ersetzt werden. Diese Speicherblöcke bestehen aus drei multiplikativen Einheiten: Input, Output und Forget Gate. Durch diese Architektur wird es dem LSTM Netz möglich Informationen über längere Zeiträume zu speichern und darauf zuzugreifen. Das LSTM lernt Zusammenhänge aus den Daten selbst. [35]

Transformer-Netzwerke sind sog. Attention Networks und gehören zum neuesten Stand der Technik im Bereich der Sequenzverarbeitung in neuronalen Netzen. Attention-Funktionen mappen eine Query und eine Menge an Key-Value Paaren zu einem Output, in dem die Queries, Keys, Values und Output Vektoren sind. Der Output ist eine gewichtete Summe von weights, die durch eine Kompatibilitäts-funktion für den dazugehörigen Key berechnet werden. [36]

2.6.3 Auswahl der Verfahren

Der vorläufige Auswahlprozess für die Netzwerkarchitektur und die Kriterien für die Entscheidung werden folgend beschrieben:

- Gute Ergebnisse aus ähnlichen Forschungsfeldern in anderen Arbeiten
- Existieren bereits Frameworks, die eine Implementierung vereinfachen?
- Effiziente Nutzung der vorhandenen Hardware im Zusammenhang mit Parallelisierbarkeit und Nutzung der GPUs
- Anzahl von konfigurierbaren Hyperparametern (weniger ist besser)

Aufgrund der bereits existierenden Implementierung aus dem CANN-Projekt werden FFNN als erste Architektur betrachtet. In der folgenden Aufzählung werden die gewählten Architekturen in der Auswahlreihenfolge aufgezählt:

- Feed-Forward Neural Network (FFNN)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Transformer
- Support Vector Machine (SVM)
- Entscheidungsbaum (EB)
- Restricted Boltzmann Machine (RBM)
- Deep Belief Network (DBN)
- Naive Bayes Networks (NBN)

Kapitel 3 Stand des Wissens

Im folgenden Kapitel ist der Stand der Technik und die Ausgangsstellung detailliert beschrieben. Abschnitt 3.1 bietet einen Überblick zu klassischen Chiffren und deren Unterscheidungsmerkmalen. Der Stand der Technik im Bereich der Cipher Type Detection ist in Abschnitt 3.2 zu finden. Die Ausgangsstellung, welche bereits mit Abschnitt 1.2 beginnt, ist in Abschnitt 3.3 weiter präzisiert worden.

3.1 Überblick zu Klassischen Chiffren

Klassische Chiffren können grundsätzlich in die drei Kategorien Substitution, Transposition und deren Kombination unterteilt werden [37]. Bei der monoalphabetischen Substitution wird jeder Klartextbuchstabe durch einen anderen ersetzt. Polyalphabetische Chiffren unterscheiden sich dadurch, dass ein Klartextbuchstabe durch mehrere Buchstaben ersetzt werden kann. Ein Beispiel für polyalphabetische Substitution stellt die Vigenère-Chiffre dar [37, p. 86]. Mithilfe eines Schlüsselworts wird das zu verwendende Alphabet bestimmt. Als polygraphische Substitution wird das Ersetzen von Blöcken von Buchstaben durch andere Blöcke bezeichnet. Polygraphische Substitutionschiffren haben, aufgrund der Verschleierung von Buchstabenhäufigkeiten, gegenüber monoalphabetischen Chiffren bessere kryptographische Eigenschaften. Ein Beispiel aus der ACA-Liste von Chiffren-Typen und gleichzeitig eine Spezialform von polygraphischen Chiffren, nämlich die bigraphische Substitution, bei der Blöcke der Länge zwei verschlüsselt werden, ist die Playfair-Chiffre [4, p. 63]. Transpositionschiffren verändern nur die Reihenfolge der Buchstaben eines Klartextes, ohne diese zu ersetzen.

Entropie ist ein wichtiges Maß für die Bestimmung der Sicherheit einer kryptographischen Funktion, wie z.B. einer Chiffre. Im Bereich der Informationstheorie bedeutet eine hohe Entropie ein hohes Maß an Zufälligkeit in der Häufigkeitsverteilung von Zeichen. Sie wird folgendermaßen berechnet [38]:

$$H = -\sum_{i} p_i log_2(p_i).$$

wobei i der Index über den Text und p_i die Auftrittswahrscheinlichkeit des Zeichens an der Stelle i ist.

Die Auftrittshäufigkeiten einzelner Buchstaben für verschiedene natürliche Sprachen sind bekannt, weshalb die Entropie ein Indiz für die verwendete Sprache sein kann. Transpositionschiffren verändern den Wert der Entropie nicht, wodurch sie gut klassifizierbar sind.

3.2 Cipher Type Detection

Die Erkennung des verwendeten Verschlüsselungsalgorithmus aus einem Geheimtext ist eine zentrale Aufgabe der Kryptoanalyse. Nuhn und Knight [5] haben die Extraktion eines Geheimtextes aus einer verschlüsselten Nachricht in drei Schritte unterteilt:

1. Erkennung des Algorithmus, mit dem die Nachricht verschlüsselt wurde
2. Finden des verwendeten Schlüssels
3. Entschlüsselung der Nachricht

Dieser Abschnitt beschreibt verschiedene Ansätze zur Erkennung von Chiffren und den Ergebnissen dieser Arbeiten. Dabei ist es wichtig zwischen klassischen und modernen Chiffren zu unterscheiden, denn die letzteren lassen sich zumeist nicht mit Statistiken klassifizieren.

Einen Meilenstein in der Klassifizierung klassischer Chiffren haben Nuhn und Knight im Jahr 2014 mit neuronalen Netzen erreicht [5]. Die Forscher haben ein neuronales Netz mit einem linearen Klassifikator und einem SGD Optimizer mit Standardparametern für 50 ACA-Chiffren trainiert. Mit einer quadratischen Loss-Funktion und adaptiven Lernraten konnte mit 1 Million Datensätzen und 20 Durchläufen 58,5% Genauigkeit erreicht werden. Sie haben mit 55 Features von Bion [11] und drei selbst entwickelten Features gearbeitet.

Einen kleiner dimensionierten Ansatz haben Sivagurunathan et al. [39] entwickelt. Sie haben ein einfaches neuronales Netz mit einem Input, Hidden und Output Layer für die Chiffren Playfair, Vigenère und Hill mit jeweils 300 Texten der Länge 1.000 und Schlüssellängen zwischen 2 und 26 trainiert. Das Ergebnis war eine sehr gute Erkennungsrate von ca. 90%, jedoch konnten Geheimtexte der Hill-Chiffre mit zu kurzer Schlüssellänge, beginnend bei 2, aufgrund der ähnlichen Eigenschaften zu der Vigenère-Chiffre nicht richtig klassifiziert werden.

Abd und Al-Janabi [40] haben die Schwäche der Ein-Layer-Klassifizierung der von Nuhn und Knight [5] entwickelten Architektur aufgezeigt. Mit sieben Features und drei Klassifizierungs-Layern konnten sie, mit eingeschränkten Voraussetzungen, eine Genauigkeit von 99,6% für Klartexte und zehn verschiedenen Chiffren erreichen.

Diese sind Transposition, Caesar, Simple Substitution, Combination, Vigenère, Playfair, Hill mit jeweils 3x3, 4x4 und 5x5 Matrizen und Autokey. Der erste Layer ist für die Klassifizierung in Substitution, Combination, Transposition und Klartext verantwortlich. Layer 2 wurde nur für die Klasse der Substitution implementiert und besteht aus Monoalphabetic, Polyalphabetic und Polygraph. Wiederum wurde der 3. Layer nur für Polygraphische Substitutionen implementiert und in Playfair, Hill mit 3x3 und 4x4 Matrizen unterteilt. Ein markanter Nachteil dieser Ergebnisse ist die sehr große Textlänge von über einer Million Buchstaben, die für das Berechnen der einzelnen Geheimtexte für das Training und die Testung des neuronalen Netzes verwendet wurden. Aufgrund dieser enormen Menge an Textdaten, die in etwa mit einem 500-seitigen Buch vergleichbar ist, sind die Ergebnisse mit anderen Arbeiten nicht vergleichbar, jedoch kann die Idee der Multi-Layer-Architektur nützlich in weiterer Forschung sein.

Ansätze, die von anderen Autor*innen noch nicht benutzt wurden, hat Krishna [41] 2019 für die vier Chiffren Simple Substitution, Vigenère, Transposition und Playfair entwickelt. Ein für die Vergleichbarkeit wichtiger Punkt ist, dass die Hill-Chiffre hier nicht verwendet wurde. Der erste Ansatz, eine Support Vector Machine (SVM), verwendet direkt die in einen Zahlenraum gemappten Geheimtexte der Länge 10 bis 10.000 als Trainingsdaten. Die SVM verwendet die Implementierung der Sklearn Bibliothek [32] mit einer 10-fachen stratifizierten Kreuzvalidierung und einem One-vs-Rest-Klassifikator für die Berechnung der Konfusionsmatrix. Das bedeutet, dass 9 von 10 Datensatzteilen für das Training und ein Datensatzteil zum Testen verwendet wurde, um die am besten geeignete Klasse zu finden. Im zweiten und dritten Ansatz wurde für 1.000 Geheimtexte pro Klasse ein Hidden Markov Model (HMM) trainiert und mittels Umwandlung als Input für ein Convolutional Neural Network (CNN) im zweiten Ansatz und eine SVM im dritten Ansatz verwendet. Der erste Ansatz erreicht eine Genauigkeit von 100% ab einer Textlänge von 200, der zweite 71% ab einer Textlänge von 155 und der dritte 100% ab einer Textlänge von 155.

Zhao et al. [42] haben 54 Features aus 15 verschiedenen NIST 800-22 Randomness-Tests [43] extrahiert. Die Effizienz wurde in mehreren 10-fachen Kreuzvalidierungen mit One-vs-One-SVMs für sechs moderne Blockchiffren (AES, Blowfish, Camellia, DES, 3DES und IDEA) isoliert für jedes Feature getestet. Das Ergebnis war, dass 42 Features bessere Ergebnisse als zufälliges Raten, also 50%, liefern. Die besten 12 dieser Features liefern jeweils sogar eine Erkennungsrate über 60%.

Ein sehr ähnliches Modell haben Tan und Ji [44] 2016 mit den fünf modernen Chiffren AES, Blowfish, DES, 3DES und RC5 entwickelt. Die Experimente wurden in dieser Arbeit mit zwei Szenarien durchgeführt: Einmal wird das gleiche Schlüsselmaterial für Trainings- und Testdaten verwendet und das andere Mal wird mit unterschiedlichem Schlüsselmaterial getestet. Für die gleichen Schlüssel ist das Ergebnis ab 20 kB Daten 85% und ab 100 kB Daten 96%. Für unterschiedliche Schlüssellängen mit denselben Datenmengen sind es 35% bzw. 39%. Die verwendeten Parameter der SVM wurden nicht genauer erläutert.

Der C4.5-Algorithmus [45] erzeugt einen Entscheidungsbaum, welcher auf dem Verhältnis des Informationsgewinns (information gain ratio) basiert. Manjula und Anitha [46] haben 2011 einen C4.5-Klassifikator für elf moderne Chiffren entworfen und eine Erkennungsrate von über 70% für Geheimtexte mit einer variablen Länge von 1-2.000 Bytes erreicht. Insgesamt wurden zehn Features entworfen, wobei sieben davon auf der maximalen Entropie unterschiedlicher Zeichen basieren. Weitere Features sind die Entropie aller Zeichen, Correlation coefficient von Großbuchstaben und die Länge der Geheimtexte, da von dieser die erwarteten Entropien abhängen.

Verschiedene, auf Backpropagation basierende, Algorithmen für die One-vs-One-Klassifizierung, also einer Gegenüberstellung einzelner moderner Strom- und Blockchiffren, wurden 2007 von Chandra et al. [47] vorgestellt. Durchschnittlich haben alle Algorithmen über 80% Genauigkeit erzielt, jedoch konnte die Resiliente Backpropagation besonders im Vergleich Stromchiffre vs. Stromchiffre über 6% bessere Ergebnisse erzielen. Das Training wurde mit Texten der Länge 12,8 kB und 46 Features, die nicht genauer beschrieben sind, durchgeführt.

Um eine einfache und übersichtliche Zusammenfassung der in diesem Abschnitt vorgestellten Arbeiten zu geben, werden in Tabelle 1 allgemeine Eigenschaften, wie z.B. die verwendete Textlänge oder der Typ der Chiffren zusammengefasst. Ein direkter Vergleich der Arbeiten durch diese Tabelle ist nicht möglich, denn die Ergebnisse der Arbeiten wurden nicht mit gleichartigen Problemstellungen geschaffen.

Autor*innen	# Features	Genauigkeit in %	Textlänge	Größe des Datensatzes	# Epochen	# Chiffren	Chiffren-Typ	verwendete Technologien
Nuhn & Knight [5]	58	58,50	zufällig	1.000.000	20	50	klassisch	Vowpal Wabbit
Kopal [3]	4	90	100	4.500	20	5	klassisch	FFNN
Siv. et al. [39]	12	84,75	1.000	900	k. A.	3	klassisch	FFNN
Abd & Al-Janabi [40]	7	99,60	1.000.000	k. A.	500	11	klassisch	3-Level-Klassifikator
Krishna [41]	k. A.	100	155	4.000	k. A.	4	klassisch	SVM, HMM
Zhao et al. [42]	54	47,8-89,5	512.000	6.000	k. A.	6	modern	One-vs-One SVM
Tan & Ji [44]	k. A.	39%	100.000	1.100	k. A.	5	modern	SVM
Manjula & Anitha [46]	10	72,20	1-2.000	150	k. A.	11	modern	EB
Chandra et al. [47]	46	80	12.800	1.000	k. A.	3	modern	One-vs-One FFNN

Tabelle 1: Zusammenfassung des Stands der Forschung im Bereich Cipher Type Detection

3.3 CANN – CryptAnalysis with Neural Networks

Das CANN-Projekt bietet die Möglichkeit zum automatisierten Training von neuronalen Netzen sowie deren Evaluierung durchzuführen. In der Trainingsphase werden mit automatisiert generierten Schlüsseln verschiedene Geheimtexte für das Training vorbereitet. Aus diesen Geheimtexten werden verschiedene Statistiken, die sog. Features, welche als Eingaben für das neuronale Netz dienen, berechnet.

Abbildung 3.1 zeigt den Trainingsprozess des Chiffren-Klassifizierungs-Modells. Für diesen Zweck werden die frei verwendbaren, englischen Texte des Gutenberg-Projekts [48] eingesetzt. Das Laden und Vorberechnen der Statistiken passiert mittels eines eigens dafür entwickelten Dataloaders und wird im nächsten Absatz genauer beschrieben. Das Vorberechnen und Trainieren des neuronalen Netzes passiert sequenziell und bietet daher einen Verbesserungspunkt für die Anpassung der Laufzeit eines Trainingsprozesses. Die Statistiken werden schon parallel berechnet, jedoch ist es möglich während des Trainings des Modells gleichzeitig die Statistiken für den nächsten Teil des Trainings vorzubereiten. Nachdem der Trainingsprozess abgeschlossen ist, wird das Modell gespeichert und evaluiert.

In Abbildung 3.2 ist der beschriebene Dataloader zu sehen. Dieser lädt eine oder mehrere Textzeilen aus dem Datensatz und filtert diese mit der für die Chiffre passenden Filter-Funktion. Beispielsweise müssen in der Playfair-Chiffre [4, p. 63] alle J mit I ersetzt werden. Jede Chiffre verwendet eine eigene Filter-Funktion, welche nach den Spezifikationen der ACA [4] implementiert wurde. Der Längenbereich des Gesamttextes ist mittels Parameter einstellbar. Dieser Bereich ist für die Laufzeit des Trainings fixiert. Grundsätzlich ist anzunehmen, dass längere Nachrichten besser zu klassifizieren sind, da die berechneten Features mehr statistische Aussagekraft besitzen. Der beschriebene Prozess muss so lange durchgeführt werden, bis die Anzahl der generierten Klartexte gleich der Größe des Datensatzes, welcher im Programm als Parameter definiert ist, durch die Summe der Chiffren und deren konfigurierten Schlüssellängen entspricht. Damit kann jeder Klartext einmal für jede Chiffre mit jeder der konfigurierten Schlüssellängen für das Training des Modells verwendet werden. Nachdem ausreichend Textzeilen zur Verfügung stehen, werden mehrere Prozesse, sog. Worker, gestartet, um die Features parallel zu berechnen.

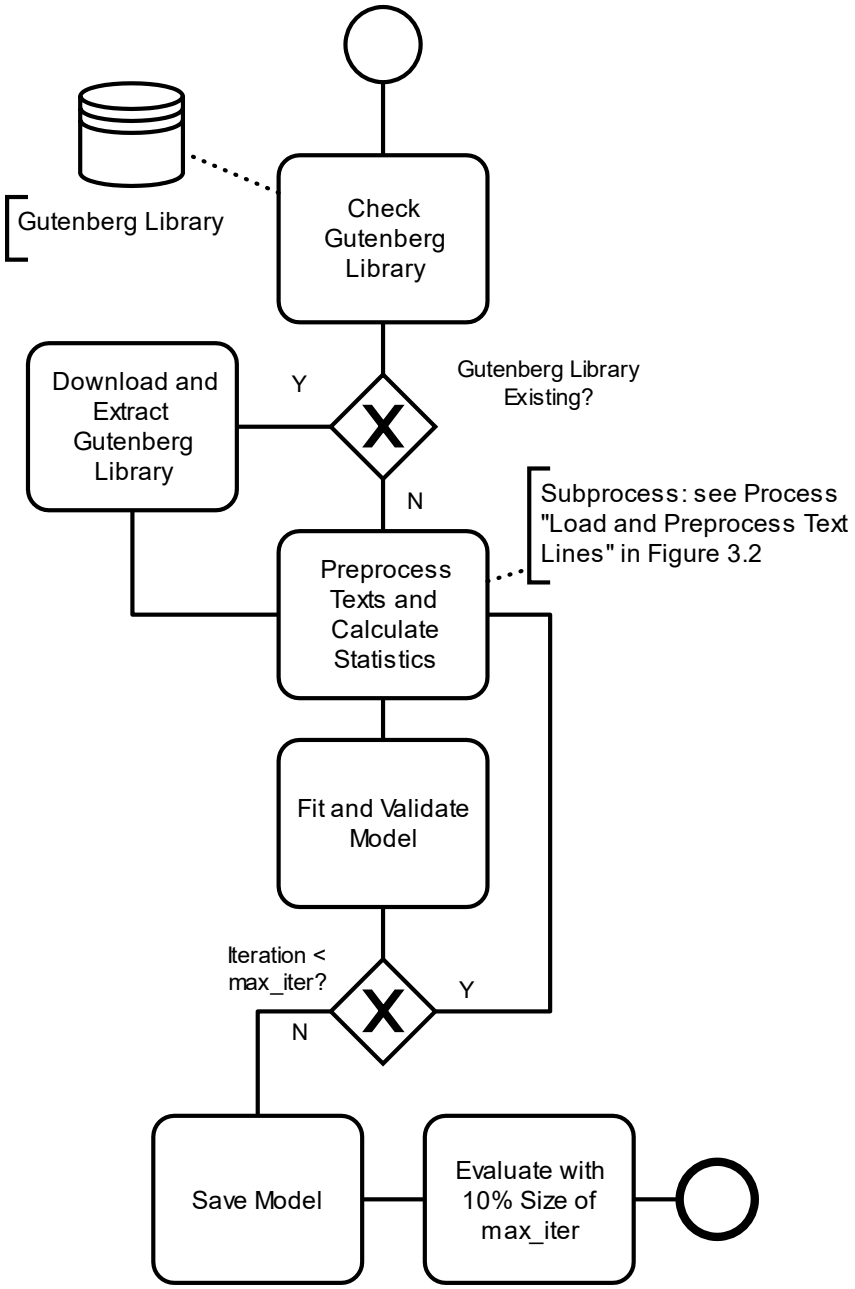


Abbildung 3.1: Trainingsprozess des Klassifizierungsmodells

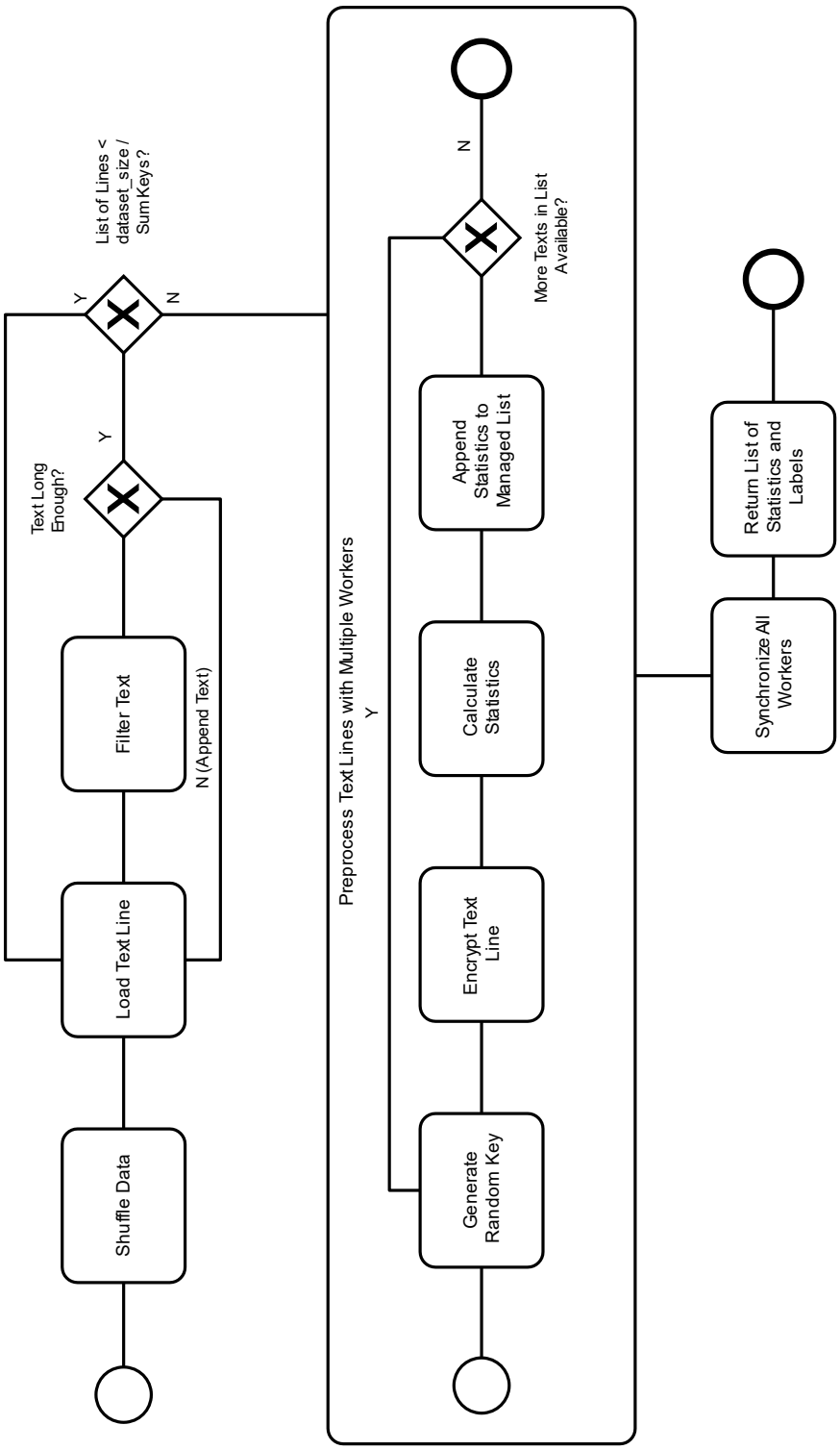


Abbildung 3.2: Prozess für das Laden und Vorverarbeiten der Texte zur Berechnung der Werte der Features (DataLoader)

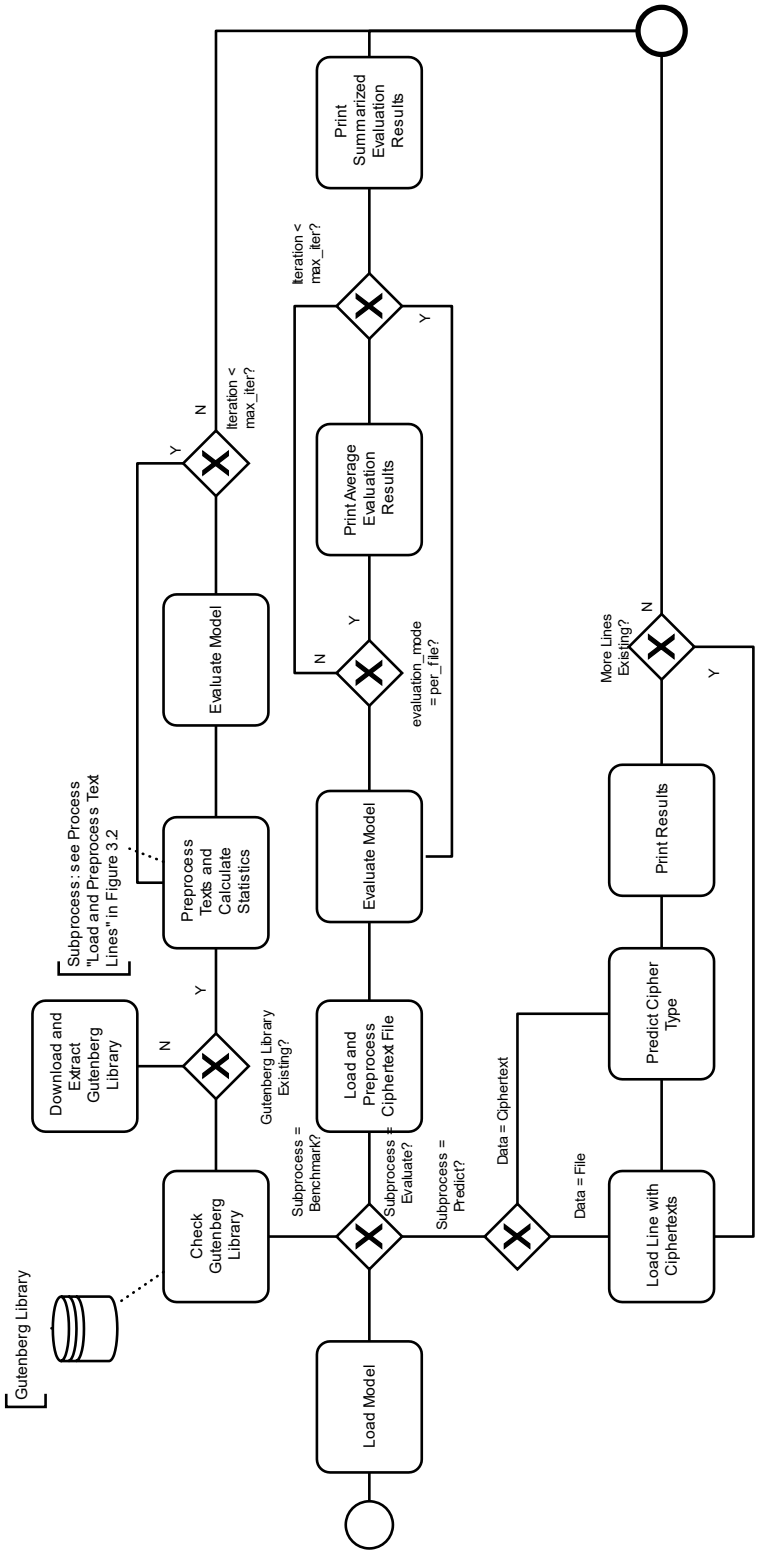


Abbildung 3.3: Evaluierungsprozess der drei Szenarien (Benchmark, Evaluate, Predict)

Für die Nutzung des trainierten neuronalen Netzes werden die folgenden drei Szenarien betrachtet. Der genaue Ablauf der einzelnen Szenarien ist in Abbildung 3.3 zu sehen.

- **Benchmark**: Dieses Szenario ist sehr ähnlich zu dem Trainingsprozess in Abbildung 3.1 und ist dazu gedacht, ein Modell mit zufällig generierten Geheimtexten mit Metriken, wie z.B. Genauigkeit und Loss zu evaluieren. Das Ziel dieses Szenarios ist, dass eine quantitative Aussage über die Qualität eines gegebenen neuronalen Netzes mit unbekannten Testdaten errechnet wird.
- **Evaluate**: Im Evaluation-Szenario werden eine oder mehrere verschlüsselte Textdateien mit bekannten Chiffren ausgewertet. Die Evaluierung funktioniert äquivalent zu der Evaluierung im Benchmark-Szenario mit dem Unterschied, dass Geheimtexte aus Dateien gelesen werden. Daher eignet sich dieses Szenario besser für den Vergleich von trainierten Modellen, da das Ergebnis direkte Aussagekraft zu der Qualität der einzelnen Modelle besitzt. Mit dieser Methode lassen sich wichtige Aspekte der Testung von Modellen, wie in [49] beschrieben, abdecken:
  - **Abdeckung der Testfälle**: Um möglichst genaue Aussagen über die Qualität des neuronalen Netzes treffen zu können, ist es notwendig eine ausreichende Menge an Testdaten für alle verwendeten Chiffren bereitzustellen.
  - **Wiederholbarkeit und Reproduzierbarkeit**: Die Ergebnisse können unabhängig von Tester und Testumgebung reproduziert werden.
  - **Ökonomisch**: Testergebnisse können ohne großen Aufwand produziert werden.
- **Predict**: Im Prädiktionsszenario wird für einen einzelnen Geheimtext oder eine Datei mit mehreren Zeilen an Geheimtexten, bei denen die Verschlüsselungsmethode unbekannt ist, eine Klassifizierung durchgeführt. Diese Methode soll schlussendlich für die Klassifizierung von verschlüsselten Dokumenten mit unbekannter Verschlüsselungsmethode verwendet werden.

Interessanterweise konnte für die Transposition und Simple Substitution eine Genauigkeit von über 99% und für die Playfair-Chiffre über 95% erreicht werden. Die höchste Anzahl der Falschklassifikationen der Hill- und Vigenère-Chiffren liegt bei der jeweils anderen. Hill wird bei diesen Falschklassifikationen meist als Vigenère und Vigenère meist als Hill klassifiziert. Zu dieser Erkenntnis sind 2010 auch Sivagurunathan, Rajendran und Purusothaman gekommen [39]. Diese haben die Chiffren-Erkennung mit einem neuronalen Netz für die Chiffren Playfair, Vigenère und Hill implementiert

Kapitel 4 Empirische Messungen der Qualität von Features

Implementierungsspezifische Details zu einzelnen ACA-Chiffren, den Features und der Datengenerierung werden in diesem Kapitel behandelt. Außerdem werden verschiedene Szenarien für die Schlüsselgenerierung vorgestellt, die mittels Experimenten evaluiert wurden. Diese Arbeit beschäftigt sich nur mit englischen Texten, daher wird eine zusammengesetzte Liste von 270.000 [50] [51] ausgewählten, häufig verwendeten englischen Wörtern für die Schlüssel- und Schlüsselalphabet-Generierung verwendet. Kategorisierung nach Merkmalen von Geheimtexten, sowie nach den im ACA-Handbuch [4, pp. 26-28] beschriebenen Varianten von Schlüsseln, werden in diesem Kapitel durchgeführt.

4.1 Varianten der Schlüsselerzeugung in ACA-Chiffren

Aus praktischen Gründen werden in diesem Abschnitt vor der Beschreibung der ACA-Chiffren kurz die verschiedenen Typen von Schlüsseln nach dem ACA-Handbuch vorgestellt [4, pp. 26-28]. Dafür werden jeweils Klartext- (PT) und Geheimtextalphabete (CT) für jeden Schlüsseltyp aufgelistet. Grundsätzlich kann ein PT als das Ausgangsalphabet und CT als Variationen dieses Alphabets gesehen werden.

K1 Schlüsseltyp: Das Klartextalphabet enthält ein Schlüsselwort. Das Geheimtextalphabet ist normal.

PT: **p o u l t r y** a b c d e f g h i j k m n q s v w x z
CT: R S T U V W X Y Z A B C D E F G H I J K L M N O P Q

K2 Schlüsseltyp: Das Klartextalphabet ist normal. Das Geheimtextalphabet enthält ein Schlüsselwort.

PT: a b c d e f g h i j k l m n o p q r s t u v w x y z
CT: V W X Z **K E Y B O A R D** C F G H I J L M N P Q S T U

K3 Schlüsseltyp: Beide Alphabete verwenden das gleiche Schlüsselwort.

PT: **c o n q u e s t** a b d f g h i j k l m p r v w x y z
CT: H I J K L M P R V W X Y Z **C O N Q U E S T** A B D F G

K4 Schlüsseltyp: Beide Alphabete verwenden unterschiedliche Schlüssel.

PT:s h o p t a l k b c d e f g i j m n q r u v w x y z
CT:V W X Y Z **J U P I T E R** A B C D F G H K L M N O Q S

5x5 Polybius Quadrat

Bei dieser Schlüsselgenerierung wird ein Schlüsselwort gewählt und in eine 5x5 Matrix in eine bestimmte Richtung (horizontal, vertikal, im Uhrzeigersinn, gegen den Uhrzeigersinn oder spiralförmig) gelegt und in horizontaler Richtung ausgelesen. Bei der Verschlüsselung muss ein Buchstabe (meistens J) durch einen anderen (I) ersetzt werden, um das englische Alphabet auf 25 Zeichen abzubilden. Anhand des rechten Beispiels kann ein Polybius Quadrat mit dem Schlüsselwort EXTRAORDINARY und einer vertikalen Route [4, p. 27] betrachtet werden.

	1	2	3	4	5
1	E	O	Y	H	Q
2	X	R	B	K	S
3	T	D	C	L	U
4	R	I	F	M	W
5	A	N	G	P	Z

Mixed Alphabetic Keyword Types

Mit einem Schlüsselwort wird das normale Alphabet mit spaltenweiser Transposition verschoben. Beim Verschlüsseln werden Buchstaben der Reihe nach aus den Spalten oder der Reihenfolge der Buchstaben des Schlüsselwortes nach vertikal gelesen. Beide Varianten können Anhand des folgenden Beispiels betrachtet werden, wobei die erste Zeile das Schlüsselwort darstellt [4, pp. 27-28]:

R O M A N C E	Der Reihenfolge nach:
B D F G H I J	Alphabet: RBKVODLWMFPXAGQYNHSZCITEJU
K L P Q S T U	Alphabetisch nach dem Schlüsselwort geordnet:
V W X Y Z	Alphabet: AGQYCITEJUMFPXNHSZODLWRBKV

Manche Chiffren, wie z.B. die Monome-Dinome-Chiffre [4, p. 52], sehen bei der Schlüsselgenerierung vor, ein Schlüsselwort zu verwenden, um aus der Reihenfolge der Buchstaben im Alphabet eine Zahlenfolge zu generieren. In der Implementierung wurden aus performancetechnischen Gründen stattdessen zufällige Zahlen gewählt und verwendet.

Die verwendeten Schlüssellängen sind standardmäßig 5-8 Zeichen. Ausnahmen dazu sind die Cadenus-Chiffre [4, p. 36], bei der die Länge des Schlüsselwortes von der Länge des Textes abhängt, Checkerboard [4, p. 37], bei der die Länge der Schlüsselwörter durch 5 teilbar sein muss, Nihilist Transposition [4, p. 57], bei der die Länge

des Schlüssels zum Quadrat gleich der Länge des Textes sein muss und Route Transposition [4, p. 75], bei der die Schlüssellänge die Länge des Textes teilen muss.

4.2 Details zu der Implementierung der ACA-Chiffren

Für die vorliegende Arbeit wurde eingeschränkt, dass nur 5x5 Polybius Quadrate für Chiffren, die mithilfe dieses Schlüsseltyps verschlüsseln, verwendet werden. Aus diesem Grund wurden die Klartexte zuerst in Kleinbuchstaben umgewandelt und alle Zeichen, die nicht zum Alphabet gehören, gefiltert. Folgende Chiffren enthalten kein J und ersetzen dieses mit I [4]:

Baconian	CM Bifid	Nihilist Substitution	Ragbaby
Bazeries	Foursquare	Phillips	Seriated Playfair
Bifid	Headlines	Phillips RC	Tri-Square
Checkerboard	Homophonic	Playfair	Two-Square

Die folgenden Chiffren inkludieren bei der Verschlüsselung auch Leerzeichen in den Klartexten und Geheimtexten [4]:

Fractionated Morse	Morbit	Pollux	Tridigital
Key Phrase	Null	Ragbaby	

Eine Ausnahme bildet die Ragbaby-Chiffre [4, p. 72], die aus historischen Gründen, die Buchstaben J und X (wird ersetzt durch W) nicht verwendet. Eine weitere Ausnahme bilden die Digrafid- und Trifid-Chiffren [4, pp. 40, 81]. Die Geheimtexte dieser Chiffren können auch #-Zeichen enthalten. Die Baconian-Chiffre [4, p. 32] enthält neben J auch kein V (wird durch U ersetzt).

Jede Chiffre hat ein eigenes Alphabet, welches bestimmte Zeichen nicht enthalten muss. Beispielsweise beinhaltet das Alphabet der Playfair-Chiffre [4, p. 63] kein J. Aufgrund des Umstandes, dass Geheimtexte aus Indizes des Alphabets einer bestimmten Chiffre bestehen, müssen diese für eine Kryptoanalyse normalisiert werden. Dafür werden alle Indizes, die größer oder gleich dem Index eines fehlenden Zeichens sind, um eins erhöht. Das Zielalphabet sieht folgendermaßen aus:

abcdefghijklmnopqrstuvwxyz #0123456789

Einige Chiffren, wie z.B. Cadenus [4, p. 36], benötigen Klartexte mit bestimmten Längen (in diesem Fall durch 25 teilbar). Aus diesem Grund bestehen die Trainings- und Testdaten, die aus verschiedenen Datensätzen stammen, aus Klartexten mit fixen Längen, jedoch können diese in der Testphase unterschiedlich zu der Trainingsphase sein.

Von den insgesamt 60 gelisteten ACA-Chiffren werden in dieser Arbeit 55 plus die Klasse Klartext als eigene Chiffre verwendet. Zu den nicht verwendeten Chiffren gehören Incomplete Columnar Transposition [4, p. 49], Interrupted Key [4, p. 50], Syllabary [4, p. 83], Twin Bifid und Twin Trifid [4, p. 83]. In Tabelle 2 werden alle verwendeten ACA-Chiffren gelistet.

Amsco	Digrafid	Morbit	Pollux	Route Transp.
Autokey	Foursquare	Myszkowski	Porta	Running Key
Baconian	Fractionated Morse	Nicodemus	Portax	Seriated Playfair
Bazeries	Grandprè	Nihilist Subst.	Progr. Key	Slidefair
Beaufort	Grille	Nihilist Transp.	Quagmire1	Swagman
Bifid	Gromark	Null	Quagmire2	Tridigital
Cadenus	Gronsfeld	Numbered Key	Quagmire3	Trifid
Checkerboard	Headlines	Period. Gromark	Quagmire4	Tri-Square
Columnar Transp.	Homophonic	Phillips	Ragbaby	Two-Square
Condi	Key Phrase	Phillips-RC	Railfence	Variant
CM Bifid	Monome-Dinome	Playfair	Redefence	Vigenère

Tabelle 2: In der Arbeit betrachtete ACA-Chiffren

Die Incomplete Columnar Transposition [4, p. 49] unterscheidet sich zu der Complete Columnar Transposition [4, p. 38] nur in dem Punkt, dass keine Füllzeichen (x) verwendet werden, um die Spalten aufzufüllen. Aus diesem Grund lassen sich die zwei Varianten nicht unterscheiden und daher wird nur die Complete Columnar Transposition verwendet. Interrupted Key [4, p. 50] ist keine eigenständige Chiffre, sondern eine beliebige Chiffre, bei der der Schlüssel nach jedem Wort im Klartext neu beginnt. Auch hier lassen sich die Chiffren nicht unterscheiden.

Aufgrund des Kontextes der vorliegenden Arbeit, und zwar die Erkennung von historischen klassischen Chiffren, wurde die Syllabary-Chiffre [4, p. 83] nicht implementiert. Diese Chiffre wurde von R. J. Friedman [52] 2012 entwickelt und hat daher im Kontext der vorliegenden Arbeit keine Relevanz.

Twin Bifid- und Twin Trifid-Chiffren [4, p. 83] benötigen zwei Klartexte mit einer gleichen Phrase, um einen Geheimtext zu berechnen. Aus diesem Grund lassen sich diese Verfahren grundsätzlich nicht mit den anderen Chiffren vergleichen und wurden daher nicht implementiert.

4.3 Auswahl des Optimizers

Die Auswahl des Optimizers wurde mittels empirischer Testläufe mit den Standardparametern durchgeführt. Für den Vergleich wurde das in Punkt 4.6.1 beschriebene Baseline-Modell mit exakt 100 Zeichen langen Texten und das zweite in Abschnitt 4.4

beschriebene Trainingsszenario verwendet. Getestet wurden Stochastic Gradient Descent (SGD) mit Momentum=0,9, RMSprop, Adam, Adadelta, Adagrad, Adamax und Nadam. Um herauszufinden, welcher Optimizer am besten für die vorliegende Problemstellung geeignet ist, wurde jeweils ein Modell mit einem anderen Optimizer mit 100 Millionen Datensätzen trainiert, also 1,8 Millionen pro Chiffre. Für jede Chiffre wurden jeweils, wenn anwendbar, die Schlüssellängen 5-8 Zeichen verwendet.

Optimizer	**Genauigkeit in %**	**Top-K-kategorische-Genauigkeit in %**	**Trainingszeit**	**Konvergenz nach 100 Millionen Iter.?**
SGD with Momentum	64,78	81,50	4h 21m	Ja
RMSprop	69,96	84,60	4h 27m	Ja
Adam	72,97	87,18	4h 21m	Nein
Adadelta	48,04	68,82	4h 23m	Nein
Adagrad	56,31	74,74	4h 23m	Nein
Adamax	73,71	87,80	4h 25m	Nein
Nadam	72,42	86,91	4h 33m	Ja

Tabelle 3: Ergebnisse des Optimizer-Vergleichs

Aus der Tabelle 3 lässt sich evaluieren, dass Adam und Adamax mit den Standard-Parametern die besten Ergebnisse in Hinsicht auf Genauigkeit und Trainingszeit liefern. Der Adamax Algorithmus ist eine Variante von Adam, die jedoch genau dieselben Hyperparameter verwendet [18]. Aufgrund der höheren Verbreitung des Adam Optimizers und der damit besseren Vergleichbarkeit zu anderen Arbeiten im Bereich der Cipher Type Detection, werden weitere Tests und die Suche der besten Hyperparameter mit dem Adam Algorithmus durchgeführt.

4.4 Trainingsszenarien

Alle ACA-Chiffren, deren Schlüssel nicht aus Ziffern bestehen, wählen die Schlüsselwörter und -alphabete historisch bedingt nicht zufällig, sondern sie verwenden englische Wörter. Schlüsselalphabete verwenden ein Schlüsselwort und füllen das restliche

Alphabet in lexikalischer Reihenfolge auf. Dadurch lassen sich folgende drei Trainingsszenarien, der Klassifizierungsschwierigkeit nach sortiert, definieren:

1. Schlüsselwörter werden aus einem Wörterbuch gewählt. Schlüsselalphabete verwenden Schlüsselwörter aus einem Wörterbuch und das restliche Alphabet wird in richtiger Reihenfolge angeordnet. (SZEN1)
2. Schlüsselwörter werden zufällig gewählt. Schlüsselalphabete verwenden zufällige Schlüsselwörter und füllen das restliche Alphabet in richtiger Reihenfolge auf. (SZEN2)
3. Schlüsselwörter werden zufällig gewählt. Schlüsselalphabete haben eine zufällige Anordnung. (SZEN3)

Aufgrund der Häufigkeitsverteilung des englischen Alphabets, die über eine große Anzahl von Wörtern berechnet wurde und zum Domain-Knowledge gehört, ist es wahrscheinlich, dass selten verwendete Buchstaben, wie z.B. Z, am Ende des Alphabets zu finden sind. Die Verteilungen der daraus entstehenden Geheimtexte sind dadurch in der Kryptoanalyse einfacher zu klassifizieren als zufällige Schlüsselalphabete.

4.5 Verwendete Features

Dieser Abschnitt behandelt die Auswahl der verwendeten Features für Feature-Engineering-Algorithmen. Grundsätzlich lassen sich die Features in folgende Gruppen unterteilen:

- Häufigkeitsstatistiken (z.B. Unigramme, Bigramme)
- Verteilungsstatistiken (z.B. Index of Coincidence)
- Binäre Features (z.B. HAS_J, HAS_X)
- Chiffren-spezifische Features (z.B. A_LDI)

Der Wertebereich der Features, die in Tabelle 4 detailliert beschrieben sind, wird auf [0..1] normiert, damit kleine Veränderungen bei einem Feature mit einem höheren numerischen Wertebereich nicht überproportionale Auswirkungen auf die Entscheidung und auf die Bewertung anderer Features im Lernprozess haben. Spalte 1-3 bestehen aus den Bezeichnungen der einzelnen Features, sowie einer Beschreibung der Funktionsweise der Features. Die vierte und fünfte Spalte beschreibt die Anzahl der Neuronen und deren Wertebereiche für jedes Feature. Die durchschnittliche Laufzeit der Feature-Berechnung wurde mit dem Durchschnitt aus 1.000 Durchläufen in der sechsten Spalte gemessen.

Abkürzung	**Bezeichnung**	**Beschreibung**	**# Neuronen**	**Wertebereich**	**Ø Zeit in ms**
SDD	Average Single Letter – Digraph Discrepancy Score [11]	Dieses Feature verwendet eine Tabelle der Differenzen zwischen Unigrammen und Bigrammen. Der Score wird berechnet, indem jeder Wert an der Stelle des ersten Buchstaben im Alphabet mal 26 plus der Position des zweiten Buchstaben im Alphabet aus der SDD-Tabelle addiert wird. Der Score wird dann durch die Länge des Textes minus 1 dividiert. Für die Normalisierung der Scores wird durch 10 dividiert.	1	0-1	<1
CHI^2	Chi Square	Mit der Chi^2-Funktion kann die Abweichung gegenüber der Verteilung von englischen Buchstaben, die bekannt ist, berechnet werden. Dieser Wert wird durch 100 dividiert.	1	0-1	<1
DIC	Digraphic Index of Coincidence [11]	Der DIC ist die Summe aller Wahrscheinlichkeiten des Auftretens von zwei gleichen Zeichenpaaren in einem Text.	1	0-1	1
DBL	Double Letter [11]	DBL ist ein binärer Wert der 1 ist, wenn ein doppeltes Zeichen an gerader Stelle und die Gesamtlänge gerade ist.	1	0,1	<1
AUTO	Estimated Auto Correlation	Die Autokorrelation ist nützlich bei der Erkennung von sich wiederholenden Mustern in einer Sequenz. Aufgrund der unterschiedlichen Längen der Geheimtexte (die Null-Chiffre [4, p. 58] hat maximal 10x so lange Geheimtexte wie Klartexte)	10x Textlänge	-1-1	<1

		müssen die restlichen Datenpunkte mit 0 gefüllt werden.			
FREQ	Frequencies	FREQ ist die rekursive Berechnung der Auftrittswahrscheinlichkeiten bis inklusive Bigrammen.	1.444	0-1	3
HAS_0	Has Digit 0 [11]	HAS_0 ist ein binärer Wert der 1 ist, wenn die Ziffer 0 auftritt.	1	0-1	<1
HAS_H	Has Hash [11]	HAS_0 ist ein binärer Wert der 1 ist, wenn das Zeichen # auftritt.	1	0,1	<1
HAS_J	Has Letter J [11]	HAS_J ist ein binärer Wert der 1 ist, wenn der Buchstabe J auftritt.	1	0,1	<1
HAS_X	Has Letter X [11]	HAS_X ist ein binärer Wert der 1 ist, wenn der Buchstabe X auftritt.	1	0,1	<1
HAS_SP	Has Space	HAS_SP ist ein binärer Wert der 1 ist, wenn ein Leerzeichen auftritt.	1	0,1	<1
IoC	Index of Coincidence [11]	Der IoC ist die Summe aller Wahrscheinlichkeiten des Auftretens von zwei gleichen Zeichen in einem Text.	1	0-1	<1
LDI	Log Digraph Score [11]	Bigramme in einem Text werden in einer Liste von vorberechneten Häufigkeiten der englischen Sprache gesucht und summiert. Der Durchschnitt dieser Summe ist der Score. Bei Bion [11] werden stattdessen die echten Anzahlen verwendet, jedoch sind das zu große Werte, weshalb die Auftrittswahrscheinlichkeit dividiert durch 10 besser geeignet ist.	1	0-1	<1
A_LDI, B_LDI, P_LDI, S_LDI, V_LDI, PTX	Log Digraph Score for Autokey, Beaufort,	Diese Menge an Vigenère-Statistiken berechnet den LDI für verschiedene Chiffren, indem die Geheimtexte mit den jeweiligen Verschiebungsfunktionen umgerechnet werden.	6	0-1	96 PTX 1

	Porta, Slidefair, Vigenere and Portax [11]	Der Score wird durch 1.000 dividiert. Für Geheimtexte, die andere Zeichen als Buchstaben enthalten ist das PTX Feature gleich 0.			
LR	Long Repeat [11]	LR ist der Prozentsatz der sich genau drei Mal wiederholenden Zeichen. Dafür werden für jedes Zeichen ab der Position + 1 alle gleichen Zeichen gezählt. Die Wurzel dieses Ergebnisses wird durch die Länge des Textes geteilt.	1	0-1	<1
BDI	Max Bifid DIC for periods 3-15 [11]	Texte werden, wie in der Bifid-Chiffre [4, p. 35], in Perioden von 3-15 gelesen und der DIC daraus berechnet. Der höchste Score wird durch 1.000 dividiert und zurückgegeben. Für Geheimtexte, die andere Zeichen als Buchstaben enthalten ist dieses Feature gleich 0.	1	0-1	2
CDD	Max Columnar SDD Score for periods 4-15 [11]	Texte werden, wie in der Columnar Transposition-Chiffre [4, p. 38], in Perioden gelesen und der SDD Score dafür berechnet. Das Ergebnis dieses Features ist der maximale SDD Score dividiert durch 1.000. Für Geheimtexte, die andere Zeichen als Buchstaben enthalten ist dieses Feature gleich 0.	1	0-1	80
MKA	Max Kappa [11]	Texte werden für die Perioden 1-15 um p Stellen nach rechts verschoben. Die restlichen p Zeichen werden mit Werten aufgefüllt, die nicht im Text enthalten sind (z.B. -1). Das Ergebnis dieser Statistik ist der maximale Prozentsatz der Übereinstimmung zwischen dem verschobenen Text und dem Originaltext.	1	0-1	8

NIC	Max Nicodemus IC [11]	Texte werden in die Perioden 3-15 unterteilt. Der höchste NIC wird berechnet indem der Text wie bei der Nicodemus-Chiffre [4, p. 55] unterteilt und gelesen wird. Der höchste Wert wird zurückgegeben.	1	0-1	2
SSTD	Max STD Score for Swagman periods 4-8 [11]	Texte werden, wie in der Swagman-Chiffre [4, p. 79], in Perioden gelesen und der STD Score wird berechnet. Das Ergebnis dieses Features ist der maximale STD Score dividiert durch 100.	1	0-1	1
MIC	Maximum Index of Coincidence [11]	Texte werden in die Perioden 1-15 unterteilt. Der höchste IoC aller Teilgruppen wird berechnet, indem der Text in p Gruppen unterteilt wird. Jede Gruppe besteht aus allen Zeichen mit dem Abstand p. Bei p = 3 gibt es 3 Gruppen, wobei die erste Gruppe jedes dritte Zeichen, beginnend bei 0 beinhaltet; die zweite Gruppe jedes dritte Zeichen, beginnend bei 1 und die dritte Gruppe jedes dritte Zeichen beginnend bei 2. Der höchste Wert wird zurückgegeben.	1	0-1	28
NO-MOR	Normal Order [11]	Die Frequenz jedes Zeichens wird berechnet und der Größe nach sortiert. Die normale Ordnung ist die Summe der Abstände aller Zeichen zu ihrer Normalposition dividiert durch 1.000.	1	0-1	<1
PHIC	Phillips IC [11]	PHIC ist der IoC mit einer fixen Spaltengröße von 5 und einer fixen Periode von 8. Das Ergebnis wird mit 10 multipliziert. Für Geheimtexte, die andere Zeichen als Buchstaben enthalten ist dieses Feature gleich 0.	1	0-1	<1

REP	Repetition Feature [5]	Dieses Feature besteht aus der normalisierten Anzahl von exakt n-mal auftretenden gleichen Zeichen für $2 \leq n \leq 5$. Die Normalisierung wird durch Division der Gesamtanzahl von Wiederholungen berechnet.	4	0-1	<1
ROD	Repetition Odd [11]	ROD ist der Prozentsatz der sich wiederholenden Zeichen mit ungeradem Abstand zu der Summe sich wiederholender Zeichen. Dafür werden für jedes Zeichen ab der Position + 1 alle gleichen Zeichen gezählt. Das Ergebnis ist die Summe der sich wiederholenden Zeichen mit ungeradem Abstand dividiert durch die Summe der sich wiederholenden Zeichen.	1	0-1	<1
RDI	Reverse Log Digraph [11]	Bigramme in einem Text werden in einer Liste von vorberechneten Häufigkeiten der englischen Sprache gesucht und summiert, wobei jedoch die Reihenfolge der Buchstaben vertauscht wird, also z.B AB -> BA. Der Durchschnitt dieser Summe ist der Score. Bei Bion [11] werden stattdessen die echten Anzahlen verwendet, jedoch sind das zu große Werte, weshalb die Auftrittswahrscheinlichkeit dividiert durch 10 besser geeignet ist.	1	0-1	<1
SHAN	Shannon's Entropy Equation	Die Entropie ist ein Maß zur Bestimmung des Informationsgehalts eines Textes. Grundsätzlich zeigt eine höhere Entropie auf, dass Daten verschlüsselt sind. Dieser Wert wird durch 10 dividiert.	1	0-1	<1

Tabelle 4: Definitionen der Features

4.6 Architekturen

Die in diesem Abschnitt vorgestellten Architekturen wurden im Zuge der empirischen Tests implementiert und bis zur Konvergenz trainiert. Ein Modell ist konvergent, wenn sich der Loss über mehrere Epochen nicht mehr oder nur geringfügig ändert. Das Stoppen des Trainings passiert mittels eines Mechanismus, der nach jedem Mini-Batch prüft ob es eine Mindestverbesserung der Genauigkeit um 10^{-5} über die letzten 250 Mini-Batches gibt. Die implementierten Architekturen verwenden Standard-Hyperparameter, da eine umfassende Suche der besten Parameter für alle Versuche praktisch nicht durchführbar ist. Es wird angenommen, dass Abweichungen zu den besten Ergebnissen in allen Modellen, aufgrund der nicht-optimalen Hyperparameter, ähnlich hoch sind und daher ein Vergleich möglich ist. Die gleiche Argumentation wird für die Textlänge, die bei allen Traingsdurchläufen exakt 100 beträgt, durchgeführt. Hier wird angenommen, dass ein Modell, welches mit Texten der Länge 100 besser umgehen kann auch ähnlich gut mit zufälligen Textlängen arbeiten kann. Mit „schnellen Features" sind in diesem Kapitel alle Features gemeint, die eine maximale Berechnungsdauer von 3ms pro Aufruf benötigen. Diese Einschränkung wird durchgeführt, da ansonsten durch die kumulative Trainingsdauer die Arbeit nicht durchführbar wäre. Die wichtigsten Aspekte der Architektur sind:

- **Konvergenzgeschwindigkeit:** Wie viele Epochen muss ein Modell zur Konvergenz trainieren? (Anzahl der Trainingsdurchläufe/Trainingsdaten)
- **Genauigkeit:** Wie genau kann ein Modell arbeiten? (Genauigkeit)
- **Rechenkomplexität:** Wie viel Rechenzeit benötigt ein Modell, um bis zur Konvergenz trainiert zu werden? (Trainingszeit)

4.6.1 Baseline-Modell

Als Baseline-Modell wird das im CANN-Projekt konstruierte FFNN mit den Features CHI^2, FREQ, HAS_H, HAS_J, HAS_SP, HAS_X, IoC, DIC, SHAN und 5 Dense-Hidden-Layern mit der ReLU-Aktivierungsfunktion verwendet. Die Input-Länge beträgt 1.490 Neuronen und die Größe aller Hidden Layer beträgt:

$$\frac{2 \cdot input\_size}{3} + output\_size = 1.049.$$

Mit einem Dense-Layer mit der Softmax-Aktivierungsfunktion wird die Vorhersage des neuronalen Netzwerks bestimmt. Das Training fand in Mini-Batches mit der Größe von 64.960 Datensätzen, einer Batch-Größe von 512 Datensätzen und dem Adam-Optimizer statt. In Tabelle 5 wurde das Training für alle drei Trainingsszenarien bis zur oben definierten Konvergenz durchgeführt. Die Spalte „ID" ist eine fortlaufende Nummer mit der die Trainingsergebnisse identifiziert werden können.

Szenario	Genauigkeit in %	Top-K-kategorische-Genauigkeit in %	Trainingszeit	Konvergenz nach Iter. in Millionen	ID
SZEN1	78,11	91,64	6h 13m	171	T1
SZEN2	73,29	87,65	6h 57m	176	T2
SZEN3	69,22	84,91	7h 37m	196	T3

Tabelle 5: Ergebnisse des Baseline-Modells mit den Features aus dem CANN-Projekt

4.6.2 FFNN mit schnellen Features und verschiedenen Aktivierungsfunktionen

Die erste anzustrebende Architektur ist ein FFNN, wie das im Baseline-Modell, mit schnellen Features. Es soll außerdem getestet werden, ob abnehmende Größe bei den Hidden Layern zu besseren Ergebnissen führt. Dabei wird die im Baseline-Modell verwendete Formel auf jeden einzelnen Hidden Layer angewandt, jedoch wird die Ausgangsgröße eines Layers zur Eingangsgröße des folgenden Layers. Das Wissen aus diesen Trainingsdurchläufen wird auf die folgenden Architekturen angewandt.

Szenario	Genauigkeit in %	Top-K-kategorische-Genauigkeit in %	Trainingszeit	Konvergenz nach Iter. in Millionen	ID
SZEN1	77,15	90,67	1d 4h 57m	166	T4
SZEN2	73,07	86,65	1d 4h 58m	168	T5
SZEN3	71,23	86,42	1d 5h 39m	171	T6

Tabelle 6: Ergebnisse des Baseline-Modells mit schnellen Features und gleichbleibender Größe der Hidden Layer

Die in Tabelle 6 gelisteten Ergebnisse zeigen, dass nicht alle Features Verbesserung herbeiführen, da die Modelle insgesamt 7-10 Prozent schlechtere Ergebnisse gegenüber dem Baseline-Modell liefern. Mögliche Begründungen für die schlechten Ergebnisse können im AUTO-Feature, der höheren Anzahl von Hidden Layern und der höheren Komplexität des Modells liegen.

4.6.3 Feature-Auswahlstrategie

Eine effiziente Methode zur Auswahl der Features ist die Berechnung einer Korrelationsmatrix für alle Features [53]. Dabei wird zwischen jedem Feature-Paar die Pearson-Korrelation berechnet. Die Pearson-Korrelation zwischen zwei Mengen kann folgendermaßen definiert werden, wobei σ_X und σ_Y die Standardabweichungen für X und Y sind [53]:

$$cor(X,Y) = \frac{\sum_{i=1}^{k}(x_i - \bar{x})(y_i - \bar{y})}{\sigma_X \sigma_Y}.$$

Aufgrund der Mehrdimensionalität der Features AUTO und FREQ wurden diese in der Pearson-Korrelation nicht ausgewertet. In Abbildung 4.1 ist die Korrelationsmatrix aller Features zu sehen. Hohe positive Korrelationswerte (hell in der Abbildung) signalisieren einen bivariaten positiven Zusammenhang. Das bedeutet, dass eine hohe Ausprägung eines Features zu einer hohen Ausprägung des anderen führt. Hohe negative Korrelationswerte (dunkel in der Abbildung) signalisieren einen bivariaten negativen Zusammenhang. Das bedeutet, dass eine hohe Ausprägung eines Features zu einer niedrigen Ausprägung des anderen führt. Niedrige Absolutwerte (rötlich in der Abbildung) deuten auf wenig Korrelation zwischen zwei Features hin.

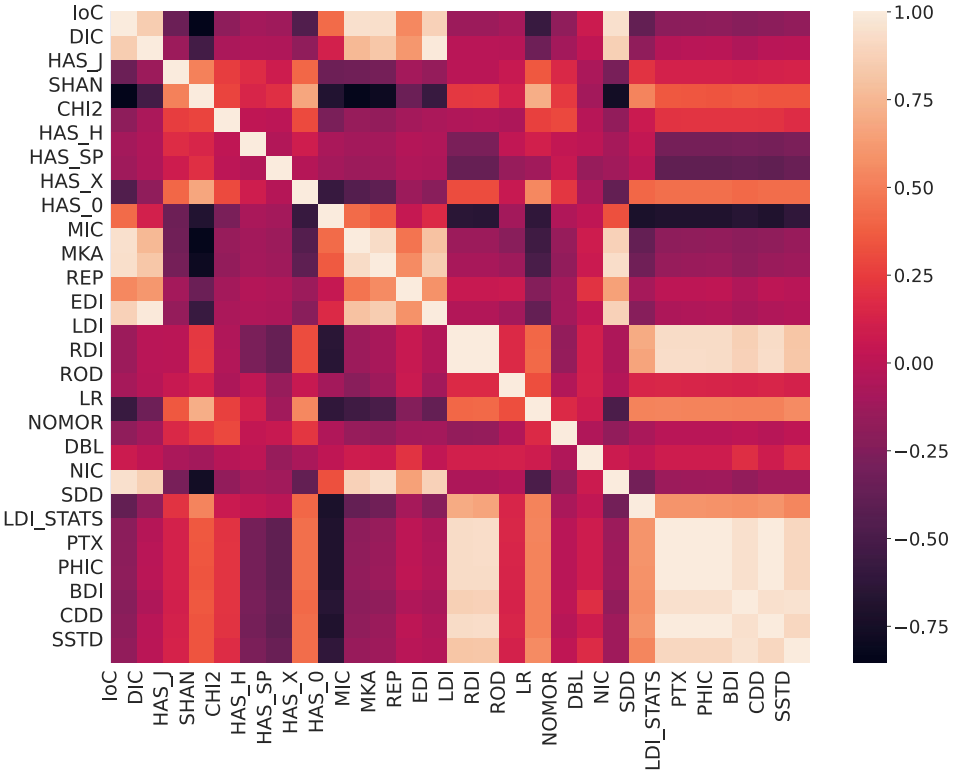


Abbildung 4.1: Correlation Heatmap

Für die Auswahl der Features wurde ein Schwellwert von 0,9 gewählt, da ansonsten entweder zu viele oder keine Features entfernt wurden. Die Korrelationsmatrix kann nur auf Features angewendet werden, die einstellig sind, weshalb sie nur auf 27 der 32 Features anwendbar ist. Nachdem jeweils ein Feature aus den Korrelationen mit einem Absolutwert von 0,9 oder mehr aussortiert wurde, sind aus den 27 Features 16 übriggeblieben. Diese sind IoC, DIC, HAS_J, SHAN, CHI^2, HAS_H, HAS_SP, HAS_X, HAS_0, REP, LDI, ROD, LR, NOMOR, DBL, SDD. Dieses Setup mit dem FREQ-Feature führt zu den Ergebnissen in Tabelle 7.

Szenario	Genauigkeit in %	Top-K-kategorische-Genauigkeit in %	Trainingszeit	Konvergenz nach Iter. in Millionen	ID
SZEN1	77,93	91,50	14h 13m	284	T7
SZEN2	73,19	87,16	9h 48m	193	T8
SZEN3	69,66	85,40	9h 48m	192	T9

Tabelle 7: Ergebnisse mit Features mit absolutem Korrelationswert kleiner 0,9

Wie Tabelle 7 zeigt, sind die Ergebnisse der Feature-Auswahl mit der Korrelationsmethode schlechter als die des Baseline-Modells, jedoch gleich gut wie die des Modells mit schnellen Features. Aus diesem Grund wurde eine alternative Feature-Auswahlstrategie definiert, die ausgehend vom Baseline-Modell iterativ jedes Feature bzw. jede Feature-Gruppe testet und bei marginal schlechterer (maximal 0,5 % schlechtere Genauigkeit) oder besserer Genauigkeit das Feature bzw. die Feature-Gruppe in den Pool der ausgewählten Features hinzunimmt. Tabelle 8 zeigt die evolutionäre Auswahl der Features nach jedem Durchlauf. In Tabelle 9 sind Details, wie Genauigkeit, Trainingszeit und Anzahl der Iterationen, aufgezeigt. Um auch die Auswirkungen im Konvergenzbereich beobachten zu können, wurde für jedes Feature das zweite Trainingsszenario bis zur Konvergenz trainiert. Aufgrund der nicht deterministischen Auswahl der Daten, durch die zufällige Generierung von Schlüsselmaterial und Klartexten, kann es dazu kommen, dass Trainings früher abbrechen als andere, obwohl die Genauigkeit noch verbessert werden kann. Daher wurden die Trainings immer bei der gleichen Anzahl von Trainingsdurchläufen mit dem vorherigen Trainingsdurchlauf verglichen.

ID	T 10	T 11	T 12	T 13	T 14	T 15	T 16	T 17	T 18	T 19	T 20	T 21	T 22	T 23	T 24	T 25	T 26	T 27	T 28	T 29	T 30
IoC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DIC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
FREQ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HAS_0	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HAS_H	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HAS_J	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HAS_X	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HAS_SP	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CHI²	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
DBL	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
LR	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ROD	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SDD	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
REP	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
LDI	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NOMOR	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PHIC	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RDI	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
SHAN	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
AUTO	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
BDI	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
PTX	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓
SSTD	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
NIC	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓
MKA	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
MIC	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
CDD	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
LDI STATS	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

Tabelle 8: Feature-Testablauf

ID	Genauigkeit in %	Top-K-kategorische-Genauigkeit in %	Trainingszeit	Konvergenz nach Iter. in Millionen
T10	74,49	88,28	8h 13m	235
T11	73,50	87,84	6h 31m	170
T12	73,86	87,79	8h 47m	230
T13	73,61	87,68	8h 27m	238
T14	73,77	87,73	8h 35m	172
T15	73,99	87,80	10h 28m	216
T16	72,84	87,14	9h 25m	190
T17	74,26	88	11h 21m	227
T18	74,29	88,02	9h 58m	200
T19	74,36	88	14h 22m	285
T20	73,85	87,81	8h 11m	159
T21	73,77	87,62	7h 49m	150
T22	72,02	83,43	7h 40m	112
T23	74,39	87,98	13h 35m	214
T24	74,67	88,19	15h 24m	228
T25	74,29	87,92	19h 31m	137
T26	74,38	88,32	14h 44m	174
T27	74,86	88,38	21h 22m	229
T28	74,70	88,94	16h 50m	146
T29	75,19	89,09	12d 14h 40m	221
T30	77,60	92,77	7d 10h 11m	212

Tabelle 9: Feature-Testergebnisse

Aufgrund der guten Ergebnisse der LDI-Features (T30), jedoch der zu langen Rechenzeit, wurde dieses Feature im finalen Modell verwendet, jedoch nicht für weitere Experimente. Die Features aus T28 wurden für alle weiteren Trainings verwendet. In Tabelle 10 sind die Ergebnisse mit abnehmender Größe der 5 Hidden Layer dargestellt. Die Größe der Hidden Layer wird der folgenden Formel für jeden Layer berechnet, wobei *i* die Position des zu berechnenden Layers ist:

$$size_{i+1} = \frac{2 \cdot size_i}{3} + output\_size.$$

Da die Veränderung der Größe der Hidden Layer zu keiner Verbesserung führt, wird eine gleichbleibende Größe für weitere Trainingsdurchläufe verwendet.

Szenario	Genauigkeit in %	Top-K-kategorische-Genauigkeit in %	Trainingszeit	Konvergenz nach Iter. in Millionen	ID
SZEN1	78,08	91,98	15h 8m	134	T31
SZEN2	74,21	88,62	15h 55m	141	T32
SZEN3	71,57	87,54	21h 21m	189	T33

Tabelle 10: Ergebnisse des Baseline-Modells mit ausgewählten Features und abnehmender Größe der Hidden Layer

Um möglichst genau bestimmen zu können wie sich die Trainingsergebnisse mit unterschiedlichen Aktivierungsfunktionen, die in Abschnitt 2.4 definiert wurden, und 5 Hidden Layern entwickeln, wurde jeweils ein Modell mit dem zweiten Trainingsszenario und den Features aus T28 bis zur Konvergenz trainiert. Die Exponentialfunktion bildet eine Ausnahme, bei welcher nur ein Hidden Layer verwendet wurde, da ansonsten der Loss nicht berechnet werden konnte.

In Tabelle 11 sind die Trainingsergebnisse der beschriebenen Aktivierungsfunktionen angeführt. Die ReLU-Funktion lieferte, mit Ausnahme der Parametric ReLU-Funktion, die besten Ergebnisse in Hinsicht auf Genauigkeit und Trainingszeit. Aufgrund der verlässlicheren Ergebnisse wegen der niedrigeren Komplexität der ReLU-Funktion wurde diese in weiteren Tests der Parametric ReLU-Funktion vorgezogen. Auch die Exponentialfunktion lieferte mit nur einem Hidden Layer gute Ergebnisse und wurde daher in der Bestimmung der besten Hyperparameter-Konfiguration für FFNNs mitberücksichtigt.

Funktion	Genauigkeit in %	Top-K-kategorische-Genauigkeit in %	Trainingszeit	Konvergenz nach Iter. in Millionen	ID
ReLU	74,70	88,94	16h 50m	146	T28
Leaky ReLU	72,64	87,45	12h 30m	99	T34
Parametric ReLU	75,18	89,02	19h 29m	152	T35
Sigmoid	72,32	87,01	1d 22h	385	T36
tanh	65,48	81,87	9h 33m	68	T37
ELU	68,51	84,18	10h 24m	76	T38
SELU	67,54	83,58	13h 48m	103	T39
Exponential	70,10	85,62	17h 54m	140	T40
Swish	70,32	86,07	20h 33m	150	T41
RBF	1,59	4,92	3h 30m	16	T42

Tabelle 11: Performance einzelner Aktivierungsfunktionen

4.6.4 Convolutional Neural Networks

Convolutional Neural Networks (CNN) sind grundsätzlich Feature-Learning-Algorithmen, können jedoch auch auf Features angewendet werden. Hyperparameter eines 1-Dimensionalen CNNs sind die Kernel-Größe, die Anzahl der Filter, die Verschiebung in der Convolution (weiter stride genannt), die Anzahl der Convolution Layer und den Dropout. Die Kernel-Größe bestimmt wie viele Elemente in der Convolution-Operation betrachtet werden und ist aufgrund der möglichen Schlüssellängen auf 9 gesetzt. Tabelle 12 zeigt die Parameterauswahl der Rastersuche und die besten Ergebnisse, die aus der Suche herausgehen. In der iterative durchgeführten Rastersuche wurde getestet, wie sich eine mit jedem Layer steigende stride auswirkt. Ein weiteres Training sollte die Wirksamkeit eines Output-Dropout-Layers zur Regularisierung mit 20% testen. Die beste Konfiguration wurde danach mit den Features aus dem Feature Engineering trainiert und mit den Ergebnissen des Feature-Learning-Modells verglichen.

Hyperparameter	Raster	Standard	Bestes Ergebnis	Genauigkeit in %
Filters	[8, 16, 32, 64, 128, 256]	-	64	39,33
Kernels	[2, 3, 5, 7, 9, 11, 13]	9	7	44,24
# Layers	[3, 5, 7, 10]	5	3	47,54
Stride	[1, 1 iterierend]	1	1	47,54
Dropout	[0; 0,2]	0	0	47,54
Features	[False, True]	False	False	47,54

Tabelle 12: Iterative Rastersuche für CNN

Die Testergebnisse der Rastersuche für ein CNN sind gegenüber dem FFNN schlecht ausgefallen. Mit den besten Konfigurationen konnten nur 47,54% Genauigkeit erreicht werden. Mit den Features aus Tabelle 8 konnte nur eine Genauigkeit von etwa 17% erzielen. Die Experimente in diesem Bereich wurden daher nicht mehr fortgesetzt.

4.6.5 Transformer

Transformer, oder auch Attention Networks genannt, sind derzeit Stand der Technik in der Textverarbeitung und -klassifizierung [36]. Keras bietet bereits eine Implementierung eines Transformers an [54]. Nach der Anpassung der problemabhängigen Parameter wurde in Tabelle 13 aufgrund der komplexen Architektur nur die Rastersuche für alle Parameter durchgeführt.

Hyperparameter	Raster	Standard	Bestes Ergebnis	Genauigkeit in %
pooling	[Average, Max]	-	Average	58,73
ff_hidden_-layer_size	[1, 5]	1	1	58,73
vocab_size	[20.000, 50.000]	20.000	20.000	58,73
ff_dim	[32, 64, 128, 256, 512, 1024]	32	1024	61,96
embed_dim	[32, 64, 128, 256]	32	128	67,96
num_heads	[2, 4, 8, 16?]	2	8	73,71
Features	[False, True]	False	False	73,71

Tabelle 13: Iterative Rastersuche für Transformer

4.6.6 Recurrent Neural Networks – LSTM

Genauso wie CNNs sind LSTMs Feature-Learning-Algorithmen, können jedoch auch auf Features angewandt werden. Das bedeutet, dass die Suche nach den besten Hyperparametern mit einer Rastersuche möglich ist. LSTMs haben im Vergleich zu CNNs nicht so viele Hyperparameter. Der wichtigste Parameter ist die Anzahl der Units. Diese bilden die Assoziationen in einem LSTM. In einem weiteren Test wird geprüft ob ein Input- und Output-Dropout zu besseren Ergebnissen führt.

Nachdem die besten Ergebnisse gefunden wurden, wurde ein Modell mit den gleichen Hyperparametern und den Features der Geheimtexte berechnet. Tabelle 14 zeigt die Ergebnisse der Rastersuche für LSTMs.

Hyperparameter	Raster	Standard	Bestes Ergebnis	Genauigkeit in %
Units	[50, 100, 150, 200, 500]	-	500	72,84 (68,50)
Dropout	[0; 0,2]	0	0	68,50
Features	[False, True]	False	False	68,50

Tabelle 14: Iterative Rastersuche für LSTM

Die Rastersuche hat ergeben, dass eine höhere Anzahl der Units zu besseren Ergebnissen führt. Das sind in etwa 2% pro 50 Units. Aufgrund der hohen Rechenzeit wurden jedoch Dropout und Features mit nur 200 Units getestet und verglichen. Es konnte gezeigt werden, dass die Architektur mit höherer Unit-Anzahl zu besseren Ergebnissen führt, jedoch steigt die Rechenzeit direkt proportional dazu.

4.6.7 Entscheidungsbäume

EB sind Feature-Engineering-Algorithmen [32]. Wie bereits in Punkt 2.6.2 beschrieben wurde, besteht der Aufbau eines EB aus zwei Phasen, der Trainingsphase und der Pruning-Phase. Die zweite Phase soll verhindern, dass der EB zu viele unnötige Knoten beinhaltet und so das Ergebnis verschlechtert. Aufgrund des Designs des EB-Algorithmus, welcher die vollständigen Daten im Speicher benötigt, und der Implementierung in der Sklearn-Bibliothek [32] können nur vollständige Datensätze trainiert werden. Außerdem ist der EB anfällig auf Overfitting [32], weshalb in diesem Setup nur eine beschränkte Menge von 3,25 Millionen Datensätzen verwendet wird. Der EB hat zwei wichtige Parameter, die getestet wurden. Der erste Parameter ist die Auswahl des Entscheidungsalgorithmus. Die Auswahl besteht zwischen dem Gini-Index, der auch im CART-Algorithmus verwendet wird, oder Entropie, die in den ID3-, C4.5- und C5.0-Algorithmen angewandt wird. Der EB in der Sklearn Bibliothek [32] verwendet den CART-Algorithmus. Der zweite Parameter ist das Cost-Complexity-Alpha, welches im Minimal Cost-Complexity Pruning verwendet wird. Tabelle 15 zeigt die Ergebnisse der Rastersuche für den EB.

Hyperparameter	Raster	Standard	Bestes Ergebnis	Genauigkeit in %
criterion	[„gini“, „entropy“]	-	„entropy“	61,68
ccp_alpha	[0; 0,1; 0,2; 0,001; 0,01]	0	0	61,68

Tabelle 15: Iterative Rastersuche für EB

Eine spezielle Form von EB sind Random Forest Klassifikatoren [32]. Dieser Algorithmus ermöglicht es die Varianz zu senken, indem gleichzeitig mehrere EB verwendet werden. Die Auswahl der verwendeten Features wird, wie es auch schon der Name andeutet, zufällig durchgeführt. Die Anzahl der verwendeten Features pro EB kann mit dem Parameter max_features gesetzt werden. Die empfohlene Anzahl von Features ist die Quadratwurzel der Gesamtanzahl an Features. Mit bootstrap kann entschieden werden, ob für jeden EB der gesamte Datensatz oder nur eine Teilmenge verwendet wird. Der wohl wichtigste Parameter für den Random Forest Klassifikator ist die Anzahl der Untereintscheidungsbäume, die n_estimators. Eine größere Anzahl von Entscheidungsbäumen kann zu besseren Ergebnissen führt, jedoch erhöht sich auch die Rechenzeit deutlich. Die Auswahl der Hyperparameter wurde mit 100 EB durchgeführt. Aufgrund der Ergebnisse der Rastersuche für EB in Tabelle 16 wird der „entropy" Entscheidungsalgorithmus und ccp_alpha = 0 verwendet. Aufgrund des beschränkten Speicherplatzes des Endmodells war die Auswahl der Parameter auch vom Speicherverbrauch abhängig, welcher für dieses Modell 12 GB nicht überschreiten durfte, damit es auch auf gewöhnlichen Rechnern betrieben werden kann.

Hyperparameter	**Raster**	**Standard**	**Bestes Ergebnis**	**Genauigkeit in %**	**Speicherverbrauch in GB**
max_features	[„sqrt", „log2"]	-	„sqrt"	70,02	-
bootstrap	[True, False]	True	False	70,44	-
n_estimators max_depth D	[100, 300, 500 (D=50), 1000 (D=30)]	100	1000	71,20	292
algorithm	[Random-Forest, Extra-Tree]	RF	RF	71,20	292
min samples split & leaf mit 300 estimators	[3, 5, 10, 100]	1, 2	10	70,04	26,6
n_estimators	[100, 200, 300]	-	100	69,35	8,9

Tabelle 16: Iterative Rastersuche für Random Forest Klassifikatoren

4.6.8 Naive Bayes Networks

Multinomiale NBN können mithilfe diskreter Features Klassifizierung durchführen und haben nur einen wichtigen Hyperparameter α [32]. Dieser Parameter wird verwendet, um das sog. smoothing durchzuführen und hat den Wertebereich [0..1]. $\alpha = 0$ bedeutet kein smoothing, $\alpha = 1$ ist eine spezielle Form des smoothings, das sog. Laplace smoothing und $\alpha < 1$ ist das Lidstone smoothing [32]. Tabelle 17 zeigt die Ergebnisse der Rastersuche für NBN.

Hyperparameter	Raster	Standard	Bestes Ergebnis	Genauigkeit in %
alpha	[1; 10^{-10}; 0,5]	-	1	54,17
fit_prior	[True, False]	True	True	54,17

Tabelle 17: Iterative Rastersuche für NBN

Mit der Veränderung der Hyperparameter des NBN konnte weder eine Verbesserung noch eine Verschlechterung der Ergebnisse bewirkt werden. Die Parameter hatten keine Auswirkung auf die Trainingsgenauigkeit, jedoch minimale Auswirkungen auf die Laufzeit.

4.6.9 Andere Architekturen

Einige der in Punkt 2.6.3 genannten Architekturen konnten nicht oder nicht sinnvoll implementiert werden. SVM wurde, aufgrund der sehr hohen Anzahl an Hyperparametern und Kernel-Algorithmen, ausgeschlossen. Die RBM ist in der Sklearn-Bibliothek [32] als Unsupervised-Learning-Algorithmus implementiert und daher für die gegebene Problemstellung ungeeignet. Ein DBN ist ein Zusammenschluss von mehreren RBMs und basiert daher auch auf Unsupervised Learning. Außerdem gibt es zum Zeitpunkt des Verfassens der vorliegenden Arbeit keine Implementierung für die aktuelle Tensorflow-Version[3].

4.7 Bestimmung der optimalen Hyperparameter

Das Finden der optimalen Hyperparameter wurde mittels einer Rastersuche von ausgewählten Parametern durchgeführt. Dabei wurde jede mögliche Kombination der Parameter bis zur Konvergenz trainiert und geloggt. Mithilfe von Tensorboard, welches ein Modul von Tensorflow [19] ist, lassen sich die Trainingsverläufe graphisch darstellen und vergleichen. Der Auswahlprozess fand mit den im letzten Abschnitt beschriebenen Aspekten statt. Aufgrund der hohen Rechendauer konnten nicht alle Kombinationen für die Optimierung der Hyperparameter durchgeführt werden. Deshalb wurde

[3]Tensorflow Version 2.3

in dieser Arbeit eine evolutionäre Suche durchgeführt. Dabei wurde jeder Hyperparameter-Wert nur einmal angewandt und die zu jedem Zeitpunkt besten Parameter wurden für den nächsten Trainingslauf verwendet. Das FFNN ist mit 74,70% Genauigkeit (T28) gegenüber anderen Architekturen überlegen, weshalb es weiter optimiert wurde. Es ist anzumerken, dass alle Hyperparameter, außer der Anzahl der Hidden Layer, den zugehörigen Adam-Optimizer beeinflussen und nicht das FFNN selbst. Außerdem wurde der learning_rate Parameter hier nur getestet, um einen guten Wertebereich zu finden und eine Vorstellung zu bekommen, wie sich das Training entwickelt. Zum Schluss wurde ein Learning Rate Scheduler, der im folgenden Abschnitt beschrieben ist, verwendet. Tabelle 18 zeigt den Verlauf der Rastersuche für den Optimizer mit dem FFNN.

Hyperparameter	Raster	Standard	Bestes Ergebnis	Genauigkeit in %
# Hidden Layers	[3, 5, 7, 9, 30, 1, 2]	-	3	75,21
learning_rate	$[5*10^{-4}, 10^{-3}, 10^{-4}]$	10^{-3}	$5*10^{-4}$	76,89
beta1	[0,85; 0,9; 0,95]	0,9	0,9	76,89
beta2	[0,95; 0,999; 0,9995]	0,999	0,999	76,89
amsgrad	[True, False]	False	False	76,89

Tabelle 18: Iterative Rastersuche für den Adam Optimizer

4.8 Aufbau des Learning Rate Schedulers

Wie bereits in Abschnitt 2.5 beschrieben wurde, gibt es viele Strategien, um die Lernrate eines Modells anzupassen. In diesem Abschnitt wird der Auswahlprozess und Testprozess für den Learning Rate Scheduler beschrieben. Aufgrund des implementierten Trainingsprozesses werden alle Trainingsdaten standardmäßig nur einmal in Form von Mini-Batches verwendet, weshalb die Epoche immer 1 ist. Aus diesem Grund sind Epochenbasierte Verfahren, also Step-Based Decay und Exponential Decay, nicht geeignet.

Der Time-Based Decay Learning Rate Scheduler ist der erste verwendete Scheduler. Dieser berechnet die Lernrate nach jedem Mini-Batch und ist abhängig von der Anzahl der Iterationen, also wie viele Datensätze bereits trainiert wurden. Aufgrund der hohen Anzahl an Iterationen (130.000-250.000) muss der Decay entsprechend klein gewählt werden. Um das Training zu beschleunigen, wurde die initiale Lernrate mit einem Faktor, welcher in weiteren Tests gefunden wurde, multipliziert. Außerdem wurde der

passende Decay in einer Rastersuche gefunden. Nach der Rastersuche für geeignete Parameter des Time-Based Decay Learning Rate Schedulers, die in Tabelle 19 zu sehen ist, konnte ein verbesserter Trainingsverlauf erzielt werden.

Parameter	Raster	Standard	Bestes Ergebnis	Genauigkeit in %
decay	$[10^{-8}, 10^{-7}]$	-	10^{-8}	75,56
initial_lr	[0,01; 0,005; 0,001]	-	0,005	75,56

Tabelle 19: Iterative Rastersuche für den Time-Based Decay Learning Rate Scheduler

Der zweite Ansatz basiert auf dem Step-Based Decay und dem in Abschnitt 4.6 vorgestellten Mechanismus zum vorzeitigen Beenden des Trainings. Dieser bricht das Training ab, sobald nach 250 Mini-Batches kein Fortschritt festgestellt werden kann. Die Idee ist es vorzeitig die Lernrate anzupassen, um weiteren Fortschritt zu erzielen. Dafür wird die Lernrate stufenweise um einen Prozentsatz verringert, sobald nach 100 Mini-Batches kein Fortschritt festgestellt wird. Die initiale Lernrate bleibt hier die mit dem besten Ergebnis aus der Rastersuche in Tabelle 18. Tabelle 20 zeigt die Ergebnisse der Rastersuche für das Finden des besten Drop-Faktors für den Custom Step-Based Decay Learning Rate Scheduler.

Parameter	Raster	Standard	Bestes Ergebnis	Genauigkeit in %
drop	[0,05; 0,1; 0,15]	-	0,1	76,53
initial_lr	[0,01; 0,005; 0,001]	-	0,001	76,53

Tabelle 20: Iterative Rastersuche für den Custom Step-Based Decay

Abbildung 4.2 zeigt den Trainingsverlauf mit statischer Lernrate im Vergleich zu einem Trainingsverlauf mit dem Time-Based Decay Learning Rate Scheduler und dem Custom Step-Based Decay Learning Rate Scheduler. Es ist zu sehen, dass statisches Training typischerweise eine steilere Lernkurve hervorgebracht hat, jedoch ist dies der niedrigeren Lernrate anzurechnen, da sich das Training mit dem Custom Step-Based Decay Learning Rate Scheduler bis zu einem gewissen Punkt vom statischen Training nicht unterscheidet. Es ist jedoch anzumerken, dass der Custom Step-Based Decay Learning Rate Scheduler zu einer Verbesserung in den letzten Trainingsepochen geführt hat.

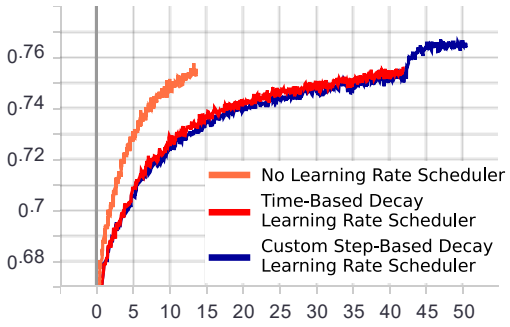


Abbildung 4.2: Trainingsverlauf mit statischer Lernrate gegenüber Time-Based Decay und Custom Step-Based Decay

Aufgrund der anzunehmenden besseren Performance mit einer initialen Lernrate von $5*10^{-4}$ und da nach dem Trainingsprozess sowieso die besten weights gewählt werden, wurde für weitere Trainings der Custom Step-Based Decay Learning Rate Scheduler mit der Lernrate $5*10^{-4}$ und drop = 0,1 verwendet.

4.9 Transfer Learning

Catastrophic Forgetting [8] wird bei neuronalen Netzwerken das Ereignis bezeichnet, bei dem durch das Training neuer Datensätze oder neuer Klassen die gelernten Gewichte „vergessen" werden und die Qualität des neuronalen Netzes sinkt. Um feststellen zu können, ob Catastrophic Forgetting im Transfer-Learning-Prozess stattfindet, wurden ein Basismodell mit 30 (B30) Chiffren, also ungefähr der Hälfte der Gesamtchiffren, und Vergleichsmodelle mit 35 (B35), 40 (B40) und 5 (B5) Chiffren trainiert. Die Auswahl der Chiffren passierte alphabetisch, wobei das Vergleichsmodell mit 5 Chiffren nur solche verwendete, die nicht im Basismodell enthalten waren. Daraus ergeben sich folgende Szenarien:

- B30 wird um 5 neue Chiffren erweitert. (t35)
- B30 wird um 10 neue Chiffren erweitert. (t40)
- B30 wird um 5 neue Chiffren erweitert, jedoch werden die Chiffren einzeln in 5 Schritten hinzugefügt, wobei das neueste Modell immer die neue Basis wird. (t35inc)
- Training eines neuen Modells mit 5 Chiffren, die in B30 nicht enthalten sind, jedoch wird dieses als Basis verwendet. (t5)

Szenario	Genauigkeit in %	Konvergenz nach Iter. in Millionen	ID
B30	94,44	261	T130
B35	92,26	189	T131
B40	85,24	166	T132
B5	99,96	138	T133
t35	92,28	76	T134
t40	86,29	75	T135
t35inc	92,70	50-106	T136-T140
t5	99,98	91	T141

Tabelle 21: Ergebnisse der Transfer Learning Tests

Wie in Tabelle 21 zu sehen ist, führte Transfer Learning in jedem der vier genannten Szenarien zu besseren Ergebnissen. Besonders bei der Erweiterung von Modellen mit einer hohen Anzahl von Klassen sank die Anzahl der benötigten Daten auf weniger als die Hälfte und führte zu mindestens genauso guten oder besseren Ergebnissen wie das Vergleichsmodell. Die inkrementelle Erweiterung des Modells (t35inc) führte sogar zu einer besseren Erkennungsrate von 0,5%. Das liegt wahrscheinlich an der höheren Anzahl an Datensätze, die in mehreren Durchläufen trainiert wurden.

Kapitel 5 Ergebnisse der empirischen Messungen

In Kapitel 4 wurden sechs unterschiedliche Architekturen vorgestellt und auf die besten Ergebnisse hin optimiert. Außerdem wurden die Hyperparameter des Adam-Optimizers in Abschnitt 4.7 optimiert. Folgend werden einige Metriken für die Beurteilung der einzelnen Architekturen vorgestellt. Einfache Entscheidungsbäume wurden in den Diagrammen nicht betrachtet, da der Random Forest Klassifikator eine erweiterte Form davon ist. Feature-Engineering-Algorithmen wurden mit den besten Ergebnissen und den Features aus T30 angeführt. Die folgende Analyse zielt auf eine bestmögliche Kombination von mehreren Klassifikatoren ab, um ein besseres Ensemble-Modell herzustellen.

5.1 Ergebnisse der Messungen

Die Precision-Metrik betrachtet alle positiv gewerteten Einheiten und kann mit der folgenden Formel beschrieben werden [55]:

$$Precision = \frac{TP}{TP + FP}.$$

Precision ist ein Indikator für die Vertrauenswürdigkeit eines Modells für eine Prädiktion für eine bestimmte Klasse, welcher die Proportion korrekt positiv gewerteter Identifikationen zu der Gesamtanzahl von positiven Identifikationen wiedergibt. Eine niedrige Precision für eine Klasse kann darauf hindeuten, dass bis zu einem gewissen Grad eine „Vorliebe“ für diese Klasse entwickelt wurde und diese besonders oft zu FP führt [55]. Abbildung 5.1 zeigt das Ergebnis des Vergleichs der Precision Scores.

Der Recall, welcher auch Erkennungsrate genannt wird, hingegen betrachtet die Rate der gefundenen True Positives gegenüber der echten Gesamtanzahl von positiv klassifizierten Einheiten [55]:

$$Recall = \frac{TP}{TP + FN}.$$

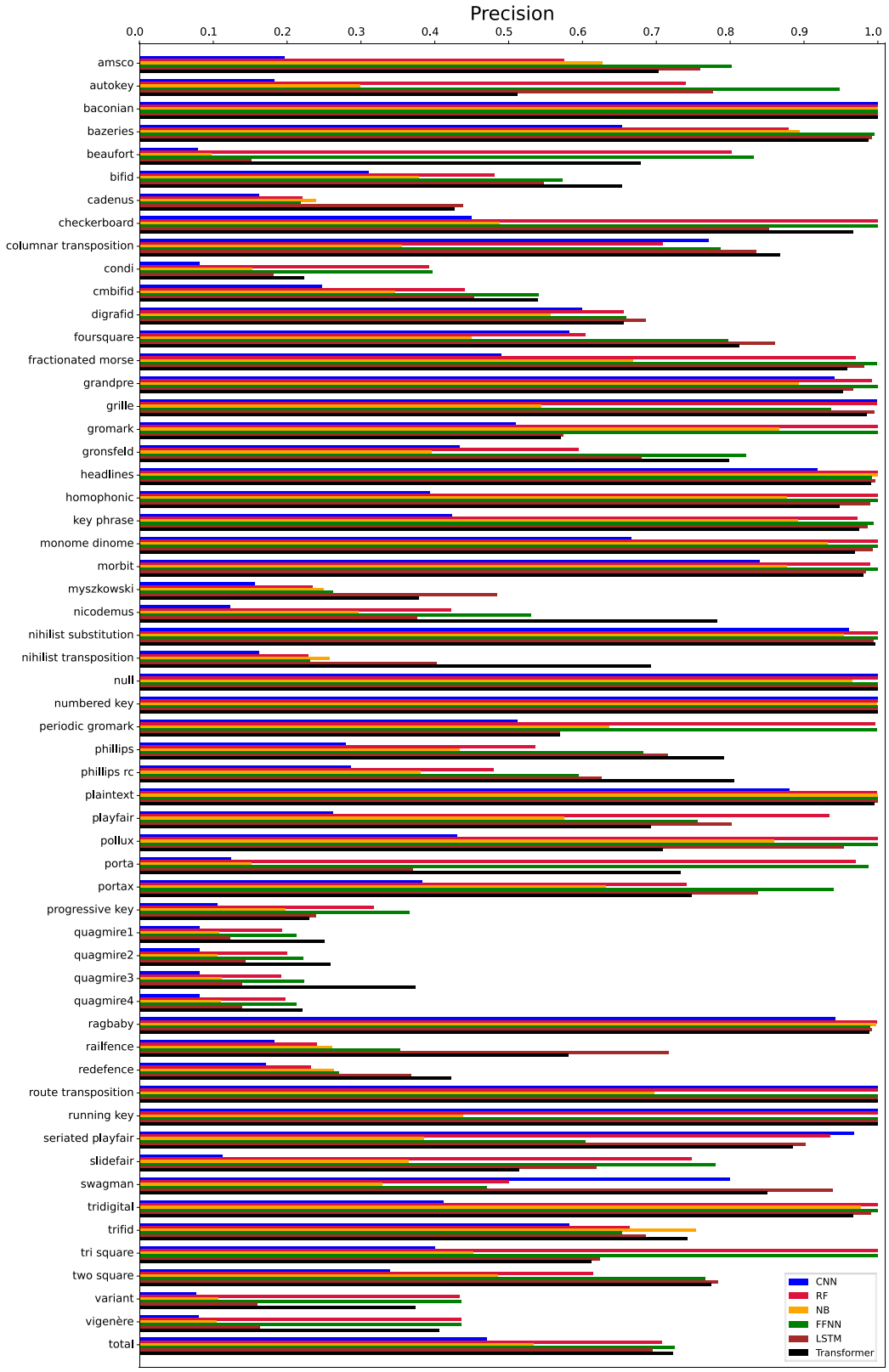


Abbildung 5.1: Vergleich von Precision Scores

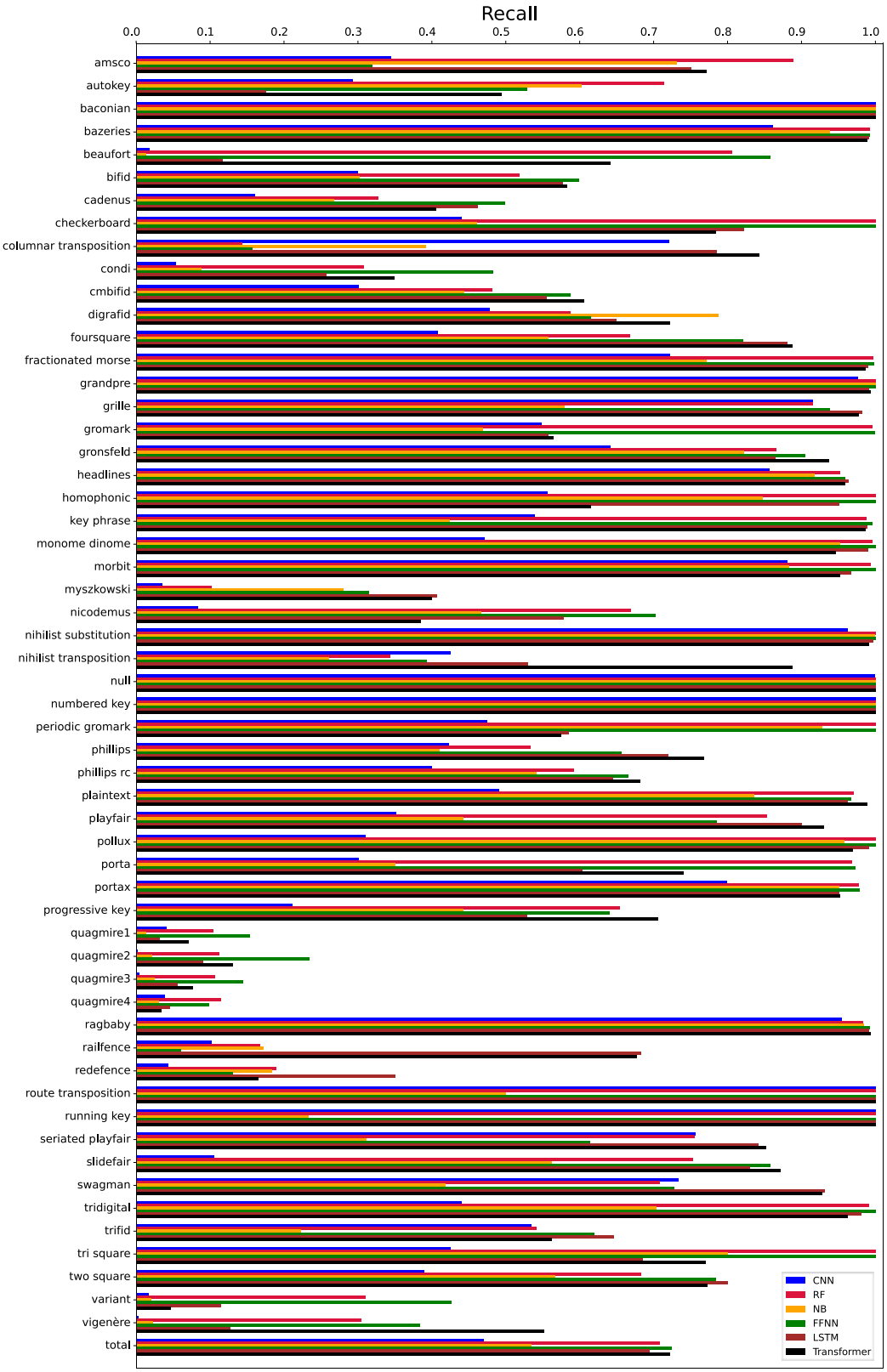


Abbildung 5.2: Vergleich von Recall Scores

Eine der beliebtesten Metriken für Multi-Klassen Klassifizierungsprobleme ist die Genauigkeit. Zwar bieten FFNN die beste Gesamtperformance, wenn das LDI_STATS-Feature verwendet wird, jedoch lässt sich aus dem Diagramm klar herauslesen, dass bei bestimmten Chiffren, wie z.B. der AMSCO-Chiffre [4, p. 30], andere Architekturen überwiegend bessere Ergebnisse liefern. Die verwendete Genauigkeit wird mit folgender Formel berechnet [55]:

$$\text{Genauigkeit} = \frac{TP + TN}{TP + TN + FP + FN}.$$

Der F1-Score ist eine Metrik, die das harmonische Mittel zwischen Precision und Recall berechnet. Modelle mit ähnlicher Precision und Recall haben üblicherweise einen höheren F1-Score gegenüber denen, die sehr unterschiedliche Werte dieser Metriken besitzen. Der F1 Score wird folgendermaßen berechnet:

$$F1_{Score} = 2 \cdot \left(\frac{precision \cdot recall}{precision + recall}\right).$$

Grundsätzlich lässt sich zwischen Macro F1-Score und Micro F1-Score unterscheiden. Der Macro F1-Score wertet die Scores für jede Klasse gleichwertig und der Micro F1-Score gewichtetet die Ergebnisse anhand der Anzahl der Daten für die jeweilige Klasse. Aufgrund der Gleichverteilung der Testdatensätze sind die in Abbildung 5.4 zu sehenden Ergebnisse gleich wie die eines Micro F1-Scores. [55]

Der Matheus Correlation Coefficient (MCC), entwickelt von Matheus [55], misst die Korrelation zwischen der Prädiktion und der Ground Truth und hat den Wertebereich [-1,1]. Wie in anderen Korrelationen bedeutet 1 stark korrelierend, Label und Prädiktion sind sehr ähnlich, -1 stark korrelierend, jedoch systematisch in die falsche Richtung und 0 bedeutet nicht korrelierend. Der MCC, sowie später Cohen's Kappa Score greifen auf folgende Definitionen zu und sind in Abbildung 5.5 zu sehen [55]:

- $c = \sum_k^K C_{kk}$ die Gesamtheit der richtig vorhergesagten Elemente
- $s = \sum_i^K \sum_j^K C_{ij}$ die Gesamtheit aller Elemente
- $p_k = \sum_i^K C_{ki}$ die Anzahl der Prädiktionen für Klasse *k*.
- $t_k = \sum_i^K C_{ik}$ die Anzahl der true positives für Klasse *k*.

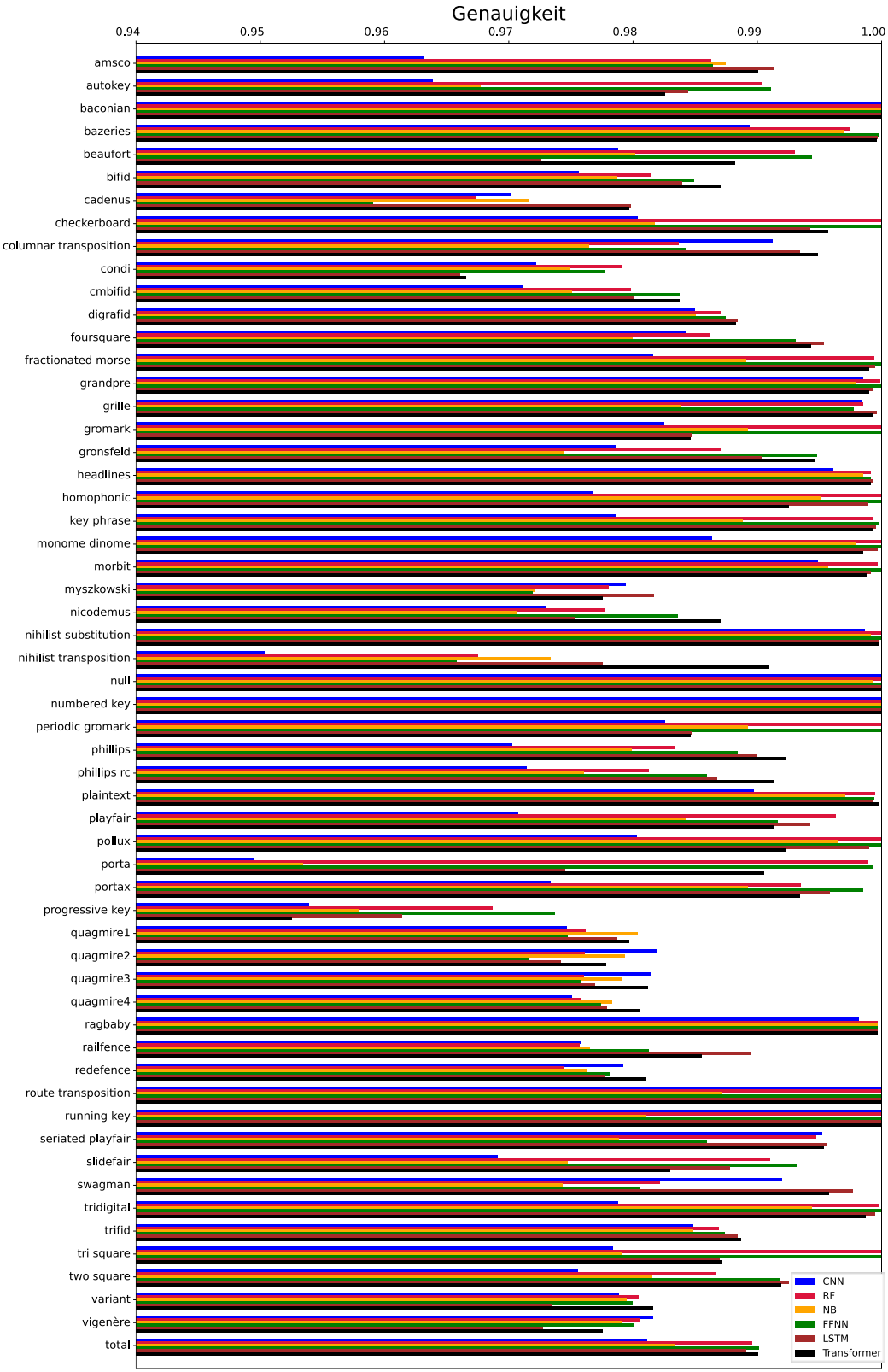


Abbildung 5.3: Vergleich von Genauigkeits-Scores

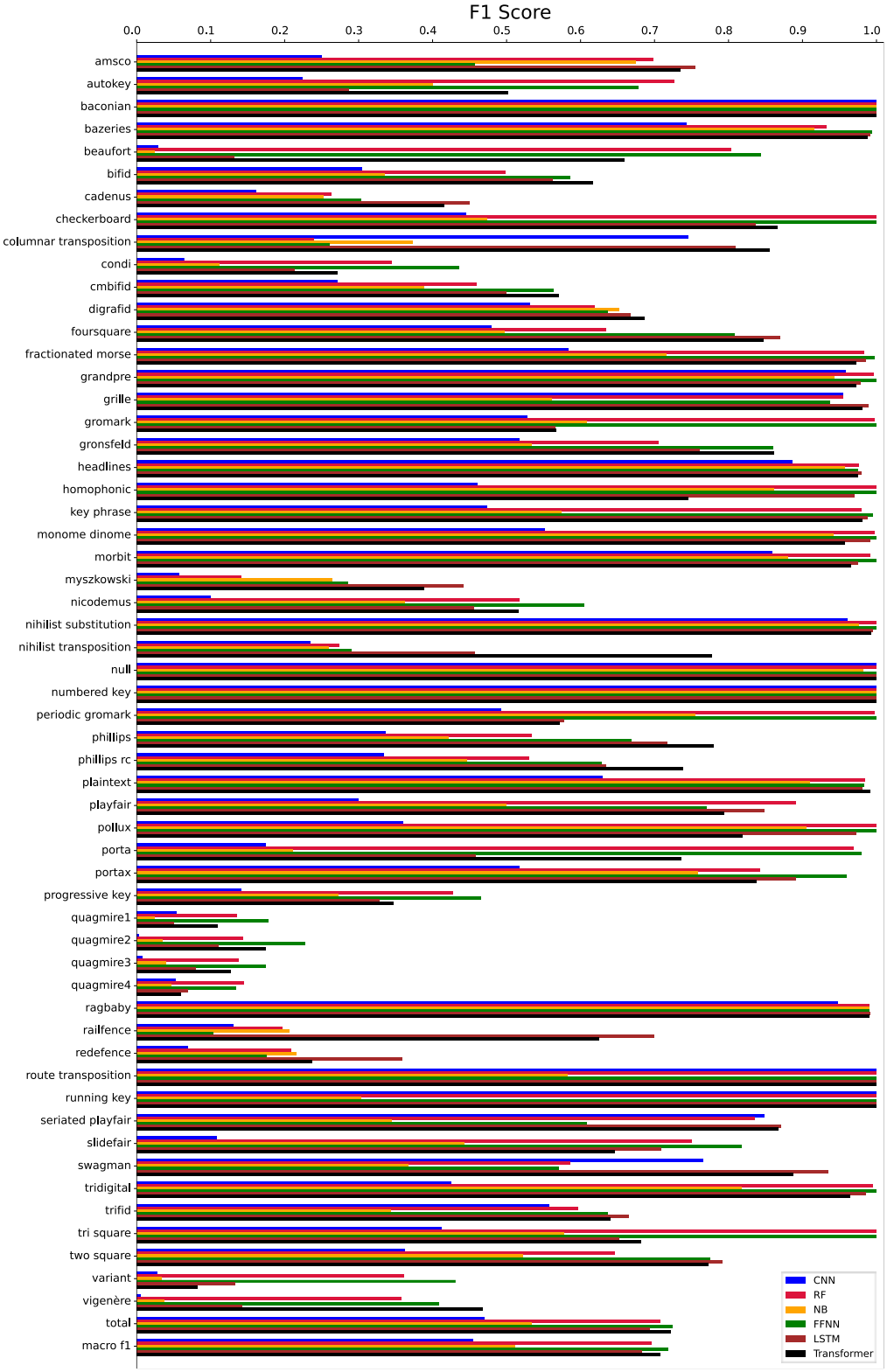


Abbildung 5.4: Vergleich von F1-Scores

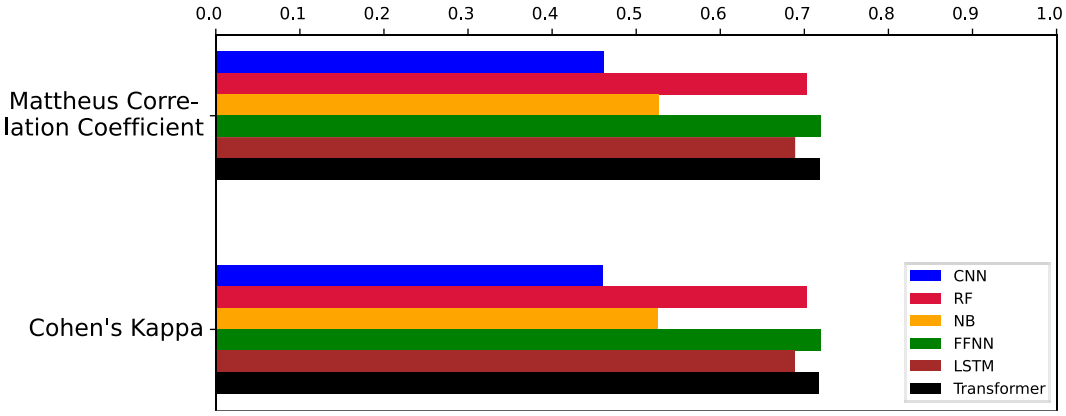


Abbildung 5.5: Vergleich von Statistischen Werten

Der MCC und Cohen's Kappa Score liefern sehr ähnliche Werte aufgrund der ähnlichen Formeln, die folgend zu sehen sind.

$$MCC = \frac{c \cdot s - \sum_k^K p_k \cdot t_k}{\sqrt{(s^2 - \sum_k^K p_k^2)(s^2 - \sum_k^K t_k^2)}}.$$

$$K = \frac{c \cdot s - \sum_k^K p_k \cdot t_k}{s^2 - \sum_k^K p_k \cdot t_k}.$$

Um eine Idee von den größten Fehlern der einzelnen Architekturen zu bekommen, wurden die Top 10 Fehleinschätzungen in Abbildung 5.6 bis Abbildung 5.11 dargestellt.

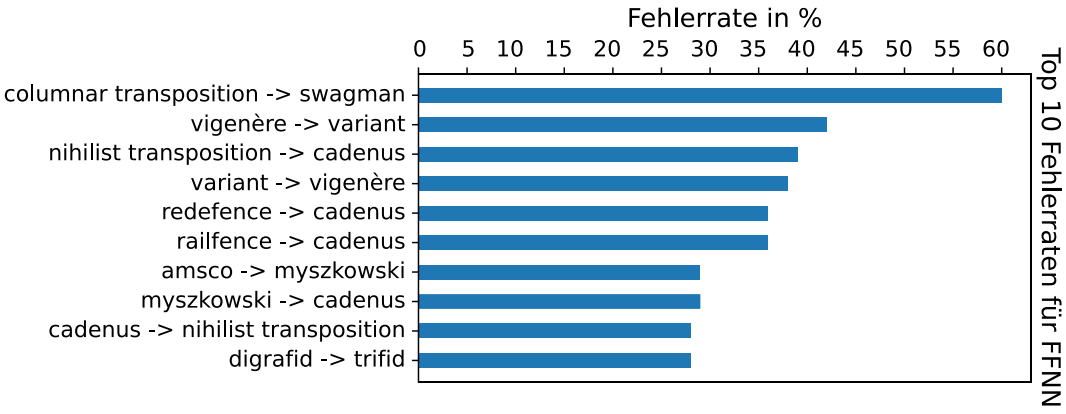


Abbildung 5.6: Top 10 Fehlerraten für FFNN

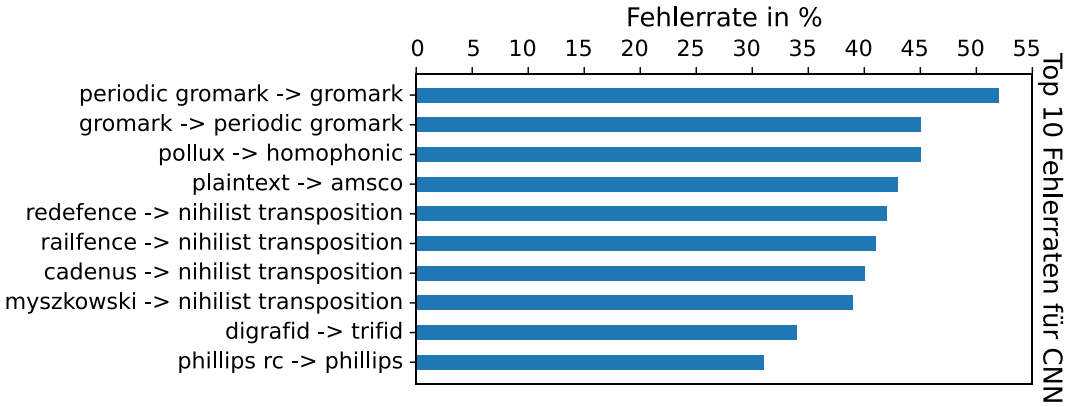


Abbildung 5.7. To 10 Fehlerraten für CNN

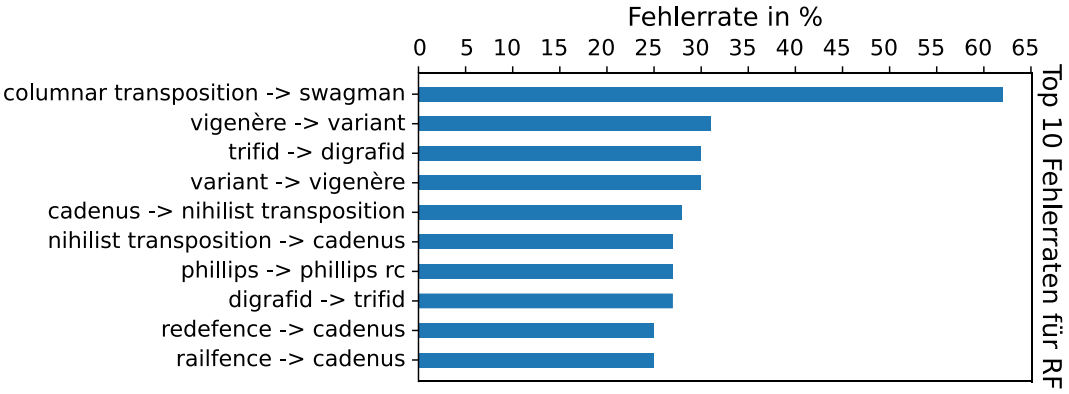


Abbildung 5.8: Top 10 Fehlerraten für RF

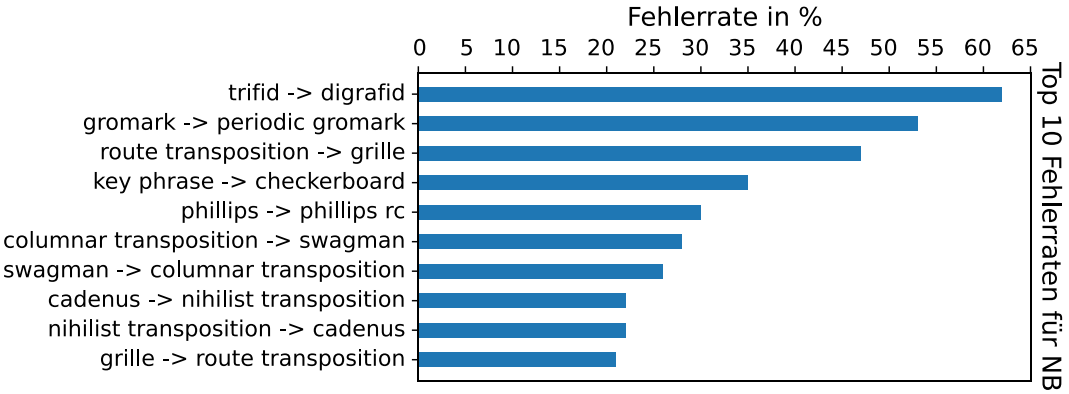


Abbildung 5.9: Top 10 Fehlerraten für NB

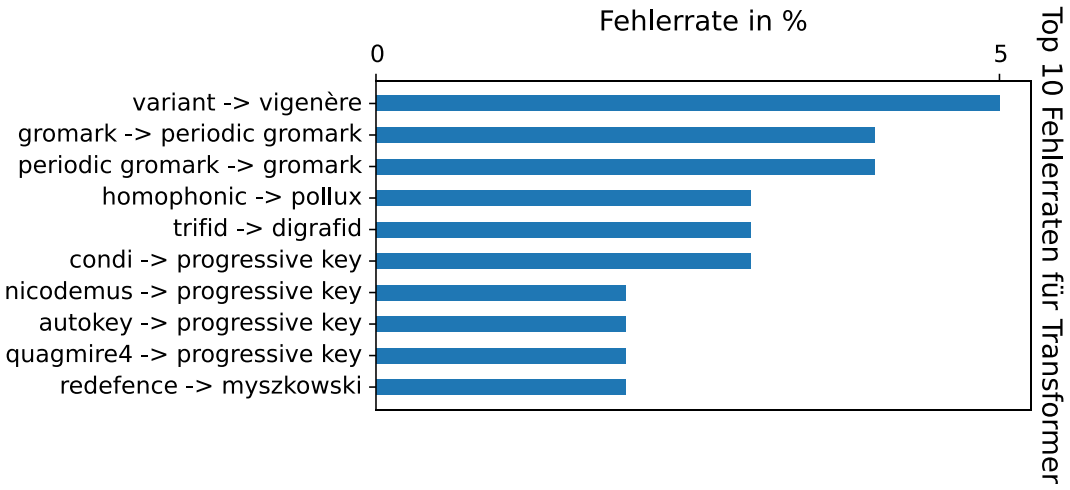


Abbildung 5.10: Top 10 Fehlerraten für Transformer

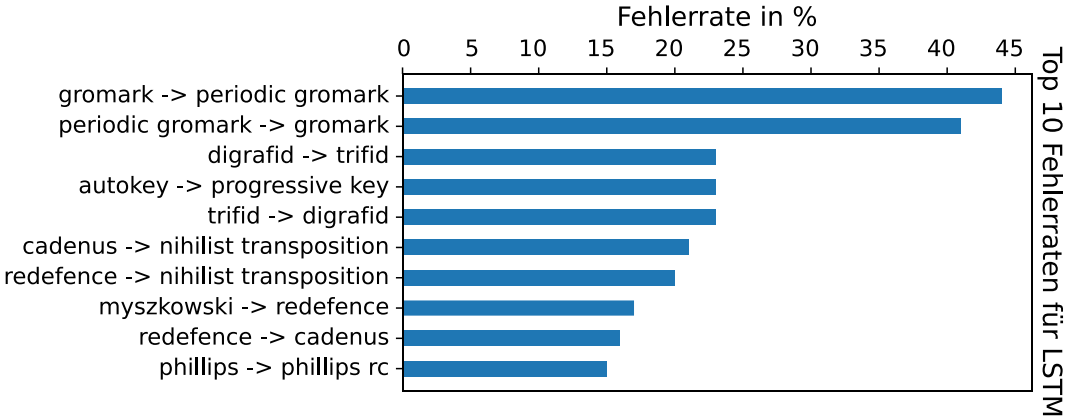


Abbildung 5.11: Top 10 Fehlerraten für LSTM

5.2 Ensemble-Learning-Modell

Als Ensemble wird ein Modell bezeichnet, welches die Prädiktionen von zwei oder mehreren Modellen kombiniert. Diese Modelle können im vornherein mit den gleichen oder unterschiedlichen Datensätzen trainiert werden. Die Prädiktion kann durch einfache Verfahren, wie ein Durchschnitts-Voting, oder komplexere Verfahren, wie z.B. die Gewichtung der einzelnen Votings, berechnet werden. [56]

Das Ensemble-Modell wird mit den Architekturen FFNN, Transformer, LSTM, NBN und RF definiert. Das CNN wird explizit ausgeschlossen, da es bei keiner Klasse gute Ergebnisse liefert und das Ergebnis möglicherweise verschlechtern würde. Als Voting-Methoden werden das Durchschnitts-Voting (weiter als mean-voting bezeichnet) und ein gewichtetes Voting (weiter als weighted-voting bezeichnet) verwendet. Bei dem weighted-voting werden die Wahrscheinlichkeiten mit den jeweiligen Statistiken, also Precision, Recall, Genauigkeit, F1-Score und MCC in einem Voting berechnet werden.

Dadurch soll erreicht werden, dass die für die jeweilige Klasse, bessere Architekturen bevorzugt werden.

Alle folgenden Modelle wurden mit jeweils 10 Millionen Datensätzen mit den Textlängen 100 und 51-428 im Hinblick auf die Genauigkeit evaluiert. Jedes Modell wurde außerdem mit 319 Testdatensätzen (ACA-Daten), welche die Längen 51-428 haben, evaluiert. Diese Testdatensätze stammen von der ACA [4] und wurden von Nuhn [5] für Testzwecke übermittelt, damit bessere Vergleichbarkeit gegeben ist. Die dritte Spalte von Tabelle 22 zeigt die Anzahl der Falschklassifizierungen mit Chiffren, die nicht in den ACA-Daten vorhanden sind. Modelle mit einer genauen Textlänge von 100 sind in Tabelle 22 zu sehen. Aufgrund der Abhängigkeit vieler Chiffren-Typen von der Textlänge, konnten nicht alle Chiffren sinnvoll mit unterschiedlichen Textlängen trainiert werden. Deshalb wurde stattdessen nur ein spezialisiertes Modell für die ACA-Daten trainiert.

Um Falschklassifizierungen mit Chiffren, die nicht in den ACA-Daten vorhanden sind, zu verhindern, wurden Modelle mit den exakten Chiffren-Typen und den Textlängen 51-428 in Tabelle 23 evaluiert. Aufgrund der sich unterscheidenden Anzahl von Chiffren sind diese Modelle nicht kompatibel mit den anderen Testdaten.

	Textlänge 100 in %	**ACA-Daten in %**	**# nicht vorhandener Chiffren in Prädiktionen**
FFNN	78,31	27,90	61
Transformer	72,33	29,15	67
LSTM	72,16	30,41	59
RF	73,50	26,33	49
NBN	52,79	32,92	36
Mean-Voting	82,67	33,54	60
Weighted-Voting	82,78	34,16	61

Tabelle 22: Vergleich der Modelle mit Textlänge 100

	Textlängen [51, 428] mit spezifischen Chiffren-Typen in %	**ACA-Daten in %**
FFNN	67,43	36,68
Transformer	59,54	26,65
LSTM	63,41	31,35
RF	59,15	32,29
NBN	50,71	31,35
Mean-Voting	70,79	35,42
Weighted-Voting	70,78	36,36

Tabelle 23: Vergleich der an die ACA-Daten angepassten Modelle

Die Modelle lieferten beste Ergebnisse für Geheimtexte mit fixen Längen. Die unterschiedlichen Längen der Chiffren macht es bei den Feature-Learning-Algorithmen notwendig ein Padding anzuwenden. Dafür wird der Geheimtext so lange aneinandergehängt, bis die maximale Textlänge erreicht ist, da ein neuronales Netzwerk in der Regel nur mit gleichbleibenden Textlängen arbeiten kann. Zu lange Texte werden auf benötigte Textlänge abgeschnitten. Dieses Padding-Verfahren ist anhand der Transformer und LSTM Modelle in Tabelle 23 zu sehen.

5.3 Diskussion der Ergebnisse

Die Auswertung der 319 ACA-Testdaten hat mit dem spezialisierten Modell zwar besser funktioniert, jedoch konnten die Ergebnisse von Nuhn [5] mit 58,49% nicht erreicht werden. Aus den von Nuhn übermittelten ACA-Testdaten lässt sich nicht auf die 305 verwendeten Datensätze schließen, weshalb die Ergebnisse nicht direkt vergleichbar sind. Außerdem wurden Chiffren, wie z.B. die Vigenère-Chiffre, in Nuhn's Arbeit ausgeschlossen. Die Schlüssellängen wurden außerdem standardmäßig auf 5, 6, 7 und 8 belassen und nicht auf die Daten spezialisiert. Jegliche spezielle Zeichen und Erkennungsmerkmale, wie z.B. der Primer am Beginn des Geheimtextes bei der Gromark-Chiffre, wurden herausgefiltert.

Grundsätzlich kann man nach den Evaluierungen mit den selbstgenerierten Testdaten die Aussage treffen, dass fixe Textlängen sehr gute Ergebnisse für 56 Chiffren-Typen liefern. Die Kombination aus unterschiedlichen Textlängen und dem Weglassen von einfach erkennbaren Chiffren-Typen führt im Durchschnitt zu 10-15% schlechteren Ergebnissen gegenüber denen aus Tabelle 22. Das bedeutet, dass eine generalisierte Aussage über die Qualität eines neuronalen Netzwerks für die Cipher Type Detection abhängig von den verwendeten Chiffren-Typen ist. Beispielsweise liefert in Tabelle 22 RF bessere Genauigkeit als Transformer und LSTMs, welche in etwa gleich gute Ergebnisse liefern. In Tabelle 23 liefern LSTMs deutlich bessere Ergebnisse als Transformer und RF.

Kapitel 6 **Fazit und Ausblick**

Durch eine Vielzahl an Experimenten im Bereich der Cipher Type Detection und der eindimensionalen Textklassifizierung konnte ein gutes Setup für ein neuronale Netz mit 56 Klassen gefunden werden. Die Architekturen der neuronalen Netze können in Feature-Engineering- und Feature-Learning-Algorithmen unterteilt werden. FFNN, RF und NBN gehören zu der Gruppe der Feature-Engineering-Algorithmen. Transformer, LSTM und CNN sind Feature-Learning-Algorithmen. Das FFNN liefert die beste Genauigkeit als einzelnes Modell, jedoch können mit Feature-Learning-Algorithmen mit unterschiedlichen Textlängen gleichwertige Ergebnisse erreicht werden. Der Vorteil von Feature-Engineering-Algorithmen ist, dass die Anzahl der Features für Texte gleichbleibt und im Gegenzug dazu Padding angewendet werden muss, wenn Feature-Learning-Algorithmen verwendet werden. Die Zweiteren haben aber den Vorteil, dass weniger Rechenleistung für das Training der neuronalen Netze notwendig ist und der größte Anteil davon auf den GPUs ausgeführt wird. In Kapitel 4 wurde die Auswahl der verwendeten Features ausführlich getestet und die Hyperparameter der neuronalen Netze wurden optimiert. Die Testung der einzelnen Features hat einen Großteil der notwendigen Rechenleistung ausgemacht, weshalb auch die Ergebnisse des FFNN etwas besser sind als die der Feature-Learning-Algorithmen.

Durch die Verwendung von eigens entwickelten Ensemble-Modellen konnten die Stärken der einzelnen Modelle genutzt werden, ohne zusätzlichen Aufwand für das Training aufwenden zu müssen. Zwar erhöht sich, aufgrund der Anzahl der verwendeten Modelle, die Evaluierungszeit deutlich, jedoch führen die Ensemble-Modelle zu einer verbesserten Erkennungsrate.

Die Verbesserung der Modelle in den einzelnen Phasen dieser Masterarbeit wurde evolutionär durchgeführt. Das heißt, dass Verbesserungen bei einem Parameter sofort in den weiteren Tests angewandt wurden, statt alle Kombinationen, wie bei der Rastersuche, zu testen. Die Metriken Genauigkeit und Top-3-kategorische-Genauigkeit, sowie der Loss wurden zur Auswertung der Ergebnisse verwendet. Eine Erweiterung der vorliegenden Arbeit könnte die Verwendung von automatisierten Feature-Auswahlverfahren, wie z.B. Auto-ViML [57], sein.

Der Custom Step-Based Decay Learning Rate Scheduler führt in manchen Fällen zu besserer Generalisierung der Modelle durch die Anpassung der Lernraten am Ende des Trainings.

In Abschnitt 4.9 konnte gezeigt werden, dass Catastrophic Forgetting in verschiedenen Szenarien nicht auftritt und durch das Transfer Learning sogar eine 40-60% kürzere Trainingszeit erreicht werden konnte.

Grundsätzlich ist die vorliegende Masterarbeit nur mit Nuhn's Klassifikator [5] in der Hinsicht auf die Anzahl und Art der verwendeten Chiffren-Typen vergleichbar. Eine Einschränkung der Vergleichbarkeit ergibt sich unter anderem auch aus den verschiedenen Datensätzen, die in den Arbeiten verwendet wurden. Es wäre sinnvoll einen Standard-Datensatz im Bereich der Cipher Type Detection von klassischen und modernen Chiffren zu haben. Neben den positiven Erkenntnissen konnte jedoch durch die ACA-Testdaten gezeigt werden, dass gewisse Chiffren-Typen schlechter erkannt werden als andere.

Weitere Arbeiten in diesem Bereich könnten daraus bestehen Modelle zu trainieren, die anderssprachige Texte oder Texte mit Verschlüsselungsfehlern verwenden, da diese historisch bedingt sehr wahrscheinlich des Öfteren aufgetreten sind. Eine weitere Fragestellung ist, ob andere Features die Schlüssellängen der vorliegenden Chiffren mithilfe des vorhergesagten Chiffren-Typs bestimmt werden können und welcher Ansatz sich für diese Problemstellung am besten eignet. In weiteren Schritten könnte die Effektivität der vorliegenden Klassifikatoren mit den verwendeten Features auch für die Erkennung von modernen Chiffren getestet werden.

Literaturverzeichnis

[1] A. v. Lieven, Grundriss des Laufes der Sterne. Das sogenannte Nutbuch, Dänemark: The Carsten Niebuhr Institute of Ancient Eastern Studies. ISBN 978-3-15-96137-8, 2007.

[2] B. Megyesi, B. Esslinger, A. Fornés, N. Kopal, B. Láng, G. Lasry, K. de Leeuw, E. Pettersson, A. Wacker und M. Waldispühl, „The DECRYPT Project," *Cryptologia,* Bd. 44, Nr. 6, pp. 545-559, Februar 2020.

[3] N. Kopal, „Of Ciphers and Neurons - Detecting the Type of Ciphers Using Artificial Neural Networks," in *Proceedings of the 3rd International Conference on Historical Cryptology, HistoCrypt*, 2020.

[4] American Cryptogram Association, „The ACA and You - A handbook for the members of the American Cryptogram Association," USA, 2005.

[5] M. Nuhn, K. Knight, „Cipher Type Detection," in *Conference on Empirical Methods In Natural Language Processing, pages 1769-1773*, Doha, Qatar, Oktober 2014.

[6] E. Leierzopf, „NCID," [Online]. Available: https://github.com/dITySoftware/ncid. [Zugriff am 20. Mai 2021].

[7] A. Grover und J. Leskovec, „node2vec: Scalable Feature Learning for Networks," in *KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, S. 855-864*, California, San Francisco, USA, August 2016.

[8] R. Kemker, M. McClure, A. Abitino, T. Hayes und C. Kanan, „Measuring Catastrophic Forgetting in Neural Networks," Rochester Institute of Technology, New York, USA, November 2017.

[9] A. R. Hevner, S. T. March, J. Park und S. Ram, „Design Science in Information Systems Research," *MIS Quarterly,* Bd. 28, Nr. 1, pp. 75-105, März 2004.

[10] I. Goodfellow, Y. Bengio und A. Courville, „Deep Learning," , pages 326-366, 416-437, MIT Press, Cambridge, MA, USA, 2016.

[11] „Bion's Gadget," [Online]. Available: https://bionsgadgets.appspot.com/. [Zugriff am 18. September 2020].

[12] K. He, G. Gkioxari, P. Dollar und R. Girshick, „Mask R-CNN," USA, März 2017.

[13] P. Burgstaller, E. Hermann und H. Lampesberger, Künstliche Intelligenz, 1. Auflage, Österreich: Manz, Oktober 2019.

[14] S. Russell und P. Norvig, Künstliche Intelligenz - Ein moderner Ansatz, 3. Auflage, Deutschland, 2012.

[15] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Graepel, T. Lillicrap, K. Simonyan und D. Hassabis, „A general reinforcement learning algorithm that masters chess, shogi and Go through self-play," *Science,* Bd. 6419, Nr. 362, pp. 1140-1144, Dezember 2018.

[16] P. Domingos, The Master Algorithm - How the Quest for the Ultimate Learning Machine Will Remake Our World, Jänner 2017.

[17] P. Tino, L. Benusova und A. Sperduti, Artificial Neural Networks. Springer Handbook of Computational Intelligence, pp. 455-471, Springer, 2015.

[18] „Keras," [Online]. Available: https://keras.io/. [Zugriff am 18. Mai 2021].

[19] „Tensorflow," Google, [Online]. Available: https://www.tensorflow.org/. [Zugriff am 20. Mai 2021].

[20] Y. Liu, J. Zhang, C. Gao, J. Qu und L. Ji, „Natural-Logarithm-Rectified Activation Function in Convolutional Neural Networks," in *IEEE 5th International Conference on Computer and Communications*, China, Dezember 2019.

[21] D. Clevert, T. Unterthiner und S. Hochreiter, „Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," Linz, Österreich, Februar 2016.

[22] P. Ramachandran, B. Zoph und Q. Le, „Searching for Activation Functions," Oktober 2017.

[23] B. Karlik und V. Olgac, „Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks," *International Journal of Artificial Intelligence and Expert Systems (IJAE),* Bd. 1, Nr. 4, pp. 111-122, Februar 2011.

[24] S. Lau, „Towards Data Science," Towards Data Science Inc., 29. Juli 2017. [Online]. Available: https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1. [Zugriff am 17. Dezember 2020].

[25] Y. Kim, C. Denton, L. Hoang und A. Rush, „Structured Attention Networks," ICLR 2017. Harvard University, Cambridge, MA 02138, USA, Februar 2017.

[26] S. Gunn, „Support Vector Machines for Classification and Regression," University of Southhampton, Southhampton, England, Mai 1998.

[27] M. Pal und P. Mather, „Support vector machines for classification in remote sensing," *International Journal of Remote Sensing,* Bd. 26, Nr. 5, pp. 1007-1011, März 2005.

[28] J. Louradour und H. Larochelle, „Classification of Sets using Restricted Boltzmann Machines," März 2011.

[29] M. Anyanwu und S. Shiva, „Comparative Analysis of Serial Decision Tree Classification Algorithms," *International Journal of Computer Science and Security,* Bd. 3, Nr. 3, pp. 230-240, Juni 2009.

[30] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu und F. Alsaadi, „A survey of deep neural network architectures and their applications," *Neurocomputing,* Bd. 234, pp. 11-26, April 2017.

[31] Y. Huang und L. Li, „Naïve Bayes classification algorithm based on small sample set," in *pp. 34-39. IEEE International Conference on Cloud Computing and Intelligence Systems*, Beijing, China, September 2011.

[32] „Sklearn," [Online]. Available: https://scikit-learn.org/stable/. [Zugriff am 19. Mai 2021].

[33] T. Ramge, Mensch und Maschine - Wie Künstliche Intelligenz und Roboter unser Leben verändern, Deutschland: Phillipp Reclam jun. Verlag, 2019.

[34] S. Hochreiter und J. Schmidhuber, „Long Short-Term Memory," *Neural Computation,* Nr. 9, pp. 1735-1780, 1997.

[35] A. Graves, „Supervised Sequence Labelling with Recurrent Neural Networks," *Studies in Computational Intelligence,* Bd. 385, Februar 2012.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser und I. Polosukhin, „Attention Is All You Need," in *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, California, USA, Dezember 2017.

[37] B. Carter, T. Magoc, „Classical Ciphers and Cryptanalysis," 2007.

[38] C. Shannon, „Communication Theory of Secrecy Systems," Bell System Technical Journal, 1949.

[39] G. Sivagurunathan, V. Rajendran und T. Purusothaman, „Classification of Substitution Ciphers using Neural Networks," *IJCSNS International Journal of Computer Science and Network Security,* Bd. 10, Nr. 3, pp. 274-279, März 2010.

[40] A. Abd und S. Al-Janabi, „Classification and Identification of Classical Cipher Type using Artificial Neural Networks," *Journal of Engineering and Applied Sciences,* Bd. 14, 2019.

[41] N. Krishna, „Classifying Classic Ciphers using Machine Learning," San Jose State University, California, USA, Mai 2019.

[42] Z. Zhao, Y. Zhao und F. Liu, „The Research of Cryptosystem Recognition Based on Randomness Test's Return Value," in *Cloud Computing and Security - 4th International Conference. pp. 1-15*, Haikou, China, Juni 2018.

[43] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray und S. Vo, „A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," NIST Special Publication 800-22 Rev. 1a, USA, April 2010.

[44] C. Tan und Q. Ji, „An Approach to Identifying Cryptographic Algorithm from Ciphertext," in *8th IEEE International Conference on Communication Software and Networks*, Chengdu, Sichuan Province, China, 2016.

[45] R. Quinlan, C4.5: Programs for Machine Learning, Australien: Morgan Kaufmann Publishers, 1993.

[46] R. Manjula und R. Anitha, „Identification of Encryption Algorithm Using Decision Tree," in *CCSIT 2011, Part III, CCIS 133, pp. 237-246*, New Delhi, India, 2011.

[47] B. Chandra, P. Pallath, P. Saxena und S. Kant, „Neural Networks for Identification of Crypto Systems," in *3rd Indian International Conference of Artificial Intelligence (IICAI-07) pp. 402-411*, New Delhi, India, Jänner 2007.

[48] „Gutenberg-Projekt," [Online]. Available: https://www.gutenberg.org/. [Zugriff am 5. April 2020].

[49] E. Leierzopf, „Systematische Evaluierung möglicher Ansätze für Unit Tests in der Software-Qualitätssicherung," FH Hagenberg, Hagenberg, Österreich, 2019.

[50] „Wikipedia Frequent Wordlists," 22. April 2021. [Online]. Available: https://en.wiktionary.org/wiki/Wiktionary:Frequency_lists. [Zugriff am 22. Mai 2021].

[51] „Github 100.000 der häufigsten englischen Wörter," 5. März 2012. [Online]. Available: https://gist.github.com/h3xx/1976236. [Zugriff am 10. Oktober 2020].

[52] R. J. Friedman, „Syllabary Cipher," *Cryptogram,* Mai-Juni 2012.

[53] S. H. Lee, K. Kim und Y. Shin, „Effective Feature Selection Method for Deep Learning-Based Automatic Modulation Classification Scheme Using Higher-Order Statistics," in *1st International Conference on Artificial Intelligence in Information and Communication (ICAIIC 2019)*, Okinawa, Japan, Februar 2019.

[54] A. Nandan, „Keras Transformer," 10. Mai 2020. [Online]. Available: https://keras.io/examples/nlp/text_classification_with_transformer/. [Zugriff am 19. Mai 2021].

[55] M. Grandini, E. Bagli und G. Visani, „Metrics for Multi-Class Classification: An Overview," Università degli Studi di Bologna, Dipartimento di Ingegneria e Scienze Informatiche, Bologna, Italien, August 2020.

[56] J. Brownlee, „Why Use Ensemble Learning?," Machine Learning Mastery Pty. Ltd, 26. Oktober 2020. [Online]. Available: https://machinelearningmastery.com/why-use-ensemble-learning/. [Zugriff am 29. Dezember 2020].

[57] R. Seshadri, „AutoViML," 17. Mai 2021. [Online]. Available: https://github.com/AutoViML/Auto_ViML. [Zugriff am 24. Mai 2021].