Erkennung japanischer Schriftzeichen mittels maschinellen Lernens

Sebastian Schmidt
Fachhochschule Südwestfalen

***Zusammenfassung*—Das japanische Schriftsystem ist mit drei verschiedenen Alphabeten (Hiragana, Katakana, Kanji) eines der komplexesten Schriftsysteme der Welt. Eine Person muss dabei im Japanischen ca. 2000 Schriftzeichen erlernen, um eine Zeitung vollständig verstehen zu können. In dem Paper „Recognizing Handwritten Japanese Characters Using Deep Convolutional Neural Networks" von Charlie Tsai wurde eine Erkennung mittels der VGGNet, speziellen Architekturen Konvolutionaler Neuronaler Netze, erprobt. Im Ansatz wurde dabei sowohl die separate Erkennung aller Alphabete als auch die gemeinsame aller möglichen Schriftzeichen erprobt. Dabei konnten Testgenauigkeiten von bis zu 99,53% erreicht werden. In diesem Paper sollen die Ergebnisse nachvollzogen und mit weiteren Mitteln die Qualität des Modells überprüft werden. Dies soll den Einsatz und klaren Vergleich weiterer Netzwerkarchitekturen wie beispielsweise der Vision Transformer oder ResNet ermöglichen. Zusätzlich wurden weitere moderne VGG und ResNet Architekturen gesichtet und mit dem implementierten Mitteln überprüft. Auch ein Einsatz weiterer Datensätze aus der vorliegenden Datenbank wurde erprobt.**

I. Einleitung

Die japanische Sprache ist eine der komplexesten Sprachen der Welt. Besonders für Muttersprachler europäischer Sprache sind die hohen Abweichungen in Aussprache, Grammatik und Schriftsystem ein Grund für große Probleme beim Erlernen der Sprache. Laut dem FSI gilt Japanisch, neben Sprachen wie Arabisch, Koreanisch und Chinesisch, als Kategory IV Sprache, welche mindestens 2200 Lernstunden für englische Muttersprachler benötigen und damit als eine der schwierigsten Sprachen gilt [1].

Geteilt in drei Alphabete (Hiragana, Katakana, Kanji) muss ein Schüler des Japanischen über 2000 Schriftzeichen erlernen, um eine Zeitung ohne Hilfe lesen zu können. Dazu kommt zusätzlich nicht nur die Bedeutung dieser, sondern im Falle der Kanji auch verschiedene Lesungen, welche von einigen bis in Dutzende pro Zeichen reichen können [2].

Schüler des Japanischen können heutzutage auf viele Tools zurückgreifen, um Ihren Lernvorgang zu unterstützen. Auch gibt es einige Apps, welche die automatische Erkennung von japanischen Schriftzeichen ermöglichen. In Charlie Tsais Thesis "Recognizing Handwritten Japanese Characters Using Deep Convolutional Neural Networks"konnte dieser bis zu 99,53% aller Schriftzeichen der ETL Character Database, einer Datenbank von handgeschriebenen japanischen Schriftzeichen, korrekt klassifizieren [3]. Diese Ergebnisse sollen in diesem Projekt durch den Einsatz der genutzten VGGNet nachvollzogen und falls möglich durch die Nutzung weiterer Techniken wie Vision Transformer verbessert werden. Für den praktischen Teil der Arbeit wurde Python 3 mit Keras und Tensorflow 2 verwendet.

Dieser Beitrag entstand im Rahmen des Master-Konferenzseminars *info@swf 2022*, das im Wintersemester 2021/22 vom Fachbereich Informatik und Naturwissenschaften der Fachhochschule Südwestfalen durchgeführt wurde.

II. Grundlagen

Im Folgenden Abschnitt sollen kurz die Grundlagen der eingesetzten Modelle dargestellt werden. Dabei wird zunächst auf die Konvolutionale Neuronale Netze und dann auf die Vision Transformer eingegangen. Grundlagen zu Neuronalen Netzen müssen aufgrund des Umfangs dabei als gegeben angesehen werden.

A. Konvolutionale Neuronale Netze

Konvolutionale Neuronale Netze oder Convolutional Neural Networks sind Neurale Netze die vom visuellen Cortex des Gehirns inspiriert wurden. Hierbei sind Neuronen in unteren Schichten für die Erkennung primitiver Formen verantwortlich, während Neuronen in höheren Schichten auf immer komplexere Muster reagieren [4, S.360].

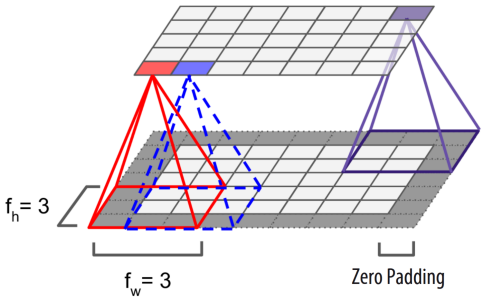

Abbildung 1. Aufbau einer Konvolutionalen Schicht mit Schrittweite eins und Zero Padding [4, S.362].

Zur Umsetzung haben sich hier in der Praxis sogenannten Konvolutionale Schichten mit einem bestimmten Wahrnehmungsfeld mehrere Pixel etabliert, welche ihre Eingabe mit einer Schrittweite (Stride) analysieren und so aus den Pixeln eine neue Ausgabe erzeugen. Dies können beispielsweise Bilder oder Ausgaben vorherigen Konvolutionaler Schichten sein. Die Ränder der aktuellen Eingabe werden dabei in der Regel mit einem Padding versehen, um

eine Ausgabe der selben Größe wie der Eingabe zu ermöglichen [4, S.361-363]. Die Eingabe des Wahrnehmungsfelds wird mittels eines sogenannten Filters gewichtet, bevor es einen Ausgabepixel erzeugt. Diese lassen sich als kleine Bilder visualisieren, welche auf bestimmte Formen in der Ausgangsschicht reagieren [4, S.363].

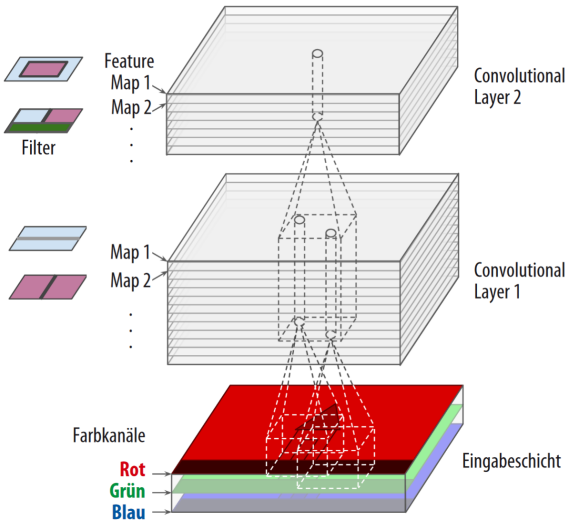

Abbildung 2. Aufbau eines CNN mit drei Schichten und vielen Feature Maps [4, S.365].

Pro Konvolutionaler Schicht können nun mehrere Filter, sogenannte Feature Maps, auftreten, welche die Erkennung verschiedener Formen in der vorherigen Schicht erlauben und in tieferen Schichten die Kombination vorheriger Features ermöglichen. Tatsächlich erstreckt sich nun das Wahrnehmungsfeld über alle vorherigen Feature Maps, was in einer dreidimensionalen Gewichtsmatrix resultiert und aufgrund der mehreren Kanäle nun auch eine bessere Analyse von Farbbildern ermöglicht [4, S.364-365]. Typischerweise wird durch eine passende Wahl von Strides und Filtern eine Komprimierung der Bildgröße bei Steigerung der Feature Maps angestrebt. Eine Komprimierung kann dabei aber nicht nur durch Strides, sondern auch durch Pooling Layer geschehen. Ein Pooling Layer betrachtet in der Regel einen gewissen zweidimensionalen Bereich einer Ausgabe und verarbeitet den auf eine gewisse Weise. Hierdurch entsteht eine Komprimierung der Ausgabe der vorherigen Konvolutionalen Schicht. Eine typische Art von Pooling ist das Max-Pooling, bei welcher die größte Zahl des Bereichs übernommen wird [4, S.369-370]. In den meisten Architekturen von CNNs, wird am Ende die Ausgabe der letzten Konvolutionalen Schicht geebnet und mit einem Feed-Forward-Netz verbunden, welches dann klassische Aufgaben wie Klassifikation vornehmen kann [4, S.371]. Einige verbreitete Architekturen von CNNs, welche aus in diesem Projekt erprobt wurden, sind das ResNet und VGGNet [4, S.378-381].

B. Vision Transformer

Vision Transformer sind eine neue Technik des Maschinellen Lernens auf Basis der Transformer. Diese basieren vom Hauptprinzip auf einer sogenannten Attention und werden vorwiegend im Natural Language Processing eingesetzt, in welchem sie die vorher prävalenten Rekurrenten Neuronalen Netze weitgehend ablösen konnten [5]. Wurden diese in Encoder-Decoder Netzen eingesetzt, so wurde bis zur Nutzung von Attention nur ein letzter Zustand des Encoders als Eingabe im Decoder genutzt. Dies hat sich nach aktueller Forschung als Bottleneck herausgestellt, welches Rekurrenten Neuronalen Netze besonders bei längeren Sätzen stark behindert hat [5, S.2]. Stattdessen wurden neue RNNs entworfen, welche es den Decoder über eine erlernbare Attention erlauben sich in jedem Schritt auf andere Aspekte aller Zustände des Encoders zu konzentrieren [6, S.4].

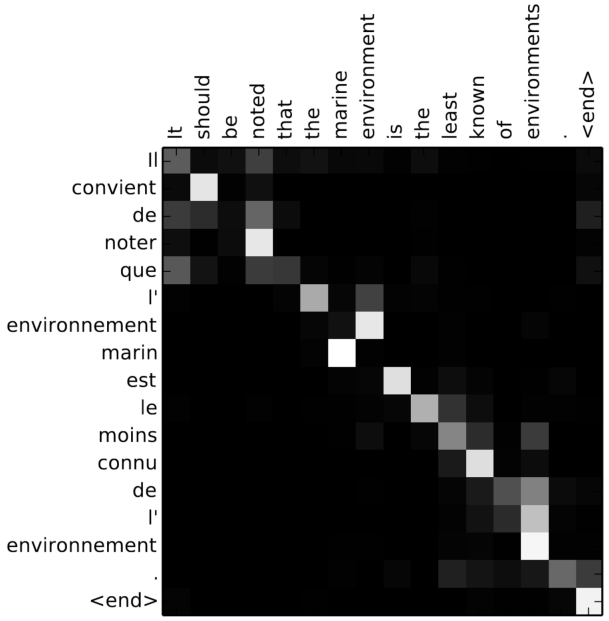

Abbildung 3. Beispiel für erlernte Attention zwischen einem englischen und französischen Satz [6, S.6].

In weiteren Forschungen im Natural Language Processing hat sich herausgestellt, dass reine Netzwerke basierend auf Attention, die weiter oben genannten Transformer, RNNs als state of the art ablösen können [5, S.2].

Vision Transformer sind vom Aufbau den Transformern im Natural Language Processing sehr ähnlich, besitzen jedoch im Fall von Klassifikation nur den Encoder Teil der Transformer Modelle. Der grundsätzliche Aufbau lässt sich in der Abbildung 4 erkennen.

Dabei werden zunächst die Eingabebilder in mehrere zweidimensionale Bereiche einer vordefinierbaren Größe unterteilt, wobei ähnlich wie bei den CNNs ein Stride zum Überlappen von Bildbereichen genutzt werden kann. Danach werden die Eingabedaten über eine trainierbare lineare Projektion auf D, eine konstante vordefinierte Vektorgröße in allen Schichten des Transformers, Dimensionen reduziert und damit die sogenannten Patch-Embeddings generiert. Schließlich werden eindimensionale Positionskennungen der Patches hinzugefügt und an den Anfang

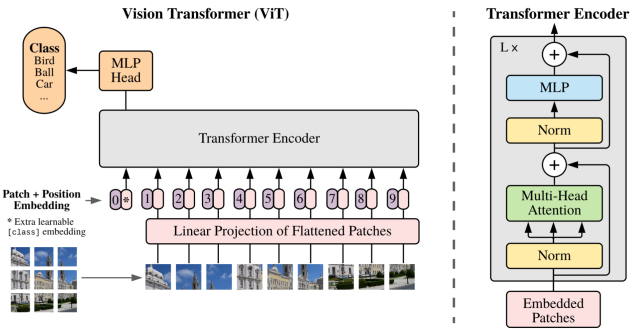

Abbildung 4. Grundsätzlicher Aufbau eines Vision Transformers [7, S.3].

der Daten ein spezielles Class-Token für das Erlernen der vorhandenen Klasse angefügt [7, S.3].

Die Schichten des Encoders bestehen im Kern aus mehreren Multiheaded Self-Attentions, darauffolgende zweischichtigen Multilayer Perceptrons und Residual Connections zwischen den inneren Schichten. Weiterhin wird eine Layernormalisierung vor jeder Multiheaded Self-Attention und jedem Multilayer Perceptron ausgeführt. Am Ende wird aus dem Encoder schließlich der Statusvektor an der Position des Class-Token entnommen und zur Klassifikation in ein weiteres Multilayer Perceptron mit einer einzelnen Schicht geleitet [7, S.3-4].

In den Ergebnissen zeigt sich, dass Vision Transformer ähnliche Ergebnisse wie aktuelle CNNs erzeugen können, wenn sie auf ausreichend großen Datensätzen trainiert werden. Dabei benötigen sie zusätzlich deutlich weniger Rechenkapazitäten [7, S.5]. Dies wird besonders ersichtlich, wenn ein Vision Transformer auf Datensätze verschiedener Größe vortrainiert wird. Hier hat sich gezeigt, dass ein Training auf dem JFT-300M, Googles Hauseigener Datensatz mit über 300 Millionen Bildern, eine grundlegende bessere Genauigkeit als state of the art CNNs erzeugt, während der vergleichsweise kleine ImageNet Datensatz mit 14 Millionen Bildern [8] die Genauigkeit schlechter werden lässt [9].

III. Übersicht der Daten

Die genutzten Bilder von japanischen Schriftzeichen aus der Thesis von Charlie Tsai sind aus der ETL Character Database [3, S.2-3], einer Datenbank erstellt zwischen 1973 und 1984 aus ca. 1,2 Millionen handgeschriebener und gedruckter Hiragana, Katakana, Kanji und weiterer Zeichen. Sie wird geteilt in neun Datensätze ETL-1 bis ETL-9 mit unterschiedlichen Inhalten [10]. Jedes Zeichen liegt als 64 x 64 Pixel Grauwertdaten vor, welche bereits segmentiert und zentriert sowie mit einer speziellen Methode in ein Binärbild umgewandelt wurden [3, S.3]. Aufgrund der speziellen Formate und Spezifikationen der Datensatzdateien, musste ein spezielles Pythonskript hinzugezogen werden, um die Daten in einfacher nutzbare PNG-Dateien umzuwandeln [11].

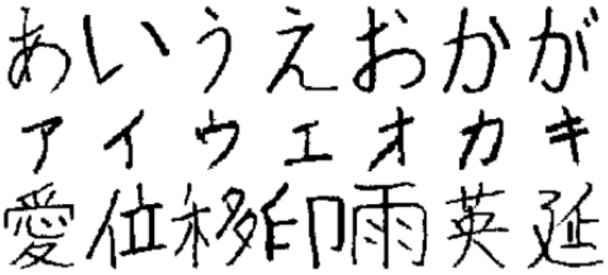

Abbildung 5. Extrahierte und invertierte Kana aus der ETL Character Database [3, S.1].

Nun, wurden aus den umgewandelten Daten verschieden zusammengesetzte Datensätze erstellt. Der erste ist an der Thesis von Charlie Tsai angelehnt und beinhaltet lediglich die den ETL-1 Datensatz mit Katakana und den ETL-8 Datensatz mit Hiragana sowie Kanji [3, S.3]. Der Aufbau von diesem mit Unterteilungen in Hiragana, Katakana, Kanji und sonstiger Zeichen sowie das arithmetische Mittel der Datensätze pro Klasse und deren Standardabweichung lässt sich in der Tabelle I erkennen.

Tabelle I
Aufbau des ersten Datensatz nach Charlie Tsai.

Alphabet	Klassen	Datensätze	Mean	Stddev
Hiragana	75	12.075	161	0
Katakana	48	71.959	1.499	342
Kanji	881	141.841	161	0
Sonstige	0	0	0	0
Gesamt	1.004	225.875	224	295

Wie klar zu sehen ist, sind Katakana im Gegensatz zu Kanji und Hiragana stark überrepräsentiert. Eine Nutzung von Klassengewichtungen, um die unterrepräsentierten Daten hervorzuheben, könnte sich daher als sinnvoll erweisen. Weiterhin stellt sich dies auch für die Katakana als möglicherweise sinnvoll heraus, da diese eine gewisse repräsentative Streuung aufweisen. Da der ETL-1 Datensatz ebenfalls Zeichen mit Beugungen (Dakuten und Handakuten) beinhaltet, besitzt der Datensatz deutlich mehr Klasse für Hiragana als Katakana, obwohl beide Alphabete sehr ähnlich aufgebaut sind. Mit einem Blick auf die originale Arbeit sind hier zwei Probleme zu erkennen. Zum einem hat sich die Anzahl der Datensätze seit dem originalen Paper um ca. 1000 erhöht [3, S.3]. Da hier kein Fehler beim Vorverarbeiten der Daten aufgefallen ist, könnte es sich um ein Hinzufügen von Daten seitens der Verwalter der ETL Character Database handeln, was auch in einer Steigerung der Klasse der Kanji um drei resultiert. Weiterhin finden sich drei Katakanatypen weniger wieder. Dies hängt mit der Kodierung ausgestorbener Zeichen im ETL-1 Datensatz zusammen, welche die Buchstaben *yi* zu *i*, *ye* zu *e* und *wu* zu *u* werden lassen [10]. Hier wurden im Gegensatz zur originalen Arbeit die daraus resultierenden Beispiele zu den passenden Vokalen umgeschrieben [3, S.3]. Eine gleichzeitge Erhöhung der Kanji und Verringerung der Katakana um drei, hat sich nach Prüfungen als

rein zufällig herausgestellt.

Mit diesem ersten Datensatz können einige praktische Probleme in unseren Augen leider nicht gelöst werden. Will man maschinelles Lernen zum Übersetzen von Sprache einsetzen, so ist es nötig die meisten Aspekte dieser Sprache zu berücksichtigen. Heutzutage hat sich das lateinische Alphabet in viele Aspekte der japanischen Sprache etabliert. Eine reine Erkennung japanischer Zeichen ist daher für praktische Anwendungen eher weniger sinnvoll. Auch Satzzeichen und Sonderzeichen wurden im originalen Datensatz nicht berücksichtigt, obwohl diese für ein späteres Verständnis ganzer Sätze essentiell sind. Zuletzt beinhaltet der Datensatz lediglich 881 der 2136 üblich genutzten Kanji [3, S.3]. Da die ETL Character Database noch weitere Datensätze beinhaltet, wurde sich in einer Auswahl an Daten entschieden die gesamte Datenbank zu nutzen. Der Aufbau dieser Gesamtauswahl lässt sich in Tabelle II erkennen.

Tabelle II
Aufbau des zweiten Datensatz als Gesamtauswahl der ETL Character Database.

Alphabet	Klassen	Datensätze	Mean	Stddev
Hiragana	77	67.335	874	437
Katakana	73	147.237	2.017	1.496
Kanji	2.967	782.561	264	83
Sonstige	74	175.308	2.369	1.013
Gesamt	3.191	1.172.441	367	507

Wie klar zu sehen ist beinhaltet die gesamte ETL Character Database noch deutlich mehr Daten, welche für das Maschinelle Lernen genutzt werden können. So konnten die vorliegenden Kanji verdreifacht und die durchschnittlichen Datensätze pro Kanji nahezu verdoppelt werden. Weiterhin sind die Hiragana nicht mehr stark unterrepräsentiert und sonstige Zeichen wie aus dem lateinischen Alphabet oder Satzzeichen können erlernt werden. Eine größere Schwankung in der Durchschnittszahl der Datensätze sowie eine höhere Streuung dieser, macht auch hier wieder eine Nutzung von Klassengewichtungen eine mögliche sinnvolle Ergänzung.

Eine beliebte Möglichkeit das Overfitting von Modellen zu verringern sowie die Toleranz von Daten in der Praxis zu verbessern, ist die sogenannte Data Augmentation. Dabei können durch bestimmten Transformationen aus vorliegenden Daten neue generiert werden. Typische Vorgehensweisen bei Bildern sind dabei das skalieren, verschieben oder rotieren der Daten. Auch eine Spiegelung kann sich bei manchen Bilddaten als sinnvoll erweisen, jedoch würde dies die Bedeutung von Schriftzeichen zerstören. Werden Transformationen miteinander kombiniert, so kann die Anzahl der vorhandenen Daten um ein vielfaches erhöht werden [4, S.311]. Im Code des Projekts können Skalierungen und Rotationen verwendet werden, um die Daten künstlich zu erhöhen. Dies geschieht über Transformationsmatrizen, welche mit der Bibliothek CV2 generiert und angewendet wurden. Eine Generation von 25 verschiedenen Transformationsmatrizen lässt sich in der Funktion im folgenden Listing 1 erkennen.

Listing 1. Python-Code für die Generierung von Transformationsmatrizen

```
def create_transforms(
  format,
  scale_values,
  rot_values):

  if rot_values is None or len(rot_values) == 0:
    rot_values = [-20, -10, 0, 10, 20]

  if scale_values is None or len(scale_values) == 0:
    scale_values = [0.8, 0.9, 1.0, 1.1, 1.2]

  transform_matrices = []
  for rot_value in rot_values:
    for scale_value in scale_values:
      transform_matrix = cv2.getRotationMatrix2D(
        (format[0]/2, format[1]/2),
        rot_value, scale_value)
      transform_matrices.append(transform_matrix)

  return transform_matrices
```

Die Transformationsmatrizen würden bei der beispielhaften Anwendung auf dem Kanji *zwei Stück* die Ausgabe in der Abbildung 6 erzeugen. Es ist klar die Skalierung zwischen den Größen und die Rotation zu erkennen, welche in einer Vergrößerung der Daten um den Faktor 25 resultieren würde.

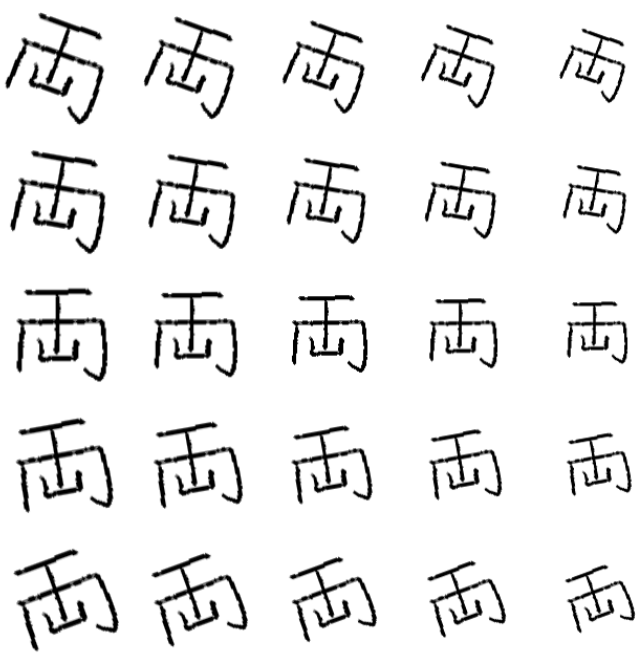

Abbildung 6. Transformationen des Kanji *zwei Stück*.

IV. Eingesetzte Modelle

Im Gegensatz zur originalen Arbeit von Charlie Tsai, wurde sich aufgrund der guten Ergebnisse bei einer gemeinsamen Erkennung aller Alphabete gegen die Nutzung von separaten Modellen für Kanji, Katakana und Hiragana entschieden. Dieses Vorgehen würde ein zusätzliche Modell zur Unterscheidung von Alphabeten benötigen und daher insgesamt vier Modelle nötig machen. Da die Ergebnisse

ConvNet Configurations										
M3	M6-1	M6-2	M7-1	M7-2	M8	M9	M11	M12	M13	M16
3 weight layers	6 weight layers	6 weight layers	7 weight layers	7 weight layers	8 weight layers	9 weight layers	11 weight layers	12 weight layers	13 weight layers	16 weight layers
input (64×64 gray-scale image)										
	conv3-32 conv3-32	conv3-64	conv3-64	conv3-64	conv3-32 conv3-32	conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-32 conv3-32	conv3-64 conv3-64
	maxpool									
	conv3-64 conv3-64	conv3-128	conv3-128	conv3-128	conv3-64 conv3-64	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-64 conv3-64	conv3-128 conv3-128
	maxpool									
		conv3-512	conv3-512 conv3-512	conv3-192	conv3-128 conv3-128	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-128 conv3-128	conv3-256 conv3-256 conv3-256
		maxpool								
				conv3-256		conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv3-512	conv3-256 conv3-256	conv3-512 conv3-512 conv3-512
				maxpool		maxpool				
									conv3-512 conv3-512	conv3-512 conv3-512 conv3-512
									maxpool	
FC-5000 FC-5000 FC-5000	FC-256	FC-4096 FC-4096	FC-4096 FC-4096	FC-1024 FC-1024	FC-1024	FC-4096 FC-4096	FC-1024 FC-1024	FC-4096 FC-4096	FC-1024 FC-1024	FC-4096 FC-4096
FC-n_{classes}										
softmax										

Abbildung 7. Genutzte VGGNet Architekturen im Paper von Charlie Tsai [3, S.3].

für der Erkennung aller Alphabete laut Tsais Arbeit bereits bei sehr hohen 99,53% in den Testfällen lagen, wurde sich gegen einen separaten Ansatz entschieden, um den Fokus auf die Erkennung aller Zeichen zu legen [3, S.4-6]. Aufgrund der teilweise hohen Streuung der Häufigkeiten von Datensätzen, wurden Ansätze sowohl mit als auch ohne Klassengewichtungen erprobt.

Während des Lernvorgangs wurden die Daten in 80% Trainings- und jeweils 10% Validierungs- und Testdaten geteilt. Alle Modelle wurden mit einem Adam-Optimizer trainiert, welcher auch in Tsais Arbeit genutzt wurde [3, S.4]. Die Startlernrate wurde dabei von Modell zu Modell verschieden angepasst. Die Genauigkeit und der Verlust auf den Validierungsdaten wird mit jeder Epoche berechnet und dient als grobes Maß des Fortschritt während des Trainings. Nach dem Training wird eine Übersicht und der gewichtete sowie ungewichtete Durchschnitt des F1-Scores aller Klassen auf den Validierungsdaten als auch für einen späteren Vergleich zwischen den Modellen auf den Testdaten gebildet.

A. VGGNet

Der Architektur der VGGNet wurde in Simonyans und Zissermans Paper *Very Deep Convolutional Networks for Large-Scale Image Recognition* entworfen und konnten 2015 state-of-the-art Genauigkeiten mit einem relativ einfachen sowie flexiblen Aufbau Konvolutionaler Neuronaler Netze erreichen [12, S.1]. Sie arbeiten typischerweise mit einem Wahrnehmungsfeld von 3 x 3 Elementen in den konvolutionalen Schichten. Weiterhin nutzen sie ein 2 x 2 Max-Pooling mit einem Stride von 2 nach einigen aber nicht jeder Schicht. Danach folgen 2 Fully-Connected Schichten mit 4096 Neuronen und schließlich eine Fully-Connected Schicht mit Neuronen in Höhe der Anzahl der vorliegenden Klassen [12, S.2].

In Tsais Arbeit wurden 11 VGGNet artige Netze umgesetzt, welche in der Tabelle 7 nach Netzgröße geordnet dargestellt werden. Diese weisen einige Veränderungen in der Struktur auf im Vergleich zum originalen VGGNet Paper auf. Zum einen liegen die Daten nur mit einem 64 x 64 Pixel Grauwertkanal vor [3, S.3]. Dadurch wird die Größe der Eingabedaten im Gegensatz zum originalen VGGNet Paper mit 224 x 224 Pixel RGB-Bildern sehr viel kleiner [12, S.2]. Weiterhin wurde die Anzahl der Neuronen und Schichten im Fully-Connected Teil von Modell zu Modell angepasst [3, S.3]. Aufgrund der guten Ergebnisse der Klassifikation in der originalen Arbeit, wurde ein besonderer Fokus auf die Modelle M7-1, M7-2, M9 und M12 gesetzt [3, S.5]. Alle diese Netzen wurden nach Tsai mit einem Dropout versehen, welcher allerdings aufgrund Probleme bei höheren Raten auf 25% verringert wurde und nur nach jeder Gruppe von Schichten platziert wurde [3, S.2]. Zusätzlich musste auf eine Batch-Normalisierung nach jeder Schicht verzichtet werden, da diese Probleme im Zusammenhang mit dem Dropout verursacht und die Ergebnisse stark verschlechtert hat [3, S.4]. Dank der direkten Unterstützung der Keras Applications API und der Möglichkeit des Transfer Learnings von *imagenet* Gewichtungen, wurden zusätzlich die von Keras implementierten VGG16 und VGG19 Netze erprobt [13].

B. ResNet

Die ResNet sind spezielle Architekturen Neuronaler Netze, welche mittels Restdarstellungen (Residual Representations) durch Abkürzungsverbindungen (Shortcut Connections) die Performanz tiefer Neuronaler Netze und im besonderen Konvolutionaler Netze verbessern sollen [14, S.1-2]. Dabei wird eine Ausgabe welche an einem bestimmten Punkt im Netzwerk ausgegeben wurde, an einem anderen Punkt identisch vor der Aktivierungsfunktion in die Ausgabe der tieferen Schicht eingerechnet [14, S.4]. Diese Funktionsweise lässt sich in der Abbildung 8 am Beispiel zweier Schichten erkennen.

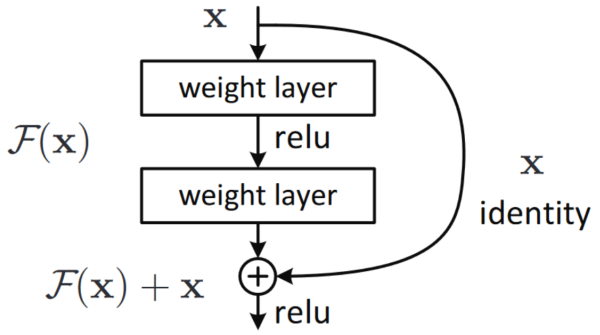

Abbildung 8. Beispiel einer Abkürzungsverbindung über zwei Schichten [14, S.2].

Ein solches Vorgehen soll den bei tiefen Neuronalen Netzen beobachteten Effekt des Zerfalls von Genauigkeiten bei einer zunehmenden Steigerung der Schichtenzahl bekämpfen [14, S.1]. Keras bietet in der Applications API für diese Netzarchitekturen sechs vordefinierte Modelle, welche 50, 101 bzw. 152 Schichten aufweisen [15]. Im besonderen wurde beim Trainieren ein Fokus auf die V2 Varianten dieser Modelle gesetzt, bei welchen es sich um eine Verbesserung des originalen Ansatz der ResNet handelt [16].

C. Vision Transformer

Vision Transformer wurden in vier verschiedenen Varianten erprobt, welche im Vision Transformer Paper dargestellt wurden. Es wurden dabei Modelle mit 12 (Base), 24 (Large) und 32 (Huge) Schichten vorgestellt, wobei die größte Variante aufgrund der wahrscheinlich nötigen Datenmenge nicht erprobt wurde. Alle Varianten der VIT mit ihren Parametern lassen sich in der Tabelle III erkennen. [7, S.5]. Die Base- und Large-Varianten wurden in der Version 16 und 32 erprobt, bei welcher es sich um die Dimensionen des Input Patchs handelt. Bei der Version 16 ist dieser 16x16 Pixel, während die Größe bei der Version 32 bei 32x32 Pixeln liegt [7, S.5]. In der Praxis wurden dieses über das Pythonpaket vit-keras, welches eine einfach und ähnlich zur Keras Applications API nutzbare Definition der ViT-Modelle ermöglicht. In diesem wurden alle nötigen Varianten implementiert und die Möglichkeit offen gehalten vorerlernte ImageNet-Gewichtungen zu laden [17]. Hier musste allerdings der Multilayer Perceptron Kopf der Vision Transformer selber definiert werden, da sonst Probleme mit der Ausgabe des Netzes auftraten.

Tabelle III
Vordefinierte Varianten der Vision Transformer Modelle [7, S.5].

Modell	Schichten	Größe D	MLP Größe	Parameter
Base	12	768	3072	86M
Large	24	1024	4096	307M
Huge	32	1280	5120	632M

V. Ergebnisse

Tabelle IV
Ergebnisse der VGGNet Modelle in Prozent.

Modell	Datensatz	Val-Acc	Test-Acc	Val-F1	Test-F1
M7-1	Tsai	0	0	0	0
M7-2	Tsai	0	0	0	0
M9	Tsai	0	0	0	0
M12	Tsai	0	0	0	0
VGG16	Tsai	0	0	0	0
VGG19	Tsai	0	0	0	0
M7-1	Full	0	0	0	0
M7-2	Full	0	0	0	0
M9	Full	0	0	0	0
M12	Full	0	0	0	0
VGG16	Full	0	0	0	0
VGG19	Full	0	0	0	0

Tabelle V
Ergebnisse der ResNet Modelle in Prozent.

Modell	Datensatz	Val-Acc	Test-Acc	Val-F1	Test-F1
50V2	Tsai	0	0	0	0
101V2	Tsai	0	0	0	0
151V2	Tsai	0	0	0	0
50V2	Full	0	0	0	0
101V2	Full	0	0	0	0
151V2	Full	0	0	0	0

Tabelle VI
Ergebnisse der ViT Modelle in Prozent.

Modell	Datensatz	Val-Acc	Test-Acc	Val-F1	Test-F1
b16	Tsai	0	0	0	0
b32	Tsai	0	0	0	0
l16	Tsai	0	0	0	0
l32	Tsai	0	0	0	0
b16	Full	0	0	0	0
b32	Full	0	0	0	0
l16	Full	0	0	0	0
l32	Full	0	0	0	0

VI. Fazit

Literatur

[1] F. S. Institute. Foreign language training. [Online]. Available: https://www.state.gov/foreign-language-training/
[2] B. von Japan in Deutschland. Feature - kennen sie kanji? - zur japanischen schrift. [Online]. Available: https://www.de.embjapan.go.jp/feature/kanji.html

[3] C. Tsai, "Recognizing handwritten japanese characters using deep convolutional neural networks." [Online]. Available: http://cs231n.stanford.edu/reports/2016/pdfs/262_Report.pdf
[4] A. Geron, *Praxiseinstieg: Machine Learning mit Scikit-Learn & TensorFlow*. O'Reilly.
[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762
[6] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016. [Online]. Available: https://arxiv.org/abs/1409.0473
[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020. [Online]. Available: https://arxiv.org/abs/2010.11929
[8] P. U. Stanford Vision Lab, Stanford University. Imagenet. [Online]. Available: https://www.image-net.org/
[9] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," *CoRR*, vol. abs/1707.02968, 2017. [Online]. Available: http://arxiv.org/abs/1707.02968
[10] N. I. of Advanced Industrial Science, J. E. Technology, and I. T. I. Association. About the etl character database. [Online]. Available: http://etlcdb.db.aist.go.jp/
[11] ——. File formats and sample script. [Online]. Available: http://etlcdb.db.aist.go.jp/file-formats-and-sample-unpacking-code
[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1409.1556e
[13] Keras. Vgg16 and vgg19. [Online]. Available: https://keras.io/api/applications/vgg/
[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/pdf/1512.03385.pdf
[15] Keras. Resnet and resnetv2. [Online]. Available: https://keras.io/api/applications/resnet/
[16] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," 2016. [Online]. Available: https://arxiv.org/pdf/1603.05027.pdf
[17] F. Morales. vit-keras. [Online]. Available: https://github.com/faustomorales/vit-keras