Convolutional Neural Network für die „Urban Sound Classification“

Pascal Stoppa

Technische Hochschule Mittelhessen
pascal.stoppa@me.thm.de

Zusammenfassung

Diese Arbeit handelt von der Klassifikation urbaner Geräusche mit Convolutional Neural Networks (CNNs). Die Grundlage ist der UrbanSound8K-Datensatz, welcher in 10 Geräuschklassen wie „Jackhammer“, „Siren“ oder „Street Music“ unterteilt ist. Um den Datensatz in das CNN zu übergeben werden Mel-Spektrogramme verwendet, wodurch charakteristische Merkmale erkannt werden können. Das Modell orientiert sich an einer bestehenden Implementierung (vgl. [1]) und besteht aus mehreren Faltungs- und Pooling- sowie Dropout-Schichten, womit eine Genauigkeit der Validierungsdaten von 79 % erreicht werden konnte. Für die tatsächliche Anwendung des Modells wurde eine Oberfläche erstellt, mit der auch selbst aufgenommene Tonspuren klassifiziert werden können. Die erzielten Ergebnisse weisen auf ein großes Potenzial für die Anwendung von CNNs in der urbanen Geräuscherkennung hin, welches mit Optimierungen in der Modellarchitektur und Datenaufbereitung ausgeschöpft werden könnte.

1 Einleitung

Um dem Ziel von Smart Cities einen Schritt näher zu kommen, spielt die automatisierte Erkennung von urbanen Geräuschen eine große Rolle. Die Anwendung könnte in Bereichen wie Verkehrsüberwachung, öffentliche Sicherheit und Lärmanalyse einen wichtigen Beitrag leisten. Methoden, die sonst zur Audiosignalverarbeitung zum Einsatz kommen, stoßen schnell an ihre Grenzen, wenn es um komplexe und gemischte Akustiksignale geht. Die Kombination von großen Datenmengen und Convolutional Neural Networks (CNNs) hat im Bereich der Mustererkennung von Audiosignalen bereits große Fortschritte gemacht.

Ziel dieser Arbeit ist es, das Modell von Min Yang [1] auf dem lokalen Rechner zu trainieren, welches mit Hilfe von Mel-Spektrogrammen in der Lage ist die zehn verschiedenen Klassen des UrbanSound8K-Datensatzes mit einer möglichst hohen Genauigkeit zu klassifizieren. Zu Beginn wird etwas Theorie zur Audiosignalverarbeitung und zu CNNs erläutert. Diese ist für das Verständnis des Modells unerlässlich. Danach wird die Programmierumgebung eingerichtet, die Daten aufbereitet und das Modell aufgebaut und trainiert. Nach der Beurteilung der Ergebnisse werden Möglichkeiten der Genauigkeitsoptimierung ausgearbeitet, angewandt und bewertet. Das Modell mit der höchsten Genauigkeit wird dann in eine Klassifikationsoberfläche eingebaut, um es mit eigenen Tonaufnahmen testen zu können.

2 Theorie

2.1 Audiosignalverarbeitung

Wir stehen vor der Aufgabe, Maschinen beizubringen, Audiosignale zu verstehen. Um das möglich zu machen, muss das rohe Signal aus dem Zeitbereich in eine Form übergeführt werden, in der charakteristische Merkmale wie Tonhöhe oder Klangfarbe gut zu erkennen sind. Diese Merkmale

sind dem Frequenzbereich zu entnehmen. Davor muss zunächst geklärt werden, wie aus einem analogen ein digitales Audiosignal entsteht und welche Kenngrößen dafür wichtig sind.

Für die Konvertierung werden bei einem kontinuierlichen Audiosignal $x(t)$ (Amplitude über die Zeit) zu diskreten Zeitpunkten Messwerte aufgenommen. Die Frequenz, mit der die Messwerte abgetastet werden, wird Abtastrate oder Sampling Rate f_{sr} genannt und gehört zu den wichtigsten Kenngrößen der Audiosignalverarbeitung. Sie ist essenziell dafür, Frequenzen korrekt darzustellen. Menschen sind in der Lage, Frequenzen bis $20\,kHz$ wahrzunehmen. Um diese Frequenzen in einem digitalen Audiosignal festhalten zu können, muss nach dem Nyquist-Shannon-Abtasttheorem

$$f_{Nyquist} = \frac{1}{2} f_{sr} \tag{1}$$

die Abtastrate mindestens doppelt so groß sein, damit die $20\,kHz$ wahrgenommen werden können. Höhere Frequenzen als die Nyquist-Frequenz werden niederfrequent dargestellt, wodurch das Signal verfälscht wird [2]. Dieser Effekt wird Aliasing genannt und sollte vermieden werden. Aus diesem Grund ist eine gängige Sampling Rate für Audiosignale $44,1\,kHz$. Damit können Frequenzen bis $22,05\,kHz$ wahrheitsgemäß dargestellt werden [2].

Für die Detektion der charakteristischen Merkmale muss das digitale Audiosignal nun in den Frequenzbereich übertragen werden. Hierfür wird die Fourier-Transformation

$$F(f) = \int_{-\infty}^{\infty} f(t) e^{-i2\pi ft} dt \tag{2}$$

verwendet, um das Signal in einzelne Frequenzanteile aufzuteilen [2]. Hierbei entspricht $F(f)$ der Fourier-Transformierten und $f(t)$ dem Zeitsignal. Die diskrete Fourier Transformation ist aufgrund der Doppelschleife extrem ineffizient, weshalb auf die „Fast Fourier transform“ (FFT) zurückgegriffen wird, welche die DFT in einem Bruchteil der Zeit ausführt.

2.2 Mel-Spektrogramme

Das Grundproblem der FFT liegt darin, dass sie die Frequenzen eines gesamten Signals erfasst und der zeitliche Bezug komplett verloren geht. Dieser ist essenziell für die Sound Klassizierung und bleibt mit der „Short-time Fourier transform“(STFT) erhalten, indem eine fensterweise Durchführung der DFT bzw. FFT gemacht wird. Dadurch wird eine zweidimensionale Zeit-Frequenz-Repräsentation erzeugt, die sich gut für ein CNN eignet. Diese Repräsentation wird Spektrogramm genannt. Für jedes Segment wird dann eine separate DFT nach

$$X(n, \lambda) = \sum_{m=-\infty}^{\infty} x[n+m] w[m] e^{-i\lambda m} \tag{3}$$

durchgeführt, wobei n den Zeitindex und λ den Frequenzindex darstellt [3]. w beschreibt eine Fensterfunktion (meistens ein Hann-Fenster), wodurch scharfe Übergang an den Segmentgrenzen vermieden werden [3]. Dadurch entsteht eine $n \times \lambda$-Matrix mit typischerweise Graustufen. Um die menschliche Hörwahrnehmung noch besser darstellen zu können, wird in das Spektrogramm die Mel-Skala eingebracht. Sie bezieht das logarithmische Hörempfinden der Menschen mit ein und kann mit

$$H(f) = 2595 \cdot log_{10}(1 + f/700) \tag{4}$$

approximativ beschrieben werden [4]. Diese Skalierung sorgt für bessere Ergebnisse bei der Sound Klassifikation.

2.3 Convolutional Neural Networks für Audiodateien

Convolutional Neural Networks (CNN) stellen eine Sonderform von künstlichen neuronalen Netzen dar. Die Bezeichnung convolutional (dt. Faltung) bezieht sich auf eine mathematische lineare Operation, die es dem Netz ermöglicht, Muster und Strukturen zu erkennen [5]. Aus diesem Grund eignen sich CNNs gut für die Bildverarbeitung. In diesem Fall speziell für die Analyse von Mel-Spektrogrammen. Mathematisch lässt sich der Faltungsvorgang im eindimensionalen Fall durch folgende Gleichung beschreiben:

$$s[i] = \sum_{m=-(k-1)/2}^{+(k-1)/2} I[i+m] \cdot K[m] \tag{5}$$

Hier enspricht I dem Input, welcher mit einem Faltungskernel K der Breite k multipliziert wird [5]. So ergibt sich eine gewichtete Summe von benachbarten Werten aus dem Input. Das Ergebnis der Faltung ist die Feature Map s. Während des Trainings werden die Gewichte des Faltungskernels (auch Filter genannt) trainiert, wodurch sich Muster erkennen lassen. Um verschiedende Charakteristiken von Inputs herauszuarbeiten, werden in der Realität viele Kernels verwendet. Das lässt den Featurespace pro Faltungslayer stark anwachsen. Es muss also eine Methode her, wie sich die wichtigsten Charakateristiken verallgemeinern lassen, um den Featurespace zu verkleinern. Hier kommt das Pooling ins Spiel. Die am häufigsten verwendete Methode ist das Max-Pooling. Dabei wird der Output in Abschnitte der Breite z unterteilt, wovon immer der größte Wert übernommen wird, wodurch der Featurespace klein gehalten wird [5].

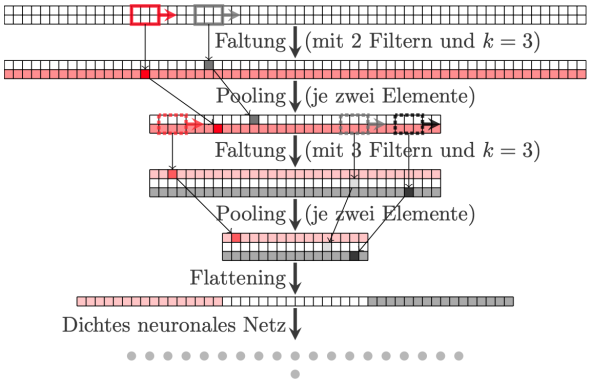

Abbildung 1: Aufbau eines 1D Convolutional Neural Network [5]

Durch die Abfolge von mehreren Convolutional- und Pooling-Layern ist es möglich sehr tiefe Netze zu bauen, die sonst zum Overfitting neigen. Das Stapeln von Convolutional- und Pooling-Layern führt zu einer Hierarchie von Merkmalen. Während frühere Schichten einfache Merkmale wie Kanten oder Übergänge erkennt, erkennen tiefere Schichten komplexere Muster [5]. Ein weiterer wichtiger Schritt ist das Flattening. Der letzte Output-Layer der letzten CNN-Schicht kann je nach Anwendungsfall verschiedene Dimensionen haben. Da dieser in ein Fully-connected dense Layer gespeist werden soll, muss der Output-Layer in einen Vektor umgewandelt werden.[5] Im Fully-connected Layer findet die eigentliche Klassifikation statt.

2.4 Datensatz „Urban Sound Classification“

Urban Sound Classification ist ein wachsendes Feld, für das es nur wenig Daten gibt. Mit diesem Hintergrund haben Forschende der Ney York University (NYU) den Datensatz UrbanSound8K [6] erstellt. Er besteht aus 8732 gekennzeichneten Sounddateien ($\leq 4\,s$) im Waveform Audio File Format (WAVE). Der Datensatz ist in 10 verschiedene Klassen unterteilt: „Air Conditioner“, „Car Horn“, „Children Playing“, „Dog Bark“, „Drilling“, „Engine Idling“, „Gun Shot“, „Jackhammer“, „Siren“und „Street Music“[7]. Neben den Sounddateien, welche in 10 ungefähr gleich große Unterordner unterteilt sind, liegen dem Datensatz auch noch Metadaten in Form von einer CSV-Datei vor. In dieser Datei werden den Sounddateien die Klassen-IDs und die zugehörige Klasse zugeteilt. Zusätzlich sind dort die Start- und Endzeiten der Aufnahme, die Nummer des Unterordners und der Name der Sounddatei dokumentiert.

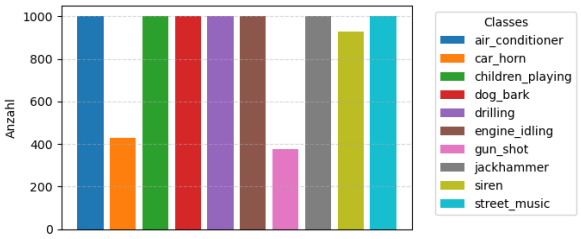

Abbildung 2: Klassenverteilung des Urban Sound Datensatzes

3 Umsetzung

Für die Umsetzung des Projekts wurde sich für folgendes Vorgehen entschieden:

1. Python-Umgebung einrichten
2. Datensatz herunterladen
3. Implementierung CNN (vgl. Min Yang [1])
4. „Mini“-Oberfläche zur Klassifikation programmieren

5. Datengrundlage erweitern

6. Modellaufbau anpassen

3.1 Einrichtung der Umgebung

Das Projekt wurde auf einem Apple MacBook Pro M1 2020 mit $8\,GB$ Arbeitsspeicher umgesetzt. Die verwendete IDE ist Visual Studio Code mit der Version 1.83.1. Das Ziel der ersten Teilaufgabe besteht darin, eine Python-Umgebung einzurichten, mit der das Jupyter-Notebook von Min Yang [1] auf dem lokalen Rechner läuft. Zuerst wurde ein Git-Repository erstellt, in dem das Projekt verwaltet wird [8]. Um Komplikationen aufgrund von verschiedenen Versionen zu vermeiden, wurde eine Requirements-Datei erstellt, welche die Versionen der Pakete beinhaltet. Mit Hilfe des Paketmanagers Conda und der Requirements-Datei konnte eine virtuelle Umgebung mit den Paketversionen erstellt werden, die auch im Original-Notebook verwendet wurden. Diese Umgebung ist als Conda-Environment-Datei (.yml) im Git-Repository zu finden, wodurch der Code von jedem ohne Paket-Komplikationen ausgeführt werden kann [8]. Einige Versionen von Paketen sind allerdings zu alt und dementsprechend nicht mehr mit Conda kompatibel, weshalb einige Pakete mit neueren Versionen installiert werden mussten. Das führte dazu, dass sich anstatt von Python 3.6.6 für Python 3.10.18 entschieden wurde. Neben ein paar Befehlsänderungen, welche versionsbedingt sind, führte diese Änderung zu keinen Komplikationen.

Im nächsten Schritt geht es um die Datenbeschaffung. Hierfür wurde eine Bash-Routine erstellt, die den UrbanSound8K Datensatz mit einer Python-API heruterlädt und im neuerstellten „data“-Ordner ablegt, wodurch die Ordnerstruktur einheitlich bleibt [6]. Genauere Informationen sind der README.md Datei des Git-Repositories zu entnehmen [8].

3.2 Datenaufbereitung

Das Notebook von Min Yang [1] beginnt zunächst mit dem Import aller wichtigen Pakete. Dazu gehören Pakete wie TensorFlow, scikit-learn, NumPy, Librosa und Pandas. Der Datensatz des Original-Notebooks ist schon vorher in Trainings- und Test-Splits unterteilt und beinhaltet keine Unterordner. Aus diesem Grund muss der Datensatz zu Beginn schon in Trainings- und Test-Splits aufgeteilt werden, damit die weitere Vorgehensweise des Original-Notebooks beibehalten werden kann. Hierbei wurde die Trainingsgröße auf $62,24\,\%$ geschätzt. Um die Reproduzierbarkeit sicherzustellen, wurde der Random State des „train_test_split“-Befehls fixiert. Anschließend wurde der Index der Trainings- und Test-Daten zurückgesetzt, wodurch der Datenimport trotz der anderen Ordnerstruktur auf den Stand des Original-Notebooks gebracht wurde.

Nun werden beispielhaft jeweils ein Sample aus den Klassen Sirene und Straßenmusik geladen, um sich auf Abtastraten und Audio-Längen festzulegen. Für dieses Modell wurde sich im Original-Notebook [1] für eine Audiolänge von $2,97\,s$ und eine Sampling-Rate von $22050\,Hz$ entschieden. Damit können nach dem Nyquist-Shannon-Theorem Frequenzen bis ca. $11\,kHz$ ohne Aliasing dargestellt werden, wodurch Rechenzeit eingespart wird [2]. Mit dieser Sampling-Rate, dieser Audiolänge und Standartwerten der Librosa Bibliothek ergeben sich für die Mel-Spektrogramme die Dimensionen 128×128. Das bedeutet 128 Mel-Bänder (Frequenzbereiche) und 128 Zeitschritte. Die Input Shape für das CNN ist also $128 \times 128 \times 1$, wobei die 1 für die Anzahl an Kanälen steht. Da das Mel-Spektrogramm in Graustufen analysiert wird, bestehen die Inputdaten nur aus einem Kanal.

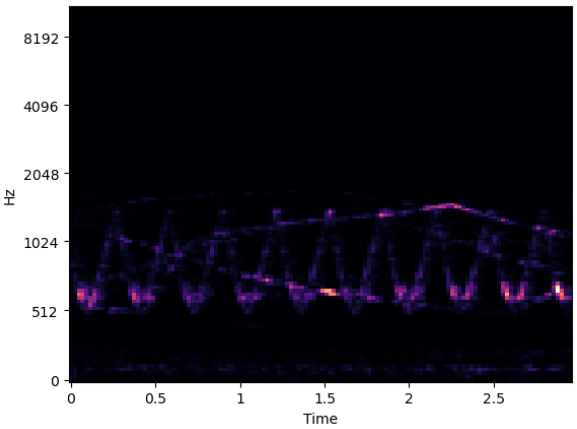

Abbildung 3: Mel-Spektrogramm Sirene

Das Mel-Spektrogramm der Sirene in Abbildung (3) zeigt, dass tatsächlich charakteristische Merkmale der Audiodatei kenntlich gemacht werden, die im Amplitudensignal verborgen geblieben wären. Besonders auffällig sind die zwei sägezahnartigen Modulationen im mittleren Frequenzbereich, die auf Sirenen mit unterschiedlichen Frequenzen hindeuten.

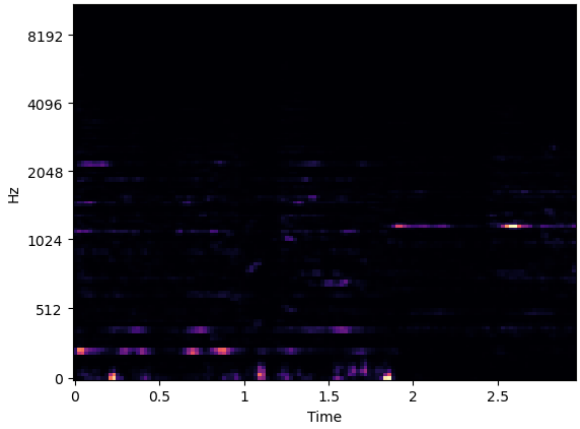

Abbildung 4: Mel-Spektrogramm Straßenmusik

In Abbildung (4) ist hingegen ein Mel-Spektrogramm einer Straßenmusikaufnahme zu sehen. Hier sind auf den ersten Blick keine klaren Muster zu erkennen. Beim genaueren Hinschauen fällt auf, dass über die gesamte Länge der Audiodatei bei gewissen Frequenzen konstante Anteile vorhanden sind, welche zusätzlich ganzzahlige Vielfache voneinander sind. Dies ist ein Hinweis für musikalische Signale in der Audiodatei.

Während diese Unterschiede für das menschliche Auge gut zu erkennen sind, stellt sich nun die Frage, wie ein Convolutional Neural Network trainiert werden muss, um solche Muster eventuell noch besser zu identifizieren und zu klassifizieren.

3.3 Modellarchitektur und Training

Bevor der Modellaufbau beginnen kann, muss den Metadaten noch der Pfad zur Audiodatei hinzugefügt werden. Damit können die Audiodateien geladen und deren Mel-Spektrogramm in die Listen „train_audio_data“ und „test_audio_data“ gespeichert werden, wodurch sie die Form eines 4D-Arrays mit (Anzahl-Bilder, Höhe, Breite, Klasse) annehmen. In diesem Schritt werden die Mel-Spektrogramme auf einheitliche Größe getrimmt bzw. erweitert. Während „train_audio_data“ durch erneute Train-/Test-Split Aufteilung zum Modelltraining und -validierung verwendet wird, dient „test_audio_data“ lediglich der finalen Modellvorhersage. Indem das Modell komplett neue Daten sieht, kann die Generalisierungsfähigkeit des Modells geprüft werden. Von dem neuen Test-Split werden erneut 30 % für den Validierungssplit verwendet, wodurch sich der Datensatz folgendermaßen mit den dazugehörigen Klassen y_train, y_test und y_val aufteilt:

<table>
<tr><th colspan="3">train_audio_data</th><th>test_audio_data</th></tr>
<tr><td colspan="3">62,24 %</td><td>37,76 %</td></tr>
<tr><td>X_train
90 %</td><td colspan="2">X_test
10 %</td><td></td></tr>
<tr><td></td><td>X_test
70 %</td><td>X_val
30 %</td><td></td></tr>
<tr><td>56,16 %</td><td>4,36 %</td><td>1,87 %</td><td>37,76 %</td></tr>
</table>

Tabelle 1: Aufteilung in Trainings-, Test- und Validierungs-Split

Das CNN besteht aus mehreren Convolution- und Pooling-Schichten, um aus den zweidimensionalen Eingangsdaten (Mel-Spektrogramme) charakteristische Merkmale zu erkennen und zu klassifizieren. Zuerst gehen die Eingangsdaten mit der Größe 128×128 in die erste Schicht, bestehend aus 24 Kernels mit der Größe 5×5. Um Overfitting zu vermeiden, wird danach eine Max-Pooling-Schicht eingebaut, welche die Auflösung reduziert gefolgt von einer ReLu-Aktivierungsfunktion. Dieser Aufbau wiederholt sich nun mit 48 Kernels. Nach der dritten Schicht (auch mit 48 Kernels) folgt keine Pooling-Schicht. Stattdessen wird nach der Aktivierungsfunktion eine Flattening-Operation eingebaut, um die zweidimensionalen Feature-Maps zu einem eindimensionalen Vektor umzuformen. Um die Generalisierungsfähigkeit des Modells zu verbessern, wird danach eine Dropout-Schicht mit einer Rate von 0,5 eingesetzt. Diese Schicht deaktiviert während des Trainings 50 % der Neuronen, wodurch das Netz gezwungen wird, robustere Merkmale zu lernen. Nun folgen zwei Fully-

connected Layer, wobei die letzte Schicht aus so vielen Neuronen besteht wie es Klassen gibt. Die zehn Neuronen werden dann von der Softmax-Aktivierungsfunktion

$$\sigma_j(z) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}},\ j = 1...n \tag{6}$$

auf 1 normiert, wodurch jedem Neuron eine Wahrscheinlichkeit zugeordnet wird [5]. Das Ergebnis der Klassifikation ist die Klasse mit dem höchsten Wahrscheinlichkeitswert.

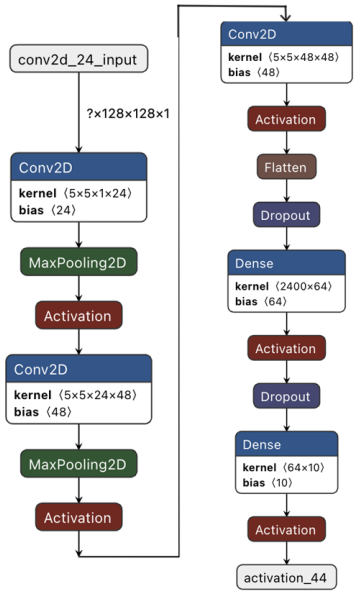

Abbildung 5: Aufbau des Modells

Eine Konvergenzanalyse zeigt, dass die Genauigkeit des Modells für die Validierungsdaten nach ungefähr 27 Epochen gegen 79 % konvergiert. Verglichen mit den Ergebnissen von Moritz Egelkraut [9], die bei 78 % für die Validierungsdaten liegen, liefert diese Implementierung vergleichbare Ergebnisse. Die Genauigkeit mit den Trainingsdaten (mit der Evaluierungs-Funktion der Keras API) liegt hier bei 88 % und mit den Testdaten bei 79 %. Dieser Offset zwischen Trainings- und Test-Split, welcher auch im Loss-Plot in Abbildung (6) zu erkennen ist, ist ein Indiz dafür, dass das Modell zum Overfitting neigt. Das ist daran zu erkennen, dass sobald das Modell neue Daten sieht, die Genauigkeit sinkt. Für viele Anwendungen sind diese Ergebnisse ausreichend. Zieht man in Betracht, dass CNNs in anderen Bereichen Genauigkeiten nahe der 100 % erzielen, besteht in der vorgestellten Implementierung Verbesserungspotenzial [9].

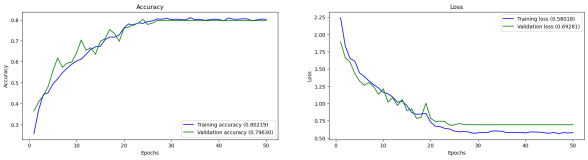

Abbildung 6: Netzgenauigkeit (50 Epochen)

In Abbildung (7) ist der Verlauf der Klassifizierungsgenauigkeit zu sehen. Hier sticht einem sofort der recht große Offset vom Trainings- und Validierungs-Split ins Auge, wobei die Genauigkeit des Validierungs-Splits höher ist. Dieser lässt sich nicht mit Overfitting begründen. Der wahrscheinliche Hauptgrund für den Offset ist die Dropout-Operation, welche während des Trainings 50 % der Neuronen deaktiviert. Da diese Funktion beim Validieren nicht aktiviert ist, ist das Modell stabiler, weil alle Neuronen aktiv sind.

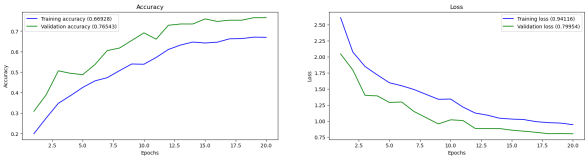

Abbildung 7: Netzgenauigkeit (20 Epochen)

Abbildung (8) zeigt die Konfusionsmatrix der 3298 Testdaten test_audio_data, die nicht für das Training verwendet wurden (vgl. Abbildung 1). Sie ist ein gutes Maß für die Performance des Modells, wenn es neue Daten bekommt. Dabei ist auf der Diagonalen die Anzahl der richtig klassifizierten Beispiele pro Klasse zu sehen. Diese hebt sich von den Nebendiagonalen stark ab, was ein Indiz für eine hohe Genauigkeit des Modell ist. Die Klassen „Car Horn“ und „Gun Shot“ sind nur ungefähr 40 % so stark vertreten wie die anderen Klassen. Dies liegt daran, dass für diese Klassen lediglich $30-40$ % der Datenmenge vorhanden sind, wie für die anderen (vgl. Abbildung 2). Zudem wird deutlich, dass die Klassen „Children Playing“ und „Dog Bark“ insgesamt 87 Mal verwechselt wurden. Das könnte daran liegen, dass sich diese beiden

Klassen in der Tonhöhe ähneln. Des Weiteren wurde „Air Conditioner“ 34 Mal fälschlicherweise als „Street Music“ klassifiziert, was wahrscheinlich den Hintergrund hat, dass die Straßenmusik meistens ein recht starkes Hintergrundrauschen hat, nämlich Menschenmengen. Dieses Hintergrundrauschen hat im Mel-Spektrogramm möglicherweise Ähnlichkeiten mit der Klimaanlage.

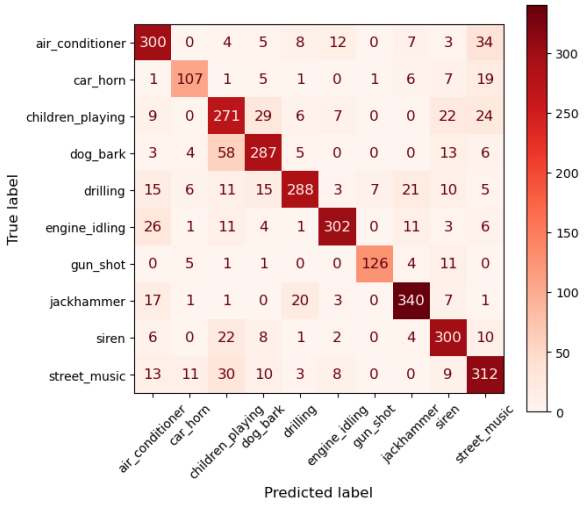

Abbildung 8: Konfusionsmatrix

3.4 Erweitern und verbessern

Im nächsten Schritt wird versucht, die Genauigkeit des Modells durch eine geeignete Datenerweiterung zu verbessern. Die Betrachtung der Klassenverteilung in Abbildung (2) zeigt deutlich, dass von den Klassen „Car Horn“ und „Gun Shot“ ca. 60 % weniger Daten zur Verfügung stehen. Dieser Datenmangel hat auch einen Einfluss auf die Klassifikation, wie in Abbildung (8) zu sehen ist. Die Klasse „Car Horn“ wird in mehr als 30 % der Fälle falsch klassifiziert. Hierfür wurde der Datensatz „HornBase - A Car Horns Dataset“[10] in den bestehenden Datensatz eingearbeitet. Dieser Datensatz besteht aus 1080 Audiodateien mit „Car Horn“-Sounds aus Verkehrsumgebungen. Da der Datensatz eigentlich der Klassifizierung dient, ob es sich um eine Hupe handelt oder nicht, wird die Hälfte aussortiert, die keine Hupen enthält, wodurch 540 verwendbare Audiodateien übrig bleiben. Für die Erweiterung der „Gun Shot“-Daten wird das „Gunshot audio dataset“ [11] verwendet. Es enthält 851 Audiodateien, eingeteilt ist neun Klassen, in denen mit verschiedenen Arten von Waffen geschossen wird. Wichtig zu erwähnen ist, dass dieser Datensatz nicht in einer urbanen Umgebung aufgenommen wurde. Dennoch wird von diesem Datensatz Gebrauch gemacht, um herauszufinden, wie sich das Modell verhält. Im Original-Notebook wird die Einordnung der Audiodateien in „Folds“ durch die zufällige Train-/Test-Aufteilung der gesamten Metadaten nicht berücksichtigt. Aus diesem Grund werden die „Car Horn“-Daten im neuen Ordner „fold11“ und die „Gun Shot“-Daten in „fold12“ gespeichert. Abschließend müssen noch die Metadaten der Erweiterung erstellt und auf die Struktur des ursprünglichen Datensatzes angepasst werden. Zudem wurden die 851 Audiodateien des „Gun Shot“-Datensatzes auf 626 Dateien getrimmt, um insgesamt auf ca. 1000 Dateien pro Klasse zu kommen. Danach wurde das Modell erneut trainiert.

	ohne Erweit.	mit Erweit.
Acc. Train	88 %	82 %
Acc. Test	79 %	76 %
Acc. Val.	79 %	78 %

Tabelle 2: Genauigkeit des Modells mit und ohne Datenerweiterung

In Tabelle (2) wird auf den ersten Blick deutlich, dass die Datenerweiterung zu einer Verschlechterung der Trainingsgenauigkeit von 88 % auf 82 % führt. Das liegt daran, dass die zusätzlichen Daten das Training schwieriger gemacht haben, wodurch es mehr Varianz besitzt. Dadurch overfittet das Modell nicht mehr so stark. Da das Modell durch die Erweiterung mehr Muster erkennen muss (z.B. 9 verschiedene Gewehre), ist auch die Test-/Validierungsgenauigkeit um $2 - 4$ % gesunken. Insgesamt ist die Differenz zwischen Trainings- und Validierungs-Genauigkeit aber gesunken (vgl. Tabelle 9) und der Prozess der Datenerweiterung hat das Overfitting des Modells reduziert.
Im Artikel von Moritz Egelkraut [9] wurden einige Vorschläge gemacht, die Genauigkeit und das Overfitting des Modells zu verringern. Dazu gehört, dass das Training vor dem Eintreten des Overfittings beendet wird. Die Betrachtung von Abbildung (9) zeigt jedoch, dass die Differenz der Loss-Funktion von Trainings- und Validie-

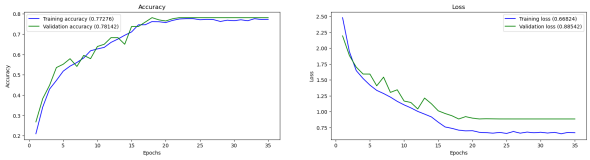

Abbildung 9: Netzgenauigkeit (35 Epochen) mit Datenerweiterung

rungsdaten ab der 13. Epochen ungefähr konstant bleiben. Dadurch würde das Training vor dem Eintreten der Konvergenz beendet werden, was zu einer Genauigkeit von unter 70 % führen würde. Aus diesem Grund wurde sich gegen diese Optimierungsmaßnahme entschieden. Eine weitere Möglichkeit der Optimierung ist eine gezielte Datenerweiterung. Diese wurde im vorherigen Abschnitt angewandt und führte zu einer Reduzierung der Überanpassung, was die die Vermutung von Moritz Egelkraut bestätigt [9].

In diesem Fall wurde sich für ein direktes Parameter-Tuning entschieden. Zunächst wurde die Fenstergröße für das Pooling von 4x2 auf 2x2 reduziert. Dadurch schrumpft das Bild beim Durchlaufen der Pooling-Schicht nicht so stark, wodurch mehr Parameter und damit auch mehr Informationen erhalten bleiben. Die Anzahl der Filter lag bisher immer bei $24 \rightarrow 48 \rightarrow 48$. Sie ist ein Indikator dafür, wie viele Merkmale sich das Modell merken kann. Da das Modell bereits zum Overfitting neigt, bietet es sich an, die Anzahl der Filter zu reduzieren. Hier wurde $20 \rightarrow 32 \rightarrow 32$ gewählt. Zudem wurde die Größe der Kernels von 5x5 auf 3x3 reduziert. Dies führt primär dazu, dass das Modell weniger Parameter hat, was die Trainingszeit reduziert [12].

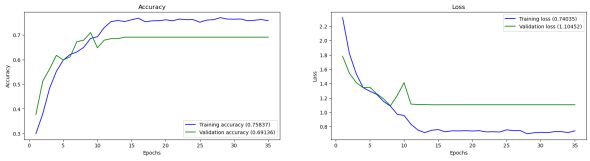

Abbildung 10: Netzgenauigkeit (35 Epochen) mit Netzanpassung - Version 1

Wie in Abbildung (10) zu sehen, haben sich diese Anpassungen sehr negativ auf das Overfitting des Modells ausgewirkt. Des Weiteren ist die Genauigkeit der Validierungsdaten auf unter 70 % gesunken und die Differenz der Loss-Funktion beträgt ca. $0,36$. Deshalb wurde die Größe der Kernel wieder auf 5x5 und das Pooling-Fenster auf 4x2 gesetzt. Somit wird in der nächsten Iteration lediglich mit der Reduzierung der Filteranzahl versucht, der Überanpassung entgegenzuwirken.

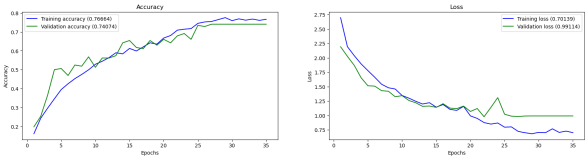

Abbildung 11: Netzgenauigkeit (35 Epochen) mit Netzanpassung - Version 2

Abbildung (11) zeigt eine Verbesserung der Überanpassung im Vergleich zu Version 1 (Abbildung 10). Verglichen mit dem ursprünglichen Modell hat sich die Überanpassung jedoch nicht verbessert, sondern ist gleich geblieben. Zusätzlich hat sich die Validierungsgenauigkeit um 5 % verschlechtert. Diese Ergebnisse weisen darauf hin, dass die ursprünglich gewählten Parameter des Modells bereits gut auf den Datensatz abgestimmt wurden. Um die Überanpassung weiter zu reduzieren, wäre ein deutlich aufwändigeres Training oder eine erhebliche Datenerweiterung erforderlich.

Damit das trainierte Modell nicht nur auf Reproduzierbarkeit geprüft wird, sondern auch auf neue, eventuell selbst aufgenommenen Tonspuren anzuwenden und zu testen ist, wurde eine Oberfläche mit der Python GUI PyQt5 5.15.11 programmiert (vgl. Abbildung 12). Die Oberfläche verfügt über vier Buttons: Mit dem Aufnehmen Button kann mit dem Mikrofon des Endgeräts selber eine Tonspur aufgenommen werden. Die Aufnahme läuft $2,97$ Sekunden, damit die Dimension der Eingabe bei 128×128 bleibt. Neben dem Aufnehmen lassen sich auch eigene Tonspuren im WAVE-Format importieren. Nach dem Laden wird die Audiodatei in voller Länge in der Wellenformdarstellung und als Mel-Spektrogramm dargestellt und kann durch betätigen des Abspielen-Buttons abgespielt werden. Mit dem Button ganz rechts kann die Klassifizierung gestartet werden. Ist diese abgeschlossen wird das Ergebnis auf der Oberfläche dargestellt. Des

Weiteren hilft eine Status-Anzeige dabei nachzuvollziehen, welcher Prozess gerade abläuft.

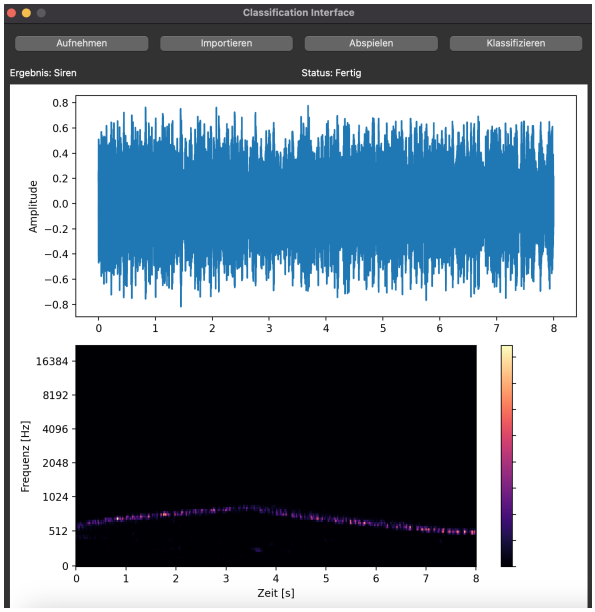

Abbildung 12: Klassifikationsoberfläche

4 Zusammenfassung

Das Ziel dieser Hausarbeit war es, ein Convolutional Neural Network für urbane Geräusche auf Basis des UrbanSound8K-Datensatzes [6] zu trainieren. Dabei wurde das Jupyter-Notebook von Min Yang [1] auf dem lokalen Rechner zum Laufen gebracht und eine GUI erstellt, mit der eigene Tonaufnahmen klassifiziert werden können. Dafür wurde zunächst die Programmierumgebung eingerichtet und die Daten vorverarbeitet. Die Daten wurden dabei in Grayscale Mel-Spektrogramme umgewandelt und in das CNN mit der Input-Shape $128 \times 128 \times 1$ eingebracht. Das Modell besteht aus drei Faltungsschichten mit 24, 48 und 48 Filtern der Kernelgröße 5×5. Diese Architektur erreichte eine Test- und Validierungsgenauigkeit von 79 % bei einer Trainingsgenauigkeit von 88 %. Das Modell neigt also zur Überanpassung, hat aber für die Größe des Datensatzes eine recht hohe Genauigkeit. Zudem zeigt die Konfusionsmatrix (Abbildung 8) Probleme bei der Klassifikation der Klassen „Car Horn“ und „Gun Shot“, da diese beiden Klassen ca. 60 % weniger Daten zur Verfügung haben. Außerdem kommt es zu Verwechslungen bei Klassen, die sich akustisch ähneln, wie z.B. die Klassen „Children Playing“ und „Dog Bark“. Um die unterrepräsentierten Daten auszugleichen, wurde eine gezielte Datenerweiterung der Klassen „Car Horn“ und „Gun Shot“ vorgenommen. Diese Änderung führte zu einer Reduzierung der Test- und Validierungsgenauigkeit um $1-3$ %, was zu erwarten war, da sich durch die Erweiterung des Spektrums der zu klassifizierenden Aufnahmen vor allem bei den Gun Shots vergrößert hat und das Training somit schwieriger gemacht wurde. Ein weiterer Effekt war die Reduzierung der Trainingsgenauigkeit auf 82 %, woraus eine Verbesserung der Überanpassung resultiert. Des Weiteren wurde Parameter-Tuning durchgeführt, indem das Pooling-Fenster von 4×2 auf 2×2 reduziert, die Kernelgröße auf 3×3 verkleinert und die Filteranzahl auf $20 \rightarrow 32 \rightarrow 32$ herabgesetzt wurde. Diese Anpassung sorgte für eine noch stärkere Überanpassung mit einer Validierungsgenauigkeit von weniger als 70 %. In einem weiteren Versuch die Überanpassung zu reduzieren wurde nur auf die Reduktion der Filteranzahl gesetzt. Diese blieb jedoch unverändert und die Validierungsgenauigkeit sank um 5 %. Die durchgeführten Experimente haben gezeigt, dass die ursprüngliche Architektur mit einer Validierungsgenauigkeit von 79 % bereits gute Ergebnisse für den UrbansSound8K-Datensatz liefert und simple Parameteränderungen nicht zu einer Verbesserung des Modells geführt haben. Eine deutliche Reduktion der Überanpassung wäre nur durch eine erhebliche Datenerweiterung oder komplexeres Training möglich, da das Grundrauschen in urbanen Umgebungen bereits überdurchschnittlich hoch ist.

Literatur

[1] Min Yang. Automatic urban sound classification with cnn. https://www.kaggle.com/code/mychen76/automatic-urban-sound-classification-with-cnn, 2019. Accessed: 2025-07-22.

[2] Stephan Marzi. *Grundlagen der Maschinendynamik*. Technische Hochschule Mittelhessen, Fachbereich Maschinenbau und Energietechnik, Gießen, 2024. Skript zur Vorlesung, Version 2024.1.

[3] Ronald W. Schafer Alan V. Oppenheim. *Discrete-Time Signal Processing: Pearson New International Edition*. Pearson Education UK, 2013.

[4] B. Pfister and T. Kaufmann. *Sprachverarbeitung*. Springer-Lehrbuch. Springer Berlin Heidelberg, 2008.

[5] Jörg Frochte. *Maschinelles Lernen: Grundlagen und Algorithmen in Python*. Carl Hanser Verlag GmbH Co KG, 2020.

[6] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. UrbanSound8K: A Dataset of Urban Sound Recordings. https://urbansounddataset.weebly.com/urbansound8k.html, 2014. Accessed: 2025-07-22.

[7] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. A dataset and taxonomy for urban sound research. In *Proceedings of the 22nd ACM International Conference on Multimedia*, pages 1041–1044. ACM, 2014.

[8] Pascal Stoppa. Urban sound classification. https://github.com/pascalsto/Urban-Sound-Classification, 2025. Accessed: 2025-08-16.

[9] Moritz Egelkraut. *Convolutional Neural Network für die Ürban Sound Classification* , chapter 17. Hochschule für Angewandte Wissenschaften Hof, 2019.

[10] Cleyton Aparecido Dim, Nelson Cruz Sampaio Neto, and Jefferson Magalhães de Morais. Hornbase – a car horns dataset, 2024.

[11] Emrah Aydemir. Gunshot audio dataset.

[12] Joy Sengupta. How to decide the hyperparameters in CNN. https://medium.com/@sengupta.joy4u/how-to-decide-the-hyperparameters-in-cnn-bfa37b608046, 2020. Aufgerufen: 2025-08-15.

[13] Roneel V. Sharan and Björn W. Schuller. Snore sound classification with mel-spectrogram and a fine-tuned cnn. In *2024 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, pages 479–482, 2024.