Urban Sound Challenge

CO2V - Keller Patrick, Kocaj Alen

Inhaltsverzeichnis

Problemstellung

- Urban Sound Challenge: Multi-Klassen Klassifikation von kurzen Audio-Samples
- 3680 Datenpunkte mit leicht unbalanciertem Datensatz
- Feature-Extraktion mittels Python-basierter libROSA Bibliothek
- Klassifikation mittels verschiedenen neuronalen Netzen Modellen
  - DNNs, CNNs, RNNs bzw. LSTMs
- Vergleich der Modelle anhand typischer ML-Metriken (Precision, Recall, Accuracy)

Methodik I

- Verwendung des großen Datensatzes zur Extraktion und Vorverarbeitung der Features
- Split in Training-, Test- und Validierungsdaten
- Schrittweise Annäherung der Hyperparameter anhand Validierungsgenauigkeit
- Testen verschiedener Hyperparameterkombinationen mittels Gridsearch
- Auswertung finaler Modelle mithilfe Metriken & Visualisierungen

Methodik II

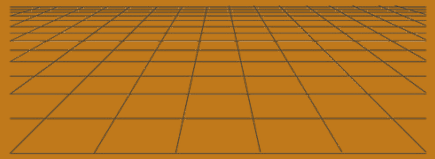
- Extraktion erfolgt mithilfe der Python-Libraries Numpy, Pandas & libROSA
- Programmierung, Training & Auswertung erfolgt in Google Colaboratory
  - `Jupyter Notebook auf Anabolika`
- Training geschieht mit Keras
- Auswertung und Darstellung mithilfe Scikit-Learn & Matplotlib

Feature Engineering - MFCC

- **M**el-**F**requency **C**eptral **C**oefficients
- Kompakte Darstellung des Frequenzraumes
- Mel-Frequenzen beschreiben die Tonhöhe
- Cepstrum = inverse-FFT des logarithmierten Frequenzspektrums (Fourier-Raum)
- Mel entstehen durch Skalierung des Frequenzspektrums anhand des menschlichen Gehör (manche Frequenzen sind für Menschen wichtiger als andere)

  → nicht lineare Fensterfunktion
- Anzahl der MFCCs = Anzahl der Fenster, die gelegt werden (initial: 40)

Gridsearch



+

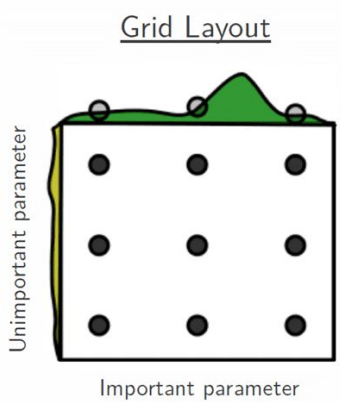
=

Gridsearch

- Unromantisch, aber effektiv
- Suche der bestmöglichen Hyperparameter für DNN, CNN, LSTM
- Training erfolgte auf Google Colab mit Shared GPU (Nvidia K80s, T4s, P4s and P100s)

Netzwerkspezifische Parameter wie folgt variiert

- Learning Rate (0.01, 0.001)
- Dropout (0.2, 0.5)
- Batch Size (10, 20, 40)



Gridsearch - DNN - I

Folgende Parameter fix gesetzt:

- Dropout Layer nach jedem Hidden Layer
- ReLU als Activation Function

Variation folgender Parameter:

- Anzahl Hidden Layer (1, 2)
- Anzahl der Neurons (128, 256, 512)

→ Alle Kombinationen summiert: ~180 Tests
→ Dauer: ~3h

Gridsearch - DNN - II

architecture	learning_rate	dropout	batch_size	accuracy	precision	recall	f1-score
l2_512_256_relu	0.001	0.5	20	93.939	0.949	0.939	0.937
l1_512_relu	0.001	0.2	10	90.909	0.934	0.909	0.910
l2_256_512_relu	0.001	0.2	10	90.909	0.929	0.909	0.908
l2_128_512_relu	0.001	0.2	20	90.909	0.912	0.909	0.907
l2_128_512_relu	0.001	0.2	40	90.909	0.944	0.909	0.912
l2_512_512_relu	0.01	0.2	40	3.030	0.001	0.030	0.002
l2_128_128_relu	0.01	0.5	10	3.030	0.001	0.030	0.002
l2_256_128_relu	0.01	0.5	10	3.030	0.001	0.030	0.002
l2_128_512_relu	0.01	0.5	40	3.030	0.001	0.030	0.002
l2_512_128_relu	0.01	0.5	40	3.030	0.001	0.030	0.002

→ Zwei Hidden Layer > ein Hidden Layer
→ Wichtigster Faktor: Learning Rate
→ Kein Overfitting mit breiterem & tieferem Netzwerk

Gridsearch - CNN - I

Folgende Parameter fix gesetzt:

- Kein Pooling Layer, weil stichprobenweise keine Verbesserung
- Batch-Normalization Layer wie empfohlen in AlexNet (Krizhevsky et al. 2012)
- Leaky-ReLU in Convolution Layer
- Dropout Layer vor DNN

Variation folgender Parameter:

- Kernelgröße (3x3, 5x5, 7x7) mit Padding = Stride = 1 des Convolution Layer
- Anzahl der Hidden Layers (1, 2)

→ Alle Kombinationen summiert: ~430 Tests
→ Dauer: ~12h

Gridsearch - CNN - II

architecture	learning_rate	dropout	batch_size	accuracy
l1_k3_d128_relu	0.01	0.2	10	93.939
l1_k3_d128_relu	0.01	0.2	40	93.939
l1_k7_d128_relu	0.01	0.2	20	90.909
l1_k5_d128_relu	0.001	0.2	20	90.909
l2_k3_k7_d128_relu	0.01	0.2	10	90.909
l2_k3_k5_d128_relu	0.01	0.2	10	27.273
l2_k5_k5_d128_relu	0.001	0.2	10	27.273
l2_k7_k7_d128_relu	0.01	0.5	20	24.242
l2_k7_k7_d128_relu	0.01	0.5	40	24.242
l2_k7_k7_d128_relu	0.01	0.5	10	18.182

→ Ein Hidden Convolution Layer mit Kernel 3x3, einem DNN mit 128 Neurons & ReLU als Act.Func.
→ Dropout von 20%, während Batch Size keinen Einfluss hat
→ Hohe Learning Rate bringt besseres Ergebnis

Gridsearch - LSTM - I

Folgende Parameter fix gesetzt:

- Ein Hidden Layer + anschließendem Dropout

Variation folgende Parameter:

- Anzahl der LSTM - Einheiten (16, 32, 48)

→ **sehr langsames Training**

→ Alle Kombinationen summiert: ~50 Tests
→ Dauer: ~8h

Gridsearch - LSTM - II

architecture	learning_rate	dropout	batch_size	accuracy	precision	recall	f1-score
l1_lstm48_relu	0.001	0.5	20	87.879	0.919	0.879	0.886
l1_lstm48_relu	0.001	0.2	40	84.848	0.899	0.848	0.856
l1_lstm48_relu	0.001	0.2	20	81.818	0.889	0.818	0.829
l1_lstm48_relu	0.001	0.2	10	78.788	0.883	0.788	0.802
l1_lstm32_relu	0.001	0.2	40	78.788	0.884	0.788	0.797
l1_lstm32_relu	0.01	0.5	40	39.394	0.339	0.394	0.321
l1_lstm32_relu	0.01	0.2	10	36.364	0.454	0.364	0.374
l1_lstm48_relu	0.01	0.2	10	36.364	0.499	0.364	0.363
l1_lstm16_relu	0.01	0.5	10	30.303	0.398	0.303	0.302
l1_lstm32_relu	0.01	0.5	10	21.212	0.155	0.212	0.158

→ Mehr LSTM - Units = besseres Modell
→ Dropout und Batch Size kaum Einfluss
→ Learning Rate niedriger als bei CNN

Gridsearch - Final

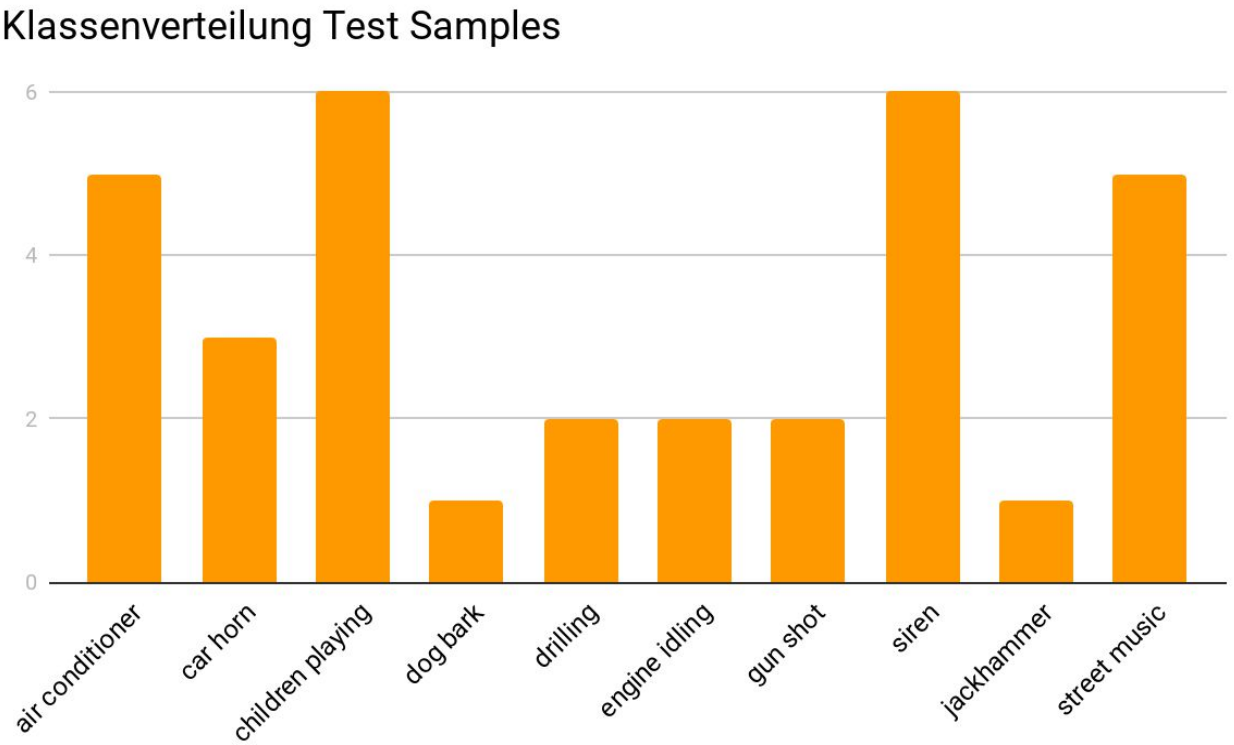
type	learning_rate	learning_rate	dropout	batch_size	accuracy	precision	recall	f1-score
dnn	l2_512_256_relu	0.001	0.5*	20	93.939	0.949	0.939	0.937
cnn	l1_k3_d128_relu	0.01	0.2*	10	93.939	-	-	-
lstm	l1_lstm48_relu	0.001	0.5*	20	87.879	0.919	0.879	0.886

* Finale Dropout-rate wurde auf 0.3 geändert

→ DNN > CNN > LSTM

Ergebnisse & Resultate

Testset



⇒ Metriken wie “gewöhnlicher” F1-Score skewed

→ daher werden gewichtete Metriken verwendet. Gewichtung erfolgt nach Anzahl der Samples

= 33 Samples - 0.9% des Datensets

Ergebnisse & Resultate - DNN I

Basis: NN mit 156K Parameter

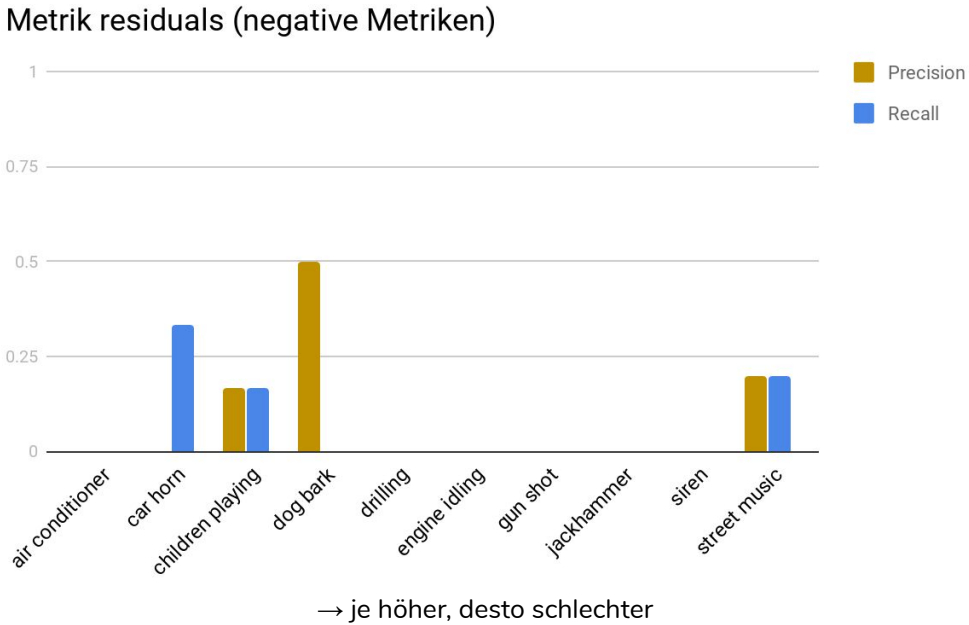
Accuracies:

- Training: 97.00%
- Test: 90.90%

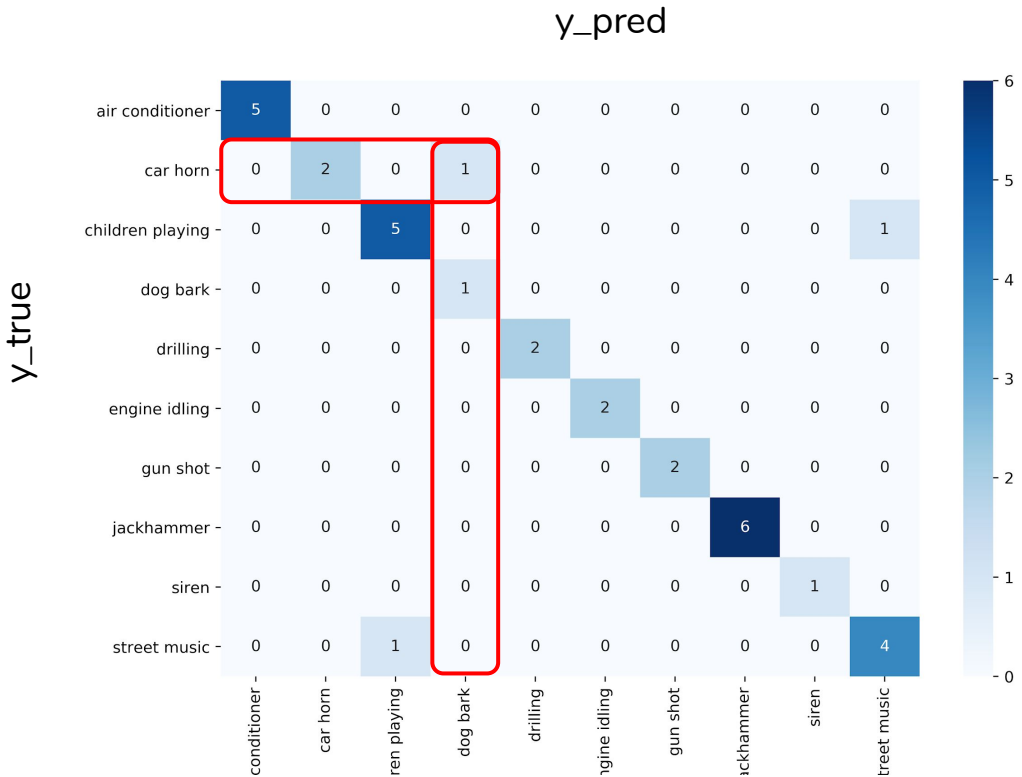
Aggregierte gewichtete Metriken:

- Precision: 92.40%
- Recall: 90.90%

⇒ Positive Klasse wird eher erkannt, als relevante Ergebnisse



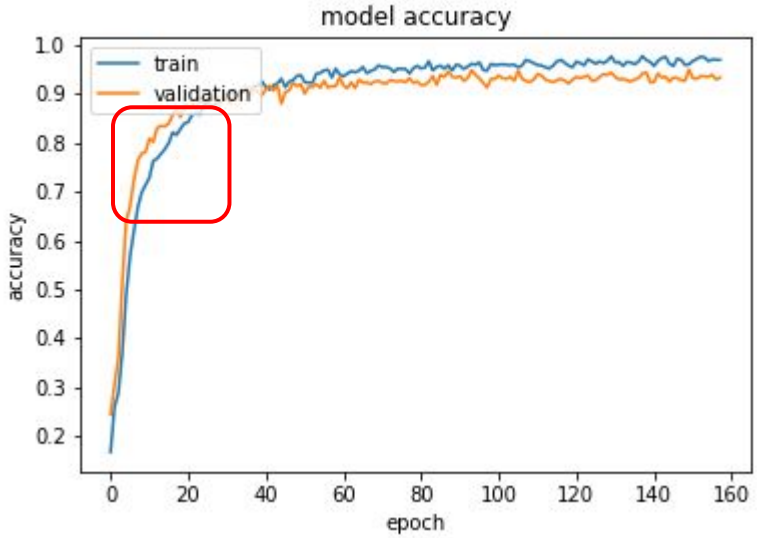
Ergebnisse & Results - DNN II

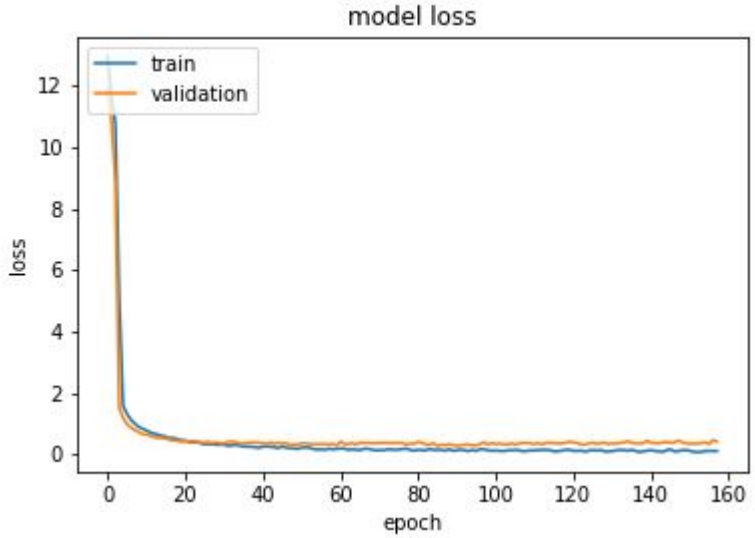


Beispiel: Autohupen wurde falsch positiv als Hundebellen klassifiziert (FP in Relation zur Klasse `dog bark`), während es falsch negativ in Relation zur Klasse `car horn` klassifiziert wurde

`Dog bark` Prec: 1 / 2 = 0.5
`Car horn` Rec: 2 / 3 = 0.67

Ergebnisse & Resultate - DNN III





Kaum Auffälligkeiten beim Training: leichtes Overfitting, kurze Phase wo Val.Acc > Train.Acc

Ergebnisse & Resultate - CNN I

Basis: CNN mit 6.3K Parameter

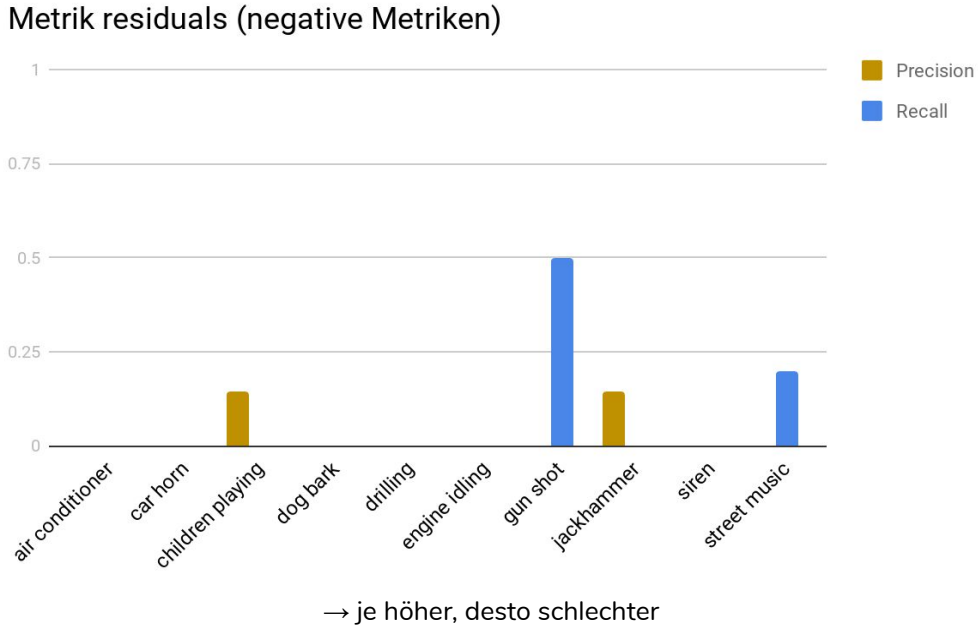
Accuracies:

- Training: 84.22%
- Test: 93.94%

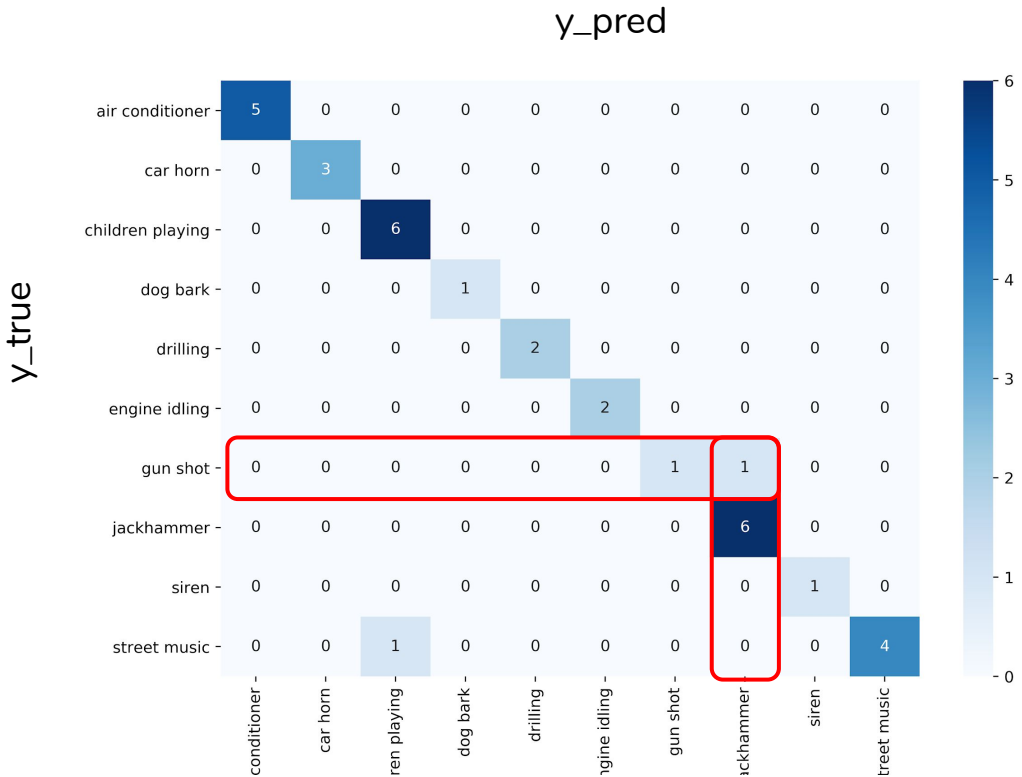
Aggregierte gewichtete Metriken:

- Precision: 94.80%
- Recall: 93.90%

⇒ Positive Klasse wird knapp eher erkannt, als relevante Ergebnisse



Ergebnisse & Results - CNN II

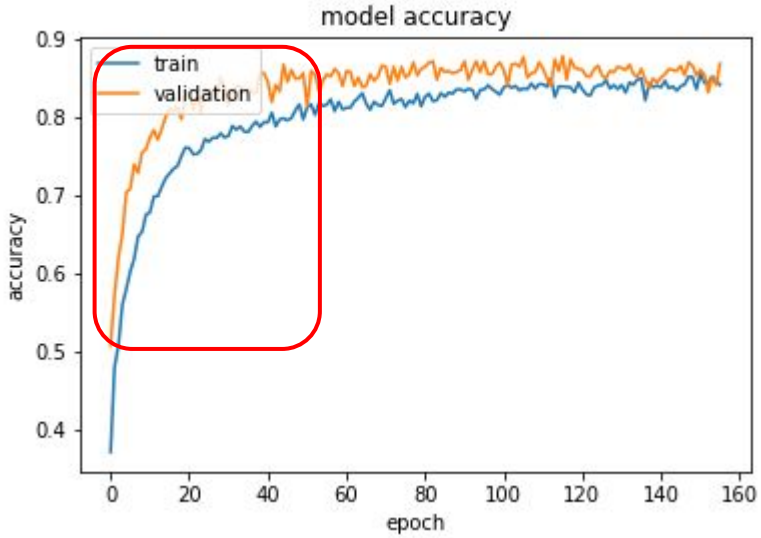


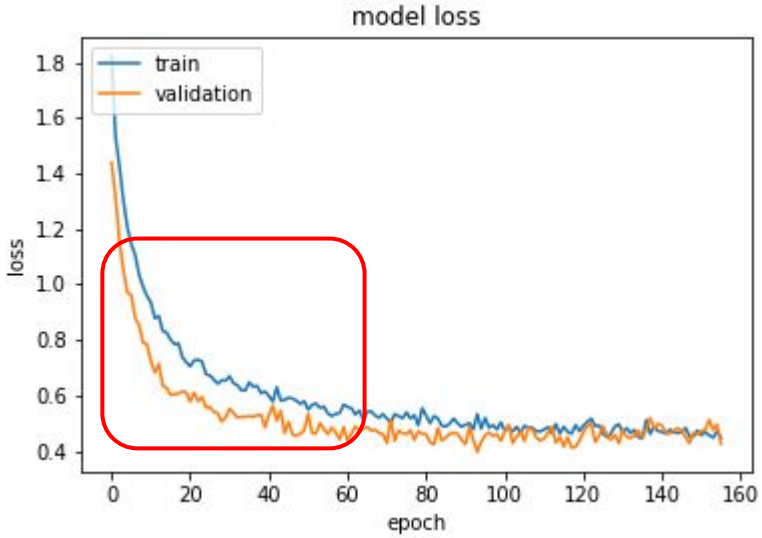
Beispiel: Schusswaffe wurde falsch positiv als Presslufthammer klassifiziert (FP in Relation zur Klasse `jackhammer`), während es falsch negativ in Relation zur Klasse `gun shot` klassifiziert wurde

`Jackham..` Prec: 6 / 7 = 0.86
`Gun shot` Rec: 1 / 2 = 0.5

Man achte auf Gewichtung der Samples!

Ergebnisse & Resultate - CNN III





Modell auf Validierungsdaten immer besser als bei Trainingsdaten
→ inhärente Eigenschaft der Daten & CNN 1D

Ergebnisse & Resultate - LSTM I

Basis: LSTM mit 9.6K Parameter

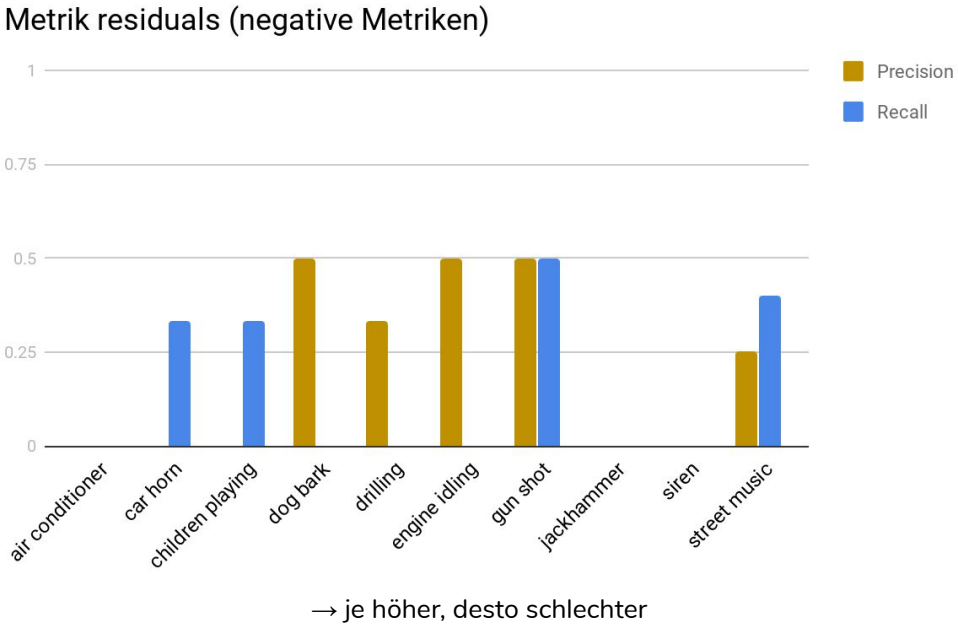
Accuracies:

- Training: 79.44%
- Test: 81.81%

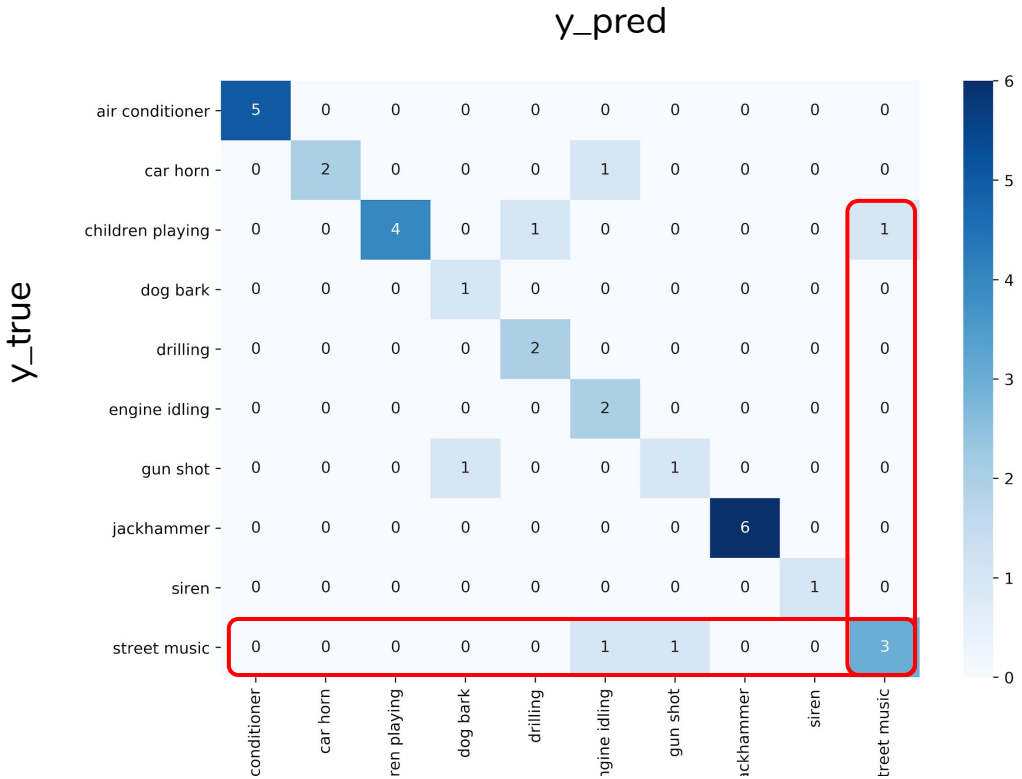
Aggregierte gewichtete Metriken:

- Precision: 86.60%
- Recall: 81.80%

⇒ Positive Klasse wird eher erkannt, als relevante Ergebnisse



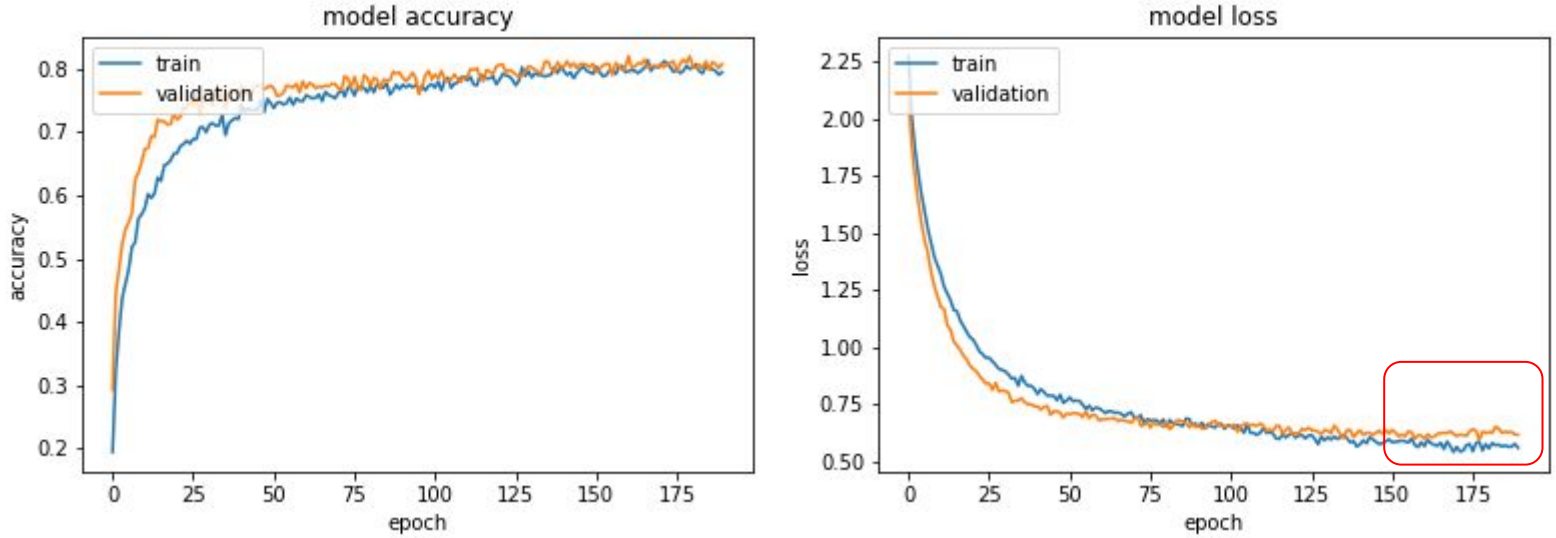
Ergebnisse & Results - LSTM II



Beispiel: Straßenmusik wurde zweimal falsch negativ als andere Klassen klassifiziert (Engine im Leerlauf & Schusswaffe), während Kinderspielen falsch positiv als Straßenmusik klassifiziert wurde

`Street M..` Prec: 3 / 4 = 0.75
`Street M..` Rec: 3 / 5 = 0.6

Ergebnisse & Resultate - LSTM III



Training in “Ordnung”: schlechte Genauigkeit und am Schluss Divergenz zwischen Train.Loss & Val.Loss. Grundsätzlich kein Overfitting vorhanden.

Schwaches Modell musste genommen werden, um starkes Overfitting zu verhindern

Ergebnisse & Resultate - Final

type	train.accuracy	test.accuracy	precision	recall
dnn	97.00	90.90	92.40	90.90
cnn	84.22	93.94	94.80	93.90
lstm	79.44	81.81	86.60	81.80

Wir gratulieren CNN zum Sieg in der Urban Sound Challenge! 👏🥳💪

Aussicht

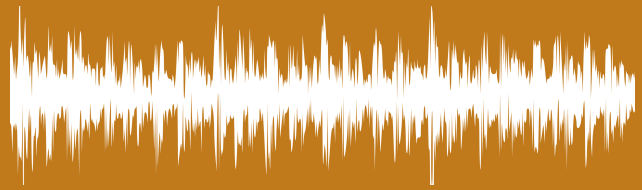
Aussicht I

- CNN und DNN beste Arten von neuronalen Netzen, um Audiodaten zu klassifizieren
  - DNN kann durch breitere Neuronenanzahl gut fitten
  - CNN besitzt inhärente Eigenschaften um 2D (Bild) bzw. 1D (Audio)
    - Daten vorverarbeiten, dann selbst mittels DNN klassifizieren
- LSTM ist sehr mächtig, aber overfittet schnell ⇒ daher schwaches Modell genommen
  - Mehr Zeit in Hyperparameter-Tuning investieren
  - Zusätzlicher Layer an LSTM-Units brächte zusätzliche Mächtigkeit
  - Mit Dropout “regularisieren”

Aussicht II

- Weiteres Feature Engineering durch diverse Features:
  - Einfach: Root-Mean-Square jedes Frames
  - Chromagram der Audio-Welle
  - Mel-skaliertes Spectrogram
  - Koeffizienten zum Fitten eines n-th Polynom an der Audio-Welle
  - Spektral “Flachness” & “Kontrast”
  - Rhythmus: Tempogram

- Andere Anzahl an Ceptral Koeffizienten



Vielen Dank für Ihre Aufmerksamkeit