Visuelle Erkennung von Gitarrenakkorden mit einer Android-App

Sascha Knapp

Zusammenfassung

Für Smartphones mit dem Betriebssystem Android wird eine Anwendung entwickelt, die mit der Kamera des Gerätes Gitarrenakkorde erkennen kann. Hierfür wird ein Datensatz erstellt, mit dem auf dem vortrainierten Machine-Learning Modell MobileNetV2 Transfer-Learning durchgeführt wird, um die Akkorde C, D, Em, F, G zu klassifizieren. Es werden Schwächen des Datensatzes aufgedeckt und untersucht. Zuletzt wird erläutert, wie das tflite-Modell für das Android-Projekt vorbereitet wird und wie sich das Projekt mit Object-Detection erweitern ließe.

1 Einleitung

Es soll eine Anwendung für das Betriebssystem Android entwickelt werden, die Gitarrenakkorde mit der Kamera erkennen kann. Die Anwendung zeigt dem Benutzer den erkannten Gitarrenakkord in Tabulaturnotation an, damit der Benutzer den Akkord selbst lernen kann. Da es eine Vielzahl an verschiedenen Formen für den gleichen Akkord gibt, werden in diesem Projekt nur fünf Akkorde in ihrer jeweiligen Grundform als "Proof of Concept" erkannt. Dafür sollen fünf Akkorde mit Hilfe von Machine-Learning klassifiziert werden können. Die Akkorde sind C-Dur, D-Dur, E-Moll, F-Dur und G-Dur (in Abbildung 1 von links nach rechts). Im Folgenden werden die Akkorde zur Vereinfachung als C, D, Em, F, G bezeichnet.

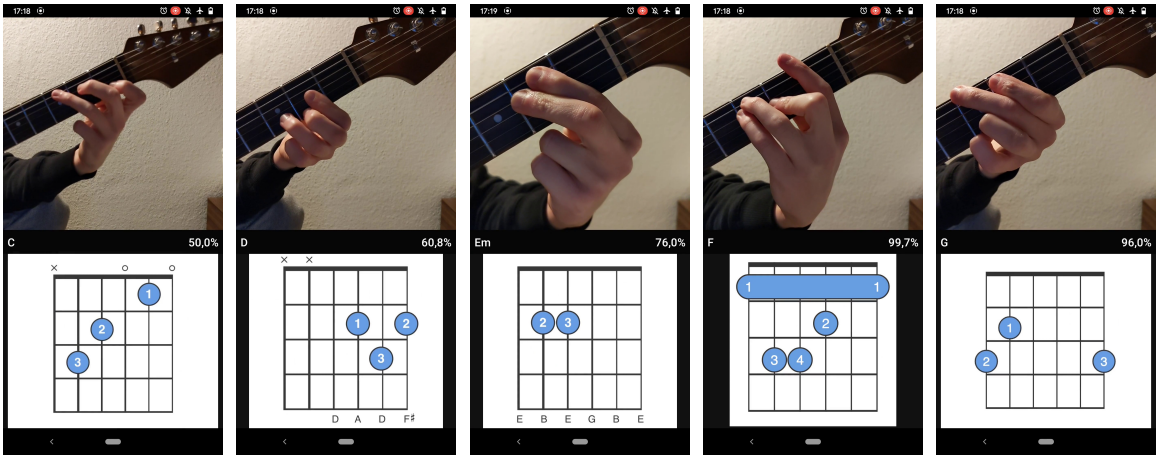

ABBILDUNG 1: Beispiel-Screenshots der Android-App

2 Ähnliche Arbeiten

In *CNN Transfer Learning for Visual Guitar Chord Classification* [1] von drei Standford-Studenten wird mit Hilfe von Transfer-Learning eine Webanwendung entwickelt, die die Akkorde C, D, Em, F und G mit der Webcam erkennt. Für die Trainingsdaten haben sie einen eigenen Datensatz erstellt. Für die Implementierung des Transfer-Learnings benutzen die Autoren PyTorch.

3 Problem

Es gibt keinen öffentlichen Datensatz mit Bildern von Gitarrenakkorden, die als Trainingsdaten verwendet werden können. Im Github-Repository [2] der Stanford-Arbeit [1] findet sich ebenso ein Datensatz mit den Gitarrenakkorden, die hier klassifiziert werden sollen, doch sind die Daten teilweise zu klein für das später genutzte Modell. Außerdem kommen die Autoren der Arbeit selber zu dem Schluss, dass ihr Datensatz nicht ausreichend für das richtige klassifizieren der Akkorde ist. Aus diesem Grund wird ein eigener Datensatz erstellt.

Das Erkennen von Gitarrenakkorden erscheint zuerst sehr ähnlich wie das Erkennen von Zeichensprache (Sign Language), nur dass sich die Zeichen auf dem Griffbrett der Gitarre befinden. Somit ist auf dem untersuchten Bild immer eine Gitarre zu sehen, die für das Erkennen des Akkords keine Relevanz hat, sich aber ohne weiteres nicht aus dem Bild entfernen lässt. Deswegen wird versucht, ein vortrainiertes Machine-Learning Modell für die Bildklassifizierung zu trainieren, was bei einem einfachen Fingeralphabet für Zeichensprachen zu guten Ergebnissen führt [3].

4 Erzeugen der Trainingsdaten

Um die Trainingsdaten zu erzeugen, werden vor einer weißen und beleuchteten Wand jeweils Videos von den Akkorden gemacht. Dabei wird im Video auch die Gitarre zwei mal gewechselt, sodass das Modell später nicht nur auf eine Gitarre trainiert wird. Von diesen Videos werden Frames exportiert. Da vor der weißen Wand immer gleiche Lichtverhältnisse herrschen, werden die Frames mit OpenCV nachbearbeitet, um andere Lichtverhältnisse zu simulieren. Dabei wird der Kontrast und die Helligkeit zufällig verändert.

ABBILDUNG 2: Exportierter Beispielframe

In Abschnitt 6 wird darauf eingegangen, wieso der Datensatz vor der weißen Wannd nicht auszureichen scheint und weitere Daten erzeugt werden müssen.

ABBILDUNG 3: Smartphone-Halterung für die Aufnahme der Trainingsdaten

Um die Diversität des Datensatzes zu verbessern, wird die in Abbildung 3 gezeigte Konstruktion verwendet um Videos von den Akkorden in einem Raum zu machen, der von einer Seite durch ein Fenster beleuchtet ist. Bei der Aufnahme eines Akkordes wird eine 360°-Drehung auf der Stelle gemacht, um verschiedene Lichtverhältnisse und Hintergründe einzufangen. Wie im ersten Datensatz werde auch hier die Gitarren gewechselt.
Im finalen Datensatz werden etwa 30 Prozent des alten Datensatzes vor der weißen Wand verwendet. Eine Klasse im finalen Datensatz hat insgesamt etwa 200 Bilder.

ABBILDUNG 4: Beispiele für C-Akkord aus dem Trainingsdatensatz nach Verwendung der Smartphone-Halterung.

5 Transfer-Learning

Dieser Abschnitt wird zum großen Teil vom Tensoflow-Tutorial zu Transfer-Learning und Fine-Tuning beeinflusst [4].

Als vortrainiertes Basismodell für das Training wird MobileNetV2 von Google verwendet. Auf dieses wird Transfer-Learning angewendet. Für die Implementierung wird Tensorflow und Keras verwendet.

Im ersten Schritt wird das Basismodell eingefroren, sodass das neue Modell nur für die Parameter der letzen Schicht trainiert wird, wie es in Abbildung 5 zu sehen ist.

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
tf.math.truediv (TFOpLambda) (None, 224, 224, 3)       0
_________________________________________________________________
tf.math.subtract (TFOpLambda (None, 224, 224, 3)       0
_________________________________________________________________
mobilenetv2_1.00_224 (Functi (None, None, None, 1280)  2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dropout (Dropout)            (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 5)                 6405
=================================================================
Total params: 2,264,389
Trainable params: 6,405
Non-trainable params: 2,257,984
_________________________________________________________________
```

ABBILDUNG 5: Modell für Klassifizierung der Gitarrenakkorde

Im zweiten Schritt wird auch das Basismodell von Schicht 100 ab trainiert, sodass 54 Schichten von dem Basismodell trainiert werden. Dadurch wird die Accuracy erhöht und der Loss verringert, wie in Abbildung 6 dargestellt.

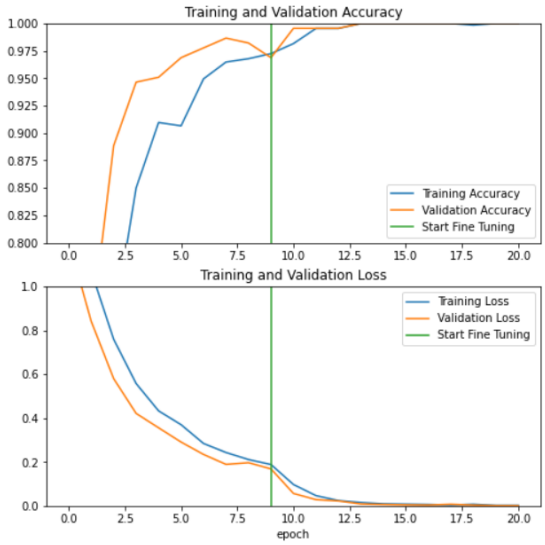

ABBILDUNG 6: Accuracy und Loss

Der finale Datensatz besteht aus 1101 farbigen Bildern. Eine Epoche dauert durchschnittlich etwa 2 Minuten, wobei die Batchsize 32 beträgt. Als erste Schicht im Modell steht eine Augmentation-Schicht, in der die Bilddaten zufällig im Bereich von 0% bis 20 % gedreht werden. Das wird vorallem gemacht, um Ungenauigkeiten im Datensatz auszugleichen, falls die Gitarre bei verschiedenen Akkorden in einem anderen Winkel gehalten wird.

6 Testen des Modells

6.1 Testen mit dem Testdatensatz

Da der Testdatensatz Teil des gesamten Datensatzes ist, aus dem auch die Trainings- und Validierungsdaten stammen, tritt hier das Phänomen auf, dass die Confusion-Matrix ein perfektes Ergebnis zeigt. Im nächsten Abschnitt 6.2 wird gezeigt, dass bei Echtweltdaten jedoch auch falsch klassifiziert wird. Das weist darauf hin, dass der Testdatensatz den Trainings- und Validierungsdaten zu ähnlich ist.

Aus diesem Grund wird, wie in Abschnitt 6 beschrieben, ein diverserer Datensatz erzeugt. Die Confusion-Matrix auf der rechten Seite zeigt, dass der neue Datensatz das gleiche Phänomen aufweist und sich die Daten trotz verschiedener Lichtverhältnisse und Hintergründe immer noch zu ähnlich sind.

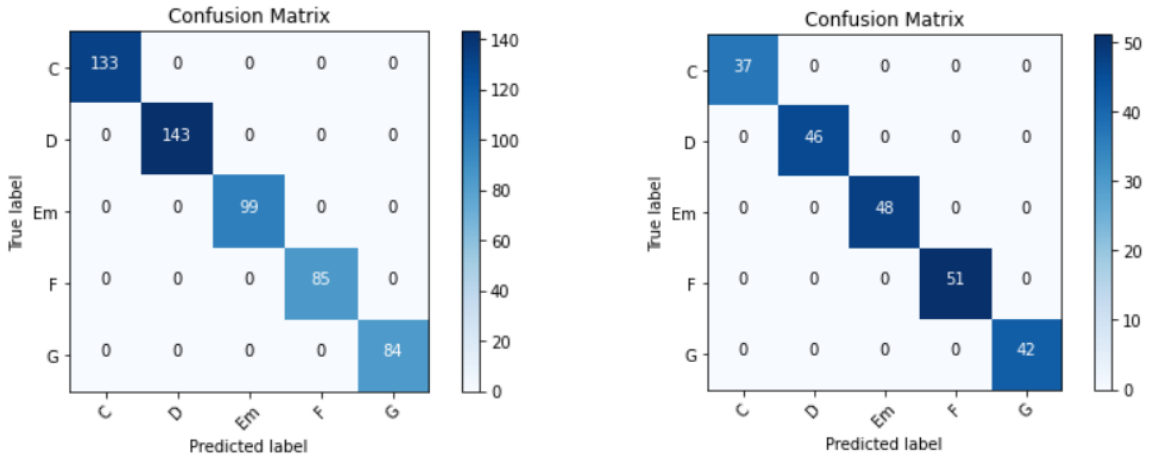

ABBILDUNG 7: Links die Confusion-Matrix zum Datensatz vor der weißen Wand, rechts die Confusion-Matrix zum diverserem Datensatz

6.2 Testen mit einem Echtweltdatensatz

Um aussagekräftigere Ergebnisse beim Testen zu erzielen wird ein Datensatz mit vier Bildern in jeder Klasse zusammengestellt. Die Bilder stammen dabei zum Teil aus dem Internet (Google Images) oder wurden aus älteren Datensätzen entnommen, die nicht mehr für das Training verwendet werden. Trotz der geringen Menge an Testdaten kann hier ein Muster erkannt werden.

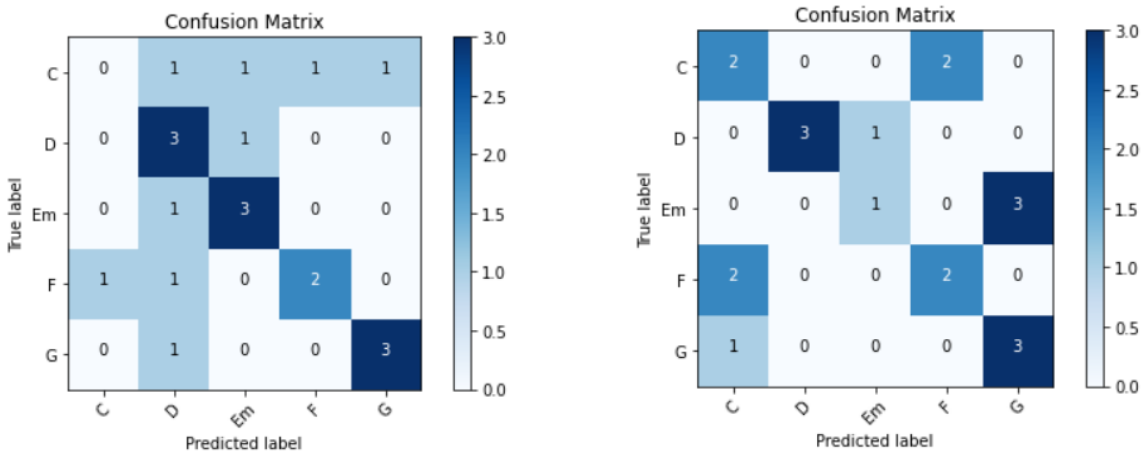

ABBILDUNG 8: Links die Confusion-Matrix zu den Testbildern, rechts die Confusion-Matrix zu den zugeschnittenden Bildern.

Zuerst wird die Confusion-Matrix in Abbildung 8 erklärt. Ein Großteil der Testdaten wird richtig klassifiziert, allerdings wird keiner der vier C Akkorde erkannt. In der Praxis beim Nutzen der Android-App wird schon erkannt, dass es zu helfen scheint, die Kamera näher an den untersuchten Gitarrenakkord zu halten. Deswegen wird der Testdatensatz

bearbeitet und nur der vermeintlich relevante Bildteil ausgeschnitten. In Abbildung 9 wird beispielhaft eine solche Region of Interest hervorgehoben.

ABBILDUNG 9: Testbild mit hervorgehobener Region of Interest.

Die rechte Confusion-Matrix in Abbildung 8 zeigt, dass auch C-Akkorde erkannt werden. Im Vergleich zur linken Confusion-Matrix werden jedoch mehr Em-Akkorde falsch klassifiziert. Es ist aber nicht nachvollziehbar, warum diese Em-Akkorde vorher überhaupt richtig klassifiziert wurden, da kein offensichtlich relevanter Teil des Akkords oder der Gitarre abgeschnitten wurde. Viel mehr lassen sich mit dem angepassten Datensatz in der Confusion-Matrix Muster erkennen und es ist nachvollziehbar, dass einige Akkorde falsch klassifiziert werden.

In Abbildung 10 werden zwei Bespiele aus dem Datensatz gezeigt. Der Em-Akkord wird als ein G-Akkord klassifiziert. Die beiden Akkorde sind sich visuell sehr ähnlich. Lediglich der Mittelfinger befindet sich sichtlich in einer anderen Position. Aus der Perspektive auf den Bildern ist nicht zu erkennen, ob der Ringfinger und der kleine Finger das Griffbrett berühren, da die Fingerspitzen nicht zu sehen sind.

Im Trainingsdatensatz ist dieses Merkmal eher beim G-Akkord aufzufinden und nicht beim Em-Akkord, da dort Zeige- und Ringfinger mehr gestreckt sind und die anderen Finger klar vom Griffbrett der Gitarre abgesetzt sind.

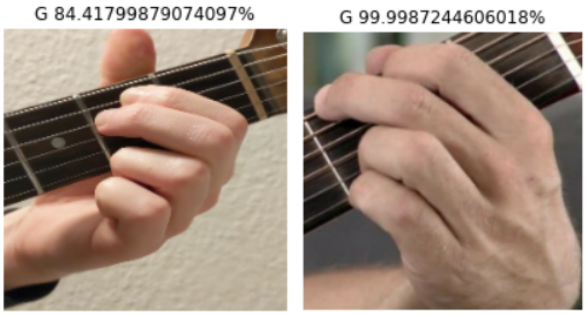

ABBILDUNG 10: Links ein falsch klassifizierter Em-Akkord, rechts ein richtig klassifizierter G-Akkord.

In Abbildung 11 wird ein F falsch als C klassifiziert und ein C als F klassifiziert. Der Unterschied zu der zuvor beschriebenen falschen Klassifizierung ist, dass die Akkorde jeweils als der andere Akkord klassifiziert werden. Die Form der Akkorde ist sehr ähnlich, da Ring- und Mittelfinger gleich positioniert sind. Dennoch sollte der gestreckte Zeigefinger ein eindeutiges Merkmal des F sein, welches in der Form nicht beim C zu sehen ist.

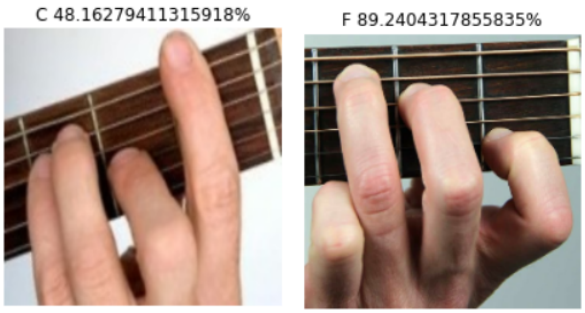

ABBILDUNG 11: Links ein falsch klassifizierter F-Akkord, rechts ein ebenso falsch klassifizierter C-Akkord.

7 Tensorflow Lite für Android

Um das trainierte Tensorflow-Modell in einer Android-App zu verwenden, muss dieses zunächst zu einem tflite-Modell konvertiert werden. Das kann durch das folgende Code-Beispiel in Abbildung 12 erreicht werden.

```
1  saved_model_dir = '/content/drive/MyDrive/EML/Guitar_Model_V4/'
2  tf.saved_model.save(new_model, saved_model_dir)
3  converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
4  tflite_model = converter.convert()
5  open('/content/drive/MyDrive/EML/Guitar_Model_V4/model.tflite', 'wb').write(tflite_model)
6  labels = '\n'.join(sorted(CATEGORIES))
7  open('/content/drive/MyDrive/EML/Guitar_Model_V4/labels.txt', 'w').write(labels)
```

ABBILDUNG 12: Beispiel-Code um TensorFlow-Modell in ein tflite-Modell zu konvertieren.

Das konvertierte Modell kann so z.B. im Beispielprojekt von Tensorflow für Android [5] direkt verwendet werden, für das dann eine Classifier-Klasse geschrieben werden muss, die getModelPath() und getLabelPath() implementiert. Den Quellcode vom Beispiel ist umfangreich und es kann leicht unübersichtlich werden, wenn eigene Anpassungen an der Implementierungen vorgenommen werden sollen.

Deswegen wird für die Android-App zum Klassifizieren der Gitarrenakkorde als Basis der Flower Classifier von Google [7] genutzt. Dieser ist in Kotlin geschrieben und nutzt einen moderneren Ansatz als den eben beschriebenen. Seit der Version 4.1 bietet Android Studio das ML Model Binding an, welches für das tflite-Modell ein einfaches Interface anbietet, um in wenigen Zeilen Code die Inferenz für ein Bild durchführen zu lassen, wie es in Abbildung 13 zu sehen ist.

```kotlin
val model = GuitarModel.newInstance(context)

// Creates inputs for reference.
val image = TensorImage.fromBitmap(bitmap)

// Runs model inference and gets result.
val outputs = model.process(image)
val probability = outputs.probabilityAsCategoryList

// Releases model resources if no longer used.
model.close()
```

ABBILDUNG 13: Beispiel-Code für Inferenz in Kotlin auf Android.

Vorraussetzung zur korrekten Nutzung des Model Bindings ist, dass das tflite-Modell Metadaten enthält. Um Metadaten zum Modell hinzuzufügen gibt es zwei Möglichkeiten:

- Die Metadaten können mit Hilfe eines Pythonskript nachträglich hinzugefügt werden. In [8] wird auf ein fertiges Script verwiesen, in welchem die **ModelSpecificInfo** für das individuelle Modell angepasst werden muss.
- Wird das Modell mit dem TensorFlow Lite Model Maker erstellt, kann das tflite-Modell direkt mitsamt den Metadaten exportiert werden [6].

Im Fall der hier entwickelten App werden die Metadaten nachträglich über das angepasste Pythonskript hinzugefügt, um beim Trainieren des Modells keine API-Features zu verlieren, die Keras bietet. Im TensorFlow Lite Model Maker ist es zum Beispiel nicht möglich einen EarlyStopping-Callback beim trainieren hinzuzufügen, der in der Vergangenheit aber hilfreich war, um bei einer großen Anzahl an Epochen Overfitting zu verhindern.

8 Ausblick

Beim Testen mit Echtweltdaten in Abschnitt 6.2 und beim Testen der App wird klar, dass die Ergebnisse verbessert werden können, wenn eine Region of Interest betrachtet wird. So kann Hintergrund entfernt werden, der nicht für die Klassifizierung relevant ist. Bei der Untersuchung, wie der Datensatz für das Modell zu erzeugen ist, wurde auch der Ansatz der Stanford-Arbeit [1] betrachtet. Hier wurden Videos vom Gitarristen gemacht und nachträglich wurde die Akkordhand mit einem Object-Detection Modell extrahiert. Dieses Object-Detection Modell [9] wurde mit dem Egohands Dataset [10] trainiert. In Abbildung 14 wird das Object-Detection Modell auf einem Bild aus einem älteren Datensatz angewendet, um eine Bounding-Box zu malen, welche die Region of Interest darstellt.

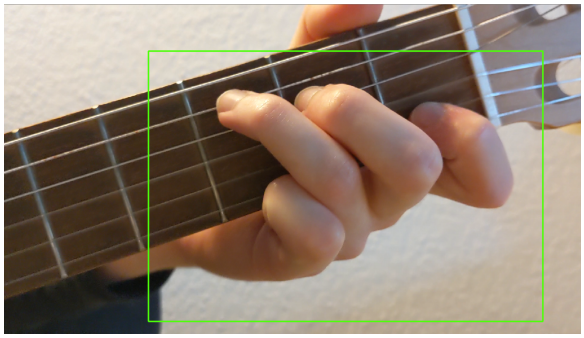

ABBILDUNG 14: Beispiel Region of Interest

Die Idee ist nun, das Object-Detection Modell nicht zum Erstellen der Trainingsdaten zu nutzen, sondern bei der Inferenz die Daten mit Hilfe der Object-Detection für die eigentliche Klassifizierung vorzubereiten. Dadurch wäre es auch möglich, die Kamera aus weiterer Entfernung zu nutzen, um Gitarrenakkorde zu erkennen.

In der Abbildung 14 ist erkennbar, dass der Daumen nicht Teil der Bounding-Box ist. Hier wird vom Modell nicht die ganze Akkordhand erkannt. Bei anderen Akkorden, wie dem D-Akkord, kann das zu Problemen führen, da dort charakteristisch ist, dass die Hälfte des Griffbretts frei ist. Das Object-Detection Modell müsste also auf den Anwendungsfall der Gitarrenakkorde angepasst werden.

Der hier präsentierte Ansatz dient lediglich der Unterscheidung der Grundformen der Gitarrenakkorde. Dass die Akkorde auch an anderen Positionen auf dem Griffbrett auftauchen können, wird nicht betrachtet. Eine Möglichkeit zu lokalisieren, in welchem Bund auf dem Griffbrett sich die Hand befindet, würde der Anwendung die Möglichkeit eröffnen, auch andere Akkorde zu bennenen, obwohl nur die schon bekannten Grundformen C, D, Em, F und G auf dem Griffbrett verschoben werden.

9 Github

Das Android Studio Projekt und die genutzten Jupyter Notebooks sind im folgenden Repository abgelegt:
`https://github.com/KnappSas/visual_guitar_chord_classifier`
In der Beschreibung wird auch auf den Datensatz verwiesen, der in Google Drive freigegeben wird.
Hinweis: Die Entwicklung des Machine-Learning Modells hat komplett in Google Colab stattgefunden und die Jupyter Notebooks wurden nur dort ausgeführt.

Literatur

[1] CNN Transfer Learning for Visual Guitar Chord Classification,
https://shawnbzhang.github.io/assets/PDFs/CS_230_Report.pdf

[2] Github zu CNN Transfer Learning for Visual Guitar Chord Classification,
https://github.com/leonkt/visual-guitar-chord-classifier

[3] Process Images for Fine-Tuned MobileNet with TensorFlow's Keras API,
https://deeplizard.com/learn/video/FNqp4ZY0wDY

[4] Transfer Learning and fine-tuning,
https://www.tensorflow.org/tutorials/images/transfer_learning

[5] TensorFlow Android Example,
https://github.com/tensorflow/examples/tree/master/lite/examples/image_classification/android

[6] TensorFlow Lite Model Maker,
https://www.tensorflow.org/lite/guide/model_maker

[7] Recognize Flowers with TensorFlow Lite on Android,
https://codelabs.developers.google.com/codelabs/recognize-flowers-with-tensorflow-on-android

[8] Adding metadata to TensorFlow Lite models,
https://www.tensorflow.org/lite/convert/metadata

[9] Real-time Hand-Detection using Neural Networks (SSD) on Tensorflow.,
https://github.com/victordibia/handtracking

[10] EgoHands: A Dataset for Hands in Complex Egocentric Interactions,
http://vision.soic.indiana.edu/projects/egohands/