Technische Dokumentation

Einleitung

Das inverse Oztaskop verarbeitet ein Eingabevideo und ersetzt die Tonspur mit einer neu generierten Tonspur. Diese basiert auf den Farben des Videos und wird von einem neuronalen Netz erzeugt. Wir haben uns hier bei den vortrainierten Modellen von magenta.js bedient. Speziell haben wir uns für ein Modell entschieden, welches auf Basis einer Eingabe eine Melodie fortführen kann.
Wir haben die BPM der generierten Tonspur auf 120 festgesetzt. Dies vereinfacht den Prozess der Bildanalyse und der Verarbeitung der Farben zu Eingabewerten für das neuronale Netz.

Python

Da das Projekt im jetzigen Zustand lokal auf dem Rechner des Nutzers laufen muss, ist es wichtig, dass jedes Video im Vorfeld analysiert wird, weil nicht jeder Nutzer einen Rechner besitzt, der eine Echtzeit-Verarbeitung zulässt.

Das Server-Backend, welches die erste Analyse der Videos durchführt, wurde in Python geschrieben. Dafür wurde als Framework Flask benutzt.
Es bietet neben der leichten Konfigurierung viele Funktionen, um die Anbindung an JavaScript und unser Frontend zu erleichtern.
Eine dieser Funktionen ist die Template-Engine Jinja, die von Flask standardmäßig verwendet wird, um HTML-Dokumente sichtbar darzustellen.
Jinja wurde für Python konzipiert und bietet neben Standardfunktionen einer solchen Template-Engine auch die Möglichkeit globale Variablen zu übergeben.
Dies erlaubt beispielsweise durch den Server gesteuerte Veränderung des HTML-Codes. So wird auch der Pfad zu unserer abzuspielenden Videodatei übergeben. Bei Änderungen der Quelldateien müssen nur an einer Stelle im Code Anpassungen vorgenommen werden. So werden auch die Ergebnisse der Videoanalyse ausgegeben.

Die Analyse wird in Python mit Hilfe der Bibliotheken OpenCV und Numpy durchgeführt. Es wird mit der Generierung eines VideoCapture-Objektes (im Code cap genannt, Kurzform für capture) begonnen. Dieses speichert den Pfad zu der Quelldatei und gibt mit dem Befehl cap.read() das nächste Bild bzw. den frame aus.
Neben dem frame wird auch ein boolean namens ret zurückgegeben. Dieser gibt nur an ob es einen return-Value gibt und wird daher dafür verwendet zu testen, ob noch Quellmaterial vorhanden ist.
Die Variable frame ist vom Typ Numpyarray, was eine Python spezifische Variante des normalen Arrays ist. Numpy ist konzipiert worden, um das Arbeiten mit großen Datenmengen zu erleichtern und bietet viele Funktionen, die es einem ermöglichen diese Arrays zu manipulieren. Darauf greift OpenCV zurück und gibt deswegen in Python Bilder in diesem Format aus. Der Numpyarray frame speichert seine Daten in einer Form, die in etwa so aussieht: frame[x][y] = (b, g, r). Dabei steht x für die Breite des Bildes in Pixeln, y für die Höhe und (b, g, r) ist ein Tupel mit den normalen RGB-Werten, die wegen Eigenheiten von OpenCV in umgekehrter Reihenfolge abgespeichert werden.

Um die Rechenzeit zu verkürzen, wird das Quellmaterial reduziert. Denn alleine bei einem Videoclip von 32 Sekunden Länge, einer Wiederholungsrate von 24 Bildern die Sekunde und einer Auflösung von 1280 x 720 Pixeln wären es 707.788.800 verschiedene RGB-Tupel.
Diese Reduktion des Materials geschieht in mehreren Schritten. Zuerst wird die Auflösung auf 32 x 16 Pixel geschrumpft. Normale Downscale-Algorithmen würden hierfür einen Pixelwert nehmen, meist einen der Eckpunkte, und dann davon ausgehen, dass dieser im ganzen Feld zu sehen war. Da dies zu ungenau ist, wird nicht direkt verkleinert, sondern es wird eine neues Numpyarray kreiert, das genauso aufgebaut ist wie das Ursprungsbild frame, also mit einer Seitenlänge von 32 x 16 Feldern. Danach wird in einem 40 x 45 Pixelfeld (1280/32 = 40, 720/16 = 45) im Original ein Durchschnittsfarbwert errechnet, der an der passenden Stelle in das neue Bild eingetragen wird. So reduzieren wir die Anzahl der Werte auf 393.216 verschiedene RGB-Tupel in dem 32 Sekunden Video.
Aufgrund der Frequenz, in der Akkorde geändert werden müssen, wird nur alle 8 Sekunden ein Wert übertragen. Darum wird jedes reduzierte Bild der letzten 8 Sekunden übereinandergelegt und der Durchschnitt pro Pixel in ein letztes Bild eingespeichert. So werden die Werte auf 2.048 Tupel beschränkt. Da diese Zahl zwar im Vergleich zu dem Quellmaterial verschwindend gering ist, aber nie so viel unsortierte Werte benötigt werden, werden jetzt alle Farbwerte verglichen und nur die 5 häufigsten Farben in jeder 8 Sekunden Periode werden gespeichert und an das Frontend übergeben. So wurden aus 707.788.800 verschiedenen unsortierten RGB-Tupeln in einem 32 Sekunden Video 20 RGB-Tupel, die nach Zeit und Häufigkeit sortiert sind.

JavaScript

Nach kurzer Recherche wurde uns schnell klar, dass es keine so leichte Aufgabe ist aus Farbwerten irgendwie Musik zu erzeugen.
Unsere beste Chance erkannten wir in der letzten Bonusaufgabe aus der Vorlesung, bei der man ein neuronales Netz, basierend auf vier Eingabeakkorden, Musik generieren lassen konnte. Das klang zwar nicht perfekt, aber lieferte teilweise schon musikähnliche Ergebnisse. So konnten wir das Problem weiter vereinfachen: „Wie erzeugt man aus Farbwerten Akkorde?“.
Zuerst haben wir uns den Code aus der Bonusaufgabe genau angeschaut, weil es da eine Funktion gab, die eingegebene Akkorde auf Gültigkeit überprüft hat, sprich ob sie in dieser Schreibweise existieren. Gelandet sind wir dann bei der Musikbibliothek tonal.js. Diese liefert für unsere Zwecke zwei nützliche Funktionen:

- Tonal.Note.names() -> (C, C#, Db, D, D#, …)
- Tonal.Scale.chords(“major”) -> (5, 64, M, M13, M6, M69, M7add13, …)

Also haben wir jetzt eine Funktion, die uns zu einer Tonleiter (bei tonal.js Scale genannt) passende Akkorde liefert. Die Akkorde brauchen dann nur noch eine Grundnote, die wir mit Hilfe der zweiten tonal-Funktion bekommen.
Aus allen Tonleitern, die es bei tonal.js gibt, haben wir uns insgesamt 15 ausgesucht: 8 moll- und 7 dur-Tonleitern. Um dem ganzen noch mehr musikalischen Charakter zu verleihen, haben wir uns zu jeder Tonleiter je zwei Instrumentenpaare überlegt. Diese bestehen aus einem Lead-Instrument und einem Bass-Instrument. Man findet eine Liste dieser Instrumente im JSON-Format auf einem Google-Server, von dem wir sie auch live abgreifen.

Um nun aus den Farben aus dem Eingabearray Akkorde zu generieren, konvertieren wir die Farben zunächst vom RGB-Farbraum in den HSL-Farbraum und berechnen Werte wie Durchschnitt und Median. In einer langen if-else-Kette mit Schwellenwerten, die aus Testvideos abgeleitet wurden, werden je eine Tonleiter und ein Instrumentenpaar ausgewählt. Das Instrumentenpaar wird in die passenden Arrays zur Übergabe in den nächsten Programmabschnitt geschrieben. Zusätzlich gibt es Zufallswerte, die entscheiden welches der beiden Instrumentenpaare, die zu einer Tonleiter gehören, verwendet wird. Anschließend werden die beiden Arrays mit den passenden Akkorden und den Grundtönen zufällig durchmischt und die ersten vier Akkorde in das finale Akkordarray geschrieben.
Wenn das gesamte Eingabearray mit den Farben verarbeitet wurde und die drei Arrays mit den Leadinstrumenten, den Bassinstrumenten und den finalen Akkorden gefüllt wurden, wartet das Programm darauf, dass der Nutzer den Play-Button drückt.
Die drei Arrays werden nun wieder aufgeteilt. Ein Instrumentenpaar und je vier Akkorde gehören zusammen zu einem Abschnitt. Für jeden Abschnitt wird die Funktion continueSequence() des Modells des neuronalen Netzes einmal aufgerufen. Diese Funktion führt die Melodie basierend auf den Akkorden weiter. Hier werden den Noten auch die entsprechenden Instrumente zugewiesen und in der Sequenz abgespeichert. Die Basslinie ist immer dieselbe und unterscheidet sich nur durch die Instrumente und die Tonhöhe, die wiederum auf den Eingabeakkorden basiert.
Beim letzten Aufruf der continueSequence-Funktion wird auch ein boolean-Wert mitgeschickt, der anzeigt, dass die Sequenz und das Video abgespielt werden sollen. Der Player lädt dann alle benötigten Samples für die Instrumente der Sequenz. Anschließend wird die Tonspur des Videos stumm geschaltet und das Video und die Sequenz gestartet. Zum Schluss und wenn die Sequenz fertig abgespielt wurde, werden alle wichtigen Variablen und das Video zurückgesetzt. Außerdem wird das Eingabearray mit den Farben erneut verarbeitet, um nicht die exakt gleiche Tonspur zweimal zu hören.