AVPRG-PROJEKT
Absolutes Gehör

Ein Deep Learning Versuch von Daniel Ollhoff

In diesem experimentellem Projekt, wollte ich herausfinden ob es mir möglich ist eine KI zu schreiben, die Musik anhand ihrer strukturellen Eigenheiten in ihre zugehörigen Genres klassifizieren kann. Das Konzept beinhaltete Anfangs die Idee vier Genres unterscheiden zu können, was schnell schrumpfte auf zwei. Weil man eine große Menge an Trainingsdaten benötigt. Immerhin hatte ich für zwei, nämlich Klassik und Lounge, in etwa 30 – 40 Songs pro Genre, zur Verfügung. Nun werde ich ein paar Begriffe vorstellen die für das Konzept einer KI von großer Wichtigkeit sind. Folgende Frameworks habe ich verwendet. Für das Neuronale Netz habe ich Keras verwendet, eine High Level API für Tensorflow. Für CSV Manipulation Pandas, sklearn für label encoding und skalieren der Spaltenwerte in der CSV. Librosa für die Audio-Analyse und Feature extraktion.

Input-Vektor:

Oft als (n, m, x) dargestellt, n = Reihen, m = Spalten usw., bei Bildern sieht der Vektor so aus → (Pixel, Pixel, Channel) . Als Beispiel habe ich in meinem Projekt einen Vektor von (1, 28). Kurz → ein Song und 28 Feature Spalten bzw. Werte die ihn beschreiben sollen.

Output/Label:

Beschreibt die Klasse, der der Input zugeordnet werden soll. Kann alles mögliche sein, bei mir sind es eben Genre.

Data-Set:

Beschreibt die gesamte Datenmenge die in ein KI-Programm eingespeist wird. Häufig sind es CSV, h5, TFRecord Dateien mit oft mehreren 1000 Zeilen in einer Datei. Wird immer unterteilt in Training-Set, Test-Set und Validation-Set. 60% train - 20% test - 20% validation ist eine Verteilung die sich Bewährt hat.

Training-Set:

Beschreibt die Daten die verwendet werden um den Algorithmus/“Activation Function“ zu trainieren bzw. die Gewichte zwischen den Neuronen anzupassen.

Test-Set:

Beschreibt den Datensatz den unser KI-Model „noch nie gesehen“ hat. Er wird dazu verwendet die Genauigkeit einer Vorhersage zu errechnen und passt die Parameter innerhalb eines Neuronalen Netzes nicht weiter an

Validation-Set:

Beschreibt den Datensatz der auch dazu verwendet wird unser Model zu trainieren allerdings kennt unser Model diese Daten auch noch nicht.

Loss:

Beschreibt sozusagen wie oft unser Model in der Trainingsphase daneben gelegen hat, mit seiner Output Vorhersage.Worauf die „Loss Function“ die Parameter im Model anpasst

Accuracy:

Genauigkeit der Vorhersage, wird als Fließkommazahl eines Intervalls zwischen null und eins dargestellt. Wird am ende eines kompletten Trainingsdurchlaufs angezeigt. Dazu muss ich erwähnen das 0.5 bzw 50% schlecht bzw. zufällig ist.

Sample:

Beschreibt ein Objekt unseres Datensatzes zum Beispiel ein Bild = ein Sample

Batch:

Ein Batch beschreibt einfach die Anzahl an Samples die in einer Runde in das Model gespeist werden.

Epoch:

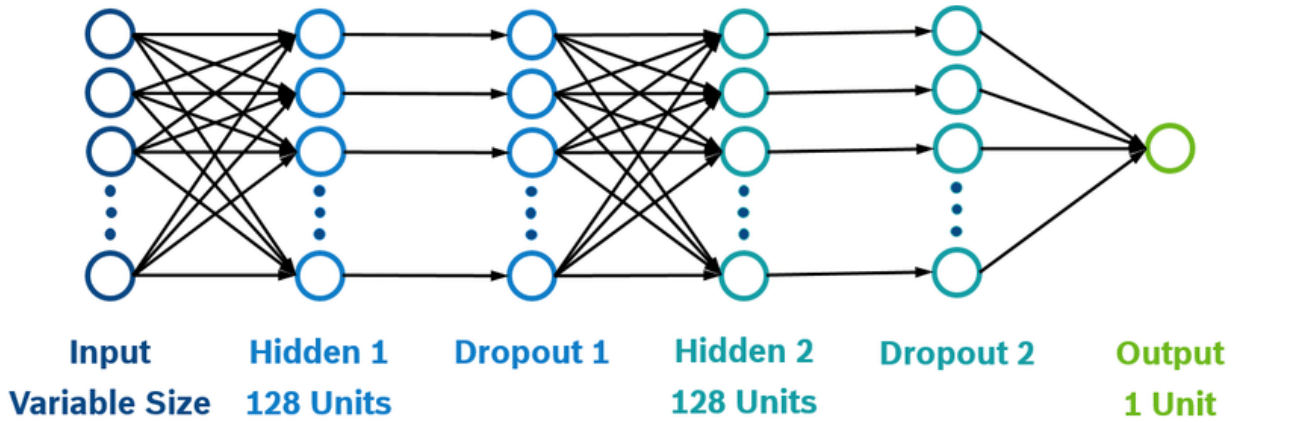
Eine Epoche ist vollzogen sobald alle Batches durchlaufen sind. So kann man mit Code zum Beispiel die Anweisung geben wie viele Epochen durchlaufen werden sollen bis der Test-Set zur Evaluierung der Vorhersagegenauigkeit eingesetzt wird.

Overfitting-Problem:

Beschreibt den Zustand dass das Training-Set eine hohe Genauigkeit aufweist aber das Test-Set eine niedrigere besitzt. Das Model also zur Vorhersage ungeeignet ist bzw. Daten schlecht einordnen kann, die es nicht kennt.

Mein Deep Learning Projekt besteht aus drei Scripts, komplett in Python geschrieben denen folgender Ablauf zugrunde liegt. Als allererstes markiert man alle zu einem Genre zugehörigen Songs drück F2(Umbenennen) und gibt z.B.: „Klassik-(1)“ als neuen Namen an. „-(1)“ führt dazu das nun alle markierten Dateien aufgezählt werden. Warum man das machen muss!? Klassik ist hier unser Output/Label. Anhand diesem werden wir später unterscheiden können welcher Song, zu welchem Genre gehört. Denn Pfad zu dem Song Ordner muss man leider manuell im Script selbst eintragen. Nun öffnet man die Bash Console und gibt „py slice.py“ ein.
Slice.py, wie der Name schon sagt, schneidet unsere Songs in 30 Sec. Blöcke – die Zahl 30 ist zufällig ausgewählt -, verwandelt Stereo zu Mono (Wichtig da Stereo mehr und doppelte Frequenzen besitzt im Vergleich zu mono), wandelt die Datei in .wav um (unkomprimiert → verschnellert die Feature Extraktion, kein Daten Verlust beim Samplen) und einen Window-Slide von 15 Sec. (Window-Slide: Song-1 geht bis 0:30, Song-2 fängt bei 0:15 an bis 0:45 usw.)
Window-Slide hab ich verwendet um ein eh schon kleinen Datensatz sozusagen zu strecken. Man würde hier auch von „Data Augmentation“ sprechen. Damit vergleichbar wäre zum Beispiel alle Bilder in einer Datenmenge zusätzlich noch zu drehen. Mit solche Prozeduren sorgt man unter anderem dafür das unser Model schneller konvergiert aber dazu später mehr.
Wieder alle Songs eines Genres markieren und wieder wie ob umbenennen.

So jetzt müsste man eine unterscheidbare Datenmenge haben.
Als nächstes wieder Bash öffnen und „py extract_audio.py [Pfad zum Ordner mit Song Dateien]“ eingeben.
Hier folgt nun die eigentliche Verarbeitung der Titeltracks. Im Prinzip holt sich das Script erst mal alle Dateien aus dem eingegebenen Ordner als Pfad Liste, mischt diese, was man auch wieder braucht um Overfitting vorzubeugen, gerade in meinem Fall wo sich zwei aufeinander folgende Songs durch Window-Slide ähneln. Eigentlich sind die Rückgabewerte der Librosa Feature-extraction Listen von Listen aber hier reduziere ich diese mit np.mean() was mir den Mittelwert zurück gibt. Warum? → Der Zentrale Grenzwertsatz Theorem hatte mich dazu bewogen.
Nun lädt es alle Songs einzeln, verarbeitet diesen und schreibt das Ergebnis in eine CSV Datei, bis die gesamten .wav Dateien in diesem Ordner durchlaufen sind.
Die Methode extract audio sorgt dafür, mithilfe von librosa, dass die extrahierten Features einzeln in ein Array geschrieben und in die CSV Datei geschrieben werden.
Hat das alles geklappt, können wir diese CSV Datei ganz einfach wieder per Bash Befehl „py model.py [Pfad zur CSV]“ in unser Model einpflegen.
Model.py kümmert sich dann erst mal um die Aufteilung des Datensatzes in Training-Set, Validation-Set und Test-Set. Außerdem skalieren wir die Werte einer jeden Spalte in unserem Datensatz so, dass sie in etwa einen gemeinsamen Wertebereich besitzen und weisen jedem Genre einen Integer Wert zu, in meinem Fall eben 0 für Klassik und 1 für Lounge. Das alles passiert in der Methode Preprocess in model.py
Bei meinem Model handelt es sich um ein Dense Neural Network kurz DNN. Es hat fünf Dense Schichten und zwei Dropout Schichten. Dropout Schichten haben den Zweck Overfitting vorzubeugen in dem, zufällig gewählt, Neuronen während des Trainieren abgeschaltet werden.



Nur um eine grobe Vorstellung zu geben, mein erster Dense Layer z.B.: – was hier Hidden heißt – hat 512 Outputs

Variabeln im Model wie „relu“ als activation function in den Dense Layern, „sigmoid“ im letzten Dense, „binary_crossentropy“ als loss function und „adam“ als optimization algorithm zu erklären würde mehrere Seiten beanspruchen, deswegen ganz kurz adam ist der „go-to“ Optimization Algorithmus im Deep Learning bereich. Sigmoid und Binary Crossentropy werden in den Fällen verwendet in denen zwischen zwei Output-Labeln enschieden wird, z.B.: ist Krebs – ist nicht Krebs. Relu oder rectified linear unit ist unsere activation function und ist auch wieder der „go-to“ zur Zeit.

Fazit

Gelungen ist das Projekt in meiner Hinsicht auf jedenfall, vor allem da ich immens viel über Machine Learning, Audio Verarbeitung und Statistische Modele gelernt habe. Es sind am Ende zwar nur zwei Genres geworden aber dafür ist Vorhersage bei ~95%!