בית ספר: מקיף יא' ראשונים

שם פרוייקט:Claptone – music creation

שם תלמיד: רן פרידמן

מספר תעודת זהות: 328201371

שם המנחה: דינה קראוס

תאריך: 6/6/2024

תוכן

מבוא

רקע לפרוייקט

בפרויקט זה נעסוק בתחום של יצירת מוזיקה אוטומטית באמצעות רשתות נוירונים חוזרות (RNN). תחום זה משלב בין מדעי המחשב, למידת מכונה ומוזיקה, ומציע דרכים חדשות ליצירת תוכן אמנותי באמצעות אלגוריתמים מתקדמים.

יצירת מוזיקה באופן אוטומטי באמצעות למידת מכונה הוא תחום חדשני ומתפתח שמציע הזדמנויות חדשות עבור מוזיקאים, מפיקים ואמנים. שימוש ב RNN-ליצירת מוזיקה מאפשר למכונות ללמוד ולהבין דפוסים מוזיקליים מורכבים, ובכך לייצר מוזיקה חדשה ומקורית. המטרה העיקרית של הפרויקט היא לפתח מערכת שיכולה לייצר מוזיקה חדשה על בסיס נתוני מוזיקה קיימים, לחקור את טכנולוגיות הלמידת מכונה המתקדמות ולהפוך את התהליך לפשוט ונגיש למשתמשים שאינם מומחים בתחום.

בפרויקט זה נעשה שימוש ברשתות נוירונים חוזרות מסוג LSTM (Long Short-Term Memory) שהן טכנולוגיה מתקדמת בלמידת מכונה, המתאימה במיוחד לעיבוד רצפים, כגון תווים מוזיקליים. רשתות אלו יכולות ללמוד דפוסים מורכבים ולזכור מידע לאורך רצף ארוך, מה שהופך אותן לאידיאליות עבור יצירת מוזיקה.

תהליך המחקר

כיום, השוק של יצירת מוזיקה באמצעות בינה מלאכותית (AI) ורשתות נוירונים חוזרות (RNN) מתפתח במהירות. ישנם מספר מוצרים מתקדמים כמו OpenAi jukebox,Amper Music ו Magenta שמאפשרים לאנשים ליצור מוזיקה במספר סגנונות.

הכלים הללו בדרך כלל בעלי ממשק מסובך ומיועדים לחברות ומפיקי מוזיקה ולא לאדם הפרטי שמעוניין ליצור מוזיקה.

השתמשתי במקורות מידע כמו StackOverflow, GitHub כדי ללמוד ולפתור בעיות טכניות בפרויקט. הדוגמאות והפתרונות שנמצאו בפלטפורמות אלו עזרו להבין את האתגרים וליישם פתרונות לבעיות בפרויקט. הפרויקט פותח על בסיס פרויקטים קיימים, תוך התאמה אישית ושיפור של הקוד והאלגוריתמים כדי להתאים לצרכים הספציפיים של יצירת מוזיקה אוטומטית.

בפרויקט יש שימוש ברשת מסוג RNN ובשכבות LSTM(long short-term memory).
בנוסף מבוצע שימוש בפרוטוקול MIDI (Musical Instrument Digital Interface) אשר מאפשר תרגום של מונחים מוזיקלים לממשק שניתן לבצע בו שינויים ולפרש אותו.

אתגרים

במהלך הפרויקט, אחד האתגרים העיקריים היה לבחור את סוג רשת הנוירונים המתאימה ביותר. לאחר מחקר מעמיק והשוואה בין אפשרויות שונות, נבחרה רשת נוירונים חוזרת (RNN)מסוג LSTM בשל יכולתה להתמודד עם נתוני רצף ולהבין דפוסים מוזיקליים מורכבים לאורך זמן.

העומס הלימודי היווה אתגר משמעותי נוסף. ניהול הזמן בין המטלות הלימודיות והעבודה על הפרויקט דרש תכנון קפדני וסדר עדיפויות, במיוחד בהתחשב במשימות המורכבות הכרוכות באימון מודלים של למידת מכונה.

נפגשתי עם בעיות טכניות רבות בתהליך הפיתוח, כולל באגים בתוכנה ותקלות בביצועים. האימון על רשתות נוירונים דורש משאבי מחשוב גבוהים וזמן רב, ולכן היה צורך למצוא פתרונות אופטימיזציה כמו שימוש ב GPU-ב Google Colab-כדי להאיץ את התהליך.

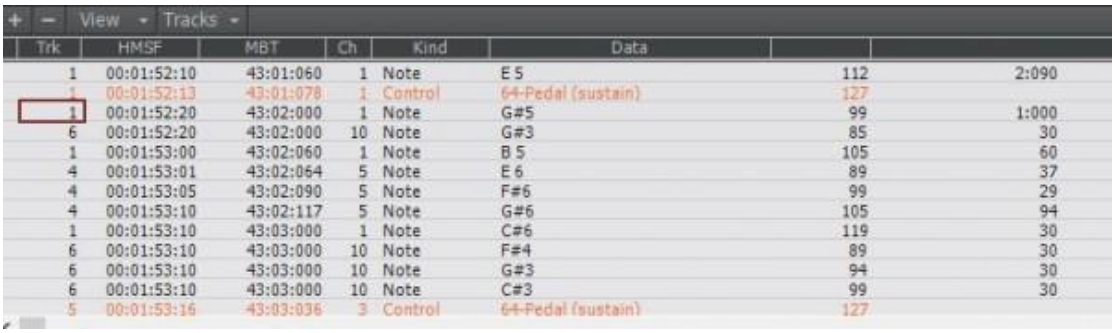
ארכיטקטורה

שלב איסוף הכנה וניתוח הנתונים

בפרויקט אני עושה שימוש בDataSet הנקרא Maestro, הוא נוצר על ידי Google בעבור הפרויקט Magenta ויש בו מעל 200 שעות של מוזיקה. הDataset מחולק לקבצי Wav שבהם אני לא מצבע שימוש בפרויקט ובקבצי Midi שהם לב ליבו של המאגר נתונים.

במאגר ישנם 1200 קבצי MIDI שונים באורך ממוצע של 565 שניות(כמעט 10 דקות), היצירות הן של מלחינים שונים כמו רוברט שוומן, סרגיי רחמנינוב, בטהובן, טצ'יקובסקי, ברהמס ואחרים.

המאגר מחולק לשם הקובץ, שם המלחין, אורך היצירה. קובץ Midi בנוי ממבנה של תווים שמאכסן את סוג הצליל, הכלי, אורך הצליל ומתי הוא מתחיל להתנגן.



Trk	HMSF	MBT	Ch	Kind	Data		
1	00:01:52:10	43:01:060	1	Note	E 5	112	2:090
1	00:01:52:13	43:01:078	1	Control	64-Pedal (sustain)	127	
1	00:01:52:20	43:02:000	1	Note	G#5	99	1:000
6	00:01:52:20	43:02:000	10	Note	G#3	85	30
1	00:01:53:00	43:02:060	1	Note	B 5	105	60
4	00:01:53:01	43:02:064	5	Note	E 6	89	37
4	00:01:53:05	43:02:090	5	Note	F#6	99	29
4	00:01:53:10	43:02:117	5	Note	G#6	105	94
1	00:01:53:10	43:03:000	1	Note	C#6	119	30
6	00:01:53:10	43:03:000	10	Note	F#4	89	30
6	00:01:53:10	43:03:000	10	Note	G#3	94	30
6	00:01:53:10	43:03:000	10	Note	C#3	99	30
5	00:01:53:16	43:03:036	3	Control	64-Pedal (sustain)	127	

ויזואליזציה של המבנה של פורמט MIDI

את קבצי הMIDI אנחנו מעוניינים להעביר לנתונים שעליהם אנחנו יכולים לגשת ולבצע חישובים, לשם כך מתבצע העברה של הנתונים לטבלה שמאכסנת את המידע על התווים. שגרה זו מתבצעת על ידי הפעולה midi_to_notes(midi_file) אשר מקבלת כקלט קובץ MIDI ומחזירה כפלט מופע של pandas DataFrame שמאכסן את הנתונים של הקובץ.

המודל ואימון בנייה שלב

תיאור של קלט פלט האלגוריתם -

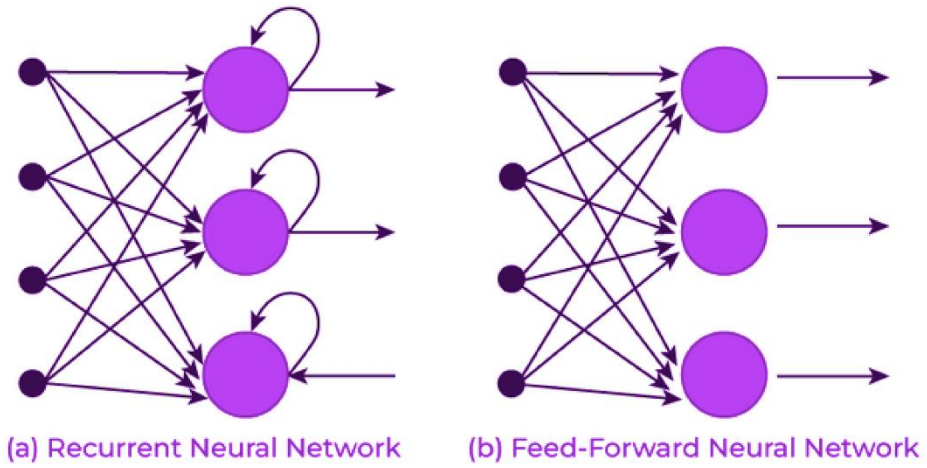

```
Model: "model_21"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape                 Param #   Connected to
==================================================================================================
 input_22 (InputLayer)          [(None, 20, 3)]              0         []

 lstm_21 (LSTM)                 (None, 128)                  67584     ['input_22[0][0]']

 duration (Dense)               (None, 1)                    129       ['lstm_21[0][0]']

 pitch (Dense)                  (None, 64)                   8256      ['lstm_21[0][0]']

 step (Dense)                   (None, 1)                    129       ['lstm_21[0][0]']

==================================================================================================
Total params: 76098 (297.26 KB)
Trainable params: 76098 (297.26 KB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

בשכבת הInput מתקבלים קבצי MIDI, את הקבצים אנחנו ממירים לטבלה שבה שמורים הנתונים ומוצמדים אחד לשני. על ידי מיון של התווים בקובץ הMIDI מכניסים לטבלה את זמני ההתחלה והסוף של כל תו, את הצליל שלו. כך מבצעים פעולה זו בעבור כל הקבצים.

	pitch	start	end	step	duration
0	60	0.967448	0.994792	0.000000	0.027344
1	51	0.973958	1.028646	0.006510	0.054688
2	48	0.973958	1.029948	0.000000	0.055990
3	55	0.975260	1.020833	0.001302	0.045573
4	60	1.024740	1.082031	0.049479	0.057292

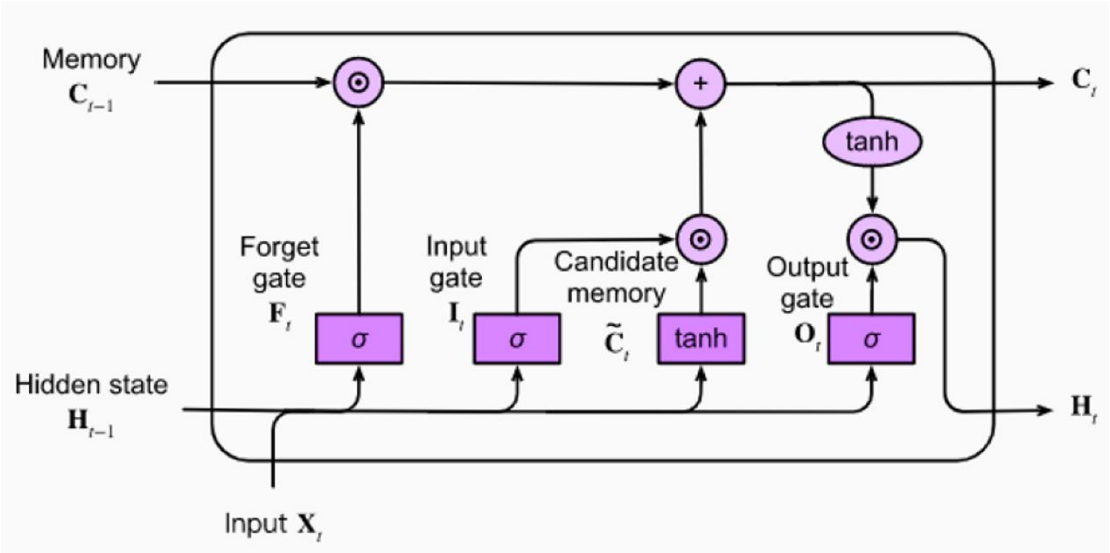
,

לאחר תרגום של קובצי הMIDI לנתונים הנתונים מחולקים לשלושה קטגוריות, Duration, Pitch ו step. כל הנתונים הללו מוכנסים לתוך שכבת LSTM. אנחנו משתמשים בLSTM משום שבמוזיקה יש חשיבות לצלילים הקודמים. מוזיקה עם לא חיבור של קטעי מוזיקה ישנים, אלא יש הקשר מסוים שבו כל צליל יכול להישמע. רק בהתבסוסות על הצלילים הקודמים ניתן ליצר צלילים חדשים שיתאימו וישמעו הגיוני ביחס לקודם. לשם כך נשתמש ברשת נוירונים חוזרת ונשנת(RNN) שמשתמשת בפלט שלה כקלט כדי להמשיך וליצר יצירה שלמה.



תיאור של ההבדל בין סוגי רשתות הנוירונים. A היא זו שבה אני משתמש.

ברשת LSTM לנוירונים יש גם ערך פנימי וגם יש חישוב על פי הפלט האחרון, הערך הפנימי(State) מתעדכן אף הוא לא רק בשלב הסופי של הTrain אלא גם בזמן ההרצה. LSTM נחשבת כמודל יעיל מאוד ובעל ביצועים טוב מאוד.

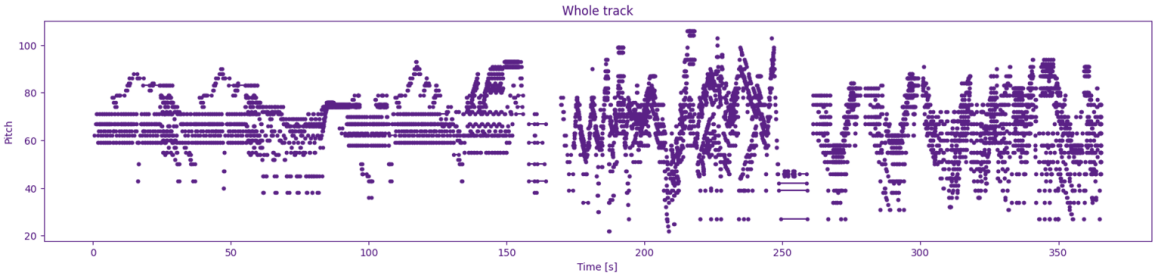


תרשמים מבנה שכבות הLSTM

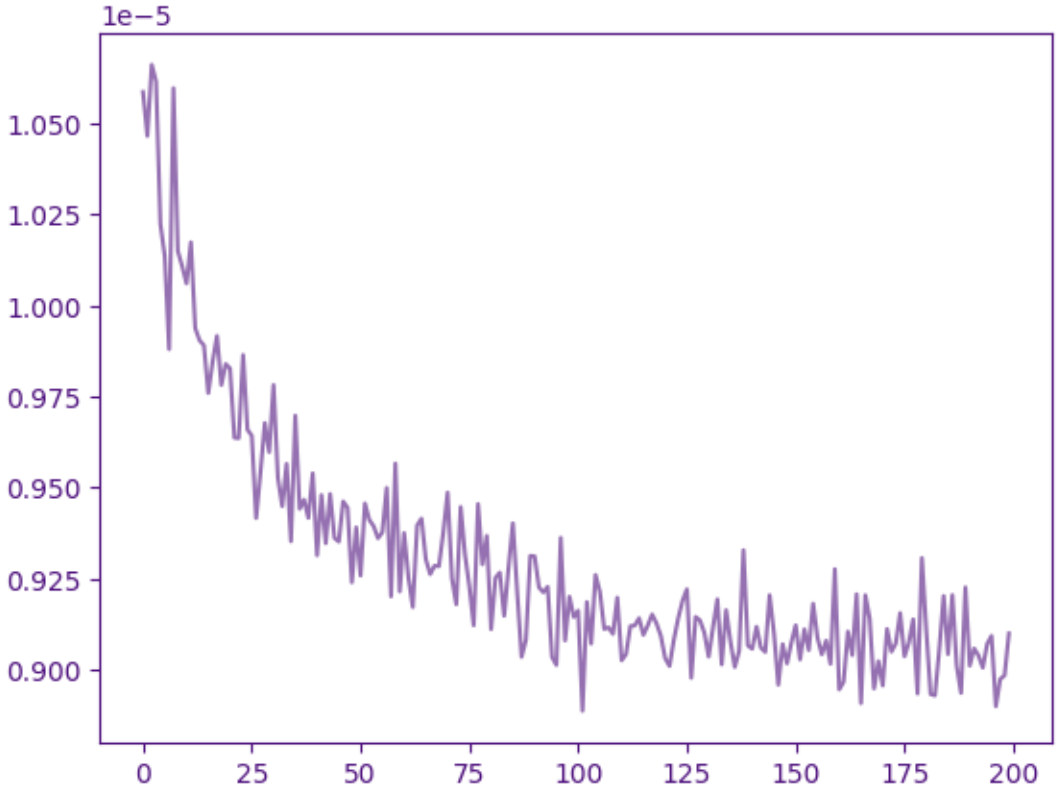
בנוסף לשכבה זו אנחנו מבצעים העברה של הנתונים שצינתי קודם(Pitch,Duration ו step) ומחזירים אותם למבנה אחיד שיהיה אפשר לפרש אותו ולתרגם אותו בחזרה לקובץ MIDI. לשם כך אנחנו משתמשים בשכבת Dense.

שכבת Dense היא רכיב בסיסי ברשתות נוירונים. בכל נוירון בשכבת Dense מתקבל קלט מכל הנוירונים בשכבה הקודמת, מה שמבטיח מקסימום קישוריות. שכבה זו מבצעת סכימה משוקללת של הקלטים, מוסיפה ערך הטיה, ואז מעבירה את התוצאה דרך פונקציית הפעלה. שכבות Dense חיוניות ללמידת ייצוגים מורכבים ומשמשות לרוב בשלבים הסופיים של רשת נוירונים לביצוע משימות סיווג או רגרסיה.

לבסוף כלל הנתונים מועברים לטבלה זהה לזו שבנינו בשכבת הקלט ובה שמור התוצר.
את כל התווים השמורים בטבלה זו כעת צריך להעביר בחזרה לקובץ MIDI.

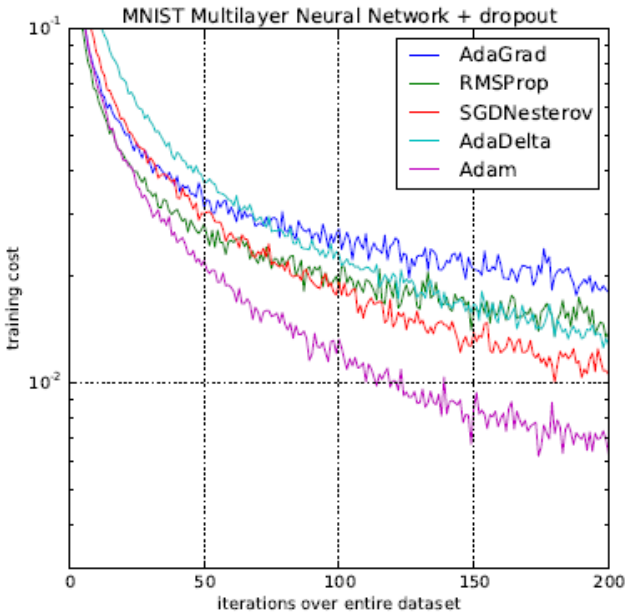


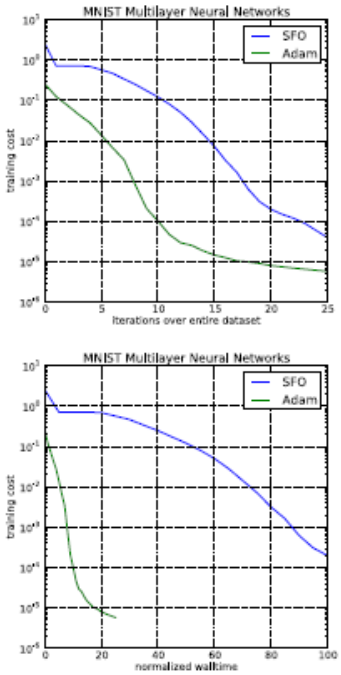
גרף של כל התווים, צליל כתלות בזמן.



גרף Loss כתלות במספר ה.Epoch

בפרויקט אני משתמש ב Adam-קיצור של Adaptive Moment Estimation אופטימזייר . Adam הוא מאיץ שמשלב את היתרונות של שני מאיצים אחרים AdaGrad ו RMSProp- הוא מתאים את קצב הלמידה של כל פרמטר באופן דינמי, על בסיס אומדן רגעי של ממוצע הנתונים. השימוש ב Adam-מאפשר אימון מהיר ויעיל יותר של המודל, וכתוצאה מכך השגת ביצועים טובים יותר במגוון רחב של משימות למידת מכונה, כולל רשתות נוירונים חוזרות (RNN)ליצירת מוזיקה.





פירוט על היעילות של ADAM ביחס לשאר הoptimizers בשוק.

שלב היישום

תיאור כללי

בממשק המשתמש יש למשתמש יש מספר אפשרויות:

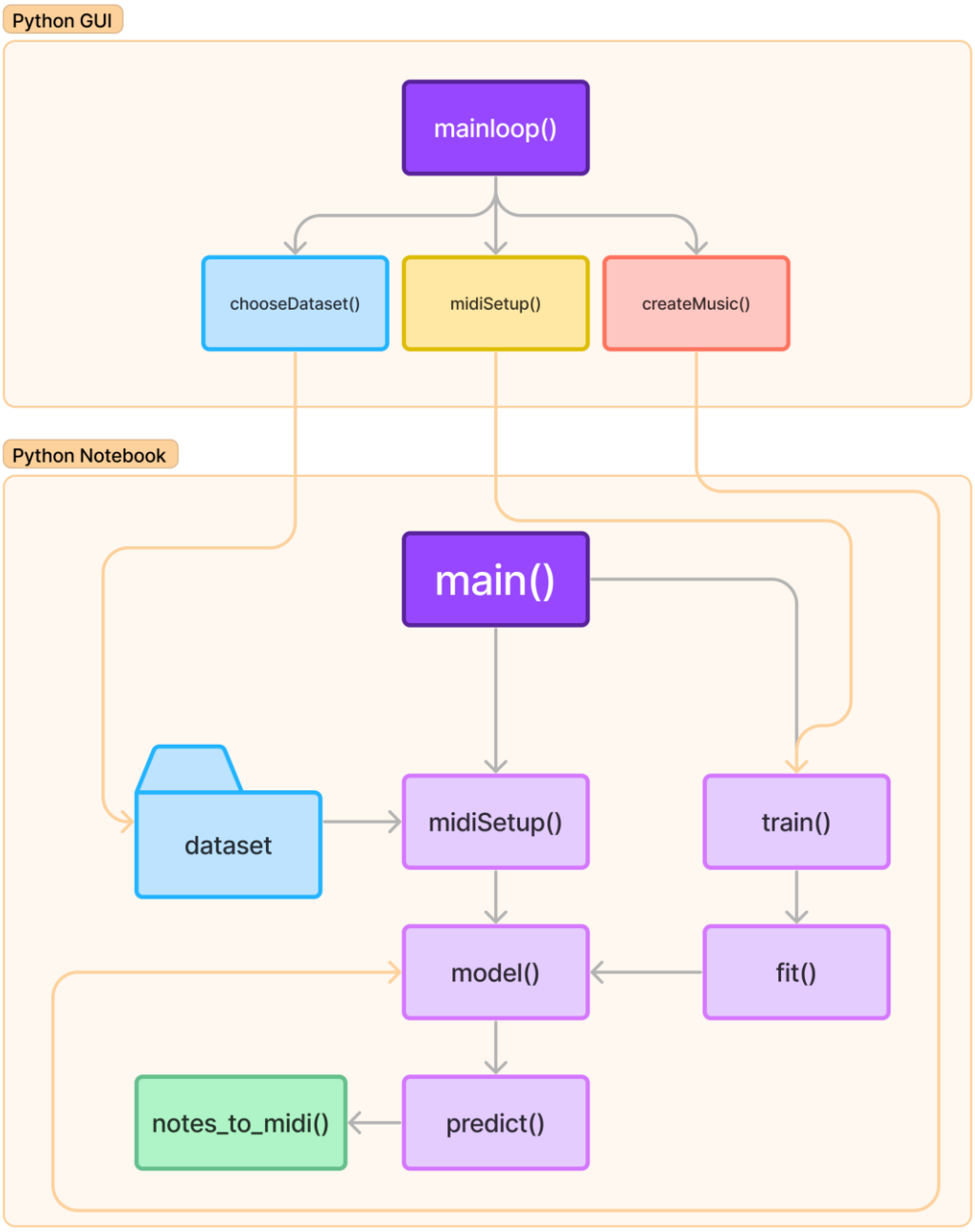
- לסמן את הDataset
- לאמן מחדש את הTrain
- ליצור מוזיקה חדשה

במידה והוא בוחר באפשרות הראשונה אז נפתח למשתמש תפריט בחירה של מיקום במחשב.

במידה והמשתמש בוחר באפשרות השנייה, התוכנה מריצה מחדש את הTrain.

במידה והוא בוחר באפשרות השלישית, נפתח דיאלוג שבו המשתמש מזין את הפרטים על הפרויקט ויוצר את היצירה המוזיקלית.

תיאור כללי



תיאור טכנולוגיה של הממשק

כדי ליצור ממשק משתמש שבו יהיה ניתן לבצע פעולות וליצור מוזיקה בחרתי להשתמש בספרייה Tkinter, שבה ניתן ליצור GUI - Graphical user interface.

יש מספר סיבות טובות ששבגללן בחרתי להשתמש בTkinter:

- הספרייה מובנית בפייתון
- תומכת בכל הפלטפורמות המרכזיות(Windows, macOS, Linux)
- Tkinterמספקת ביצועים טובים לאפליקציות קטנות ובינוניות
- ישנה קהילה גדולה ותומכת של מפתחים שמשתמשים בTkinter

תיאור קוד הקולט את הDATA

הDATA המתקבל כפלט מבחינת המשתמש הוא אורך של היצירה. לשם כך במשק ישנו Input Entry שאליו המשתמש מזין את אורך היצירה ובנוסף יכול לבחור את הכלי נגינה שבה הוא ינוגן.

מדריך למפתח

חלוקת הפרויקט

הפרוייקט שלי מחולק לשני קבצים מרכזיים

- Python Notebook – שאחראי על כל הרשת נוירונים ומחולק לSections
- Python File – שאחראי על ממשק המשתמש GUI

תדפיס הקוד של קובץ הGUI

```
from tkinter import *
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
from tkinter.simpledialog import askstring

from claptone import *
window = Tk()

#Colors#################
elementColor = "#8a32db"
textColor = "white"
mainColor = "#222222"
lighterMainColor = "#333333"
SecondColor = "#444444"
#Colors#################


instruments = [
   "Acoustic Grand Piano", "Bright Acoustic Piano", "Electric Grand Piano", "Honky-tonk Piano",
   "Rhodes Piano", "Chorused Piano", "Harpsichord", "Clavinet", "Celesta", "Glockenspiel",
   "Music box", "Vibraphone", "Marimba", "Xylophone", "Tubular Bells", "Dulcimer",
   "Hammond Organ", "Percussive Organ", "Rock Organ", "Church Organ", "Reed Organ", "Accordion",
   "Harmonica", "Tango Accordion", "Acoustic Guitar (nylon)", "Acoustic Guitar (steel)",
   "Electric Guitar (jazz)", "Electric Guitar (clean)", "Electric Guitar (muted)", "Overdriven Guitar",
   "Distortion Guitar", "Guitar Harmonics", "Acoustic Bass", "Electric Bass (finger)",
   "Electric Bass (pick)", "Fretless Bass", "Slap Bass 1", "Slap Bass 2", "Synth Bass 1", "Synth Bass 2",
   "Violin", "Viola", "Cello", "Contrabass", "Tremolo Strings", "Pizzicato Strings", "Orchestral Harp",
   "Timpani", "String Ensemble 1", "String Ensemble 2", "Synth Strings 1", "Synth Strings 2",
   "Choir Aahs", "Voice Oohs", "Synth Voice", "Orchestra Hit", "Trumpet", "Trombone", "Tuba",
   "Muted Trumpet", "French Horn", "Brass Section", "Synth Brass 1", "Synth Brass 2", "Soprano Sax",
   "Alto Sax", "Tenor Sax", "Baritone Sax", "Oboe", "English Horn", "Bassoon", "Clarinet", "Piccolo",
   "Flute", "Recorder", "Pan Flute", "Bottle Blow", "Shakuhachi", "Whistle", "Ocarina",
   "Lead 1 (square)", "Lead 2 (sawtooth)", "Lead 3 (calliope lead)", "Lead 4 (chiffer lead)",
   "Lead 5 (charang)", "Lead 6 (voice)", "Lead 7 (fifths)", "Lead 8 (brass + lead)", "Pad 1 (new age)",
   "Pad 2 (warm)", "Pad 3 (polysynth)", "Pad 4 (choir)", "Pad 5 (bowed)", "Pad 6 (metallic)", "Pad 7 (halo)",
   "Pad 8 (sweep)", "FX 1 (rain)", "FX 2 (soundtrack)", "FX 3 (crystal)", "FX 4 (atmosphere)", "FX 5 (brightness)",
   "FX 6 (goblins)", "FX 7 (echoes)", "FX 8 (sci-fi)", "Sitar", "Banjo", "Shamisen", "Koto", "Kalimba",
   "Bagpipe", "Fiddle", "Shana", "Tinkle Bell", "Agogo", "Steel Drums", "Woodblock", "Taiko Drum",
   "Melodic Tom", "Synth Drum", "Reverse Cymbal", "Guitar Fret Noise", "Breath Noise", "Seashore",
   "Bird Tweet", "Telephone Ring", "Helicopter", "Applause", "Gunshot"
]

isThereTab = False
tabControl = NONE
instrumentName = StringVar()
style = ttk.Style()
inputLenght = None
style.theme_create( "yummy", parent="alt", settings={
        "TNotebook": {"configure": {"tabmargins": [2, 5, 2, 0] ,"background": lighterMainColor,"title_color":"blue", "selected_title_color":"red"} },
        "TNotebook.Tab": {
            "configure": {"padding": [5, 1], "background": SecondColor,"foreground":elementColor },
            "map":       {"background": [("selected", mainColor)],
                         "expand": [("selected", [1, 1, 1, 0])]} } } )

style.theme_use("yummy")

def createMusic():
   e_text=inputLenght.get()
   if(not e_text.isnumeric() ):
      messagebox.showerror("Error!", "Lenght has to be a Integer")
      return
   e_text = int(e_text)
   print(e_text)
   if(e_text<=0 ):
      messagebox.showerror("Error!", "Lenght has to be a positive number")
      return
   if(e_text>100 ):
      messagebox.showerror("Error!", "Lenght can't be above 100s")
      return
   if(instrumentName.get()=="" ):
      messagebox.showerror("Error!", "Please select an instrument")
      return
   print(instrumentName.get())
   notes_to_midi(generated_notes,"w.mid",instrumentName.get())
```

```
def createOptionsSection(frame):
    global inputLenght
    l1 = Label(frame,text="Enter Paremeters",width=20,fg=elementColor,bg=mainColor, font =
            ('calibri', 35, 'bold','underline'))
    lenghtOfFile = 30
    # l1 = Label(frame,textvariable=v,width=10,fg=elementColor)
    l1.place(relx=0.5, rely=0.1, anchor=CENTER)
    l2 = Label(frame,text="Whats the lenght of the music peice?",width=30,fg=textColor,bg=mainColor, font =
            ('calibri', 30))
    l2.place(relx=0.5, rely=0.2, anchor=CENTER)


    inputLenght = Entry(window,textvariable = lenghtOfFile, font=('calibre',15,'normal'),bg=mainColor,fg=textColor,width=10)
    inputLenght.config(highlightbackground=elementColor)
    inputLenght.place(relx=0.5, rely=0.27, anchor=CENTER)

    l2 = Label(frame,text="Whats instrument do you want to play",width=40,fg=textColor,bg=mainColor, font =
            ('calibri', 30))
    l2.place(relx=0.5, rely=0.32, anchor=CENTER)
    combo = ttk.Combobox(
        state="readonly",
        values=instruments,
        textvariable = instrumentName
    )
    combo.place(relx=0.5, rely=0.38, anchor=CENTER)

    createBtn = Button(text="Create",width=20,fg=elementColor,bg=mainColor,command=createMusic, font =
            ('calibri', 20, 'bold'))
    createBtn.place(relx=0.5, rely=0.45, anchor=CENTER)
def createTab():
    name = askstring("New Project","What is the Name of the project?")
    if name == "": return
    if not isinstance(name, str): return
    global isThereTab
    global tabControl
    if not isThereTab:
        tabControl = ttk.Notebook(window)
        tabControl.place(width=1600, height=900)
        isThereTab = True

    tab1 = Frame(tabControl, bg= mainColor)
    tabControl.add(tab1, text=name)

    createOptionsSection(tab1)

def firstTab():
    createTab()


window.title("Claptone")
# window.resizable(False, False)
canvas1 = Frame(window, width=1600, height=900,bg="#222222")
canvas1.pack()
window.config(bg='#222222')
commadMenu =  Menu(window)
window.configure(menu=commadMenu)
filemenue = Menu(commadMenu, tearoff=0)
filemenue.add_command(label="Create another music peice",command=createTab)
filemenue.add_command(label="Train",command=createTab)
filemenue.add_command(label="Locate Dataset",command=createTab)
commadMenu.add_cascade(label="File", menu=filemenue)

B = Button(canvas1, text ="Create your\n first sound!",fg=textColor,bg=elementColor, font =
            ('calibri', 20, 'bold') ,command = firstTab ,borderwidth = 0,width=15,height=3)

B2 = Button(canvas1, text ="Train the model",fg=textColor,bg=elementColor, font =
            ('calibri', 20, 'bold') ,command = None ,borderwidth = 0,width=15,height=3)

B3 = Button(canvas1, text ="Locate Data",fg=textColor,bg=elementColor, font =
            ('calibri', 20, 'bold') ,command = None ,borderwidth = 0,width=15,height=3)

B.place(relx=0.3, rely=0.5, anchor=CENTER)
B2.place(relx=0.7, rely=0.5, anchor=CENTER)
B3.place(relx=0.5, rely=0.5, anchor=CENTER)

window.wm_attributes('-transparentcolor','black')

def showError():
    messagebox.showerror("Error!", "There is an error in your input")

mainloop()
```

תדפיס קוד של קובץ הNotebook

```
!pip install pretty_midi
!sudo apt install -y fluidsynth
!pip install --upgrade pyfluidsynth

import numpy as np
import tensorflow as tf
import pandas as pd
import collections
import fluidsynth
# import glob
import pretty_midi
from IPython import display
from typing import Dict, List, Optional, Sequence, Tuple
import os
from matplotlib import pyplot as plt

"""# MIDI Setup

"""


dir_path = "/content/"
filenames = next(os.walk(dir_path), (None, None, []))[2]  # [] if no file
print(filenames)
filenames = [dir_path+"/"+s for s in filenames]

def plot_piano_roll(notes: pd.DataFrame, count: Optional[int] = None):
  if count:
    title = f'First {count} notes'
  else:
    title = f'Whole track'
    count = len(notes['pitch'])
  plt.figure(figsize=(20, 4))
  plot_pitch = np.stack([notes['pitch'], notes['pitch']], axis=0)
```

```
  plot_start_stop = np.stack([notes['start'], notes['end']], axis=0)
  plt.plot(
      plot_start_stop[:, :count], plot_pitch[:, :count], color="b", marker=".")
  plt.xlabel('Time [s]')
  plt.ylabel('Pitch')
  _ = plt.title(title)

sample_file = filenames[33]
def midi_to_notes(midi_file):
    pm = pretty_midi.PrettyMIDI(midi_file)
    instrument = pm.instruments[0]
    notes = collections.defaultdict(list)
    sorted_notes = sorted(instrument.notes , key=lambda note:note.start)
    prev_start = sorted_notes[0].start

    for note in sorted_notes:
        start = note.start
        end = note.end
        notes["pitch"].append(note.pitch)
        notes["start"].append(start)
        notes["end"].append(end)
        notes["step"].append(start - prev_start)
        notes["duration"].append(end - start)
        prev_start = start
    return pd.DataFrame({name:np.array(value) for name,value in notes.items()})

raw_notes = midi_to_notes(sample_file)
print(raw_notes)
note_names = np.vectorize(pretty_midi.note_number_to_name)
sample_note_names = note_names(raw_notes["pitch"])
plot_piano_roll(raw_notes)

"""# Train"""

num_files = 5
all_notes = []
for f in filenames[:num_files] :
    notes = midi_to_notes(f)
    all_notes.append(notes)
all_notes = pd.concat(all_notes)
print(all_notes)
key_order = ["pitch" , "step" , "duration"]
train_notes = np.stack([all_notes[key] for key in key_order] , axis = 1)
notes_ds=tf.data.Dataset.from_tensor_slices(train_notes)
notes_ds.element_spec

seq_length = 20
vocab_size = 128
def create_sequences(dataset,seq_length,vocab_size=128):
  sequences = []
  targets = []
  num_seq = train_notes.shape[0] - seq_length
  for i in range(num_seq):
    sequence = train_notes[i:i+seq_length - 1,:] / [vocab_size, 1 ,1]
    target = train_notes[i+seq_length] / vocab_size
    sequences.append(sequence)
    targets.append(target)
  sequences = np.array(sequences)
  targets = np.array(targets)
  print(sequences.shape , targets.shape)
  dataset = tf.data.Dataset.from_tensor_slices((sequences,{"pitch":targets[:,0] , "step":targets[:,1] ,"duration" :targets[:,2]}))
  return dataset
seq_ds = create_sequences(notes_ds, 21, vocab_size)
batch_size =64
buffer_size = 5000
train_ds = seq_ds.shuffle(buffer_size).batch(batch_size)
train_ds.element_spec

layer = tf.keras.layers
learning_rate = 0.005
input_data = tf.keras.Input(shape=(seq_length , 3))
x= layer.LSTM(128)(input_data)
outputs = {
  "pitch":tf.keras.layers.Dense(64 , name = "pitch")(x),
  "step":tf.keras.layers.Dense(1 , name = "step")(x),
  "duration":tf.keras.layers.Dense(1 , name = "duration")(x),
}
model = tf.keras.Model(input_data , outputs)

loss ={
  "pitch" : tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True),
  "step": tf.keras.losses.MeanSquaredError(),
  "duration":tf.keras.losses.MeanSquaredError(),
}
optimizer = tf.keras.optimizers.Adam(learning_rate = learning_rate)
model.compile(loss=loss , loss_weights={
    'pitch': 0.05,
    'step': 1.0,
    'duration':1.0,
  }, optimizer = optimizer)

model.summary()

# Commented out IPython magic to ensure Python compatibility.
# %%time
# epochs = 200
#
# history = model.fit(
#     train_ds,
#     epochs=epochs,
# )
#
# plt.plot(history.epoch, history.history['loss'], label='total loss')
# plt.show()

"""# Run the model"""

def predict_next_note(
notes , keras_model , temperature):

  assert temperature > 0
  inputs = np.expand_dims(notes , 0)
  predictions = model.predict(inputs)
  pitch_logits = predictions['pitch']
  step = predictions["step"]
  duration = predictions["duration"]
  pitch_logits /= temperature
  pitch = tf.random.categorical(pitch_logits , num_samples = 1)
  pitch = tf.squeeze(pitch , axis = -1)
  duration = tf.squeeze(duration , axis =-1)
  step = tf.squeeze(step,axis = -1)
```

```
  step = tf.maximum(0,step)
  duration = tf.maximum(0 , duration)
  return int(pitch) , float(step) , float(duration)

temperature = 2.0
num_predictions = 120
seq_length = 20
vocab_size= 128
sample_notes = np.stack([raw_notes[key] for key in key_order], axis=1)

# The initial sequence of notes and the pitch is normalized similar to training sequences
input_notes = (
  sample_notes[:seq_length] / np.array([vocab_size, 1, 1]))

generated_notes = []
prev_start = 0
for _ in range(num_predictions):
  pitch, step, duration = predict_next_note(input_notes, model, temperature)
  start = prev_start + step
  end = start + duration
  input_note = (pitch, step, duration)
  generated_notes.append((*input_note, start, end))
  input_notes = np.delete(input_notes, 0, axis=0)
  input_notes = np.append(input_notes, np.expand_dims(input_note, 0), axis=0)
  prev_start = start

generated_notes = pd.DataFrame(
  generated_notes, columns=(*key_order, 'start', 'end'))

"""# Create MIDI File"""

sampling_rate = 16000
def display_audio(pm, seconds=30):
  waveform = pm.fluidsynth(fs=sampling_rate)
# Take a sample of the generated waveform to mitigate kernel resets
  waveform_short = waveform[:seconds*sampling_rate]
  return display.Audio(waveform_short, rate=sampling_rate)

def notes_to_midi(
  notes: pd.DataFrame,
  out_file: str,
  instrument_name: str,
  velocity: int = 100, # note loudness
  ) -> pretty_midi.PrettyMIDI:

  pm = pretty_midi.PrettyMIDI()
  instrument = pretty_midi.Instrument(
    program=pretty_midi.instrument_name_to_program(
      instrument_name))

  prev_start = 0
  for i, note in notes.iterrows():
    start = float(prev_start + note['step'])
    end = float(start + note['duration'])
    note = pretty_midi.Note(
      velocity=velocity,
      pitch=int(note['pitch']),
      start=start,
      end=end,
    )
    instrument.notes.append(note)
    prev_start = start

  pm.instruments.append(instrument)
  pm.write(out_file)
  display.display(display_audio(pm))
  return pm

notes_to_midi(generated_notes,"w.mid","Electric Grand Piano")

"""# GUI"""
```

רשימה של כל המשתנים בקובץ הPython

משתנה	תפקיד
window	**האובייקט הראשי של חלון ה-Tkinter**
elementColor	**צבע עבור האלמנטים בממשק (למשל כפתורים)**
textColor	**צבע הטקסט בממשק**
mainColor	**הצבע הראשי של הרקע**
lighterMainColor	**צבע רקע בהיר יותר**
SecondColor	**צבע שניוני עבור רכיבים בממשק**
instruments	**רשימה של כלי נגינה שיכולים לשמש בפרויקט**

isThereTab	**משתנה בוליאני שמציין אם יש לשונית כלשהי פתוחה**
tabControl	**אובייקט של בקרת לשוניות (Notebook) שמנהל את הלשוניות הפתוחות**
style	**משתנה לשמירת אובייקט הסגנון (Style) של Tkinter**
lenghtOfFile	**משתנה שמאחסן את אורך קובץ המוזיקה שהמשתמש רוצה ליצור**

רשימה של כל המשתנים בקובץ הPython Notebook

משתנה	תפקיד
dir_path	**נתיב הספריה בה נמצאים קבצי ה-MIDI**
filenames	**רשימה של שמות קבצי ה-MIDI בספריה הנתונה**
sample_file	**קובץ ה-MIDI שנבחר לדוגמא לצורך ניתוח והדגמה**
pm	**אובייקט של PrettyMIDI המייצג את קובץ ה-MIDI**
instrument	**כלי הנגינה הראשון בקובץ ה-MIDI**
notes	**אובייקט המייצג את תווי הנגינה בכלי הנגינה שנבחר**
sorted_notes	**רשימת תווים ממוינת לפי זמן התחלה**
prev_start	**זמן התחלה של התו הקודם, לצורך חישוב מרווחי הזמן בין התווים**
raw_notes	**את כל התווים מהקובץ הנבחר לדוגמא**
note_names	**פונקציה הממירה מספרי תווים לשמות תווים**
sample_note_names	**שמות התווים מהקובץ הנבחר לדוגמא**
num_files	**מספר קבצי ה-MIDI שישמשו לאימון המודל**
all_notes	**רשימה של כל התווים מכל קבצי ה-MIDI שנבחרו לאימון**
key_order	**סדר המפתחות של התווים (pitch, step, duration)**
train_notes	**מערך של תווי האימון ממוינים לפי המפתחות**
notes_ds	**המורכב מתווי האימון TensorFlow של Dataset**
seq_length	**אורך הרצף של תווי האימון**

vocab_size	**גודל הלקסיקון (מספר התווים האפשריים)**
sequences	**רשימה של רצפים (sequences) של תווי האימון**
targets	**רשימה של תווי המטרה (targets) לרצפים**
seq_ds	**המורכב מהרצפים ותווי המטרה TensorFlow של Dataset**
batch_size	**גודל הבאטצ' (batch) לאימון המודל**
buffer_size	**גודל הבופר (buffer) לאימון המודל**
train_ds	**המורכב מרצפים ממושקלים TensorFlow של Dataset ומאורגנים לבאטצ'ים**
layer	**שכבה במודל ה-Keras**
learning_rate	**קצב הלמידה של המודל**
input_data	**שכבת הקלט של המודל**
x	**שכבת LSTM במודל**
outputs	**שכבת הפלט של המודל, המכילה את הפלטים לתווי pitch, step, ו-duration**
model	**המודל הכולל של Keras**
loss	**פונקציות ההפסד לכל אחד מתווי הפלט**
optimizer	**האלגוריתם לאופטימיזציה של המודל**
temperature	**הטמפרטורה (temperature) לקביעת האקראיות בתחזיות המודל**
num_predictions	**מספר התחזיות שהמודל ייצור**
sample_notes	**מערך של התווים שנלקחו לדוגמא לצורך התחזיות**
input_notes	**רצף התווים המוזן למודל לצורך התחזיות**
generated_notes	**של תווי התחזיות שנוצרו על ידי המודל DataFrame**
sampling_rate	**קצב הדגימה (sampling rate) של היצירה המוזיקלית**
waveform	**גל הקול של היצירה המוזיקלית**
waveform_short	**דגימה קצרה של גל הקול של היצירה המוזיקלית**
out_file	**שם הקובץ של יצירת ה-MIDI שנוצרה**

instrument_name	**שם כלי הנגינה ליצירת ה-MIDI**
velocity	**עוצמת הנגינה של התווים**

פירוט על כל הפונקציות בקובץ Python Notebook

פונקציה	תפקיד	פרטים
createOptionsSection	יצירת אזור אפשרויות	**יוצר וממקם את אזור האפשרויות בתוך המסגרת שניתנה, כולל תוויות, שדה קלט וקומבובוקס לבחירת כלי נגינה.**
createTab	יצירת טאבים חדשים	**מבקש מהמשתמש שם לפרויקט, יוצר טאב חדש בנוטבוק ומאתחל אותו עם אזור האפשרויות.**
firstTab	יצירת הטאב הראשון	**קורא לפונקציה createTab כדי ליצור את הטאב הראשון באפליקציה.**
createMusic	יוצר את המוזיקה לפי הקלט של המשתמש	**קורא את הפרמטרים מהממשק משתמש וקורא לפונקציית היצירה של המוזיקה**

פירוט על כל הפונקציות בקובץ Python GUI

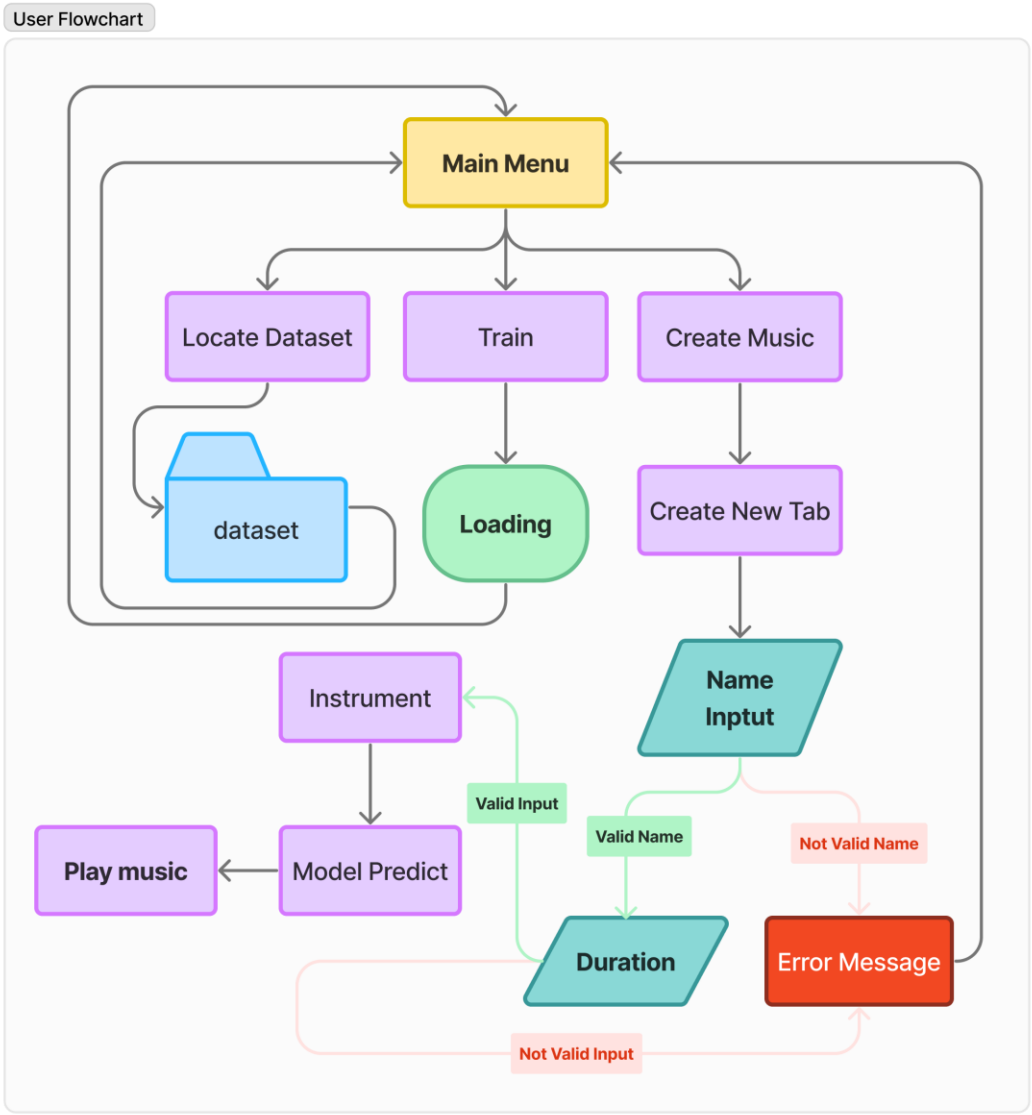
פונקציה	תפקיד	פרטים
plot_piano_roll(notes: pd.DataFrame, count: Optional[int] = None)	מצייר תרשים של תווי הנגינה על פני הזמן	**מקבלת DataFrame של תווים ויוצרת גרף המראה את התווים לפי הזמן והגובה שלהם. אם ניתן פרמטר count, הוא קובע כמה תווים ראשונים יוצגו.**
midi_to_notes(midi_file)	ממיר קובץ MIDI ל-DataFrame המכיל מידע על התווים	**טוענת קובץ MIDI באמצעות PrettyMIDI, לוקחת את כלי הנגינה הראשון, וממיינת את התווים לפי זמן התחלה. יוצרת DataFrame עם מידע על הגובה, זמן התחלה, זמן סיום, שלב, ומשך של כל תו.**
create_sequences(dataset, seq_length, vocab_size=128)	יוצר רצפים של נתונים לצורך אימון מודל	**מקבלת את המידע על התווים ויוצרת רצפים עבור כל תו על בסיס אורך הרצף הנתון (seq_length). מחלקת את הנתונים לרצפים ומטרות (targets), ומחזירה אובייקט של TensorFlow Dataset המוכן לאימון המודל.**

predict_next_note(notes, keras_model, temperature)	מנבא את התו הבא בהתבסס על הרצפים הנתונים והמודל	**מקבלת את הרצפים הנוכחיים והמודל, ומנבאת את התו הבא על בסיס טמפרטורת הדגימה (temperature).**
display_audio(pm, seconds=30)	מציג את האודיו המנוגן	**יוצר גל קול מ-PrettyMIDI ומציג אותו עם IPython.**
notes_to_midi(notes: pd.DataFrame, out_file: str, instrument_name: str, velocity: int = 100)	ממיר DataFrame של תווים לקובץ MIDI	**יוצר אובייקט PrettyMIDI חדש עם כלי הנגינה הנתון, מוסיף את התווים, ושומר את הקובץ עם השם הנתון. מציג את האודיו שנוצר.**
createPrediction()	יוצר מנגינה לפי ניחושים של הרבה תווים	**עובר על כל התווים ביצירה ומנחש מה יהיה הצליל האורך והמיקום שלהם ברצף של הצלילים**
createModel()	מכיל את המודל RNN של הפרוייקט	**פונקציה המתכללת את מודל השכבות שתיארתי למעלה**

מדריך למשתמש

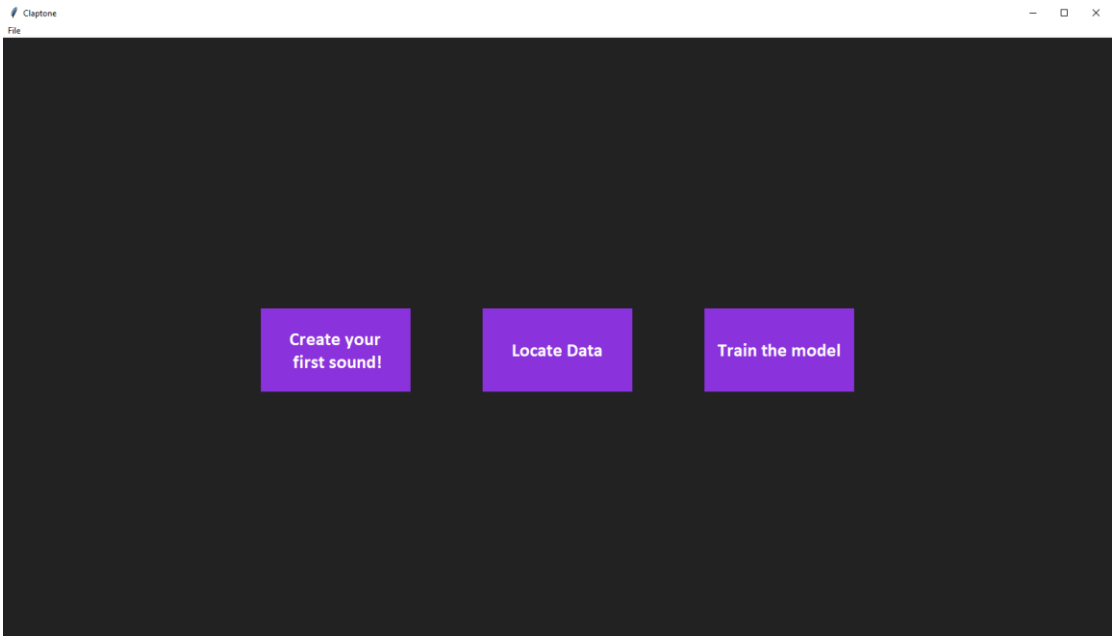
בעת הפעלת התוכנה נפתח מסך ראשי(Main Menu) שממנו המשתמש בוחר מה לעשות.
המשתמש יכול לתפעל את כל המערכת מתוך הממשק GUI ואין צורך בהתעסקות בקוד.

תרשים ממשק משתמש



תיאור מסכים

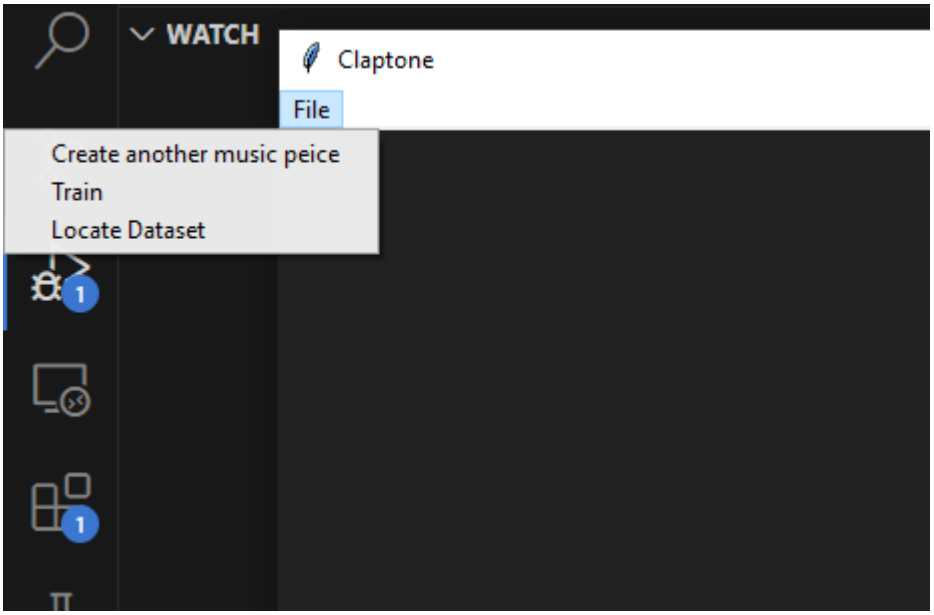
מסך ראשי



צילום מסך מתוך מחשב של המסך פתיחה/מסך ראשי של הממשק

המסך הראשי של הממשק משמש כתפריט המאפשר ליצור מוזיקה, לאמן את המודל ולבחור את הDataset שעליו ירוץ המודל. כדי לבצע כל אחת מהפעולות האלו צריך ללחוץ על הכפתור המתאים מבין השלושה.

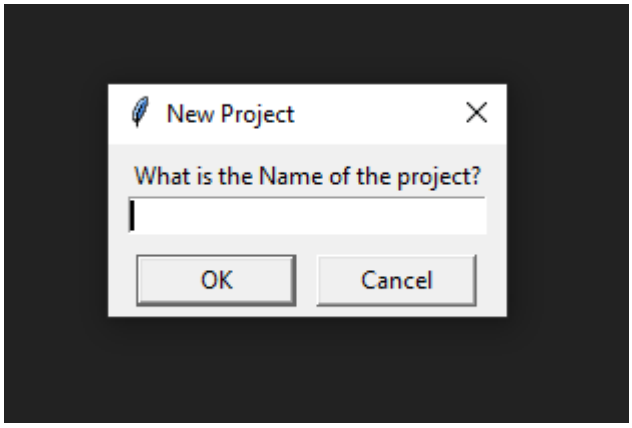
בנוסף לכל אורך הממשק תמיד ניתן לפתוח את האפשרויות של האלו מתוך תפריט העזרה הנמצא במעלה המסך.



צילום מסך של התפריט שניתן לפתוח מכל מסך במערכת

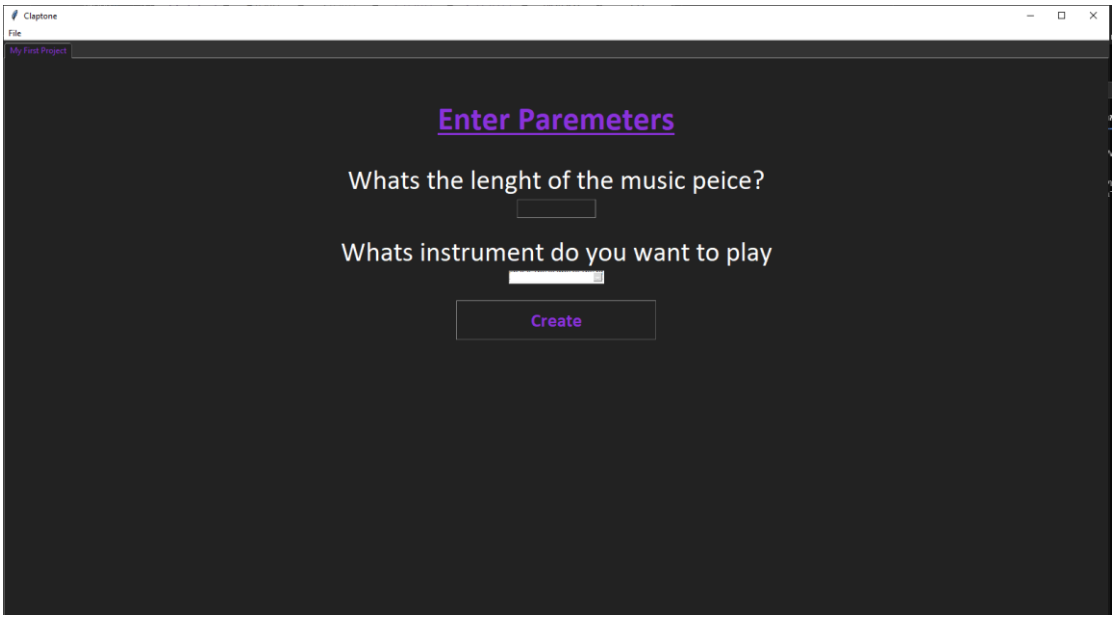
מסך יצירת הפרויקט

בעת לחיצה על הכפתור "Create your first Sound" או דרך תפריט התוכנה יפתח הדיאלוג השואל את המשתמש משהו השם של הפרויקט שהוא יוצר.

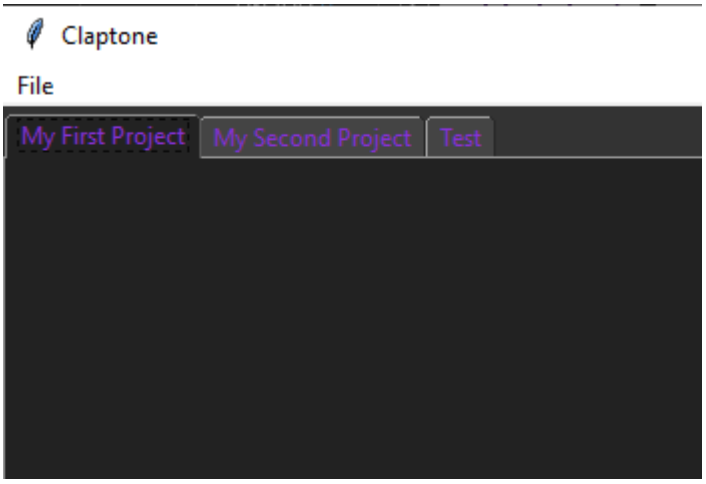


צילום מסך של הדיאלוג שנפתח

לאחר הזנת שם הפרויקט של המשתמש נוצר Tab חדש במסך שבו המשתמש יכול להזין את האורך של היצירה שהוא מעוניין ליצור ובאיזה כלי נגינה הוא רוצה שהיא תנוגן.

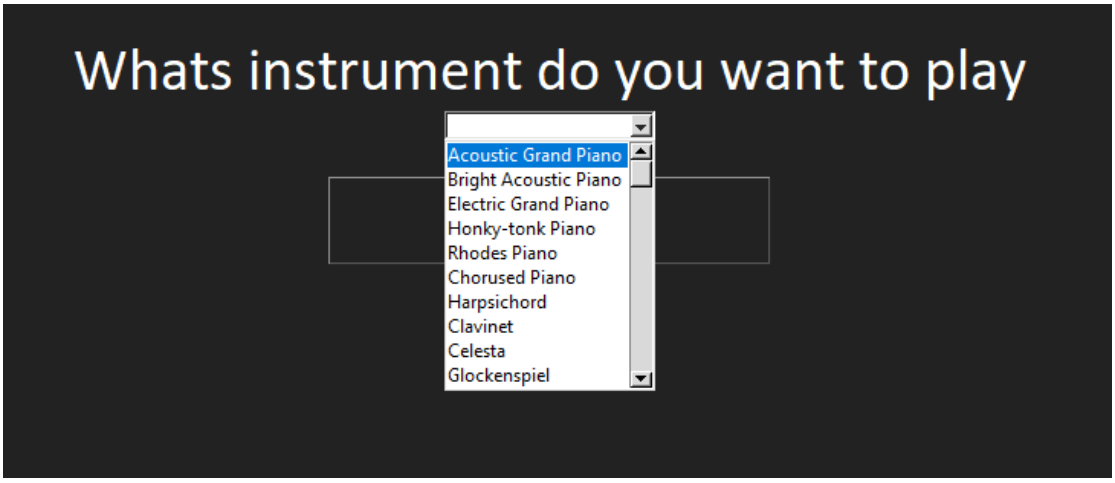


צילום מסך של הTab שנפתח והנתונים שיש להזין



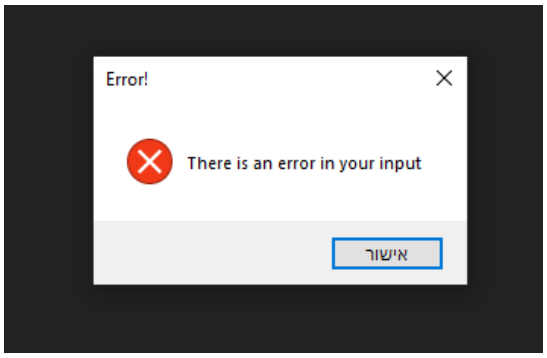
צילום מסך של שורת הTabs שנוצרת למשתמש כשבכל Tab ישנו פרויקט

כדי לבחור את הכלי נגינה שבו ינוגן היצירה המשתמש יכול לפתוח Dropdown שבו יש מעל מאה כלי נגינה אפשריים שמהם ניתן ליצור את היצירה.



צילום מסך של הDropdown שנפתח למשתמש

כאשר המשתמש מבצע שגיאה, למשל מזין קלט שגוי(str במקום int) תוצג הודעת שגיאה מתאימה



צילום מסך של הודעת השגיאה שיכולה להופיע למשתמש

סיכום אישי

העבודה על הפרויקט הייתה חוויה מעשירה ומרגשת. כמישהו שאוהב מוזיקה ואוהב תכנות, השילוב בין השניים בפרויקט הזה היה מושלם עבורי. העבודה על פרויקט שמאפשר יצירת מוזיקה באמצעות תכנות פתחה בפניי עולמות חדשים ומרתקים, ואיפשרה לי לבטא את היצירתיות שלי בדרכים שלא חשבתי עליהן בעבר.

במהלך העבודה על הפרויקט, קיבלתי הרבה ידע חדש במגוון תחומים. למדתי להשתמש בספריות מוזיקליות בתכנות כמו PrettyMIDI ו TensorFlow-ליצירת מוזיקה ולמידת מכונה. כמו כן, שיפרתי את המיומנויות שלי בתכנות ב Python-ועבודה עם Tkinter ליצירת ממשקים גרפיים. בנוסף, הפרויקט תרם לי המון בפיתוח יכולות חשיבה יצירתית ובמציאת פתרונות לבעיות מורכבות.

היו מספר קשיים ואתגרים שעמדו בפניי במהלך הפרויקט. אחד האתגרים המרכזיים היה שילוב בין המוזיקה והתכנות, מציאת הדרך הנכונה לעבוד עם קבצי MIDI ולהפיק מהם את המידע הנדרש. בנוסף, העבודה עם למידת מכונה הייתה מאתגרת במיוחד, כאשר נדרש להבין כיצד להתאים את המודלים למוזיקה ולוודא שהם מפיקים תוצאות מספקות. גם ניהול הזמן היה אתגר, כי פרויקט כזה מצריך עבודה ממושכת ומוקפדת.

המסקנה העיקרית שלי מהפרויקט היא שהשקעה בעבודה קשה והתמדה מובילים לתוצאות מרשימות. למדתי שגם אם דברים נראים קשים ומסובכים בתחילה, עם מספיק השקעה ונחישות ניתן להתגבר על האתגרים ולהשיג את המטרות. בנוסף, אני מבין כעת את החשיבות של תכנון נכון ושל ניהול זמן טוב, דברים שיכולים לעשות את ההבדל בין פרויקט מוצלח לבין כזה שנתקע באמצע.

לו הייתי מתחיל את הפרויקט מהתחלה, הייתי שם יותר דגש על תכנון מוקדם. הייתי מנסה לתכנן את השלבים השונים בצורה מפורטת יותר, ולוודא שאני מבין היטב כל חלק בתהליך לפני שאני מתחיל לעבוד עליו. בנוסף, הייתי מנסה לעבוד יותר בצוות או להיעזר ביותר במשוב מאנשים אחרים, כדי לקבל פרספקטיבות שונות ולשפר את העבודה שלי.

בסיכומו של דבר, העבודה על הפרויקט הייתה חוויה מהנה ומלמדת. השילוב בין אהבתי למוזיקה לבין היכולות הטכנולוגיות שלי אפשר לי ליצור משהו ייחודי ומיוחד. נהניתי ללמוד דברים חדשים ולהתמודד עם אתגרים מורכבים, ואני גאה בתוצאה שהשגתי. הפרויקט הזה היה צעד משמעותי בהתפתחות האישית והמקצועית שלי, ואני מצפה להשתמש בכלים ובמיומנויות שלמדתי בו גם בפרויקטים עתידיים.

ביבליוגרפיה

מקורות אינטרנט

- Matson, J. (2018, September 15). Artificial intelligence music creation with recurrent neural networks (RNN). *Medium*. https://medium.com/@james.matson_64120/artificial-intelligence-music-creation-with-recurrent-neural-networks-rnn-be08d1d3c759
- Author unknown. (2020, April 5). Neural networks, RNNs, and music generation. *Analytics Vidhya*. https://medium.com/analytics-vidhya/neural-networks-rnns-and-music-generation-a1838dfa6472
- Author unknown. (2016, June 10). Recurrent neural network generation tutorial. *Magenta*. https://magenta.tensorflow.org/2016/06/10/recurrent-neural-network-generation-tutorial
- Python Software Foundation. (n.d.). Tkinter — Python interface to Tcl/Tk. *Python Documentation*. https://docs.python.org/3/library/tkinter.html
- Author unknown. (n.d.). Principles of UI/UX design. *Geeks for Geeks*. https://www.geeksforgeeks.org/principles-of-ui-ux-design/

תיעוד וספריות קוד פתוח

- PrettyMIDI documentation. (n.d.). *GitHub*. Retrieved June 2, 2024, from https://github.com/craffel/pretty-midi
- TensorFlow documentation. (n.d.). *TensorFlow*. Retrieved June 2, 2024, from https://www.tensorflow.org

נספחים

דברים שלולא הם היה לי הרבה יותר קשה לממש את הפרויקט

- The MAESTRO Dataset https://magenta.tensorflow.org/datasets/maestro#dataset
- /https://chatgpt.com – My Best Friend Aka LLM
- – My favorite developing environment for ML https://colab.research.google.com