

Příloha k bakalářské práci
ZDROJOVÉ KÓDY AUTOMATICKÉHO SYSTÉMU
HUDEBNÍ TRANSKRIPCE
Tomáš Netolický

OBSAH

1. Zdrojový kód pro předzpracování hudebního signálu, trénink modelu neuronové sítě a jeho predikci
2. Zdrojový kód pro převod predikce modelu neuronové sítě do souboru ve formátu .mid pomocí knihovny MIDIUtil

1. Zdrojový kód pro předzpracování hudebního signálu, trénink modelu neuronové sítě a jeho predikci

Zdrojový kód v této části slouží k automatizovanému předzpracování hudebních nahrávek do podoby vhodné pro vstup do neuronové sítě. Obsahuje implementaci výpočtu CQT (Constant-Q Transform) a STFT (Short-time Fourier Transform), rozdělení dat na trénovací a testovací sady, definici architektury modelu neuronové sítě a samotný proces trénování. Dále je zde zahrnuta část kódu pro provedení predikce nad novými hudebními vstupy. Zdroj: vlastní tvorba autora bakalářské práce.

```
import librosa
import librosa.display
import numpy as np
import tensorflow as tf
from sklearn.metrics import precision_score, recall_score, f1_score

data = np.load('musicnet.npz', allow_pickle=True, encoding='bytes')
def load_and_process(track_id):
    y = data[track_id][0]
    labels = data[track_id][1]

    hop_length = 512
    sr = 44100
    bins_per_octave = 12
    n_bins = 88
    frame_length = np.round(hop_length / sr, 5)

    cqt = np.abs(librosa.cqt(y, sr=sr, hop_length=hop_length,
bins_per_octave=bins_per_octave, n_bins=n_bins))

    n_fft = 4096
    stft = np.abs(librosa.stft(y, n_fft=n_fft, hop_length=hop_length))

    piano_freq = librosa.midi_to_hz(np.arange(21, 109))
    piano_stft = np.zeros((n_bins, stft.shape[1]))

    freq_bins = librosa.fft_frequencies(sr=sr, n_fft=n_fft)
    for i, freq in enumerate(piano_freq):
        idx = np.argmin(np.abs(freq_bins - freq))
        start_idx = max(0, idx - 2)
        end_idx = min(len(freq_bins) - 1, idx + 3)
```

```

piano_stft[i] = np.sum(stft[start_idx:end_idx, :], axis=0)

harmonic_cqt = np.zeros_like(cqt)
for i in range(n_bins):
    base_idx = i
    harmonic_cqt[base_idx] = cqt[base_idx]
    if base_idx + 12 < n_bins:
        harmonic_cqt[base_idx] += cqt[base_idx + 12]*0.5
    if base_idx + 19 < n_bins:
        harmonic_cqt[base_idx] += cqt[base_idx + 19]*0.3

cqt_norm = librosa.amplitude_to_db(cqt, ref=np.max)
cqt_norm = (cqt_norm - np.min(cqt_norm)) / (np.max(cqt_norm) -
np.min(cqt_norm))

piano_stft_norm = librosa.amplitude_to_db(piano_stft, ref=np.max)
piano_stft_norm = (piano_stft_norm - np.min(piano_stft_norm)) /
(np.max(piano_stft_norm) - np.min(piano_stft_norm))

harmonic_cqt_norm = librosa.amplitude_to_db(harmonic_cqt, ref=np.max)
harmonic_cqt_norm = (harmonic_cqt_norm - np.min(harmonic_cqt_norm)) /
(np.max(harmonic_cqt_norm) - np.min(harmonic_cqt_norm))

input_data = np.concatenate([cqt_norm.T, piano_stft_norm.T,
harmonic_cqt_norm.T], axis = 1)

labels_np = np.array([(a.begin, a.end, a.data[1]) for a in
sorted(labels)], dtype=float)
labels_np = labels_np[np.argsort(labels_np[:, 0])]
labels_np[:, :2] = labels_np[:, :2] / sr
labels_np[:, :2] = np.round(labels_np[:, :2] * sr / hop_length) *
hop_length / sr
labels_np[:, :2] = np.round(labels_np[:, :2], 5)
for row in range(labels_np.shape[0]):
    labels_np[row, 0] = np.round(labels_np[row, 0] / frame_length)
    labels_np[row, 1] = np.round(labels_np[row, 1] / frame_length)
labels_np = labels_np.astype(int)

output_data = np.zeros(cqt_norm.T.shape, dtype = int)
frame_indices = np.arange(output_data.shape[0])
for start, end, midi_pitch in labels_np:

```

```

        note_indices = (frame_indices >= start) & (frame_indices <= end)
        output_data[note_indices, midi_pitch - 24] = 1

    return input_data, output_data

tracks_train = ['2203', '2204', '1733', '1734', '1735', '2186', '2191',
                '2217', '2218',
                '2318', '2319', '2320', '2330', '2334', '2335', '2336',
                '2570', '2571',
                '2241', '2242', '2243', '2244', '2289', '2300', '2302',
                '2303', '2304',
                '2293', '2294', '2295', '2296', '2297']
tracks_test = ['2202']

x_train = []
y_train = []
x_test = []
y_test = []

for track_idx in tracks_train:
    x, y = load_and_process(track_idx)
    x_train.append(x)
    y_train.append(y)

for track_idx in tracks_test:
    x, y = load_and_process(track_idx)
    x_test.append(x)
    y_test.append(y)

x_train = np.vstack(x_train)
x_test = np.vstack(x_test)
y_train = np.vstack(y_train)
y_test = np.vstack(y_test)
x_train = np.expand_dims(x_train, axis=1) # shape (rows, 1, BINS)
x_test = np.expand_dims(x_test, axis=1)
y_train = np.expand_dims(y_train, axis=1) # shape (rows, 1, BINS)
y_test = np.expand_dims(y_test, axis=1)

model = tf.keras.Sequential([
    tf.keras.layers.Input(shape = (None, 88*3)),
    tf.keras.layers.Conv1D(128, kernel_size=3, activation='relu',

```

```

padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(256,
return_sequences=True, dropout=0.3, recurrent_dropout=0.2)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128,
return_sequences=True, dropout=0.3, recurrent_dropout=0.2)),
    tf.keras.layers.Dense(88, activation='sigmoid')
])

model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

history = model.fit(x_train, y_train,
                    epochs = 25,
                    validation_data = (x_test, y_test),
                    batch_size = 512)

loss, acc = model.evaluate(x_test, y_test)
print("ACC: ", acc, "LOSS: ", loss)

prediction = model.predict(x_test)
np.savetxt("prediction_float.csv", prediction.squeeze(), delimiter=",")
prediction = (prediction > 0.5).astype(int)
print("PREC: ", precision_score(y_test.flatten(), prediction.flatten()),
      "RECALL: ", recall_score(y_test.flatten(), prediction.flatten()),
      "F1: ", f1_score(y_test.flatten(), prediction.flatten()))
np.savetxt("y_test.csv", y_test.squeeze(), delimiter=",")
np.savetxt("prediction_bin.csv", prediction.squeeze(), delimiter=",")

```

2. Zdrojový kód pro převod predikce modelu neuronové sítě do souboru ve formátu .mid pomocí knihovny MIDIUtil

Tato část kódu slouží k převodu výstupních dat z predikce neuronové sítě do formátu MIDI. Pomocí knihovny MIDIUtil jsou ze získaných informací o výšce tónu, jejich začátcích a délkách generovány odpovídající MIDI události. Výsledkem je .mid soubor, který je možné přehrát v běžném hudebním softwaru nebo dále analyzovat. Zdroj: vlastní tvorba autora bakalářské práce.

```
from midiutil import MIDIFile
import numpy as np

hop_length = 512
sr = 44100

y_pred = np.loadtxt('prediction_bin.csv', delimiter=',')

midi = MIDIFile(1)
midi.addTempo(0, 0, 90)
midi.addProgramChange(0, 0, 0, 74) #piano - 0, flute - 74
velocity = 90
time_per_frame = hop_length / sr
active_notes = {}
for time_idx in range(y_pred.shape[0]):
    current_time = time_idx * time_per_frame
    for midi_pitch in range(21, 109):
        note_active = y_pred[time_idx, midi_pitch - 21] == 1
        if note_active:
            if midi_pitch not in active_notes:
                active_notes[midi_pitch] = current_time
        else:
            if midi_pitch in active_notes:
                start_time = active_notes.pop(midi_pitch)
                duration = current_time - start_time
                midi.addNote(track=0, channel=0, pitch=midi_pitch,
                             time=start_time, duration=duration,
                             volume=velocity)
with open('output.mid', 'wb') as f:
    midi.writeFile(f)
```