Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Rozpoznávanie farebných notových zápisov

Bakalárska práca

2024
Adam Dávid Detko

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Rozpoznávanie farebných notových zápisov

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	RNDr. Zuzana Berger Haladová, PhD.

Bratislava, 2024
Adam Dávid Detko

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Adam Dávid Detko
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Rozpoznávanie farebných notových zápisov
Color notes recognition

Anotácia: Pre výučbu hry na xylofón sa používa zápis formou farebných nôt, rôzne xylofóny môžu mať rôznu farebnosť a farby v notovom zápise nemusia sedieť. Cieľom práce bude rozpoznať noty v notovom zápise a previesť ich na midi, resp. notový zápis s inými farbami.

Vedúci: RNDr. Zuzana Berger Haladová, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 13.10.2022

Dátum schválenia: 16.10.2023 doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Ďakujem mojej školiteľke, RNDr. Zuzane Berger Haladovej, PhD., za ochotu, odborné rady a nápady pri písaní tejto práce. Tiež ďakujem svojej rodine za podporu.

Abstrakt

Cieľom tejto práce je vytvoriť systém na rozpoznávanie hudobných nôt z fotografie farebných notových záznamov pre xylofón a generovanie notového záznamu s iným farbením. Zameriavame sa na využitie optického rozpoznávania hudby (OMR), konkrétne neurónových sietí na detekciu nôt v notovom zázname. Pre tento účel využívame model YOLOv8 dotrénovaný na vlastných dátach, ktoré sme vygenerovali, oanotovali a augmentovali na zvýšenie úspešnosti modelu na reálnych dátach. Ďalej sa zaoberáme rekonštrukciou notového záznamu z výsledkov detekcie a následne vyhodnocujeme úspešnosť dotrénovaných modelov.

Kľúčové slová: xylofón, farebný notový zápis, OMR, detekcia

Abstract

The goal of this work is to create a system for recognizing musical notes from photographs of colored musical notations for the xylophone and generating a musical score with different coloring. We focus on utilizing Optical Music Recognition (OMR), specifically neural networks for note detection in the musical score. For this purpose, we employ a model YOLOv8 fine-tuned on custom data that we generated, annotated, and augmented to improve the model's performance on real data. Furthermore, we deal with the reconstruction of the musical score from the detection results and subsequently evaluate the success of the fine-tuned models.

Keywords: xylophone, colored musical notation, OMR, detection

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Xylofón je bicí hudobný nástroj. Má drevené ploché doštičky usporiadané podľa výšky tónu, na ktoré sa udiera dvoma paličkami. Pre výučbu hry na xylofóne pre deti sa zvyčajne používa zápis formou farebných nôt. Takýto zápis sa väčšinou viaže ku konrétnemu xylofónu. To znamená, že farba každej noty zodpovedá farbe konkrétnej doštičky daného xylofónu. Xylofóny môžu mať rôznu farebnosť a rôzny rozsah, preto môže nastať situácia, v ktorej konkrétny farebný notový zápis nebude sedieť s daným xylofónom.

Ak by sme chceli použiť notový zápis pre xylofóny, ktoré majú iné zafarbenie, museli by sme upraviť farebnú štruktúru notového zápisu. Jedna z možností je manuálne prepísať alebo prekresliť notový záznam tak, aby zodpovedal danému xylofónu. Toto riešenie je pracné a časovo náročné. V dnešnej dobe nám technológia umožňuje pracovať s notovými záznamami v digitálnej forme. Je to omnoho efektívnejšie a rýchlejšie.

V našej práci sa budeme zaoberať návrhom riešenia na prevod farebného notového zápisu na notový zápis s inými farbami. Cieľom je vytvoriť systém, ktorý dokáže automaticky rozpoznať noty a previesť ich na notový zápis s inými farbami. Toto riešenie by mohlo uľahčiť prácu hudobníkom, skladateľom a učiteľom hudby pri práci s rôznymi formátmi notových zápisov.

Vstupom pre tento systém je fotografia farebného notového zápisu. Pomocou optického rozpoznávania hudby budeme získavať informácie o jednotlivých notách, ich pozíciach a výškach. S týmito informáciami budeme schopní vytvoriť digitálny notový zápis s farbami podľa potreby. Zameriame sa na využitie neurónových sietí, konkrétne knižnice YOLOv8, na detekciu a klasifikáciu hudobných symbolov v notovom zápise. Vygenerujeme dáta, na ktorých natrénujeme model, ktorý bude schopný rozpoznať noty a následne spracujeme potrebné informácie na vygenerovanie notového záznamu.

Kapitola 1

Analýza problému

Pre výučbu hry na xylofóne pre deti sa používa zápis formou farebných nôt. Zvyčajne sa tento zápis viaže ku konkrétnemu xylofónu, ktorý môže mať rôznu farebnosť. Z tohto dôvodu môže nastať situácia, že farby v notovom zápise sú odlišné ako farby na danom xylofóne. Preto by sme chceli vedieť rozpoznávať noty nezávisle od farebnosti a prípadne ich previesť do správneho formátu. V tejto kapitole si priblížime xylofón a potrebnú hudobnú teóriu. Taktiež si ukážeme ako vyzerá detský xylofón a farebný notový zápis.

1.1 Xylofón

Xylofón je hudobný nástroj, ktorý patrí do skupiny bicích nástrojov. Má lichobežníkový tvar a drevené ploché obĺžnikové doštičky. Tieto ploché doštičky sú usporiadané v poradí podľa výšky tónu a udiera sa do nich dvoma paličkami, podobnými vydutej lyžici alebo môžu byť aj s guľatými hlavami z tvrdej gumy, plastu alebo dreva. Sú usporiadané podľa klaviatúry klavíra a sú naladené na špecifickú hudobnú stupnicu. Môžeme ho nájsť v rôznych veľkostiach a konfiguráciách, od orchestrálnych xylofónov so širokou škálou tónov až po menšie, prenosnejšie verzie alebo verzie pre deti. [1] V našej práci sa budeme zaoberať iba verziou xylofónu pre deti.

Xylofón pre deti je zábavný a vzdelávací hudobný nástroj. Je to zjednodušená verzia klasického. Skladá sa z menšieho počtu doštičiek, ktoré sú farebne odlíšené. Znázornené na obrázku 1.1. Mnoho xylofónov určených pre deti sa dodáva so vzdelávacími materiálmi. Väčšinou sa jedná o notové zápisy k danému xylofónu.

1.2 Notový zápis

Hudobné dielo zapisujeme pomocou notovej osnovy. Notová osnova sa skladá z piatich vodorovných čiar a štyroch medzier. Okrem toho môže obsahovať aj pomocné čiary,

Obr. 1.1: Ukážka detského xylofónu.

ktoré majú dĺžku len mierne presahujúcu samotnú notu. Hlavné čiary notovej osnovy sa počítajú zdola nahor, zatiaľ čo pomocné čiary sa počítajú v smere od hlavných čiar. Ak sú pod osnovou, počítajú sa smerom nadol, a ak sú nad osnovou, počítajú sa smerom nahor. Noty môžu byť umiestnené na čiare alebo v medzere a čítajú sa zľava doprava. Tieto hudobné symboly reprezentujú danú výšku, dĺžku tónu a mnohé ďalšie vlastnosti. Možeme ich zapísať aj pomocou hudobnej abecedy (A, B, C, D, E, F, G), pričom každé písmeno reprezentuje určitú výšku. Výška noty je tiež závislá od značky (kľúča) na začiatku notovej osnovy. [15]

Podľa dĺžky poznáme tieto základné noty:

- 𝅝 - celá
- 𝅗𝅥 - pólová
- ♩ - štvrtinová
- ♪ - osminová
- 𝅘𝅥𝅯 - šestnástinová

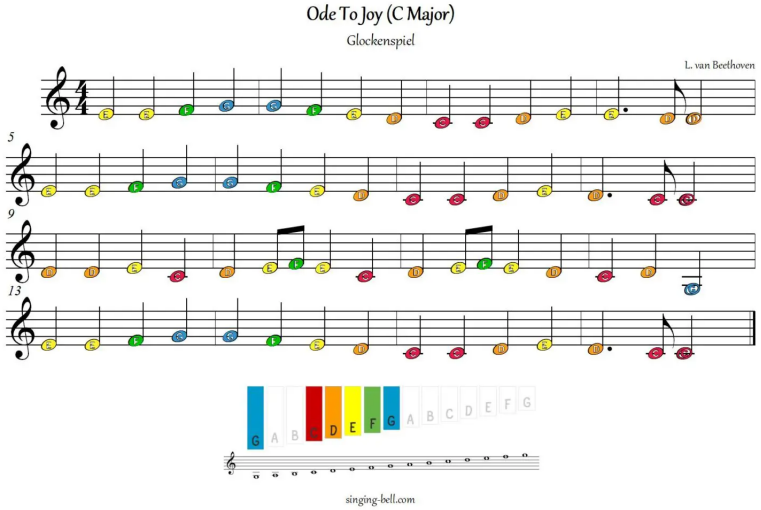
Pre zjednodušenie budeme v našej práci využívať iba prvé tri typy nôt (celá, pólová a štvrtinová) a husľový kľúč.

1.2.1 Notový zápis pre detský xylofón

Notový zápis pre detské xylofóny je navrhnutý tak, aby pomohol deťom naučiť sa hrať na nástroj jednoduchým a pútavým spôsobom bez toho, aby museli poznať hudobnú teóriu. Tieto zápisy často používajú farebné kódovanie a kódovanie hudobnej abecedy na označenie doštičiek, na ktoré má dieťa udrieť.

1. Farebné kódovanie: Každá nota má priradenú farbu, ktorá zodpovedá farbe doštičky na xylofóne.

Obr. 1.2: Ukážka jednoduchšieho zápisu.



Obr. 1.3: Ukážka pokročilejšieho notového zápisu.

2. Kódovanie hudobnej abecedy: Každá nota je označená písmenom, ktoré zodpovedá danej note.

V jednoduchších zápisoch sú noty reprezentované farebnými guličkami s daným písmenom a zaznačené v notovej osnove, alebo sú jednoducho usporiadané v riadkoch a môžu byť viazané na text. Príklad vidíme na obrázku 1.2. Pokročilejšie zápisy sa skladajú z klasických hudobných symbolov a čiar, ktorým sú priradené farby podľa príslušného xylofónu a môžeme to vidieť na obrázku 1.3.

V tejto práci budeme pracovať s pokročilejším notovým zápisom, ktorý používa farebné kódovanie.

Kapitola 2

Optické rozpoznávanie notových zápisov

Spracovanie a rozpoznávanie notových zápisov zahŕňa konverziu notového záznamu z fyzických alebo digitálnych notových záznamov do formátu, ktorý možno pochopiť a spracovať pomocou počítača. V tejto kapitole si priblížime spôsob optického rozpoznávania a spracovania notových záznamov.

2.1 Optické rozpoznávanie hudby (OMR)

Optické rozpoznávanie hudby, bežne označované ako OMR, je systém, ktorý umožňuje konverziu tlačených alebo ručne písaných hudobných symbolov do digitálneho formátu. Tento spôsob otvára veľa možností pre hudobníkov, skladateľov a hudobných nadšencov. Napríklad umožňuje efektívne upravovať notové záznamy a spracovávať hudbu pomocou digitálnych technológií. Optické rozpoznávanie hudby je aktívna výskumná téma od roku 1967 [18]. Ľahší prístup k digitálnej technológií koncom 80. rokov 20. storočia prispel k rozšíreniu výskumných aktivít OMR. Objavilo sa niekoľko komerčných softvérov OMR, avšak žiadny s uspokojivým výkonom z hľadiska presnosti a robustnosti, najmä pre ručne písané notové záznamy. Doteraz aj tie najpokročilejšie rozpoznávacie produkty vrátane Notescan v Nightingale, Midiscan vo Finale, Photoscore v Sibelius a iných ako Smartscore, a Sharpeye nedokážu identifikovať všetky hudobné symboly [16].

2.1.1 Architektúra OMR

Typická kostra pre automatické rozpoznávanie notových záznamov zahŕňa štyri hlavné etapy:

1. predspracovanie obrazu

2. rozpoznávanie hudobných symbolov

3. rekonštrukcia hudobných informácií s cieľom zostaviť logický popis notového záznamu

4. konštrukcia notového modelu, ktorý má reprezentovať symbolický opis notového záznamu

Pre každú z vyššie opísaných etáp existujú rôzne metódy na vykonávanie príslušnej úlohy [16] .

2.1.2 Predspracovanie obrazu

Predspracovanie obrazu je základným krokom v mnohých úlohách počítačového videnia. Cieľom tejto etapy je upraviť obrázok tak, aby sa s ním pracovalo oveľa ľahšie. Väčšina digitálnych obrázkov je ovplyvnených náhodným šumom, ktorý môže zmeniť ich farbu a jas. Uhol, pod ktorým je obrázok zachytený, tiež spôsobuje náhodný šum. Odstránenie šumu je jedna z hlavných techník predspracovania obrazu. Možeme to dosiahnuť aplikovaním rôznych filtrov, ktoré sú závislé od vlastností obrázka, ako napríklad Gaussov filter, vyhladzujúce lineárne alebo nelineárne filtre [5]. Ďalšou technikou na odstránenie šumu je napríklad odstránenie zošikmenia s využitím Houghovej transformácie pre nájdenie čiar pomocou ktorých môžeme identifikovať správny uhol [3]. Tiež sem patrí aj rozmazanie a binarizácia. Výstup z tejto etapy používame ako vstup pre ďalšiu etapu. [20]

2.1.3 Rozpoznávanie hudobných symbolov

Ďalšia fáza sa zaoberá rozpoznávaním hudobných symbolov. Táto etapa sa zvyčajne delí na tri ďalšie časti [16]:

- detekcia a odstraňovanie čiar s cieľom dostať obraz obsahujúci iba hudobné symboly;
- izolácia hudobných symbolov;
- rozpoznávanie symbolov;

Spracovanie čiar

Notová osnova pozostáva z piatich vodorovných čiar a štyroch medzier. Každá čiara a medzera predstavujú inú výšku tónu. Odstránenie týchto čiar sa stalo bežným prístupom, lebo umožňuje dosiahnuť efektívnejšie a presnejšie rozpoznávanie. Výskumníci

používajú dva prístupy: jeden sa zameriava na detekciu čiar a následne ich izoluje, zatiaľ čo druhý ide o krok ďalej a úplne ich odstraňuje.

V tlačených notových záznamoch sú čiary osnovy zvyčajne rovné, paralelné a horizontálne. V ručne písaných partitúrach však môžu byť tieto čiary naklonené, zakrivené a nemusia byť rovnobežné. V našej práci sa však nebudeme zaoberať ručne písanými notovými záznamami. Vzhľad týchto čiar môže byť tiež ovplyvnený ich zakrivením alebo zošikmením v závislosti od faktorov ako je uhol skosenia obrazu alebo degradácia papiera.

Pri odstraňovaní očakávame, že nestratíme informáciu o žiadnych symboloch a žiadna časť symbolu nebude odstránená. Na izoláciu a odstránenie čiar sa zvyčajne používa Houghova transformácia. Ďalším spôsobom je použitie vertikálnych skenovacích čiar, ktoré sa opierajú o line adjacency graph (LAG) alebo mnohé kombinované techniky, ktoré zlepšujú základný prístup [20]. Existujú práce, v ktorých autori rozpoznávajú noty bez odstránenia notovej osnovy, aby sa predišlo problémom so segmentáciou, alebo ak je notová osnova potrebná na rozpoznávanie [8, p. 758]. [14]

2.1.4 Izolácia a klasifikácia hudobných symbolov

Ďalším krokom po spracovaní notovej osnovy je izolácia a klasifikácia hudobných symbolov. Jedným z bežných prístupov je hierarchická dekompozícia. Najprv sa notový záznam rozdelí do riadkov podľa čiar a potom sa vytiahnu noty a ostatné symboly.

Následne sú symboly klasifikované na základe ich tvarov a podobností. Tento proces je dosť zložitý, nakoľko sú symboly častokrát natlačené na seba a môže sa stať, že sa prekrývajú. Preto je tento krok závislý aj od typu a variácie notového zápisu.

V oblasti rozpoznávania a klasifikácie objektov boli skúmané rôzne metódy na efektívnu kategorizáciu objektov. Tieto prístupy zahŕňajú využitie projekčných profilov na klasifikáciu a porovnávanie šablón na identifikáciu objektov. Okrem toho sa v rôznych štúdiách použili support vector machines (SVM), k-nearest neighbour (kNN), neurónové siete (NN) a hidden Markov models (HMM). Aj keď tradičné metódy fungujú dobre, v súčasnosti sa výskumníci viac zameriavajú na používanie pokročilých hlbokých neurónových sietí (DNN) v tejto oblasti. [16] [20]

2.1.5 Konštrukcia notového záznamu a výsledná reprezentácia

Posledným krokom je rekonštrukcia notového záznamu. V tomto kroku musí program pochopiť, čo znamenajú jednotlivé symboly a usporiadať ich do štruktúrovaného hudobného formátu. To zahŕňa pohľad na to, ako sú symboly usporiadané na hárku. Na rozdiel od jednoduchého čítania textu sa musíme pozerať aj na vertikálne rozloženie, ktoré nám udáva výšku noty, a horizontálne, ktoré nám udáva dĺžku. Rovnaký grafický

tvar môže znamenať rôzne veci v rôznych situáciách. Preto je pozícia symbolu veľmi dôležitá.

Výsledná reprezentácia by mala byť v strojovo čitateľnom formáte. Medzi bežné výstupné formáty OMR patria MIDI, MusicXML, MEI, NIFF, Finale a v niektorých softvéroch je hudobný záznam dokonca prevedený do súborov WAVE.

MIDI

Digitálne rozhranie pre hudobné nástroje (MIDI) je výmenné médium medzi počítačom a digitálnymi nástrojmi. Na základnej úrovni MIDI zahŕňa časovú polohu, kedy tón začína, končí, aký hlasný je tón, výšku tónu, nástroj a kanál. Hlavnou nevýhodou MIDI je, že nedokáže zachytiť spojenia medzi hudobnými symbolmi alebo vygenerovať preformátovaný štruktúrovaný súbor, čo obmedzuje jeho výstup len na možnosti prehrávania.

MusicXML

Medzi významné formáty, ktoré umožňujú štruktúrované kódovanie a ukladanie notácií, patrí MusicXML. Tento formát umožňuje ďalšie úpravy v softvéri pre hudobnú notáciu. MusicXML sa viac zameriava na rozloženie kódovania. Je určený na archiváciu a zdieľanie notových záznamov medzi aplikáciami. [16] [20]

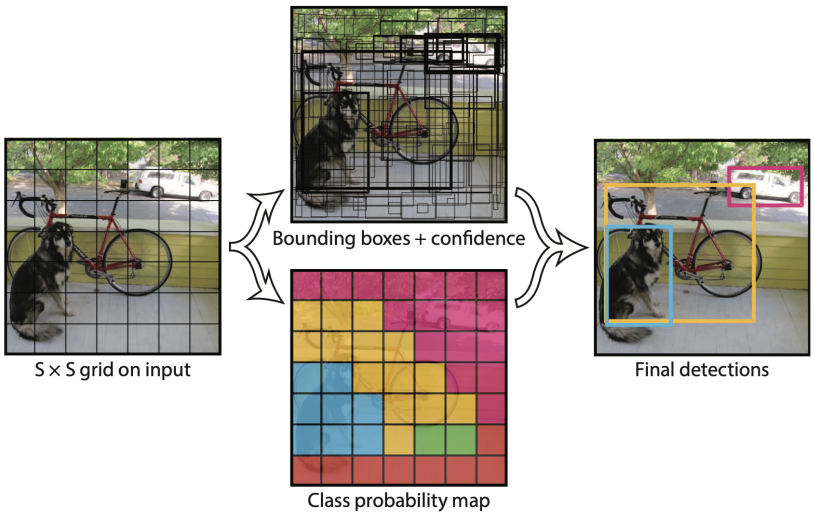
2.2 Neurónové siete v OMR

Hlboké neurónové siete preukázali pozoruhodný úspech v rôznych úlohách strojového učenia, čo podnietilo výskumníkov, aby využili ich potenciál v kontexte rozpoznávania a kategorizácie hudobných objektov. Tento prístup využíva schopnosť neurónových sietí sa automaticky učiť hierarchické reprezentácie údajov, čo pomáha zachytávať zložité vzory a prvky v hudobných objektoch.

Pre tento problém sa používajú konvolučné neurónové siete, ktoré sú určené na spracovanie obrazových súborov. Tieto siete sa trénujú na maticiach reprezentujúcich obrázky a využívajú sa napríklad na klasifikáciu. Prvá sieť tohto typu, LeNet, bola publikovaná v roku 1998, ale nevzbudila veľkú pozornosť. Zlom nastal až v roku 2012, keď vznikla AlexNet, ktorá viedla k rýchlemu rozvoju v tejto oblasti. Odvtedy vzniklo množstvo detekčných algoritmov. Môžeme ich rozdeliť do dvoch kategórií na jednostupňové a dvojstupňové detekčné algoritmy. [11] [14]

Jednostupňové detekčné algoritmy

Typickým príkladom sú modely ako YOLO (You Only Look Once), SSD (Single Shot Multibox Detector) a retina-net. Tieto algoritmy eliminujú potrebu samostatnej etapy



Obr. 2.1: Ukážka fungovania YOLO algoritmu [17].

regionálneho návrhu a priamo vytvárajú kategórie pravdepodobností a polohy súradníc objektov. Vo všeobecnosti sú jednoduchšie a rýchlejšie, vďaka čomu sú vhodnejšie pre aplikácie v reálnom čase. [11]

Dvojstupňové detekčné algoritmy

Tradičné dvojstupňové algoritmy pozostávajú z dvoch krokov. V prvej fáze algoritmus navrhne množinu oblastí, ktoré môžu obsahovať objekty. Druhá fáza zahŕňa klasifikáciu navrhovaných oblastí do rôznych kategórií a spresnenie súradníc ohraničujúceho rámčeka. Patria sem napríklad Fast R-CNN, Faster R-CNN, and R-FCN. Na rozdiel od jednostupňových sú pomalšie, ale za to presnejšie. [11]

2.2.1 YOLO

YOLO je jednostupňový detekčný algoritmus, ktorý využíva konvolučnú neurónovú sieť na detekovanie objektov v reálnom čase. Rozdeľuje obrázok do oblastí, pričom pre každú oblasť predpovedá samostatne ohraničujúce obdĺžniky a pravdepodobnosť objektov [17]. Podrobneší postup môžeme vidieť na obrázku 2.1.

Kapitola 3

Návrh

Cieľom tejto práce je navrhnúť riešenie prevedenia farebného notového zápisu pre detský xylofón (ukážka v podkapitole 1.2.1) na notový zápis s inými farbami.

Vstup nám predstavuje fotografia farebného notového zápisu. Aby sme mohli spraviť konverziu z jedného farebného formátu na druhý, je nutné vedieť, ktorá nota sa nachádza na ktorej pozícii. Tieto informácie budeme získavať pomocou optického rozpoznávania notového zápisu. V kapitole 2.2 sme si spomenuli neurónové siete ako jednu z možností riešenia daného problému. Neurónové siete dosahujú pozoruhodné výsledky v tejto oblasti, preto sme sa tiež rozhodli použiť tento prístup v našej práci. Postup si môžeme rozdeliť do niekoľkých krokov.

1. tvorba dát
2. trénovanie modelu na vytvorených dátach
3. detekcia na reálnych dátach
4. vyhodnotenie a rekonštrukcia zápisu

Výstupom by mal byť zápis v digitálnom formáte, ktorý vieme ľahko previesť na formát podľa našej potreby.

V notovom zápise pre detský xylofón nám farebné guličky namiesto nôt spôsobujú strácanie informácie o dĺžke noty. V našej práci budeme používať štandarný notový zápis, ale pre zjednodušenie sa budeme zaoberať iba výškou noty. Nič nám ale nebráni rozšíriť naše riešenie aj o dĺžku noty a ostatné vlastnosti.

3.1 OMR model

V kapitole 2.2 sme si vymenovali niekoľko detekčných algoritmov. Mnohé z nich ako napríklad YOLO, Faster R-CNN a RetinaNet preukázali vysokú úspešnosť v úlohách optického rozpoznávania znakov (OCR) a hudby (OMR) [19]. Na základe dostupných

materiálov, dokumentácie, tutoriálov a open-source licencie sme sa rozhodli použiť Ultralytics YOLOv8 knižnicu.

3.1.1 Ultralytics YOLOv8

Ultralytics YOLOv8 je špičkový state-of-the-art (SOTA) model, ktorý stavia na úspechu predchádzajúcich verzií YOLO a prináša nové funkcie a vylepšenia. V čase začiatku písania našej práce to bola najnovšia verzia. Tento model je dostupný ako python knižnica. Môžeme ho použiť v rozhraní príkazového riadka alebo priamo v prostredí pythonu. Ultralitics poskytuje predtrénované modely rôznej veľkosti a s rôznymi parametrami, ktoré môžu byť dotrénované na vlastných dátach. [22]

Vlastné dáta pre YOLO

Vytváranie vlastnej množiny dát na dotrénovanie modelu zahŕňa anotáciu každého obrázku. To znamená, že každý objekt, od ktorého sa má model učiť, musí byť označený obdĺžnikom. Tieto ohraničujúce obĺžniky sú zápísane v samostatnom textovom súbore [21]. Špecifikácie tohto súboru sú:

- každý objekt v samostatnom riadku
- každý riadok obsahuje 5 hodnôt (trieda objektu, x-súradnica, y-súradnica, šírka obdĺžnika, výška obdĺžnika)
- súradnice ohraničujúceho odĺžnika musia byť v normalizovanom formáte xywh (od 0 do 1)
- čísla tried sú indexované od 0

Príklad môžeme vidieť na obrázku 3.1.

3.2 Tvorba dát

Aby sme natrénovali model s dobrou úspešnosťou, potrebujeme množstvo kvalitných dát. Musíme si uvedomiť, že na vstupe máme reálnu fotku, na ktorú vplýva okolitý šum. Preto by bolo najlepšie, keby trénujeme model na reálnych fotografiách. To znamená, že by sme potrebovali veľké množstvo fotografií rôznych notových záznamov. Navyše by sme museli priradiť ohraničujúci obĺžnik každej jednej note nachádzajúcej sa na notovej osnove. Z pohľadu času a zdrojov tento prístup nie je možný.

Nakoľko sa mnoho výskumníkov zaoberá optickým rozpoznávaním hudby, existujú voľne dostupné množiny anotovaných dát. Tieto množiny však obsahujú veľa ďalších symbolov a konceptov, ktoré v našej práci ignorujeme. Napríklad výška noty je závisla

Obr. 3.1: Ukážka anotácie v YOLO formáte.

aj od klúča na začiatku notovej osnovy. Väčšinou sú detekované čiary a symboly samostatne a výška je prepočítaná na základe pozície a klúča. My používame iba husľový klúč, takže výška noty je závislá iba od pozície na notovej osnove. Vďaka tomu môžeme detekovať výšku na základe kontextu. Preto potrebujeme, aby ohraničujúce obdĺžniky obsahovali aj informáciu o čiarach.

Ďalším možným prístupom je generovať noty a ohraničujúce obĺžniky pomocou programov. Existuje množstvo knižníc v rôzných programovacích jazykoch, ktoré umožňujú vytvárať notové záznamy. Po odskúšaní viacerých knižníc (music21, mingus, abjad) sme sa rozhodli použiť pythonovskú knižnicu music21.

music21

Music21 je sada nástrojov založená na programovacom jazyku python, ktorá vznikla na pôde MIT. Hlavnou úlohou tejto knižnice je jednoducho prepájať hudobnú teóriu s programovacími schopnosťami. Je navrhnutá pre hudoníkov aj vývojárov a poskytuje robustnú platformu na analýzu hudby a generovanie hudobných partitúr. Pomocou kódu nám umožnuje definovať rôzne hudobné symboly, a tak vytvárať rôzne notácie a hudobné diela. Taktiež podporuje rôzne výstupné formáty hudobnej notácie, vrátane musicXML alebo LilyPond. Vďaka tomu je jednoduché exportovať svoje vygenerované kompozície a ďalej s nimi pracovať alebo ich vytlačiť ako notový zápis. [7]

3.2.1 Augmentácia

Umelé dáta, generované rôznymi spôsobmi, ako sú simulácie alebo rôzne programy na generovanie notových záznamov, získali popularitu v oblasti strojového účenia a dátových odvetví. Takéto dáta sú oveľa dostupnejšie a ľahko škálovateľné v porovnaní s dátami z reálneho sveta, avšak čo sa týka presnosti a výkonu zaostávajú. Umelé dáta

častokrát nedokážu zachytiť realitu skutočného experimentu a výsledkom toho je slabo natrénovaná neurónová sieť [2]. Dôvodom je absencia informácií ako napríklad okolitý šum. V našom prípade je to jas obrázku, svetlo, tieň alebo uhol a priblíženie, z ktorého je odfotený daný notový zápis.

Aby sme vylepšili umelé dáta a zvýšili úspešnosť modelu, môžeme aplikovať augmentáciu. Augmentácia je technika, ktorú možno použiť na umelé rozšírenie veľkosti trénovacej množiny upravením existujúcej. Cieľom je zvýšiť rozmanitosť množiny a pomôcť modelu reagovať na rôzne scenáre. Techniky, ktoré budeme aplikovať sú napríklad:

1. **rotácia** - otočenie obrázka o určitý uhol aby sme simulovali rôznu orientáciu obrázka

2. **škálovanie** - zmena veľkosti obrázka na inú mierku, čím budeme simulovať variácie vo veľkostiach objektov

3. **translácia** - posunutie obrázka horizontálne alebo vertikálne, aby sme simulovali zmenu polohy objektov

4. **zmena jasu** - upravenie jasu obrazu, aby sme simulovali zmenu svetelných podmienok

5. **pridanie šumu** - zavedenie náhodného šumu, aby sme simulovali nedostatky skutočného sveta

Na prácu s obrázkami a aplikáciu augmentácie budeme používať knižnicu OpenCV.

3.2.2 OpenCV

Opencv je obrovská open-source knižnica, ktorú pôvodne vyvinula spoločnosť Intel v roku 1999. Podporuje množstvo programovacích jazykov ako C, C++, Java, Python a rôzne operačné systémy ako Windows, Linux, iOS alebo Android.

Táto knižnica obsahuje veľké množstvo funkcií a algoritmov pre počítačové videnie, strojové učenie a spracovanie obrazu. Napríklad umožňuje spracovanie obrázkov a videí, čítanie a zápis obrázkových súborov, vykonávanie základných transformácií na obrázkoch (napr. zmena veľkosti, otáčanie, orezávanie) a aplikovanie rôznych filtrov a vylepšení.

OpenCV je možné integrovať s knižnicami strojového učenia ako su TensorFlow a PyTorch, čo umožňuje kombinovať algoritmy počítačového videnia s pokročilými technikami strojového učenia pre úlohy, ako je klasifikácia obrázkov, detekcia objektov a sémantická segmentácia. [10]

3.3 Trénovanie

Aby sme boli schopný natrénovať model na množstve obrázkov, budeme potrebovať väčší výpočtový výkon. Na načítanie, predspracovanie údajov, správu pamäte a koordináciu výpočtov budeme potrebovať výkon CPU. Avšak na prácu s vysokorozmernými dátami ako sú obrázky, budeme potrebovať výkon grafickej karty. GPU urýchľuje tréningový proces vykonávaním paralelizovaných maticových operácií, ktoré vyžadujú algoritmy hlbokého učenia. Nesmieme zabudnúť na dostatok pamäte RAM, ktorú potrebujeme na ukladanie parametrov modelu, prechodných aktivácií a gradientov počas tréningu. Taktiež pomáha urýchliť prístup k údajom, najmä pri načítavaní obrázkov a anotácií z disku.

Pre túto časť budeme používať GPU servery katedry aplikovanej informatiky saturn, uran a neptun.

3.3.1 saturn, uran, neptun

Saturn, uran a neptun sú linuxové servery určené na zbiehanie výpočtov na GPU. Na týchto serveroch je nainštalovaný Ubuntu s proprietárnymi drivermi od nVidie. Prístup je možný cez terminál pomocou ssh/mosh alebo cez grafické prostredie MATE, ktoré je prístupné cez RDP.

Hardvér saturnu

CPU AMD Ryzen 7 2700 (8 jadier, 16 vlákien, 3.2–4.1 GHz)

RAM 48 GB

HDD 2 × 2 TB (RAID 1)

SDD 512 GB (používaný ako LVM cache)

GPU NVIDIA Titan V, NVIDIA XP (každá s 12 GB pamäte)

Hardvér uranu a neptunu

CPU AMD Ryzen 9 5900X (12 jadier, 24 vlákien)

RAM 64 GB

SSD 1 TB

GPU 2 × NVIDIA GeForce RTX 3060 (každá s 12 GB pamäte)

3.4 Detekcia a vyhodnotenie

Natrénovaný model môžeme použiť na detekciu na nových obrázkoch alebo videách. Výsledok z detekcie väčšinou pozostáva z ohraničujúcich obdĺžnikov okolo detekovaných objektov, ktoré obsahujú priradenú triedu a skóre spoľahlivosti. Tieto informácie môžu byť vizualizované na vstupnom obrázku.

Následne chceme vyhodnotiť úspešnosť modelu. Aby sme zistili ako dobre model funguje, musíme porovnať výsledky z detekčného modelu s tými, ktoré sú označené ako správne. Typickou metódou je napríklad Intersection over Union.

3.4.1 Intersection over Union (IoU)

IoU meria prekrytie medzi detekovaným ohraničujúcim obdĺžnikom a správnym obĺžnikom. Najprv sa vypočíta plocha prekrytia týchto dvoch obdĺžnikov. Následne sa táto plocha vydelí plochou, ktorá vznikne spojením obidvoch obĺžnikov. Matematicky to môžeme zapísať ako:

$$J(A, B) = \frac{A \cap B}{A \cup B} \tag{3.1}$$

V prípade IoU je interpretácia metrických hodnôt jednoduchá. Čím väčšie prekrytie, tým vyššie skóre a teda lepší výsledok [6]. Ale aby sme mohli porovnávať obrázky pomocou tejto metódy, museli by byť detekované obrázky a správne obrázky rovnaké, čo v našom prípade nie je také jednoduché. Správne obrázky generujeme a detekované obrázky snímame kamerou, takže rozdiely v uhloch a perspektíve kamery môžu viesť k výrazným rozdielom v polohách objektov. Takéto nesprávne zarovnanie môže viesť k nepresnému prekrytiu ohraničujúcich obdĺžnikov. Správne zarovnanie by sme mohli dosiahnuť tým, že by sme zistili zarovnanie z detekovaného obrázka a aplikovali ho na generovaný obrázok, toto však vyžaduje náročnejšie spracovanie obrázka. Preto sme sa rozhodli použiť jednoduchšiu metódu.

3.4.2 Porovnávanie reťazcov

Ďalší prístup ako môžeme porovnať výsledky je pomocou porovnávania reťazcov. Notový záznam sa číta po riadkoch zľava doprava a zhora nadol, preto si môžeme triedy generovaných objektov uložiť ako reťazec a porovnať s reťazcom tried detekovaných objektov. Tieto reťazce môžeme porovnávať napríklad pomocou Levenshteinovej vzdialenosti 3.4.2.

Levenshteinova vzdialenosť

Levenshteinova vzdialenosť je reťazcová metrika na meranie rozdielu medzi dvoma sekvenciami. Meria koľkokrát musíme nahradiť, vložiť alebo vymazať znak, aby sme zmenili jeden reťazec na druhý [9].

Pri detekcií obrázku sa môže stať, že model nesprávne priradí triedu objektu, nezdetekuje objekt, ktorý má byť detekovaný alebo zdetekuje objek, ktorý sa nenachádza na obrázku či na danej pozícií. Tieto možnosti nám predstavujú znaky, ktoré treba zmeniť, vložiť alebo vymazať z reťazca.

3.4.3 Usporiadanie zdetekovaných objektov

Problém môže nastať, keď chceme usporiadať množinu zdetekovaných objektov podľa riadkov zľava doprava a zhora nadol. Natrénovaný model nám ako výsledok dá neusporiadanú množinu týchto objektov. Pokiaľ by boli riadky len vodorovné, tak by sme usporiadali objekty najprv podľa y-ovej súradnice a potom podľa x-ovej. Notový zápis však môže byť odfotený z rôznych uhlov a kvôli tomu riadky nie sú vodorovné. Môže sa stať, že objekty z druhého riadku budú na úrovni prvého. Preto najlepším riešením bude najprv oddeliť objekty podľa riadkov a potom ich usporiadať podľa x-ovej súradnice. Na oddelenie objektov podľa riadkov môžeme použiť napríklad Houghovú transformáciu, K-means Clustering algoritmus alebo RANSAC.

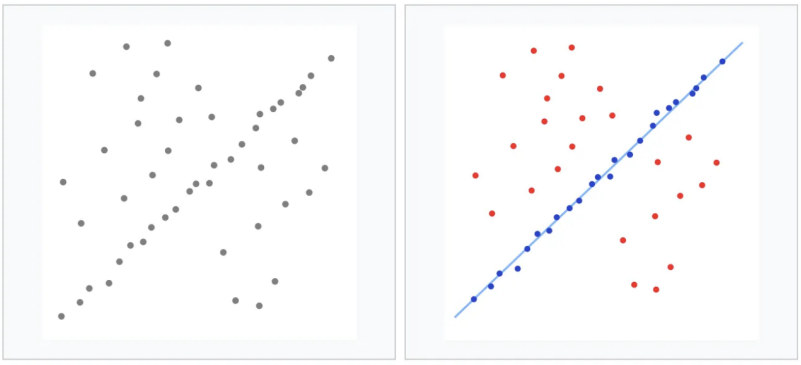
Houghova transformácia

Houghova transformácia je technika, ktorá sa používa pri spracovaní obrazu a počítačovom videní na detekciu tvarov, najmä čiar a kriviek, v rámci obrazu [13]. Keďže noty ležia na notovej osnove, ktorá sa skladá z čiar, mohli by sme tieto čiary dekovať a na základe ich pozícií rozdeliť detekované noty do skupín podľa riadkov.

K-means Clustering algoritmus

K-means clustering je populárny algoritmus strojového učenia bez učiteľa, ktorý sa používa na zoskupovanie bodov/objektov do skupín alebo zhlukov na základe podobnosti. Umožňuje nám to rozdeliť neoznačené dáta do určitého počtu skupín bez potreby akéhokoľvek učenia. Tento algoritmus je založený na centroidoch, kde každý zhluk je spojený s centroidom. Cieľom je minimalizovať súčet vzdialeností medzi dátovým bodom a ich zodpovedajúcimi zhlukmi. [12]

Nevýhodou tohto algoritmu je, že musíme poznať počet skupín, do ktorých chceme dáta rozdeliť. Tento údaj vieme získať zo správneho notového záznamu pri porovnaní s detekovaným notovým záznamom. Informáciu, podľa ktorej majú byť dáta zoskupené, mu môžeme definovať. Hlavným kritériom, podľa ktorého bude v našom prípade



Obr. 3.2: Čiara odhadnutá pomocou RANSAC s mnohými odľahlými údajmi.

algoritmus rozdeľovať dáta, je y-ová súradnica.

Random sample consensus (RANSAC)

RANSAC je robustný algoritmus používaný v strojovom učení a počítačovom videní na odhad parametrov modelu, v ktorom sa môžu nachádzať odľahlé hodnoty. Je to iteratívny algoritmus, ktorý opakovane vyberá náhodné podmnožiny a prispôsobuje model týmto podmnožinám. Cieľom je nájsť model, ktorý dobre funguje s medziľahlými hodnotami. Tento model sa použije na klasifikáciu zostávajúcich údajov, buď ako medziľahlých hodnôt, alebo odľahlých hodnôt.

RANSAC sa používa aj na prispôsobenie čiar, rovín alebo iných geometrických primitív množiny 2D alebo množiny 3D bodov, a to aj v prítomnosti odľahlých bodov. Napríklad môže pomôcť nájst najlepšiu čiaru, ktorá prechádza cez veľa bodov, aj keď existujú nejaké zlé body, ktoré narúšajú dáta. Môžeme to vidieť na obrázku 3.2. [4] Keďže noty v notovom zázname sú usporiadané, vo väčšine zápisov, v riadkoch, môžeme tento algoritmus použiť na výstup z detekovaného obrázku, aby sme detekované objekty usporiadali podľa čiar.

3.4.4 Rekonštrukcia notového záznamu

Keď sa nám podarí usporiadať objekty z detekovaného obrázka, môžeme tieto informácie použiť na rekonštrukciu nového notového záznamu s inou farebnosťou podľa preddefinovaných farieb. Pre túto časť môžeme použiť napríklad pythonovskú knižnicu music21, ktorá nám umožňuje generovať farebné noty.

Kapitola 4

Implementácia

V tejto kapitole sa budeme zaoberať konkrétnou implementáciou jednotlivých častí. Na celú prácu sme používali programovací jazyk Python, pre ktorý sme sa rozhodli na základe dostupnosti potrebných knižníc v tomto jazyku.

4.1 Tvorba notového záznamu pomocou music21

Ako sme spomenuli v časti 3.2, pre tvorbu notových záznamov sme sa rozhodli používať pythonovskú knižnicu music21. V tejto časti si priblížime podrobnejší postup ako sme postupovali pri tvorbe notových záznamov pre trénovanie a vyhodnotenie modelu.

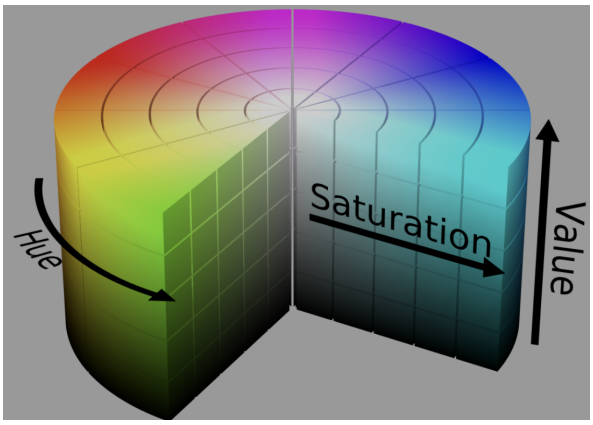
4.1.1 Vytvorenie notového objektu

Na vytváranie nôt používame `note` modul z knižnice music21. Pomocou triedy `note.Note` vytvoríme notový objekt, ktorý podrobnejšie definujeme pomocou vstupných parametrov. Napríklad na vytvorenie celej noty C4 použijeme príkaz `note.Note('C4', type='whole')`. Aby sme mohli pracovať s viacerými notami, musíme ich priradiť do triedy `Stream`. Táto trieda a jej podtriedy sú kontainery pre základné objekty ako sú noty.

Notové záznamy sme sa rozhodli generovať náhodne s určitými obmedzeniami. Pre každý notový záznam sme náhodne vybrali počet riadkov od 1 po 5 a typ nôt (celá, pólová alebo štvrtinová). Ďalej sme pre každý riadok vybrali počet nôt od 6 po 16, tie sme generovali z množiny (C4, D4, E4, F4, G4, A4, B4, C5).

4.1.2 Zafarbenie noty

Aby sme mohli rozpoznávať notové záznamy s farebnými notami, musíme vytvoriť takéto záznamy a natrénovať model na ich rozpoznávanie. Knižnica music21 nám umožňuje meniť štýl noty a teda aj farbu hlavičky noty. Keď máme vytvorený objekt noty po-



Obr. 4.1: Farebný priestor HSV.

mocou `n = note.Note()` môžeme meniť jej štýl. Napríklad príkazom `n.style.color = 'red'` zmeníme farbu noty na červenú. Na priradenie farby môžeme použiť viacero formátov. Môžeme ju definovať pomocou mena alebo pomocou rgb, rgba alebo hexadecimálneho kódovania.

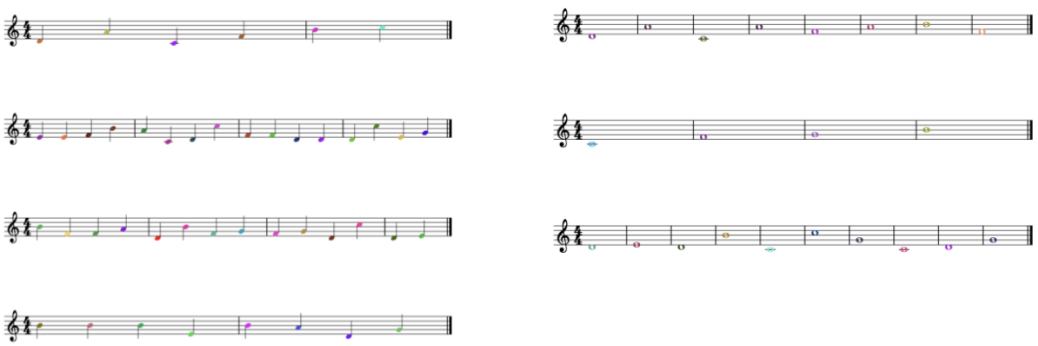
Farby sme generovali náhodne z farebného priestoru HSV. HSV priestor môžeme vidieť na obrázku 4.1. Aby sme sa vyhli príliš bledým farbám, generovali sme hodnotu S z rozsahu 0,5 až 1. Aby sme sa vyhli odtieňom čiernej farby, generovali sme hodnotu V z rozsahu 0,3 až 1. Následne sme farbu pomocou funkcie `colorsys.hsv_to_rgb(h, s, v)` previedli do RGB formátu.

Hoci chceme v našom prípade mať každú výšku noty rovnako zafarbenú, pri trénovacích dátach sme sa rozhodli generovať farbu náhodne pre každú notu, aby model očakával farebné noty, ale nenaučil sa ich rozpoznávať na základe farby.

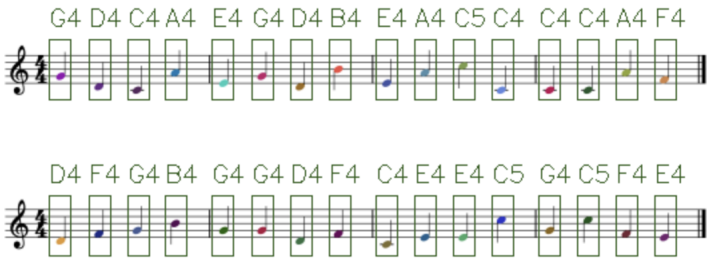
4.1.3 Vykreslenie notového objektu

Keďže chceme trénovať model na obrázkoch notových záznamov, musíme tieto záznamy vykresliť. Po priradení potrebnej noty do streamu, môžeme tento objekt zobraziť pomocou metódy `show(format)`. Pomocou argumentu `format` definujeme formát na zobrazenie notového záznamu. Podporované formáty sú napríklad musicxml, text, lilypond, lily.png, lily.pdf alebo musicxml.png.

Hoci je music21 pokročilá knižnica, sama o sebe nie je schopná vytvárať súbory typu PNG alebo PDF. V našej práci však vytvárame notové záznamy vo formáte PNG. Túto funkcionalitu môžeme dosiahnuť nainštalovaním pomocných aplikácií ako je MuseScore alebo Lilypond. Tieto nástroje umožňujú zobrazovať, upravovať a exportovať notové záznamy, a je ľahké ich integrovať s knižnicou music21. Metóda `show()` vytvorí iba dočastný súbor. Na obrázku 4.2 môžeme vidieť vykreslené notové záznamy vo formáte PNG. Ak chceme súbor uložiť, musíme použiť metódu `write(format, cesta)`, ktorá okrem formátu potrebuje ako argument aj cestu priečinku, do ktorého sa má súbor



Obr. 4.2: Vykreslené notové záznamy.



Obr. 4.3: Vykreslená anotácia notového záznamu.

uložiť.

4.1.4 Tvorba anotácie

Ďalším krokom je anotácia notového záznamu. V časti 3.1.1 sme si ukázali ako majú vyzerať ohraničujúce obdĺžniky. Tak ako notové záznamy, aj ohraničujúce obdĺžniky sme generovali v pythone. Music21 nám neumožňuje získať potrebnú informáciu, na ktorej pozícií sa nota nachádza, preto sme si museli najprv predpočítať pozície, na ktorých sa noty budú nachádzať. Dĺžka noty ovplyvňuje počet nôt v rámci taktu, a teda aj pozíciu noty v riadku. Pre tento dôvod sme sa rozhodli používať iba jeden typ noty v rámci notového záznamu, aby sme nemuseli predpočítavať všetky možné pozície. Po vygenerovaní notového záznamu sme si uložili ohraničujúce obĺžniky v potrebnom formáte do samostatného textového súboru, ktorý je rovnako pomenovaný ako PNG súbor notového záznamu. Túto anotáciu si vieme vykresliť, aby sme si overili, či sú noty správne ohraničené. Môžeme to vidieť na obrázku 4.3.

4.2 Augmentácia

Ďalej sme sa venovali aplikovaniu augmentácie na vytvorených trénovacích dátach, aby sme zlepšili presnosť trénovaného modelu na reálnych obrázkoch. Túto časť sme robili pomocou už spomenutej knižnice OpenCV. Aplikovali sme viacero rôznych transformácií na obrázky, aby sme napodobnili šum z reálneho sveta. Pri niektorých typoch transformácií sme museli ronvakú transformáciu aplikovať aj na ohraničujúce obdĺžniky, aby sme zachovali ich správnu pozíciu. Napríklad pri perspektívnej transformácií, ktorá mení pozíciu nôt.

4.2.1 Aplikácia transformácie na ohraničujúce obdĺžniky

Súradnice ohraničujúcich obĺžnikov v YOLO formáte sú v normalizovanom tvare od 0 po 1. Rozmery obrázka sú na rozdiel od súradníc obĺžnikov v pixeloch. Aby sme správne aplikovali transforáciu na anotáciu, museli sme súradnice najprv prepočitať na pixely, následne aplikovať trasformáciu a potom ich naspäť prepočítať do normalizovaného tvaru.

Keď chceme získať x-ovú súradnicu ľavého horného rohu ohraničujúceho obdĺžnika v pixeloch, musíme od pôvodnej x-ovej súradnice odčítať polovicu dĺžky ohraničujúceho obdĺžnika a následne musíme túto hodnotu prenásobiť šírkou daného obrázka. Pre y-ovú súradnicu sme použili rovnaký princíp s výškou.

4.2.2 Náhodné orezanie obrázku

Na rozšírenie množiny vytvorených dát sme sa rozhodli z každého obrázka vytvoriť náhodným orezaním ďalšie dva obrázky, aby sme získali rôzne variácie. Na orezanie obrázka sme náhodne vybrali pomer z možností (0.9, 0.8, 0.7, 0.6, 0.5). To znamená, že ak sme napríklad vybrali 0.9, tak po orezaní bolo zachovaných 90% z obrázka. Pre nižšie hodnoty bol zachovaný nižší pomer. Na orezanie obrázka nám stačí definovať počiatočné a koncové súradnice a použitím zápisu rezu `image[y:y + crop-height, x:x + crop-width]` určíme podmnožinu pôvodného obrázka. Následne sme orezanú časť obrázka zväčšili na pôvodnú veľkosť obrázka, čím sme zmenili aj veľkosti objektov na obrázku. Jednotlivé noty, ktoré boli odstránené z obrázka orezaním, sme museli odstrániť z anotácie, aby nevznikla chyba pri trénovaní tým, že by existovali neplatné súradnice.

4.2.3 Perspektívna transformácia

Na každý obrázok sme so 70% pravdepodobnosťou aplikovali perspektívnu transformáciu. Buď sme aplikovali posunutie po x-ovej a y-ovej súradnici spolu s rotáciou

a zväčšením (náhodne v menšom rozsahu) alebo projektívnu transformáciu. Pomocou `np.array([])` sme si definovali perspektívnu maticu, ktorá reprezentuje danú transformáciu. Následne sme vytvorili nový obrázok s už aplikovanou transformáciou pomocou metódy `cv2.warpPerspective(image, perspective-matrix, (width, height))`, ktorá berie pôvodný obrázok, perspektívnu maticu a dvojicu šírky a výšky ako argumenty.

4.2.4 Ďalšia transformácia

V ďalšom kroku sme na už vytvorené a čiastočne predspracované dáta použili ďaľšiu transformáciu, aby sme pridali šum. So 70% pravdepodobnosťou sme použili jednu z týchto troch možnosti na upravenie obrázka: náhodný šum, gaussovské rozostrenie alebo zmena odtieňa, sýtosti a expozície. Pri tejto transformácií zostáva pozícia nôt zachovaná, takže ju nemusíme aplikovať aj na anotáciu.

Náhodný šum

Kvôli pridaniu náhodného šumu sme menili priemer a štandardnú odchýlku, ktorú sme následne skombinovali so vstupným obrázkom. Priemer určuje tendenciu šumu, zatiaľ čo štandardná odchýlka určuje jeho rozsah a intezitu.

Gaussovské rozostrenie

Aby sme simulovali efekt rozmazania, použili sme gaussovské rozostrenie. Náhodne sme vybrali veľkosť jadra z vopred definovaných možností a aplikovali ho ako jeden z parametrov metódy `cv2.GaussianBlur(image, kernel-size, 0)`, ktorá vytvorí nový obrázok s aplikovanou transformáciou.

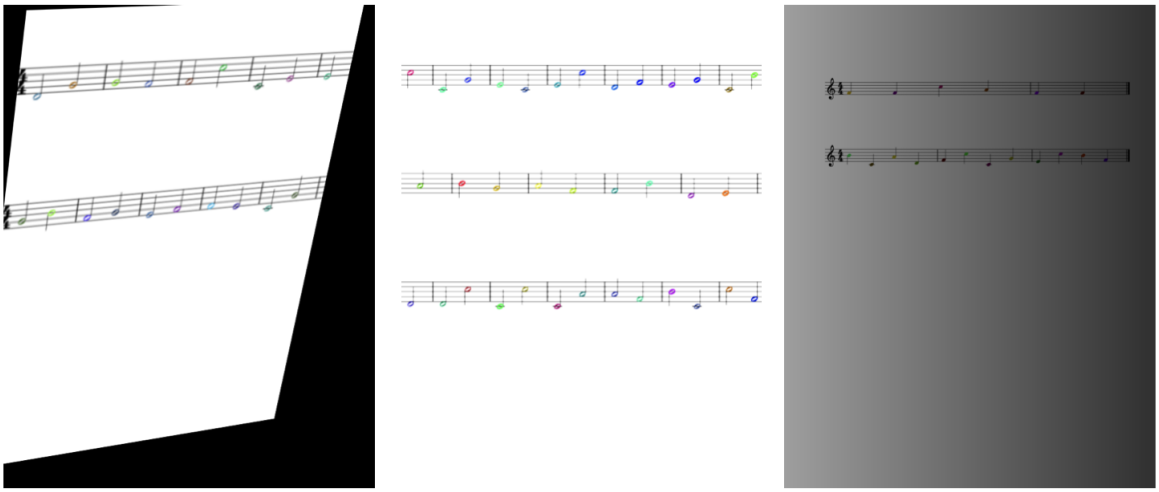
Odtieň, sýtosť a expozícia

Na vylepšenie farby obrázka sme náhodne vybrali z preddefinovanej množiny možností pre posun odtieňa, škálovanie sýtosti a škálovanie expozície. Následne sme tieto úpravy aplikovali na obrázok pomocou farebného priestoru HSV (Hue, Saturation, Value) a skonvertovali naspäť do farebného priestoru BGR.

4.2.5 Svetlo a tieň

Ďalšia vec, ktorá dosť vplýva na fotografiu obrázka je svetlo a tieň. Aby sme napodobnili tieto podmienky, vytvorili sme funkciu, ktorá s 50% pravdepodobnosťou generuje efekt tieňa na vstupných obrázkoch. Táto funkcia vytvára masku s rôznymi úrovňami jasu na základe náhodných parametrov smeru a polohy. Efekt tieňovania sa aplikuje na obrázok zmiešaním pôvodného obrázka s maskou jasu.

Výsledok týchto transformácií možeme vidieť na obrázku 4.4.



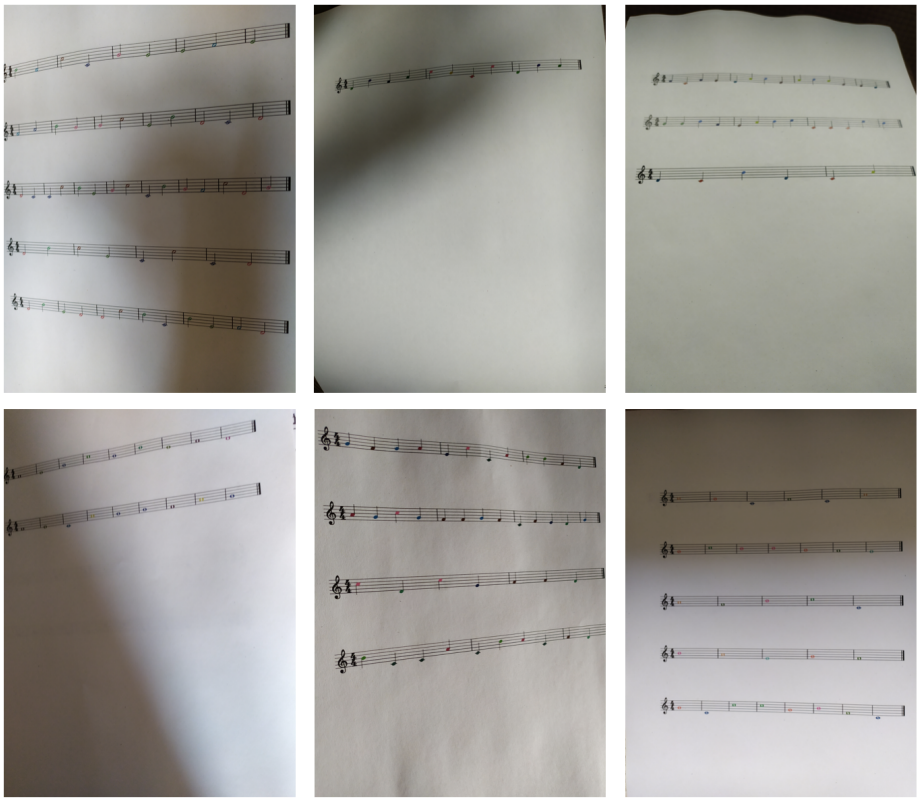
Obr. 4.4: Notové záznamy po aplikácií augmentácie.

4.3 Trénovanie modelu

Na trénovanie modelu sme použili už spomenuté grafické servery katedry aplikovanej informatiky spomenuté v kapitole 3.3. Pomocou spomenutého postupu v kapitolách 4.1 a 4.2 sme vygenerovali 3000 oanotovaných notových záznamov vo formáte PNG. Vygenerované dáta sme rozdelili do dvoch skupín, 2400 obrázkov sme použili ako trénovacie dáta a 600 ako validačné dáta. Tieto dáta sme použili na dotrénovanie modelov. Na serveroch sme pracovali cez príkazový riadok. Vytvorené dáta sme si preniesli pomocou príkazu `scp` na server. Najprv sme si museli nainštalovať CONDA prostredie, v ktorom sme boli schopný nainštalovať potrebné knižnice. Po nainštalovaní balíka ultralytics sme dotrénovali dva predtrénované modely rôznej veľkosti, `yolov8n.pt` a `yolov8m.pt`, na 100 epochách. Pomocou YAML súboru sme definovali triedy a cesty k súborom s obrázkami a anotáciou trénovacích a validačných dát. Natrénované modely sme si naspäť preniesli a testovali na vlastnom zariadení.

4.4 Reálne dáta

Pre potrebu vyhodnotenia jednotlivých modelov sme vygenerovali testovaciu množinu notových záznamov. V tejto množine sme generovali farby podľa výšky noty, aby sme čo najviac napodobnili noty pre detský xylofón. To znamená, že pre každú výšku noty v rámci záznamu, sme na začiatku náhodne vygenerovali jedinečnú farbu, ktorú sme priradili danej note. Takýmto spôsobom sme vytvorili a vytlačili 30 notových záznamov. Následne sme každý notový záznam nafotili trikrát z rôznych uhlov a za rôzneho osvetlenia, aby sme získali väčšiu testovaciu množinu. Vzorku množiny fotografií notových záznamov môžeme vidieť na obrázku 4.5. Na týchto dátach sme testovali natrénované modely.



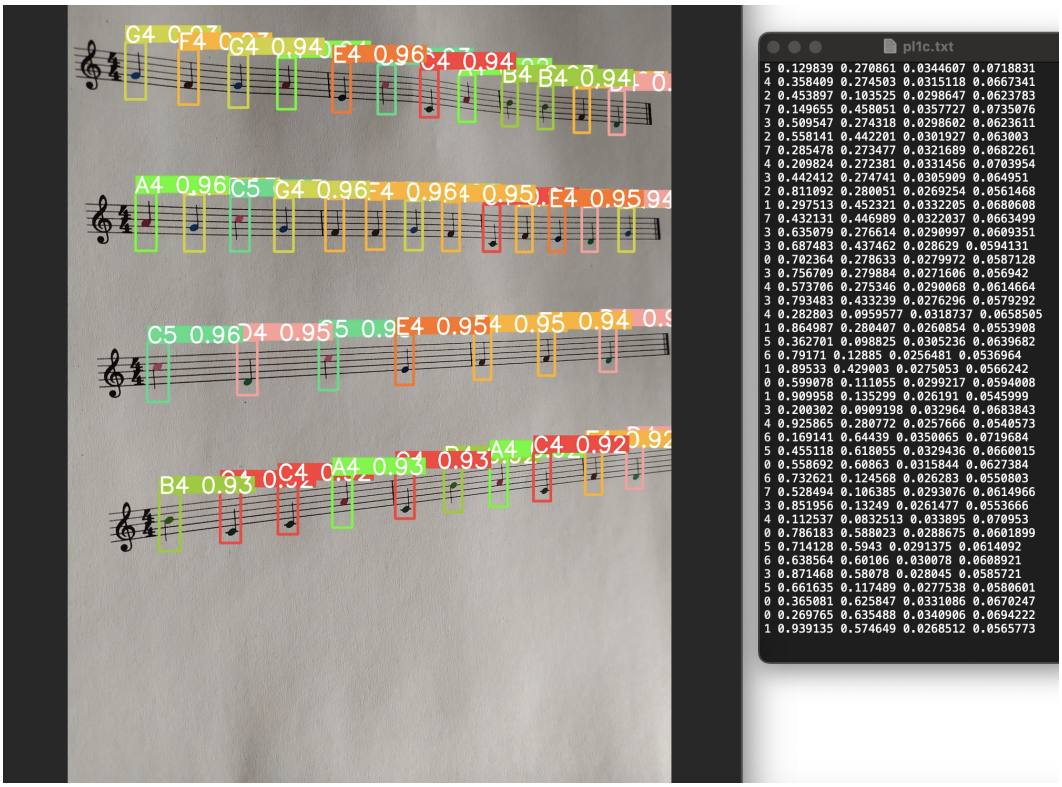
Obr. 4.5: Vzorka testovacej množiny notových záznamov.

4.5 Detekcia, usporiadanie, porovnanie

Natrénovaný model je súbor vo formáte PT. Tento model sme inicializovali pomocou triedy `YOLO()` z knižnice ultralytics, ktorá nám umožňuje ohodnotiť nové obrázky, s ktorými sa model ešte nestretol. Výsledok z tohto hodnotenia môžeme uložiť v rôznych formátoch. Jednou z možností je vykresliť výsledky na vstupnom obrázku. Ďalšou možnosťou je uložiť si výsledok v JSON formáte, alebo ako textový súbor. Aby sme mohli ďalej pracovať s výsledkami, ukladali sme si ich ako textový súbor. V textovom súbore je každý ohraničujúci obdĺžnik uložený do samostatného riadku v YOLO formáte. Na obrázku 4.6 môžeme vidieť výsledky v textovom súbore a ich vizuálnu reprezentáciu na vstupnom obrázku. Tieto obdĺžniky sme sa snažili usporiadať tak, aby sme ich mohli porovnať pomocou už spomenutej Levenshteinovej vzdialenosti (3.4.2).

4.5.1 Houghova transformácia

Prvou technikou, ktorú sme použili bola Houghova transformácia. Pomocou metódy `cv2.HoughLinesP()` a viacerých vstupných parametrov sme sa pokúšali zdetekovať čiary notovej osnovy, aby sme na základe nich mohli zoskupiť detekované objekty v rámci riadkov. Problém nastal, keď papier s notovým záznamom nebol na rovnom povrchu a teda ani čiary neboli úplne rovné. Na obrázku 4.7 môžeme vidieť príklad.



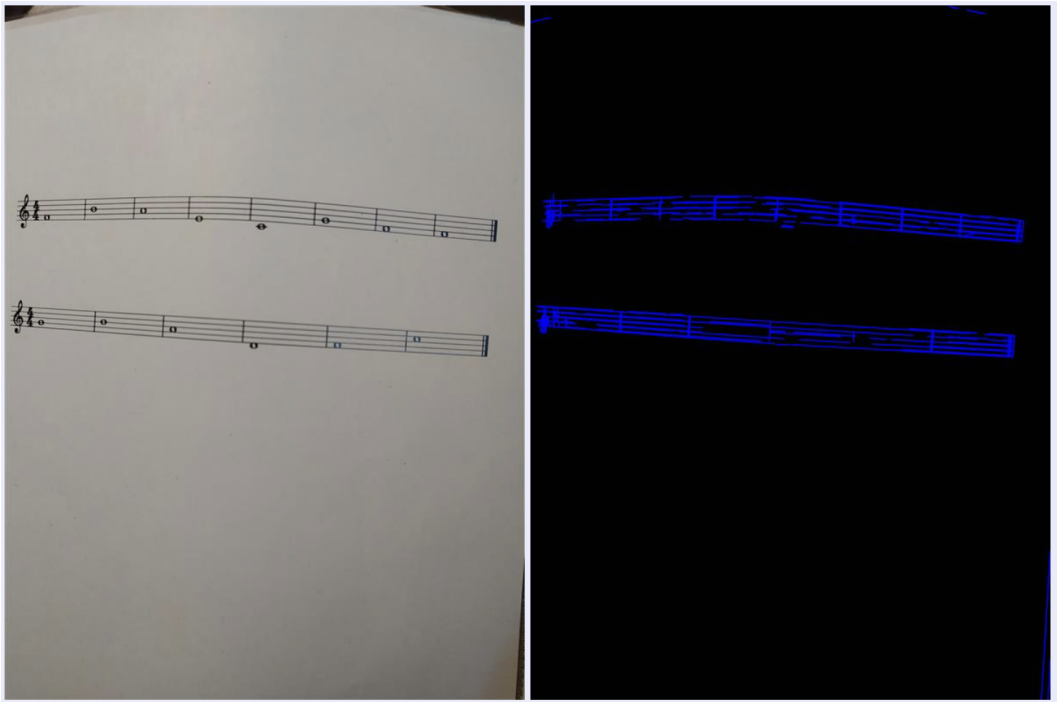
Obr. 4.6: Výsledky detekcie v textovom súbore a vykreslené na vstupnom obrázku.

V takomto prípade by sme potrebovali lepšie nastaviť parametre transformácie alebo predspracovať obrázok. Prvotné testy ukázali, že táto metóda nie je až tak vhodná pre takéto prípady bez potreby extenzívneho predspracovania, preto sme sa rozhodli ju nevyužit ale sme pokračovali s inými technikami.

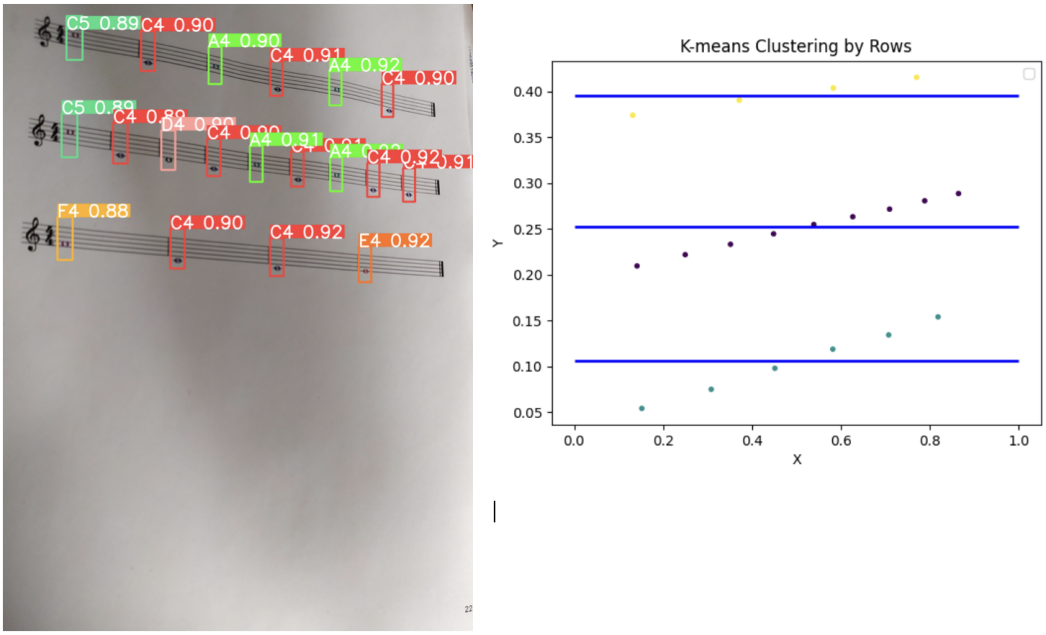
4.5.2 K-means Clustering algoritmus

Ďalšou technika, pomocou ktorej sme sa pokúsili oddeliť noty po riadkoch je K-means Clustering algoritmus. Tu sme si museli pamätať, alebo manuálne definovať, počet riadkov z pôvodného obrázka, do ktorých má objekty rozdeľovať. V prípade, že by sme túto informáciu nemali, mohli by sme si ju dopočítať počas predspracovania obrázka pomocou iných detekčných techník. Z knižnice sklearn.cluster sme použili triedu `KMeans()`, ktorej sme definovali počet klastrov ako parameter. Následne sme použili metódu `fit()` do ktorej sme vložili stĺpcový vektor y-ových súradníc. Táto metóda nám výpočíta stredy klastrov a následne každý bod priradí k tomu najbližšiemu.

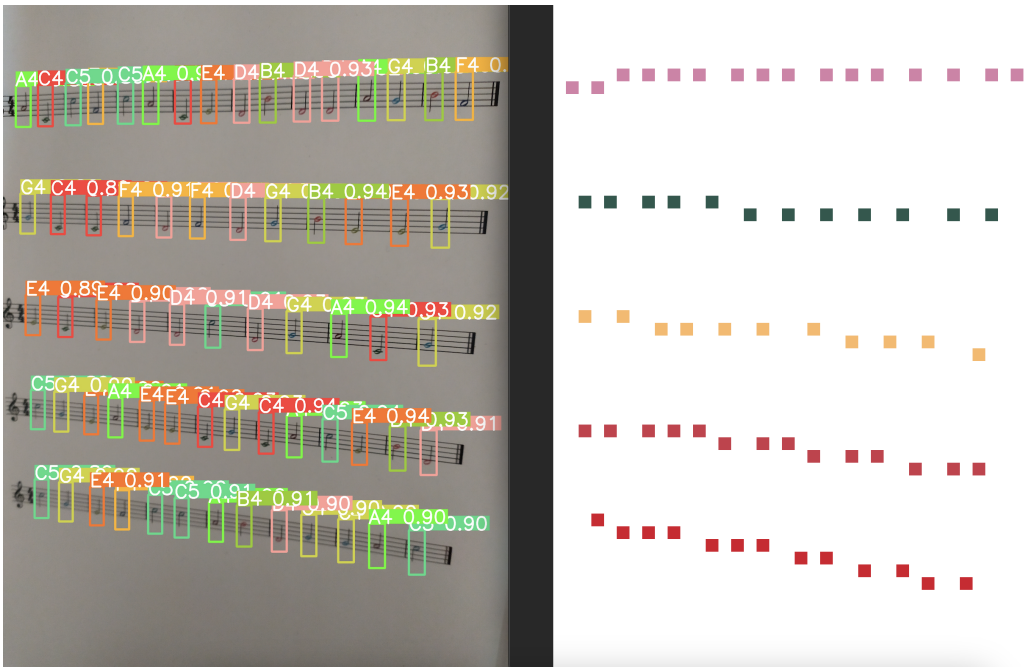
K-means Clustering algoritmus funguje vo väčšine prípadov správne, hlavne ak máme riadky vo vodorovnej polohe. Problém môže nastať pri prílišnom natočení obrázka alebo pri veľkom množstve nesprávne detekovaných objektov. V takom prípade sa môže stať, že stredy klastrov sa nesprávne vypočítajú a tým pádom objekty z jedného riadku môžu byť priradené do nesprávneho.



Obr. 4.7: Detekcia čiar pomocou Houghovej transformácie.



Obr. 4.8: Usporiadanie detekovaných nôt pomocou K-means algoritmu (na obrázku s notamy je pozícia 0,0 v ľavom hornom rohu).



Obr. 4.9: Usporiadanie detekovaných nôt pomocou RANSAC algoritmu.

4.5.3 RANSAC

Posledná technika, ktorú sme použili na oddelenie nôt do riadkov je RANSAC. Na aplikáciu tejto techniky sme použili funkciu `ransac()` z knižnice skimage.measure. Táto funkcia nám na základe predspracovaných súradníc a modelu LineModelND vráti medziľahlé body, ktoré reprezentujú najlepšie vyhovujúcu čiaru. Ako ďalšie parametre môžeme definovať minimálny počet bodov, ktoré majú tvoriť čiaru, rozptyl (vzdialenosť bodov od čiary) a maximálny počet pokusov algoritmu. Po viacerých testoch sme nastavili minimálny počet bodov na číslo 3 (predpokladáme, že menej ako 3 noty sa nebudú nachádzať v riadku) a odchylku na číslo 2. Následne sme iteratívne odstraňovali body ohodnotené ako najlepšia čiara, kým nám neostalo príliš málo bodov na to, aby vznikla ďalšia čiara alebo algoritmus nepriradil žiadnu čiaru. Na obrázku 4.9 môžeme vidieť rozdelené body pomocou RANSAC algoritmu.

4.5.4 Porovnanie

Po oddelení nôt po riadkoch sme usporiadali noty na základe x-ovej súradnice v rámci skupiny a následne sme usporiadali skupiny na základe y-ovej súradnice. Takto usporiadaný reťazec nôt sme porovnávali s reťazcom nôt, ktorý sme si uložili pri vytváraní záznamu. Na porovanie sme použili funkciu `distance()` z knižnice Levenshtein, ktorá porovnáva Levenshteinovu vzdialenosť dvoch reťazcov. Táto funkcia nám vráti vzdialenosť a nie podobnosť. Podobnosť sme si vypočítali odčítaním normalizovanej Levenshteinovej vzdialenosti (`vzdialenosť/max(len(str1), dĺžka(str2))`) od 1. Hodnota 0

predstavuje žiadnu podobnosť a 1 označuje identické reťazce. Nakoniec sme prenásobili výsledok číslom 100, aby sme dostali percentuálnu hodnotu podobnosti.

4.5.5 Zlepšenie

Počas testovanie sme si všimli, že niektoré noty sú detekované viackrát s rôznou presnosťou. Tieto detekcie sme sa rozhodli odfiltrovať a nechali sme iba detekcie s najvyššou presnosťou. Takýmto spôsobom sa nám podarilo zvýšiť úspešnosť modelov.

Kapitola 5

Vyhodnotenie výsledkov a rekonštrukcia

V tejto kapitole si zhrnieme a porovnáme výsledky detekcie jednotlivých modelov, ktoré sme validovali na reálnych dátach a popíšeme si postup rekonštrukcie notového záznamu s vlastným farbením.

5.1 Výsledky

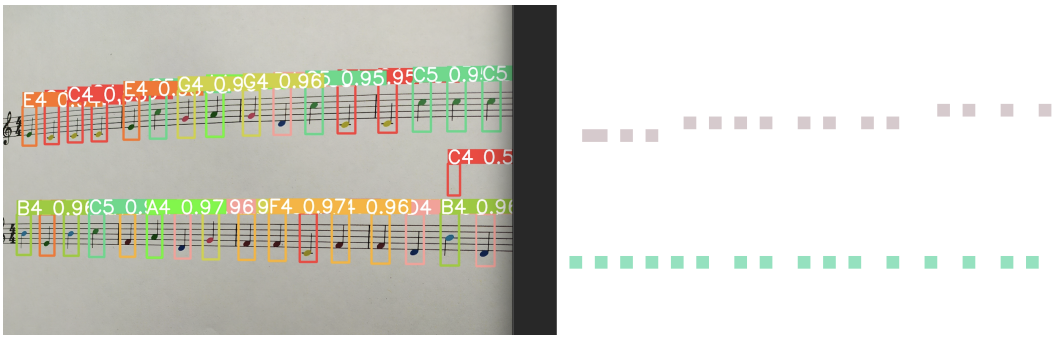
Ako sme spomenuli v časti 4.4, model sme vyhodnocovali na deväťdesiatich reálnych fotografiách. Výsledky detekcií sme porovnali so správnymi výsledkami, čím sme dostali percentuálnu hodnotu úspešnosti jednotlivých fotografií. Nakoniec sme vypočítali priemernú úspešnosť pre obidve metódy, K-means clustering a RANSAC, ktoré sme použili na usporiadanie nôt.

Yolov8n.pt

Ako prvý sme testovali dotrénovaný model `yolov8n.pt`. Napriek tomu, že to je parametrovo najmenší predtrénovaný model, ktorý poskytuje YOLOv8, po našom dotrénovaní dosahuje dosť dobré výsledky. Zachytil takmer všetky noty a vo veľkej väčšine im priradil správnu triedu. Celkovú úspešnosť tohto modelu môžeme vidieť v tabuľke 5.1. Ako vidíme, druhá metóda RANSAC dosahuje výššiu úspešnosť. Je to preto, lebo táto technika hľadá noty, ktoré sa nachádzajú na čiare a tým prirodzene odfiltruje nesprávne detekcie, ktoré sa nachádzajú mimo notovej osnovy. Môžeme to vidieť na obrázku 5.1.

Yolov8m.pt

Následne sme testovali druhý dotrénovaný model `yolov8m.pt`. Tento model je robustnejší a precíznejší, čo sa prejavilo aj na výsledkoch, ktoré môžeme vidieť v tabuľke 5.1.



Obr. 5.1: Odfiltrovaná detekcia pomocou RANSAC algoritmu.

Tabuľka 5.1: Výsledky testovania modelov bez ďaľšieho spracovania.

Model	K-means	RANSAC
yolov8n.pt	95.41%	95.60%
yolov8m.pt	99.33%	99.66%

Tak ako v predošlom prípade, metóda RANSAC mala trocha väčšiu úspešnosť.

Zlepšenie

Ako sme spomenuli v kapitole 4.5.5 podarilo sa nám zvýšiť úspešnosť modelov tým, že sme odfiltrovali duplicitné detekcie. Výsledky po tomto spracovaní môžeme vidieť v tabuľke 5.2. Po tomto procese sa nám podarilo s RANSAC algoritmom dosiahnuť 100% úspešnosť. Jednoduchosť a zúženie problémovej domény výrazne prispela k dosiahnutiu tejto presnosti. Notové záznamy, na ktorých sme trénovali a vyhodnocovali modely neobsahovali nadbytočné symboly a text, ktoré by mohli ovplyvniť detekciu. Taktiež sme pracovali len s generovanými notovými záznamami a nie ručne písanými, ktoré nemajú konzistentú štruktúru.

Tabuľka 5.2: Výsledky testovania modelov po odfiltrovaní duplicitných detekcií.

Model	K-means	RANSAC
yolov8n.pt	98.43%	98.66%
yolov8m.pt	99.67%	100%

5.2 Rekonštrukcia notového zápisu

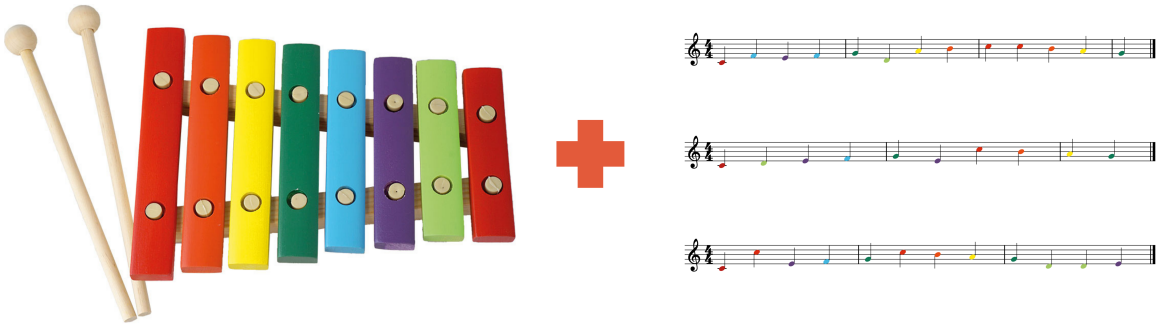
Pomocou natrénovaného modelu sme boli schopní získať informácie o jednotlivých notách, ich výškach a pozíciach. Pomocu RANSAC algoritmu sme získali informáciu o počte riadkov a boli sme schopní správne usporiadať noty. Vďaka týmto informáciam sme dokázali zrekonštruovať notový záznam. Stačí nám vytvoriť jednoduché rozhranie, kde si používateľ navolí farby jednotlivých nôt. V práci sme sa nezaoberali tvorbou rozhrania, ale sme si farby navolili priamo v kóde. Následne s použitím knižnice music21 sme vytvorili nový farbený notový záznam vo formáte PNG alebo PDF, ktorý si môže používateľ v prípade potreby vytlačiť. Na obrázku 5.2 máme znázornený proces prevodu jedného notového záznamu špecifického k danému xylofónu na druhý notový záznam, ktorý je špecfický k inému xylofónu. Tieto rozdielne xylofóny môžeme tiež vidieť na obrázku 5.2.

Ak by sme chceli notový záznam previesť napríklad do formátu MIDI, existujú rôzne knižnice, ktoré su schopné na základe ziskaných informácií vytvoriť záznamy v tomto formáte.

5.3 Budúca práca

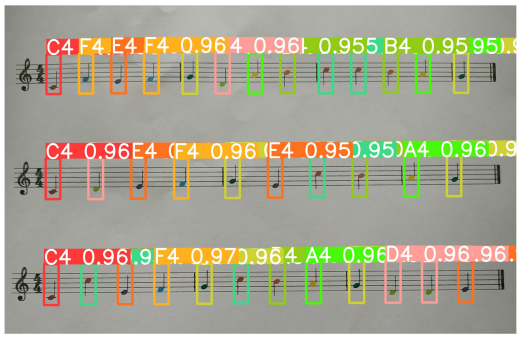
Hoci sa nám podarilo splniť ciele práce a vytvoriť riešenie na rozpoznanie nôt v notovom zápise a previesť ich na notový zápis s inými farbami, stále je miesto na zlepšenie. Mnohé farebné notové záznamy pre xylofón okrem nôt obsahujú aj text piesne. V rámci práce sme sa nezaoberali prípadom, v ktorom notový záznam obsahuje aj text. Túto možnosť je možné otestovať, a v prípade, že by bol model nedostatočný, dotrénovať pre tento prípad. Ďaľšou možnosťou je rozšíriť model o ďaľšie tóny, nakoľko niektoré farebné xylofóny majú rozsah väčší ako 8 tónov.

Pre budúcu prácu je možné uviesť tento systém rozpoznania a rekonštrucie notového záznamu do praxe. Napríklad by sme mohli vytvoriť mobilnú aplikáciu, ktorá by obsahovala intuitívne rozhranie na voľbu farieb pre konkrétne noty. Následne by stačilo odfotiť notový záznam, ktorý by bol automaticky transformovaný podľa navolených farieb. Tento nový záznam by mohol byť zobraziteľný v aplikácií alebo exportovateľný do iných formátov pre ďalšie použitie.

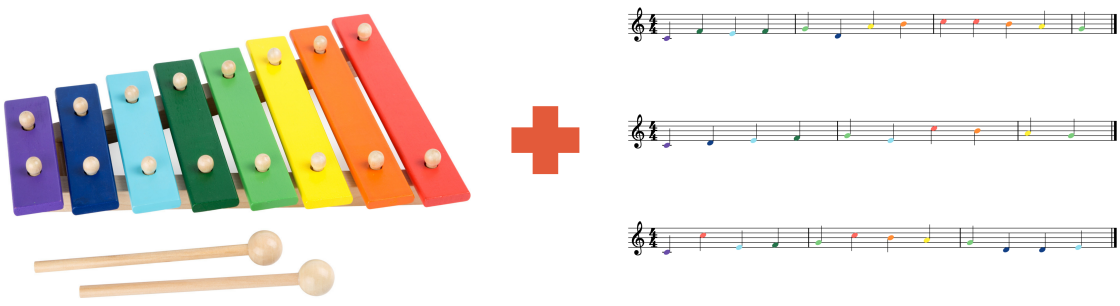


Xylofón 1

Notový záznam pre xylofón 1



Odfotený notový záznam pre xylofón 1
po detekcií



Xylofón 2

Notový záznam pre xylofón 2

Obr. 5.2: Proces prevodu notového záznamu špecifického k jednému xylofónu na notový záznam špecifický k druhému xylofónu.

Záver

V tejto práci sme sa oboznámili s hudobným nástrojom xylofón, jeho verziou pre deti a s farebnými notovými záznamami. Tiež sme si priblížili Optické rozpoznávanie hudby a rôzne techniky a postupy na spracovanie a rozpoznávanie notového záznamu. Rozhodli sme sa použiť prístup pomocou neurónových sietí a natrénovať model YOLOv8 na rozpoznávanie nôt v notových záznamoch.

Aby sme mohli natrénovať model, museli sme vytvoriť a oanotovať dáta. Pomocou pythonovskej knižnice music21 sme náhodne vytvorili notové záznamy. Na zvýšenie úspešnosti rozpoznávania na reálnych fotografiách sme aplikovali augmentáciu na umelo vytvorené dáta. Pomocou knižnice OpenCV sme napríklad aplikovali perspektívnú transformáciu, náhodne orezali obrázok a prídavali náhodny šum zmenou odtieňa, sýtosti a expozície. Takýmto spôsobom sa nám podarilo napodobniť umelo vytvorené dáta tým reálnym. Na týchto dátach sme dotrénovali modely yolov8n.pt a yolov8m.pt na detekciu nôt. Následne sme vygenerovali množinu notových záznamov, ktoré sme odfotili z rôznych uhlov a za rôznych podmienok, na ktorých sme vyhodnocovali dotrénované modely.

Úspešnoť modelov sme vyhodnocovali pomocou Levenshtein algoritmu. Aby sme mohli porovnať detekované noty so správnymi a zrekonštruovať notový záznam, museli sme detekované noty správne usporiadať. Na túto úlohu sme vyskúšali K-means Clustering algoritmus, RANSAC algoritmus a Houghovu transformáciu. Pomocou K-means Clustering algoritmu a RANSAC algoritmu sa nám podarilo úspešne usporiadať noty a otestovať natrénované modely. Pre úspešnosť použitím Houghovej transformácie by sme potrebovali extenzívnejšie predspracovanie, tak sme sa viacej venovali predošlým dvom metódam. S použitím RANSAC algoritmu a knižnice music21 sme zrekonštruovali notový záznam podľa iného farbenia.

Napriek tomu, že sme modely trénovali na umelých dátach, podarilo sa nám dosiahnuť dostatočne vysokú presnosť na reálnych dátach. S robustnejším modelom a dodatočným odfiltrovaním nechcených detekcií sa nám pomocou RANSAC algoritmu podarilo dosiahnuť 100% úspešnosť.

Literatúra

[1] Vienna symphonic library: Xylophone. https://www.vsl.info/en/academy/percussion/xylophone, 2024.

[2] Tariq Alkhalifah, Hanchen Wang, and Oleg Ovcharenko. Mlreal: Bridging the gap between training on synthetic data and real data applications in machine learning. *Artificial Intelligence in Geosciences*, 3:101–114, 2022.

[3] Dana H Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122, 1981.

[4] Chandra Prakash Bathula. Machine learning concept 69: Random sample consensus (ransac). Medium https://medium.com/@chandu.bathula16/machine-learning-concept-69-random-sample-consensus-ransac-e1ae76e4102a, 2023.

[5] Maxence Boels. Image processing: Filters for noise reduction and edge detection. *Medium*, Oct 2019.

[6] CloudFactory Docs. Intersection over union (iou). https://wiki.cloudfactory.com/docs/mp-wiki/metrics/iou-intersection-over-union.

[7] Michael Scott Asato Cuthbert. music21 documentation. https://web.mit.edu/music21/doc/about/what.html.

[8] David Doermann and Karl Tombre. *Handbook of document image processing and recognition*. Springer Publishing Company, Incorporated, 2014.

[9] GeeksforGeeks. Introduction to levenshtein distance. https://www.geeksforgeeks.org/introduction-to-levenshtein-distance/.

[10] GeeksforGeeks. What is opencv library? https://www.geeksforgeeks.org/opencv-overview/.

[11] Zhiqing Huang, Xiang Jia, and Yifan Guo. State-of-the-art model for music object recognition with deep learning. *Applied Sciences*, 9(13):2645, 2019.

[12] JavaTpoint. K-means clustering algorithm in machine learning. JavaTpoint https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning.

[13] Surya Teja Karri. Hough transform. Medium https://medium.com/@st1739/hough-transform-287b2dac0c70, 2019.

[14] Jozef Martiš. Omr: Optical music recognition. Master's thesis, Comenius University in Bratislava, Bratislava, May 2020.

[15] Michael Pilhofer and Holly Day. *Music theory for dummies*. John Wiley & Sons, 2019.

[16] Ana Rebelo, Ichiro Fujinaga, Filipe Paszkiewicz, Andre RS Marcal, Carlos Guedes, and Jaime S Cardoso. Optical music recognition: state-of-the-art and open issues. *International Journal of Multimedia Information Retrieval*, 1:173–190, 2012.

[17] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[18] Florence Rossant. A global method for music symbol recognition in typeset music sheets. *Pattern Recognition Letters*, 23(10):1129–1141, 2002.

[19] Gonzalo Santamaría, César Domínguez, Jónathan Heras, Eloy Mata, and Vico Pascual. Combining image processing techniques, ocr, and omr for the digitization of musical books. In *International Workshop on Document Analysis Systems*, pages 553–567. Springer, 2022.

[20] Elona Shatri and György Fazekas. Optical music recognition: State of the art and major challenges. *arXiv preprint arXiv:2006.07885*, 2020.

[21] Ultralytics. Ultralytics yolov5 documentation: Training custom data. https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/#22-create-labels.

[22] Ultralytics. Ultralytics yolov8: Open-source python library. https://github.com/ultralytics/ultralytics?tab=readme-ov-file.