Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Vizualizácia hudby

Bakalárska práca

2018 **Marián Sabat**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Vizualizácia hudby

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	9.2.1. Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	Ing. Norbert Ádám, PhD.
Konzultant:	

Košice 2018 **Marián Sabat**

Názov práce:	Vizualizácia hudby
Pracovisko:	Katedra počítačov a informatiky, Technická univerzita v Košiciach
Autor:	Marián Sabat
Školiteľ:	Ing. Norbert Ádám, PhD.
Konzultant:	
Dátum:	25. 5. 2018
Kľúčové slová:	

Abstrakt: ABSTRAKT

Thesis title:	Music Visualizer
Department:	Department of Computers and Informatics, Technical University of Košice
Author:	Marián Sabat
Supervisor:	Ing. Norbert Ádám, PhD.
Tutor:	
Date:	25. 5. 2018
Keywords:	
Abstract:	ABSTRAKT

Tu vložte zadávací list pomocou príkazu
`\thesisspec{cesta/k/suboru/so/zadavacim.listom}`
v preambule dokumentu.

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval(a) samostatne s použitím uvedenej odbornej literatúry.

Košice, 25.5.2018 .....

Vlastnoručný podpis

Poďakovanie

Obsah

Úvod **1**

1 Syntéza dát **3**
1.1 Zmena textu na reč 3
1.2 Prevod reči na text 4
1.3 Rozpoznávanie obsahu dát 4
1.4 Generovanie obrázkov 5

2 Teoretický základ **6**
2.1 Hudobné dáta 6
2.2 Neurónové siete 7
2.3 Variačné autoenkódery 9
2.4 Generatívne konkurenčné siete 10

3 Tvorba datasetu **11**
3.1 Zvukové dáta 11
3.2 Obrazové dáta 12
3.3 Zber dát 13

4 Detekcia farby **15**
4.1 Návrh 15
4.2 Implementácia 16

5 GAN Vizualizátor **18**
5.1 Návrh 18
5.2 Implementácia 19
5.3 Podmienená generatívna sieť 21

5.4 Zmena parametrov 21
5.5 Zväčšovanie obrázkov 24

6 VAE Vizualizátor **26**
6.1 Návrh 26
6.2 Implementácia 27
6.3 Trénovanie 29

Literatúra **32**

Zoznam obrázkov

2.1 Model formálneho neurónu. 8

4.1 Graf aktivačnej funkcie ReLu. 16

5.1 Chyba generátora pri trénovaní neoptimalizovanej siete. 22
5.2 Chyba diskriminátora pri využití RMSPropOptimizer (hore) a AdamOptimizer (dole). 24
5.3 Zväčšenie obrázku z datasetu využitím CPPN siete. 25

6.1 Výstup variačného autoenkódera pri Mozartovej skladbe Turkish March 30

Zoznam tabuliek

2.1 Akustické vlastnosti zvukových signálov 7

Úvod

Systémy, ktoré sa dokážu učiť z dát sú už dnes prístupné verejnosti. Je čoraz jednoduchšie študovať techniky strojového učenia a preto progres v tomto odvetví je skutočne viditeľný. Množstvo dát a dobrá výpočtová technika majú za následok, že v takmer všetkých oblastiach sa zavádza nejaký druh umelej inteligencie. Počítače, ktoré by rozumeli informáciám by znamenali revolúciu v našich životoch. Program, ktorý by dokázal vygenerovať obraz na základe hudobného podkladu, na základe emócií a nálad, ktoré sú obsiahnuté v hudbe by bol pokrok ku umelej inteligencii, ktorá by skutočne rozumela dátam.

Proces syntézy jedného druhu informácií na iní je pre ľudí prirodzený no pre stroj je to neľahká úloha. Avšak progres v neurónových sieťach a v generatívnych algoritmoch umožňuje klasifikáciu jednej informácie a jej následnú zmenu na inú formu. Stále ale existuje množstvo prekážok v realizácií tohto problému.

To všetko nás privádza k otázke, sú dnešné neurónové siete schopné previesť hudobnú skladbu na zmysluplný obraz? Prevod hudobnej informácie na obrazovú si vyžaduje určitý stupeň kreativity a znalostí. V našej práci sa budeme snažiť zodpovedať tento problém. Budeme sa snažiť vytvoriť model, ktorý by simuloval ľudskú kreativitu.

V prvom rade je dôležité upraviť dáta, ktoré budeme analyzovať. Ide o zvukové signály, ktoré ako také sú nespracovateľné dnešnými algoritmami strojového učenia. Je nevyhnutné aby sme tieto dáta upravili na použiteľnú formu. Ďalším krokom je zistenie či sú počítače vôbec schopné priradenia najjednoduchšej obrazovej formy, čiže farby, k hudobným skladbám. Úspešné splnenie tejto úlohy bude dobrým predpokladom pre vytvorenie finálneho modelu, ktorý dokáže generovať obrázky na vyššej kreatívnej úrovni.

Naša práca je preto rozdelená presne podľa týchto celkov. Prvé kapitoly poskytnú súčasné úspechy v syntéze dát a teoretický základ pre naše riešenie. V

ďalších kapitolách postupne prejdeme naše výsledky od najjednoduchších modelov až po tie zložitejšie. Na konci poskytneme porovnanie nami vytvorených systémov a odvodenie záverov.

1 Syntéza dát

Už od čias pred počítačmi ľudia využívali abstrakciu skutočných dát v podobe čísel a matematiky. S vývojom výpočtovej techniky prišli aj nové spôsoby zmeny jedného typu informácií na iný. Dnes existuje mnoho systémov určených na tento proces.

1.1 Zmena textu na reč

Už v osemdesiatych rokoch dvadsiateho storočia, keď Steve Jobs predstavil nový Macintosh, počítač vedel hovoriť. Systémy, ktoré používajú zmenu textu na reč sú dnes bežná vec, a preto je ťažké predstaviť si svet bez nich [1]. Aj keď sú zaužívané stále existuje priestor na zlepšenie. Hlas starého Macintosha bol zreteľne umelý, no dnes existujú programy, ktoré dokážu simulovať ľudskú reč takmer na nerozpoznanie od živých ľudí. Tieto syntetizátory našli svoje využitie napríklad v telekomunikáciách. Primitívne úlohy vykonávané cez telefón sú zverené počítačom. Ľudia si už zvykli, že keď volajú niekam aby si niečo vybavili je normálne ak sa im ozve stroj. Syntetizátory našli svoje využitie aj vo vzdelávaní. Učenie jazykov z pohodlia domova je možné aj vďaka tomu, že počuť ako sa slovo vyslovuje môžeme bez prítomnosti skúseného rečníka či cudzinca. Zrakovo postihnutí práve vďaka technológií zmeny textu na reč môžu využívať prístroje, ako napríklad počítače, telefóny a iné, bez ktorých sa dnes už nezaobídeme. Nie len slepím, ale aj nemým a inak telesne postihnutým pomáhajú čítačky textu každý deň. Svoje využitie našli v mnohých oblastiach od vedy a výskumu až po zábavný priemysel.

1.2 Prevod reči na text

Pre ľudí veľmi jednoduchá úloha, rozpoznanie reči, je pre číselné systémy netriviálna záležitosť. Fourierové transformácie a úprava dát nám dávajú šancu na extrakciu vhodných informácií, ktoré sa stali vhodným nástrojom na detekciu slov [2]. Systém schopný prevodu hovorenej reči na text je nevyhnutnosťou v prípadoch, keď človek nemôže alebo nedokáže použiť klávesnicu či iní mechanický vstup. Osobní asistenti ako Cortana alebo Ok Google by bez týchto syntetizátorov nedosiahli takej popularity. Prevodníky reči na text majú veľký vplyv na to ako pracujeme s našimi zariadeniami. Funkcie ako preklad z jedného jazyka do druhého v reálnom čase sa zavádzajú do programov určených na komunikáciu a zmenšujú tak priepasť medzi ľuďmi rôznych národností. To by nebolo možné ak by jadrá týchto programov nestáli na technológiách prevodu reči na text a textu na reč.

1.3 Rozpoznávanie obsahu dát

S úlohou syntézy informácií súvisí aj problém reprezentovania významu dát. Naučiť počítače rozpoznávať čo sa nachádza na obrázku je dnes silno skúmaná oblasť. Mnohé automobilové spoločnosti sa snažia vytvoriť samo jazdiace vozidlá, ktoré dokážu spozorovať prekážky okolo seba a zareagovať rýchlejšie ako by to dokázal ktorýkoľvek človek. Vďaka najnovším metodikám, ako napríklad využite konvolučných neurónových sietí, ľudia vytvorili programy, ktoré dokážu rozpoznať čo sa nachádza na obrázkoch a rozoznať jednotlivé objekty [3]. Takéto technológie sa dajú využiť napríklad pri vyhľadávaní. Spoločnosti ako Google ponúkajú cloudové služby stvorené presne na tieto účely [4].

Neurónové siete dokážu analyzovať obrázky na úrovniach aké doposiaľ neboli možné. V roku 2016 výskumníci z Montrealu a Toronta vytvorili model, ktorý dokázal analyzovať obrázky a vytvoriť textový popis týkajúci sa obsahu obrázku [5]. V spojení s čítačkou textu by sme takto mohli ešte viac uľahčiť prístup k informáciám aj pre zdravotne postihnutých ale aj využiť takúto technológiu na ešte lepšie výsledky. Detekcia tvári sa zavádza aj do mobilných telefónov a využíva sa na odomykanie uzamknutej obrazovky. Facebook vytvára systém, ktorý by využil rozpoznanie tváre a využil tieto dáta ako heslo pre používateľa [6]. Analýza

obrazu ale neostáva len pri rozpoznávaní objektov. Inteligentné systémy dokážu rozpoznať rôzne vlastnosti obrazu. Google vytvoril systém, ktorý nazval Deep dream. Deep dream dokáže rozpoznať vlastnosti obrazu a upraviť pôvodný obrázok pridaním vrstiev, ktoré majú podobné vlastnosti. Vytvorené obrázky potom vyzerajú ako halucinácie. Mnohé iné webové aplikácie zase využívajú siete, ktoré sa naučia ako rozpoznať štýl obrazu a vďaka tomu dokážu preniesť štýl na úplne iný obrázok. Vznikajú tak zaujímavé filtre na úpravu obrázkov a fotiek.

1.4 Generovanie obrázkov

V posledných rokoch systémy strojového učenia preukazujú výsledky v generovaní nových dát. Syntéza textu na obraz je jednou z najnovších prác v tomto odvetví [7]. Nakoľko ide o experiment, tak komerčné využitie ešte neexistuje. Ale pri zlepšení tejto technológie sa môže vytvoriť obrovský potenciál. Takéto výskumy prebiehajú na celom svete a na prechod na trh určite nebudeme dlho čakať. Umelo generované obrázky dokážeme vďaka tomu ako sú vytvorené ďalej upraviť. Nakoľko sú to obrázky vytvorené pomocou matematického modelu vieme upravovať obsah veľmi jednoducho. Príkladom sú generované obrázky ľudí, v ktorých sa dá upravovať napríklad to či sa osoba usmieva alebo nie, či má okuliare alebo mnoho iných vlastností [8].

2 Teoretický základ

Ako už bolo avizované naša práca sa zaoberá neurónovými sieťami. Preto v tejto kapitole zhrnieme základné pojmi a problémy, pred ktorými stojíme.

Algoritmus, ktorý sa učí z dát, poskytuje dobé výsledky len vtedy ak má dobré dáta, z ktorých sa učí. Náš model sa bude učiť z hudobných dát. Zvukové analógové signály uložené v digitálnej forme sú príliš rozsiahle nato aby sme ich mohli využiť ako vstup do neurónovej siete. Jedinou možnosťou je úprava takýchto dát na jednoduchšiu formu.

2.1 Hudobné dáta

Ľudia vnímajú hudbu na rôznych úrovniach. Rôzne vlastnosti harmonických zvukov vyvolávajú u nás rozličné emócie. Napríklad kým vzrušenie blízko súvisí s tempom, tak výška tónov a hlasitosť skôr určujú náladu skladby [9]. To ako vnímame hudbu má preto veľký vplyv na našu predstavivosť.

Hudobné dáta sú ľahko dostupné na internete. Čo sa týka hudby existuje niekoľko zdrojov pre informácie [10]:

- Hudobné metadáta,
- Akustické vlastnosti,
- Slová,
- Hudobná kritika,
- MIDI súbory,
- Hudobné skóre.

Tabuľka 2.1: Akustické vlastnosti zvukových signálov

Sada vlastností	**Extrahovateľné vlastnosti**
Energia	Dynamická hlasitosť, výkon zvuku, celková hlasitosť, špecifické koeficienty citlivosti na hlasitosť
Rytmus	Diagram úderov, vzor rytmu, histogram rytmu a tempo, sila rytmu, pravidelnosť rytmu, jasnosť rytmu, priemerná frekvencia nástupu, priemerné tempo
Časové vlastnosti	Nulové priechody, logaritmus času útoku
Spektrálne vlastnosti	Spektrálne centroidy, spektrálne vyhodnocovanie, spektrálny tok, spektrálne merania rovinnosti, spektrálne hrudkové faktory, mel-frekvenčné kepstrálne koeficienty
Harmónia	Jasnosť kľúča, hudobný režim, harmonická zmena, diagram stúpania

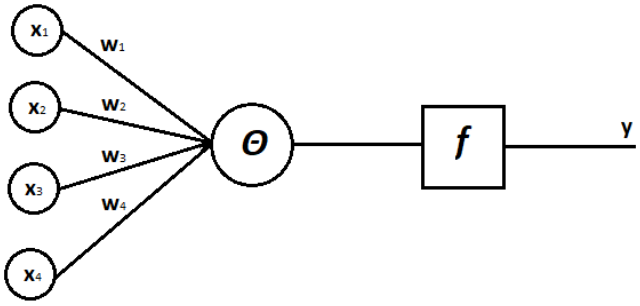
Rôzne typy informácií majú využitie v rôznych prípadoch. Nás budú zaujímať akustické vlastnosti, medzi ktoré patrí energia, rytmus, časové a spektrálne vlastnosti [9]. Pre každú z týchto sád vlastností existuje niekoľko vlastností, ktoré sa dajú extrahovať známymi algoritmami a programami. V tabuľke 2.1 sme uviedli zopár významných údajov, ktoré by sme mohli využiť pre trénovanie našich modelov. Tieto údaje opisujú hudobnú informáciu a majú značne menšiu veľkosť ako skladba uložená vo formáte mp3 alebo wav.

2.2 Neurónové siete

Tak ako mozog, neurónová sieť je spojenie viacerých neurónov do siete. Neurón na vstupe príjma informácie, ktoré pozmení a pošle na výstup, čo môže byť vstup ďalšieho neurónu.

Formálny neurón alebo perceptrón (obrázok 2.1) má na vstupe n aktivít [11]. Označme vstup ako vektor $x = (x_1, x_2, x_3, ...x_n)^T$, kde T označuje transponovaný vektor.

Všetky vstupné kanály majú svoju váhu, označme preto váhy vstupov ako vektor $w = (w_1, w_2, w_3, ...w_n)^T$. Celkový vstup perceptrónu sa potom vypočíta ako sú-



Obr. 2.1: Model formálneho neurónu.

čet súčinu vektorov x a w, a prahu excitácie Θ. Pre získanie výstupu vstupy prejdú aktivačnou funkciou. Pre výstup y potom platí

$$y = f(w^T x + \Theta)$$

Neuróny v sieťach sa spájajú do vrstiev. Nech je vrstva tvorená m neurónmi a každý neurón má n vstupných kanálov. Označme váhu j-teho vstupného kanála do i-teho neurónu ako w_{ij}. Potom môžeme vytvoriť váhovú maticu

$$W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \dots & & & \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{bmatrix}$$

Výstup takejto vrstvy potom bude $y = (y_1, y_2, y_3, ...y_m)^T$, ktorý dostaneme ako

$$y = Wx$$

Tento výstup potom slúži ako vstup do ďalšej vrstvy, ktorej celkový počet vstupných kanálov je rovný m.

Po prechode celou sieťou sa vypočíta strata s ohľadom na požadovaný výstup. Trénovanie takéhoto modelu je minimalizačný problém kedy sa upravujú hodnoty váhových matíc a prahov aktivácie tak aby bola strata čo najmenšia.

Neurónové siete sú výborné klasifikátory ale dajú sa použiť aj na generovanie nových dát. Existuje niekoľko druhov sietí, ktoré dokážu generovať obrázky. Pre nás zaujímavé budú VAE, CPPN a GAN neurónové siete.

2.3 Variačné autoenkódery

VAE alebo Variačné autoenkódery (angl. Variational autoencoders) vznikli upravením jednoduchých autoenkóderov [12]. Neurónová sieť postavená ako autoenkóder sa dokáže naučiť štruktúru vstupných dát. Pre jednoduché vysvetlenie majme na vstupe do siete rastrový obrázok. Obrazové dáta sú uložené vo forme pixlov, ktorých množstvo závisí od rozmerov obrázka. Autoenkóder dokáže uložiť štruktúru umiestnenia pixlov v obrázku, čiže jeho obsah do jednoduchých premenných nazvaných latentné premenné. Vektor tvorený z týchto premenných je komprimáciou obrázka a z tohto vektora môžeme dekódovaním dostať pôvodný obrázok. Ak by sme vedeli ako jednotlivé dimenzie latentného vektora ovplyvňujú obrázok, mohli by sme meniť jeho vlastnosti jednoduchou zmenou premennej. Tento fakt sa dá využiť pri generovaný nových dát.

V prvom rade pre všetky naše dáta X v datasete musí existovať nastavenie latentných premenných, ktoré umožňuje modelu generovať niečo veľmi podobné naším dátam. Majme vektor latentných premenných z v multidimenzionálnom priestore Z, ktorý môžeme ľahko vybrať podľa nejakej funkcie hustoty pravdepodobnosti $P(z)$ definovanej nad Z. Majme funkciu $f(z, \theta)$ parametrizovanú vektorom θ v priestore Θ, kde $f : Z \times \Theta \rightarrow X$. Ak je z náhodné a θ nemenné, potom $f(z, \theta)$ je náhodná premenná v priestore X. My chceme optimalizovať θ tak, že ak vyberieme vzorku z $P(z)$, tak $f(z, \theta)$ bude podobné naším dátam X.

VAE sieť musí zistiť akú informáciu nesie latentná premenná. VAE majú neobvyklí prístup k tejto úlohe. Pri variačných autoenkódroch predpokladáme, že neexistuje jednoduchá interpretácia dimenzií z. Namiesto toho sa vzorky z berú z Gaussovho normálneho rozdelenia.

V praxi to vyzerá, tak že pri kódovaní dát na vektor z sa vytvárajú dva vektory, vektor stredných hodnôt a vektor smerodajnej odchýlky. Tieto vektory potom spoločne tvoria výsledný vektor z. Podľa vektora stredných hodnôt sa vypočítava strata, ktorá hovorí či sú vygenerované dáta podobné chceným dátam. A podľa vektoru smerodajnej odchýlky sa vypočíta strata, ktorá meria ako blízko sú latentné premenné k normálnemu rozdeleniu. Celková strata je suma týchto čiastkových strát.

Nevýhodou takýchto sietí je že ak sa použijú na generovanie obrázkov, tak generované obrázky sú rozmazané práve kvôli strate pri kompresií, ktorú VAE

vytvárajú.

2.4 Generatívne konkurenčné siete

Generačné konkurenčné siete alebo GAN (angl. Generative adversarial networks) pôvodne navrhnuté Ianom Goodfellowom, fungujú na princípe konkurencie medzi dvoma sieťami [13]. Prvá sieť, generátor, sa snaží generovať čo najrealistickejšie dáta zo šumu. Druhá sieť, diskriminátor, rozpoznáva či sú vstupy reálne alebo vygenerované. Tieto siete sa tak snažia poraziť jedna druhú. GAN sú známe generovaním takmer realistických obrázkov. No nevýhodou je, že sa veľmi ťažko trénujú. Je viacero situácií, ktoré môžu nastať. Môže sa stať, že generátor nájde systém ako oklamať diskriminátor a generuje len jeden druh informácií. Alebo diskriminátor sa stane tak dobrým v rozpoznávaní skutočnosti, že vyhodnotí všetky generované dáta za neskutočné a tak sa generátor nebude môcť zlepšiť.

GAN zaznamenali niekoľko vylepšení a využití v rôznych oblastiach. Pri generovaní obrázkov sa osvedčilo využitie konvolučných vrstiev [8]. Takéto siete dokážu generovať oveľa kvalitnejšie obrázky aj keď stále nie vo veľkom rozlíšení. Ďalším vylepšením, práve v tomto probléme bolo využitie viacerých GAN sietí [14]. Zreťazenie viacerých neurónových sietí má za následok lepšiu kvalitu generovaných obrázkov. Niekoľko vrstiev dokáže zväčšiť obrázok s rozmermi 64x64 na rozmery 256x256 a výrazne upraviť kvalitu. A v neposlednom rade je aj výskum podmienených GAN. Tie vznikajú pridaním vlastností na vstup, pričom sa takto dá viac, či menej ovplyvniť výstup [15]. Práca z Univerzity v Michigane využíva práve podmienené GAN. V tejto práci využili neurónovú sieť na syntézu textu na obraz. Z datasetu vtáctva a kvetín vytvorili model, ktorý dokáže generovať obraz podľa popisu [7]. S dostatočne veľkými zdrojmi by mohol mať tento prístup veľké využitie.

3 Tvorba datasetu

Najdôležitejšou časťou pri vytváraní nášho systému je tvorba datasetu. Podľa toho aké dobré dáta zozbierame, také výsledky sa nám podarí získať. Náš dataset bude tvorený dvomi druhmi dát, a to obrazovými a zvukovými.

3.1 Zvukové dáta

V predošlej kapitole sme vysvetlili prečo a ako je potrebné upraviť hudobné dáta, z ktorých sa bude náš model učiť. V tejto podkapitole opíšeme ako bude vyzerať zvuková časť nášho datasetu.

Prvým krokom je výber žánru skladieb, ktoré budeme zbierať. V súčasnosti existuje nesmierne množstvo interpretov a skladieb z ktorých si môžeme vybrať. No nie každý hudobný žáner dokáže podnietiť našu predstavivosť. Pre náš projekt budú najvhodnejšie skladby klasickej hudby, nakoľko na rozdiel od populárnej modernej hudby, klasická hudba dáva väčší dôraz na emócie.

Vzorky datasetu by mali mať konštantnú veľkosť. Preto je nevyhnutné upraviť skladby tak aby tomu vyhovovali.

V závislosti od dĺžky a rôznorodosti jednotlivých skladieb vyberieme z každej niekoľko vzoriek s dĺžkou 25 sekúnd. Táto dĺžka je zvolená preto aby mal poslucháč dostatok času na vybavenie obrazu. Pomocou programu Audacity ďalej upravíme každú vzorku pomocou efektu Normalizovať. Takto dosiahneme podobnú hlasitosť, bez ohľadu na spôsob nahrávania daných vzoriek. Taktiež zmeníme vzorkovaciu frekvenciu na 22 050Hz, aby naša analýza bola súvislá.

Pre hlavnú analýzu vytvoríme skript v programovacom jazyku Python. Tento jazyk sme vybrali kvôli možstvu modulov, ktoré existujú a pomôžu nám pri tvorbe systému. Pre hudobnú analýzu použijeme modul Librosa, ktorý obsahuje množstvo algoritmov vhodných pre získanie vlastností akustických signálov, ktoré sme

uviedli v predošlej kapitole.

Vybrali sme päť akustických vlastností, ktoré budú reprezentovať naše skladby, a to mel-frekvenčné kepstrálne koeficienty, tempo, spektrálne centroidy, počet nulových priechodov a mieru týchto priechodov. Každá z týchto vlastností nesie užitočnú informáciu potrebnú pre našu úlohu. Napríklad mel-frekvenčné kepstrálne koeficienty poskytujú informáciu o tom aký tón a farbu má skladba v danom časovom úseku.

Touto analýzou sme úspešne skrátili vstupný vektor. Ak by sme chceli vložiť 25 sekundové vzorky skladieb do neurónovej siete celkový počet vstupných kanálov by bol $25 \times 22050 = 551250$. Náš dataset obsahuje pre každú vzorku 549 hodnôt z oboru reálnych čísel. Mel-frekvenčné kepstrálne koeficienty tvoria najväčšiu časť.

```
S = librosa.feature.melspectrogram(
        y=y, sr=sr, n_mels=128, hop_length=22050, n_fft=22050)
log_S = librosa.logamplitude(S, ref_power=np.max)
mfcc = librosa.feature.mfcc(S=log_S, sr=sr, n_mfcc=20)
mfcc_1d_vector = mfcc.flatten()
```

Tie sú vypočítané na základe mel-spektrogramu so vzorkovaním 22050, čo vytvára 26 hodnôt. Počet mfcc je 20, celkovo tak dostávame $20 \times 26 = 520$ čísel. Tempo, aritmetický priemer spektrálnych centroidov a celkový počet nulových priechodov sú ďalšie 3 hodnoty datasetu. Posledných 26 hodnôt tvoria miery nulových prechodov, merané so skokom 22050.

```
zcr = librosa.feature.zero_crossing_rate(
        y, frame_length=22050, hop_length=22050).flatten()
```

3.2 Obrazové dáta

Tak ako hudobné dáta tak aj obrázky, ktoré bude náš dataset obsahovať musia mať nejakú koherentnú formu.

Prvým modelom, ktorý budeme vytvárať bude neurónová sieť, ktorá dokáže priradiť k skladbe farbu. Pre tento klasifikátor ako výstup použijeme jedno číslo, ktoré bude reprezentovať farbu v našej farebnej palete. Takto sme sa rozhodli z dôvodu, že pri vytváraní datasetu bude jednoduchšie ak k jednotlivým skladbám priradíme farbu z vopred danej palety.

Pre druhý a náš hlavný model, budeme ukladať obrázky vo formáte png a každý obrázok bude uložený vo veľkosti 32x32 pixlov.

3.3 Zber dát

V tejto časti opíšeme spôsob ako sme zbierali dáta pre náš dataset. Zber údajov je veľmi podstatná časť projektu. Preto je nevyhnutné nebrať tento proces na ľahkú váhu. Vytváranie vlastností (features) a štítkov (labels) môže byť úplne manuálne alebo z časti automatizované. Zber dát manuálne môže byť veľmi pracný a zdĺhavý, preto je vhodné mať nejaký nástroj pre túto činnosť. Tento nástroj by nás oslobodil od množstva repetitívnej práce. Ďalšou výhodou pre vytváranie datasetu na jednom mieste je zapojenie viacerých ľudí do procesu. Viac ľudí prináša variabilitu do našich dát a umožňuje vytvorenie robustnejšieho riešenia.

Z týchto dôvodov sme vytvorili webovú aplikáciu, ktorá slúži na zber údajov. V našom datasete sú okrem akustických vlastností skladieb, ktoré dokážeme získať relatívne jednoducho, aj údaje, ktoré nedokážeme získať pomocou algoritmov. Presne pre získanie týchto dát sme vytvorili formulár určený na ich ukladanie. Konkrétne ide o farbu skladby, emóciu a obrázok, toho čo skladba predstavuje. Farbu je možné vybrať z palety pätnástich základných farieb. Pre emóciu možno vybrať jednu z piatich skupín emócií. Jednotlivé skupiny možno opísať ako vášnivé emócie, veselé, dojímavé, vtipné a agresívne. Najdôležitejšou informáciou, ktorú potrebujeme získať je ale obrázok. Do nášho datasetu potrebujeme dva druhy obrázkov, a to jednoduchú skicu a detailný obrázok. Pre jednouché stvárnenie skladby v podobe skice sme do formuláru pridali editor na nakreslenie obrázku. Kvôli jednoduchosti je možné požiť len čiernu farbu. V datasete potrebujeme obrázky o veľkosti 32x32, lenže kreslenie takýchto obrázkov by bolo nepraktické. Preto sme plátnu na kreslenie dali veľkosť 320x320 a zväčšili sme hrúbku štetca. Takto budeme môcť kresliť pohodlne a zároveň potom zmenšiť obrázky bez prílišnej straty. Druhý typ obrázku získavame pomocou url adresy. Detailnejšie obrázky vyhľadáme na internete a pomocou nášho nástroja uložíme adresu. Databáza takto nebude musieť ukladať surové dáta a my budeme môcť obrázok jednoducho vyhľadať a stiahnuť. Keďže je naša aplikácia verejne dostupná, pribudla nám nová úloha, a to kontrola. Skôr ako budeme môcť využiť nazbierané dáta, musíme ich skontrolovať či neobsahujú nevhodný obsah a či sú vierohodné.

Pomocou tejto webovej aplikácie sme zozbierali potrebné údaje, ktoré je nutné už len stiahnuť a jednoducho upraviť pre vstup do modelov neurónových sietí.

4 Detekcia farby

V predchádzajúcej kapitole sme opísali ako vyzerá náš dataset ako sme ho vytvorili. Táto kapitola je venovaná nášmu prvému modelu neurónovej siete. Ešte pred vytvorením zložitých a výpočtovo náročných algoritmov, určených na vizualizáciu hudby, bolo nevyhnutné dokázať, že naše dáta nesú informáciu, ktorá dovoľuje takýto počin.

Najjednoduchšia forma vizualizácie hudobnej skladby je reprezentácia farbou. Pri počúvaní hudby si ľudský mozog dokáže predstaviť obraz. Tento obraz nemusí obsahovať konkrétne objekty, ale môže byť čisto abstraktný s veľkou dominanciou farby. Náš algoritmus by mal dokázať podobnú vec.

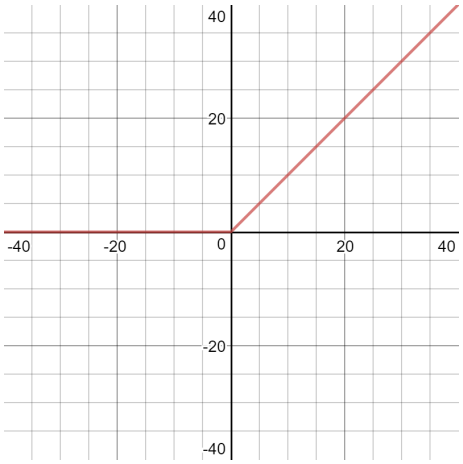
4.1 Návrh

Na klasifikáciu sme využili doprednú hlbokú neurónovú sieť s troma skrytými vrstvami a jednou výstupnou vrstvou. Pričom sa každá skrytá vrstva skladá zo sto neurónov. Na vstupe očakávame dáta na 549 kanáloch. Výstupom je vektor y s pätnástimi dimenziami. Veľkosť tohto vektora súvisí s využitím techniky One-Hot-Encoding. Zo vstupných dát model určuje jednu z pätnástich farieb, preto je pre každú farbu priradený jeden rozmer. Aktivačnou funkciou pre jednotlivé vrstvy je ReLu (obrázok 4.1). ReLu je definovaná ako

$$g(x) = max(0, x)$$

Pričom obor hodnôt je $H =< 0, \infty)$.

Trénovanie modelu je realizované v troch krokoch. Výpočty sú vykonávané na dávkach dát. V prvom kroku je vytvorená predikcia výsledku. Následne sú tieto výsledky porovnané s očakávanými výstupmi a vypočíta sa chyba. Táto chyba je vstupom do optimalizátoru, ktorý upraví premenné siete. Celý proces sa opakuje vo viacerých epochách pre dosiahnutie väčšej presnosti.



Obr. 4.1: Graf aktivačnej funkcie ReLu.

Na výpočet rozdielu medzi výstupom neurónovej siete a predpokladaným výstupom využijeme krížovú entropiu. Krížová entropia pre dve diskrétne rozdelenia pravdepodobností p a q je definovaná ako

$$H(p,q) = -\sum p(x) \log q(x)$$

Optimalizovanie premenných modelu, čiže váhových matíc a prahov excitácie zabezpečí algoritmus spätného šírenia chyby. Konkrétne ide o modifikáciu algoritmu Gradient Descent.

4.2 Implementácia

Model klasifikátora a jeho trénovanie je implementované v jazyku python, využitím modulu TensorFlow. TensorFlow obsahuje mnoho algoritmov strojového učenia, preto nás oslobodzuje od ich implementovania. Ďalšou výhodou tohto modulu je to, že separuje výpočty od pythonu, preto sú tieto výpočty rýchlejšie a je možné ich paralelizovať. Program pozostáva z dvoch krokov, vytvorenie výpočtového grafu a jeho spustenie. Výpočtový graf obsahuje celý model neurónovej siete spolu so všetkými výpočtovými krokmi pre trénovanie.

Vďaka tomu je možné implementovať funkciu pre trénovanie len niekoľkými riadkami.

```
with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
```

```
        for epoch in range(hm_epochs):
                for _ in range(num_examples/batch_size):
                        epoch_x, epoch_y = next_batch(batch_size)
                        sess.run([optimizer, cost],
                                feed_dict={x: epoch_x, y: epoch_y})
```

Na začiatku je nutné nainicializovať premenné, až potom je možné s nimi pracovať. V každej epoche sú prejdené všetky dáta a sú optimalizované premenné siete. Výpočty sa spúšťajú príkazom run. Jeho parametrami sú výpočty, vrcholy TensorFlow grafu, a vstupné premenné potrebné pre tieto výpočty. Optimizer je verzia algoritmu Gradient Descent.

```
optimizer = tf.train.AdamOptimizer().minimize(cost)
```

Na výpočet chyby (cost) je využitá TensorFlow implementácia krížovej entropie.

```
prediction = neural_network_model(x)
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
                        logits=prediction,
                        labels=y))
```

5 GAN Vizualizátor

Priradenie farby ku skladbe nie je dostačujúce. Preto sme vytvorili model, ktorý generuje obrázky. Na vyriešenie tohto problému sme zvolili generatívnu konkurenčnú sieť, spomínanú v predošlých kapitolách. Ďalej v tejto kapitole opíšeme konkrétny návrh a implementáciu vizualizátora.

5.1 Návrh

Ako sme opísali v kapitole dva, generatívna konkurenčná neurónová sieť je tvorená dvoma sieťami. V našom návrhu budú generátor tak ako aj diskriminátor tvoriť dopredné neurónové siete. Generátor na vstupe potrebuje náhodné dáta, šum, z ktorých bude generovať nové obrázky. Tento šum je v našom prípade vektor náhodných čísel z rozsahu -1 až 1. Celkový počet rozmerov vektora je závislý od počtu vstupných kanálov. Prvotný návrh generátora je tvorený dvoma vrstvami. Prvá vrstva ma 128 neurónov, pričom každý neurón má 100 vstupných kanálov. Výstupy neurónov druhej vrstvy tvoria celkový výstup siete. Ich počet je závislý od požadovanej veľkosti generovaných obrázkov. Nakoľko chceme generovať obrázky veľkosti 64x64, počet neurónov druhej vrstvy je 4096, pričom každý má 128 vstupných kanálov.

Diskriminátor je klasifikátor tvorený tak isto dvomi vrstvami. Prvá, vstupná vrstva je tvorená 128 neurónmi. Nakoľko na vstupe je obrázok, u ktorého sa posudzuje reálnosť, počet vstupných kanálov je pre 64x64 obrázok 4096. Druhá a zároveň výstupná vrstva je tvorená len jedným neurónom s počtom vstupných kanálov totožným s počtom neurónov predošlej vrstvy. Výstupom tejto vrstvy bude hodnota v intervale 0 až 1, ktorá predstavuje pravdepodobnosť či ide o skutočné dáta.

Trénovanie tohto modelu je náročné nakoľko diskriminačná sieť sa snaží maxi-

malizovať výstupnú pravdepodobnosť, na druhej strane generátor sa túto hodnotu snaží minimalizovať. Matematicky by sa tento problém dal vyjadriť ako

$$\min_G \max_D V(D,G)$$

$$V(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1-D(G(z)))]$$

, kde prvý člen funkcie je entropia, že dáta zo skutočnej distribúcie prejdú diskriminátorom. Diskriminátor sa snaží túto hodnotu maximalizovať na 1. Druhý člen je entropia, že náhodný vstup vojde do generátora, ktorý vygeneruje vstup do diskriminátora. Generátor sa snaží minimalizovať výstup.

Našťastie objavitelia GAN sietí popisujú vo svojej práci algoritmus na trénovanie tohto modelu [13]. Trénovanie prebieha v troch krokoch:

1. Trénovanie diskriminátora na reálnych dátach v niekoľkých epochách.
2. Vygenerovanie nových dát a prechod diskriminátorom.
3. Trénovanie generátora za pomoci výstupu z minulého kroku.

Celý tento proces môže prebiehať vo viacerých cykloch. V pôvodnom algoritme je pre učenie diskriminačnej siete využitý algoritmus gradient ascend. Chyba pre túto sieť je definovaná ako

$$\frac{1}{m}\sum_{i=1}^{m}[logD(x^{(i)}) + log(1-D(G(z^{(i)})))]$$

, kde m je počet vstupných dát alebo inak povedané veľkosť dávky. Pre generátor, s využitím algoritmu gradient descend, definujeme chybu ako

$$\frac{1}{m}\sum_{i=1}^{m} log(1-D(G(z^{(i)})))$$

5.2 Implementácia

V prvom rade je nevyhnutné upraviť vstupné dáta. Pre stručnosť opíšeme úpravu len obrázkov ručne kreslených skíc z nášho datasetu. Pre detailné obrázky je potrebná podobná úprava len s malými zmenami. Obrázky na disku sú uložené vo formáte png s hĺbkou 4. Aj keď sú uložené vo formáte RGB, obsahujú len čiernobiely obraz. Pozadie týchto obrázkov je nastavené na priehľadné a majú veľkosť

320x320. Pre našu sieť sú tieto obrázky príliš veľké a majú zlú farebnú hĺbku. Proces úpravy preto rieši tieto nedostatky.

```
img = Image.open(image_name)
white_back = Image.new('RGB', img.size, (255,255,255))
white_back.paste(img, mask = img.split()[3])

gray = white_back.convert('1')
small = gray.resize((HEIGHT, WIDTH))
arr = np.array(small)
flat_arr = arr.ravel()
return (1*flat_arr)
```

Táto funkcia nazačiatku vytvorí nové biele pozadie a spojí ho s existujúcim obrázkom. Následne je tento obrázok konvertovaný na dvojrozmerné pole jednotiek a núl signalizujúcich biele a čierne pixle. Ostáva už len zmeniť jeho veľkosť a upraviť dvojrozmerné pole na jednorozmerné.

Obrázky sú po úprave pripravené na vstup do siete nadefinovanej v predchádzajúcej podkapitole. Podľa návrhu sme naimplementovali model generátora aj diskriminátora v pythone za použitia modulov tensorflow a numpy. V sieti generátora sme pre prvú vrstvu zvolili aktivačnú funkciu relu, nakoľko ide o odporúčanie viacerých výskumníkov. Aktivačnú funkciu pre výstupnú vrstvu tvorí sigmoid funkcia, ktorá nadobúda hodnoty medzi 0 a 1. Diskriminátor je až na zmenu v počte neurónov implementovaný rovnako.

Výpočet chyby sme boli nútený trošku pozmeniť, nakoľko tensorflow nepozná algoritmus gradient ascend.

```
D_loss = -tf.reduce_mean(tf.log(D_real) + tf.log(1. - D_fake))
G_loss = -tf.reduce_mean(tf.log(D_fake))

D_solver = tf.train.AdamOptimizer().minimize(D_loss, var_list=theta_D)
G_solver = tf.train.AdamOptimizer().minimize(G_loss, var_list=theta_G)
```

D_loss v uvedenom kóde predstavuje chybu diskriminátora, z uvedeného dôvodu je zmenená na zápornú a namiesto maximalizovania používame optimalizátor na minimalizáciu. Takisto chyba generátora nie je totožná s pôvodným vzorcom. Odôvodnenie je, že v tomto tvare algoritmus dosahuje lepšie výsledky.

5.3 Podmienená generatívna sieť

Zatiaľ náš model dokáže generovať obrázky z náhodnej distribúcie. Zadaním tejto práce je ale generovať obrázky za pomoci hudby. Generatívnu konkurenčnú sieť, ktorú sme vytvorili zmeníme na podmienenú GAN alebo CGAN (ang. conditional generative adversarial network). CGAN na rozdiel od GAN na vstupe do generatívnej aj diskriminatívnej siete potrebuje aj nový štítok (ang. label), ktorý nesie nejakú informáciu pre požadovaný výstup. Naše siete sa zmenia z $G(z)$ a $D(X)$ na $G(z, y)$ a $D(X, y)$.

Z matematického pohľadu $G(z, y)$ modeluje distribúciu našich dát pri zadaní z a y. To znamená, že vzorec na generovanie dát je

$$X \sim G(X|z, y)$$

Podobne pre diskriminátor, ktorý sa snaží nájsť pravdepodobnosť reálnosti pre X a X_G, výsledky dostávame ako

$$d \sim D(d|X, y)$$

Čiže funkcia pre požadovaný výsledok nášho modelu so započítaním nových skutočností je

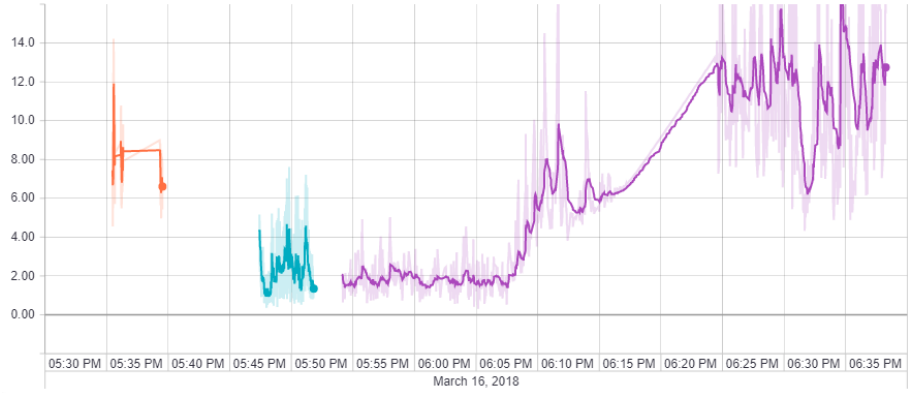
$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[logD(x, y)] + E_{z \sim p_z(z)}[log(1 - D(G(z, y), y))]$$

V našom prípade tvoria vektor y, z danej rovnice, naše hudobné dáta. Vstup hudobnej informácie do neurónovej siete sme opísali v kapitole päť.

5.4 Zmena parametrov

Pravdepodobnosť, že náš navrhnutý model dokáže generovať obrázky v požadovanej kvalite je veľmi malá. Úprava všetkých parametrov siete, ako sú počet neurónov vo vrstvách, rýchlosť učenia alebo typ optimalizátora sa nedá automatizovať.

Bez optimalizovania boli výsledky nášho modelu hudobného vizualizátora, tak ako sme predpokladali, nedostačujúce. Na obrázku 5.1 sú zaznamenané tri trénovania, v rôznych časoch a trvaniach, na skicách z nášho datasetu. Je vidno že strata generátora po prvých dvoch trénovaniach klesla, no potom sa klesanie



Obr. 5.1: Chyba generátora pri trénovaní neoptimalizovanej siete.

zastavilo dokonca trénovanie úplne zlyhalo a chyba začala rapídne divergovať. Náraz chyby spôsobil zmenu váh siete na neprípustné hodnoty, čo pri ďalšej epoche trénovania viedlo k nekonečným alebo neznámim hodnotám chýb obydvoch sietí. Tieto hodnoty tensorflow nedokáže spracovať a program trénovania spadol. Z toho dôvodu všetky naše ďalšie kroky smerovali k stablizácií trénovania.

V prvom rade sme normalizovali vstupné dáta. Jednou časťou normalizácie je zmenšenie každej vzorky o priemernú hodnotu vypočítanú zo vzoriek danej vlastnosti. Druhou časťou je min-max normalizácia, pri ktorej každú vlastnosť danej vzorky upravíme na hodnotu z intervalu $< -0,5; 0,5 >$. Trénovanie týchto dát bolo možné udržať dlhšie ale problém nebol úplne eliminovaný.

S predpokladom, že problém tkvie v typológií siete, sme zmenili počet vrstiev. Nový generátor mal vstupnú vrstvu so 128 neurónmi, dve skryté vrstvy so 128 a 512 neurónmi a výstupnú vrstvu. Na výstupe sa aktivovala sigmoind funkcia všade inde relu. Diskriminátor ostal nezmenený, nakoľko sme predpokladali, že lepší generátor bude mať výhodu nad diskriminátorom, čo by viedlo k poklesu chyby. Táto zmena nepriniesla žiaden pozitívny výsledok. Je známim faktom, že aktivačná funkcia leaky relu má lepšie výsledky ako relu. Implementovali sme túto funkciu a využili v generátore.

```
def lrelu(x, alpha):
    return tf.nn.relu(x) - alpha * tf.nn.relu(-x)
```

Nevedli sme si poradiť s výpadkami programu, tak sme pristúpili k riadenému trénovaniu sietí. Snahou bolo zabrániť chybe generátor presiahnúť vysokú hodnotu. Z toho dôvodu sme dovolili trénovať diskriminátor len v prípade, že chyba

generátora je menšia ako 4. Daná úprava dovoľovala generátoru dobehnúť diskriminátor a program už nespadol. Výsledky trénovania však neboli pozitívne.

Keď už sme mohli trénovať aj v dlhších časových úsekoch, začali sme viac experimentovať s parametrami siete. Zväčšili sme diskriminátor, zmenili aktivačné funkcie či upravili mieru trénovania v algoritme gradient descent. Nič z toho neprinieslo požadovaný úspech.

Pri hľadaní riešenia na otázku stabilizovania GAN siete, sme narazili na iný prístup k výpočte chyby. Nový výpočet spočíval vo výpočte tzv. wasserstein distance. Upravili sme diskriminátor aby na výstupe nebola aktivačná funkcia. Zmenili sme výpočty chýb na:

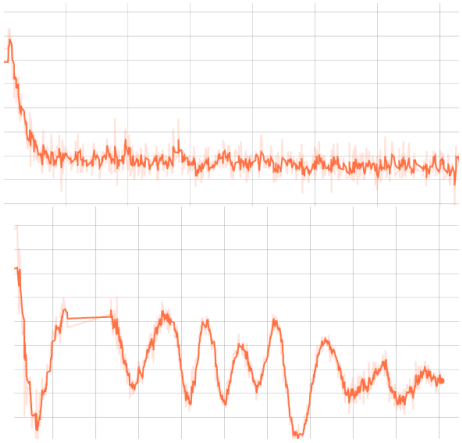
```
D_loss = tf.reduce_mean(D_real) - tf.reduce_mean(D_fake)
G_loss = -tf.reduce_mean(D_fake)
```

Použili sme menšiu mieru trénovania a nechali trénovať diskriminátor viac.

```
for _ in range(5):
        D_summary, _, D_loss_curr, _ = sess.run(
                [D_merge, D_solver, D_loss, clip_D],
                feed_dict={X: X_mb, Z: sample_Z(mb_size, z_dim), y:y_mb}
        )


G_summary, _, G_loss_curr = sess.run(
        [G_merge, G_solver, G_loss],
        feed_dict={Z: sample_Z(mb_size, z_dim), y:y_mb}
)
```

Táto zmena stabilizovala trénovanie. Program nespadol a nebolo nutné zasahovať manuálne do algoritmu. Pre porovanie sme vyskúšali dva optimalizátory, konkrétne AdamOptimizer a RMSPropOptimizer.

Chyba diskriminátora konvergovala rýchlejšie ako je vidieť na obrázku 5.2, tak sme sa rozhodli využívať pri ďalšom trénovaní už len tento optimalizátor. Opäť sme vyskúšali niekoľko typov siete. Najlepšie výsledky sme dosiahli pri Generátore, ktorý využíval Leaky ReLu aktivačné funkcie na skrytých vrstvách a Sigmoid funkciu na výstupe. Diskriminátor v tomto modely mal presne opačnú topológiu v porovnaní s generátorom a ako ativačné funkcie využíval ReLu. Táto GAN sieť



Obr. 5.2: Chyba diskriminátora pri využití RMSPropOptimizer (hore) a AdamOptimizer (dole).

dokázala generovať obrázky podobné nášmu datasetu, pri zadaní piesne z datasetu. Ak však na vstupe bola skladba, ktorú sieť pri trénovaní nemala, výstup bol bez kontextu.

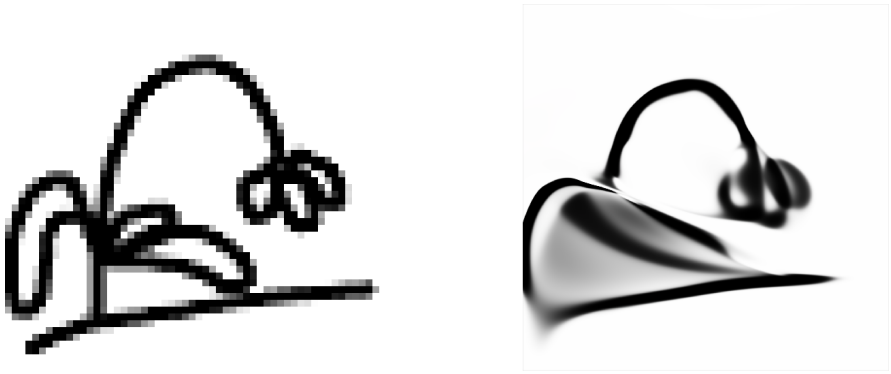
Tento problém sme sa snažili vyriešiť využitím iného datasetu, ktorý obsahoval menšie množstvo akustických vlastností pri jednotlivých vzorkách. Ani po veľmi dlhom trénovaní výsledky neboli pre nás dostatočné. Posledným krokom, ktorým sme sa snažili vylepšiť výsledok, bolo zakomponovanie konvolučných vrstiev do siete. Vytvorli sme generátor, ktorý pozostával z dvoch konvolučných vrstiev. Výsledky toho modelu naznačovali, že pre generovanie správnych obrázkov potrebujeme viac konvolúcií. Rozmery takej siete by, ale náš stroj nezniesol a tak sme od tohto riešenia ustúpili.

5.5 Zväčšovanie obrázkov

Hudobný vizualizátor popísaný v tejto kapitole je teoreticky schopný produkovať obrázky ľubovoľnej veľkosti. Veľkosť ale musí byť zadaná predom, čo nám znemožňuje akúkoľvek neskoršiu manipuláciu s rozmermi bez strát a rozmazania. Samozrejme rozmery závisia aj od obrázkov v datasete. Čo sa týka samotného algoritmu, nič nás neobmedzuje vo vytvorení vhodného setu dát a generovania obrázkov vo vysokom rozlíšení. Prakticky sme ale obmedzený operačnou pamäťou a výpočtovým výkonom. Z toho dôvodu je možné generovať len malé rozlíšenia.

Pre zväčšovanie obrázkov by sme mohli požiť nejaký algoritmus na zaostrova-

nie, no tie neprodukujú dostatočne ostré obrazy. Namiesto toho využijeme Kompozitnú vzory vytvárajúcu sieť (ang. Compositional pattern-producing network). Využijeme skript vytvorený Liviu Pirvanonm. Ním navrhnutá sieť sa dokáže naučiť štruktúru jedného obrázka a podľa nej dokáže generovať nové obrázky ľubovoľných rozmerov. Táto CPPN sieť na vstupe berie súradnice jednotlivých pixlov. Na výstupe sa snaží určiť správnu farbu pre zadané súradnice. Celá sieť tak funguje ako funkcia, ktorá aproximuje obrázok. Keďže vykresľuje po jednotlivých pixloch, tak veľkosť vykreslených obrázkov môže byť ľubovoľná a je obmedzená len operačnou pamäťou. Takto môžeme naše výstupy z CGAN vizualizátora o veľkosti 64x64 prekonvertovať aj na rozlíšenia 4K a viac. Samozrejme, že táto konverzia nie je dokonalá a stojí relatívne veľa času kvôli nutnosti trénovania siete pre každý obrázok zvlášť. Nové obrázky sú síce rozmazané ale majú o niečo lepšiu kvalitu ako, keby boli zväčšené klasickým spôsobom.



Obr. 5.3: Zväčšenie obrázku z datasetu využitím CPPN siete.

Funkčnosť tejto neurónovej siete sme overili na jednom obrázku z nášho datasetu. Výsledok je možné vidieť na obrázku 5.3. Vľavo sa nachádza pôvodný obrázok vo veľkosti 64x64 zväčšený klasickým spôsobom na veľkosť 640x640. Vpravo je vygenerovaný obrázok pomocou CPPN siete vo veľkosti 2048x2048.

6 VAE Vizualizátor

V dnešnej dobe existujú dva spôsoby ako generovať obrázky pomocou neurónových sietí. Z predchádzajúcich kapitol vieme, že ide o generatívne konkurenčné siete a variačné autoenkódery. GAN implementáciu vizualizátora sme už ukázali, v tejto časti vysvetlíme ako sme postupovali v riešení nášho zadania využitím druhej zo spomínaných metód.

6.1 Návrh

Už vieme, že VAE generuje dáta z latentnej premennej. Cieľom je preto namodelovať dáta X z rozdelenia pravdepodobnosti $P(X)$. Vzťah medzi latentnou premennou z a pravdepodobnosťou $P(X)$ môžeme vyjadriť ako:

$$P(X) = \int P(X|z)P(z)dz$$

Celou myšlienkou VAE je získať $P(z)$ za pomoci $P(z|X)$. No nakoľko nepoznáme distribúciu $P(z|X)$ využijeme jednoduchšie Gaussovo rozdelenie a minimalizujeme rozdiel medzi nimi využitím Kullback-Leiber divergencie.

Celkovú úlohu variačného autoenkódera môžeme zapísať takto:

$$\log P(X) - D_{KL}[Q(z|X)||P(z|X)] = E[\log P(X|z)] - D_{KL}[Q(z|X)||P(z)]$$

To znamená, že chceme optimalizovať logaritmickú pravdepodobnosť dát $P(X)$ so započítaním nejakej chyby. Všimnime si, že čo v skutočnosti dostávame je $P(X|z)$, čo generuje dáta so zohľadnením latentnej premennej, a $Q(z|X)$, čo kóduje naše dáta do priestoru latentných premenných.

Toto všetko nám pomôže pri implementácii, no tento variačný autoenkóder nijako neumožňuje ovplyvniť výstup. Kódovacia časť $Q(z|X)$ ako aj dekódovacia časť $P(X|z)$ modelujú výstup priamo zo vstupu a nezohľadňujú rôzne typy

vstupov. Tento problém vyriešime pridaním nového vstupu do oboch častí autoenkódera a dostávame tak $P(X|z,c)$ a $Q(z|X,c)$ čo pozmení úlohu na:

$$\log P(X|c) - D_{KL}[Q(z|X,c)||P(z|X,c)] = E[\log P(X|z,c)] - D_{KL}[Q(z|X,c)||P(z|c)]$$

Návrh modelu tejto neurónovej siete má dva časti. $P(X|z,c)$ je podsieť tvorená dvoma vrstvami. Vstupná vrstva ma 128 neurónov, pričom každý má 649 vstupných kanálov. 549 vstupov tvoria hudobné akustické vlastnosti. Ostávajúcich 100 vstupov zaberá vektor latentných premenných. Výstupná vrstva generuje obrázky, preto počet jej neurónov zodpovedá rozmerom $64 \times 64 = 4096$. Podsieť $Q(z|X,c)$ má na vstupnej vrstve, so 128 neurónmi, 4196 vstupov. Ide od spojenie latentného vektora a obrázkov z datasetu. Produkuje dva výstupy. Každý o veľkosti 100. Tieto sa neskôr spoja do jediného vektora z, čo nám pomôže pri počítaní straty.

6.2 Implementácia

Definujme obe časti, opísané v návrhu.

```
def P(z, c):
    inputs = tf.concat(axis=1, values=[z, c])
    h = tf.nn.relu(tf.matmul(inputs, P_W1) + P_b1)
    logits = tf.matmul(h, P_W2) + P_b2
    prob = tf.nn.sigmoid(logits)
    return prob, logits

def Q(X, c):
    inputs = tf.concat(axis=1, values=[X, c])
    h = tf.nn.relu(tf.matmul(inputs, Q_W1) + Q_b1)
    z_mu = tf.matmul(h, Q_W2_mu) + Q_b2_mu
    z_logvar = tf.matmul(h, Q_W2_sigma) + Q_b2_sigma
    return z_mu, z_logvar
```

Aktivačnou funkciou vstupných vrstiev oboch častí je ReLu. Rozdiel v kódery a dekódery je vidno na výstupe. $P(z,c)$ má na výstupe Sigmoid funkciu, nakoľko

jednotlivé pixle majú hodnoty v rozmedzí 0 až 1. $Q(X, c)$ výstupnú aktivačnú funkciu nepotrebuje. Ako sme vysvetlili v teoretickej časti práce, pri kódovaní vytvárame dva vektory. Vektor z_mu je vektor stredných hodnôt a vektor z_logvar určuje disperziu.

Rozoberme si celkovú úlohu VAE siete. Úlohou je maximalizovať pravdepodobnosť mapovania latentnej premennej na dáta a minimalizovať rozdiel medzi jednoduchou distribúciou $Q(z|X, c)$ a skutočnou distribúciou $P(z|c)$. Maximalizácia $E[\log P(X|z, c)]$ je v podstate odhad maximálnej pravdepodobnosti. Práve preto je $P(z, c)$ implementované ako klasifikátor a pre výpočet chyby môžeme využiť napríklad krížovú entropiu. Pri druhej časti úlohy musíme myslieť na to, že chceme neskôr nejako zvoliť $P(z|c)$. Najjednoduchším spôsobom je brať vzorky z normálneho rozdelenia $N(0, 1)$. Takže chceme aby rozdelenie $Q(z|X, c)$ bolo čo najpodobnejšie $N(0, 1)$. Zvoľme nech $Q(z|X, c)$ je takisto Gaussovo normálne rozdelenie ale s priemerom rovným $\mu(X)$ a disperziou $\sum(X)$, čiže $N(\mu(X), \sum(X))$. Potom platí

$$D_{KL}[N(\mu(X), \sum(X))||N(0, 1)] = \frac{1}{2}\sum_k(\sum(X) + \mu^2(X) - 1 - \log\sum(X))$$

Prakticky je ale stabilnejšie robiť $\sum(X)$ namiesto $\log\sum(X)$. Náš finálny vzorec je

$$D_{KL}[N(\mu(X), \sum(X))||N(0, 1)] = \frac{1}{2}\sum_k(exp(\sum(X)) + \mu^2(X) - 1 - \sum(X))$$

V kóde krížovú entropiu vypočítame za pomoci skutočných dát z datasetu. Pre druhú časť chyby, do vzorca vložíme vektory z_mu a z_logvar. Celkovú chybu tvoria obe časti. Pre optimalizáciu váh neurónovej siete využijeme Adam optimalizátor, ktorý sa snaží minimalizovať celkovú chybu.

```
z_mu, z_logvar = Q(X, c)
z_sample = sample_z(z_mu, z_logvar)
_, logits = P(z_sample, c)

# Sampling from random z
X_samples, _ = P(z, c)
# E[log P(X|z,c)]
recon_loss = tf.reduce_sum(
        tf.nn.sigmoid_cross_entropy_with_logits(
```

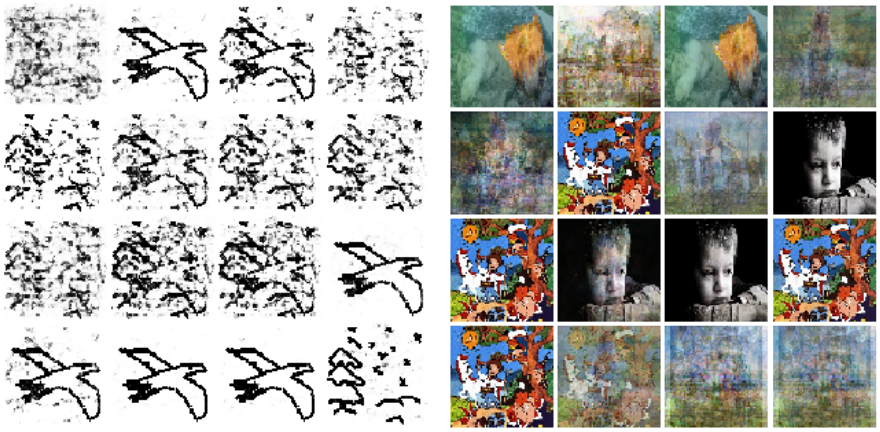
```
                logits=logits, labels=X), 1)
# D_KL(Q(z|X,c) || P(z|c))
kl_loss = 0.5 * tf.reduce_sum(
        tf.exp(z_logvar) + z_mu**2 - 1. - z_logvar, 1)

vae_loss = tf.reduce_mean(recon_loss + kl_loss)
solver = tf.train.AdamOptimizer().minimize(vae_loss)
```

6.3 Trénovanie

Trénovanie neurónových sietí môže byť zdĺhavý proces, pri ktorom jedinou technikou dosiahnutia výsledku je pokus-omyl. Pre dosiahnutie vyhovujúcich zobrazení sme vyskúšali niekoľko rôznych konfigurácií systému. Už pri trénovaní pôvodného návrhu sme spozorovali, rýchle rozloženie datasetu do priestoru normálneho rozdelenia. Ako sa ale dalo čakať, tento model nebol dostatočne komplexný aby vykreslil správny typ obrázkov. Pôvodne sme predpokladali, že problém môže byť v počte rozmerov vektora z. Zmenili sme teda $|z|$ zo 100 na 1000 a nechali spustené trénovanie niekoľko hodín. Výsledky boli takmer totožné. V obidvoch prípadoch sieť dokázala vykresliť vzorky datasetu so 100% presnosťou no pri vložení nových skladieb na vstup, vytvorené obrázky obsahovali len šum bez obsahu.

Skôr ako sme pristúpili k zmenám v typológií siete, otestovali sme inú štruktúru datasetu. Namiesto pôvodných 549 hudobných vlastností, pri ďalšom trénovaní do siete vstupovali len miery nulových prechodov a tempo, čiže 28 čísel z oboru reálnych hodnôt. Opäť sme vyskúšali varianty s malým aj veľkým latentným vektorom. Obrázky vygenerované, v tomto pokuse takisto neboli v dostačujúcej kvalite, no pri niektorých skladbách sme spozorovali náznaky obsahu. Takisto bolo zjavné, že pri $|z| = 1000$ sa lepšie výsledky objavovali častejšie.

Ďalším krokom bolo pozmeniť sieť ako takú. Pridali sme do kódovacej aj dekódovacej časti siete zhodne po dvoch skrytých vrstvách. Každá obsahovala 128 neurónov s aktivačnou funkciou ReLu. Trénovanie bolo konečne úspešné. Pri každej testovacej skladbe aspoň jeden zo šestnástich vygenerovaných obrázkov mal viditeľný obsah. Takisto sa potvrdilo, že je lepšie nastaviť počet rozmerov latentného vektora na 1000, nakoľko obrázky v tomto prípade boli kvalitnejšie.



Obr. 6.1: Výstup variačného autoenkódera pri Mozartovej skladbe Turkish March

Tento úspech nás priviedol k myšlienke, že počiatočné neúspechy nesúviseli so štruktúrou vstupných dát. Otestovali sme teda novú sieť aj pre pôvodný dataset. Trénovanie potvrdilo túto skutočnosť, čo naznačovalo, že problém v kvalite výstupov zapríčiňuje $P(X|z,c)$ časť problému. Rozhodli sme sa ďalej zväčšiť $P(z,c)$ sieť o dve nové skryté vrstvy, totožné s už existujúcimi skrytými vrstvami. Nová konfigurácia modelu generovala kvalitné výstupy.

Už len pre kontrolu sme zmenšili P aj Q o jednu vrstvu. Nespozorovali sme žiadny nárast vo variabilite výstupu, naopak kvalita obrázkov sa zhoršila. Teda môžeme povedať, že predchádzajúci model je dostatočným riešením.

Vytrénovali sme ešte raz tento model ale na dátach so zmenšeným oborom akustických hodnôt. V porovnaní s pôvodným datasetom takto vytrénovaný variačný autoenkóder produkoval rôznorodejšie obrázky. Tento výsledok sa dal čakať nakoľko rozdiel v počte nulových prechodov a tempe v dvoch rozdielnych skladbách nemusí byť dostatočným meradlom.

Takisto sa tento model ukázal byť vhodným aj pri generovaný farebných detailných obrázkov. Jedinou zmenou v sieti, je zväčšenie vstupu Q siete a výstupu P siete. Ide o to, že skice v datasete majú každý pixel uložený ako jednotku alebo nulu. Pri farebných obrázkoch je každý pixel reprezentovaný tromi číslami. Čiže tieto vstupné respektíve výstupne vrstvy namiesto 4096 neurónov obsahujú $4096 \times 3 = 12288$ neurónov. Nakoľko sieť tak obsahuje väčšie vrstvy, čas jednej epochy pri trénovaní sa približne strojnásobil. Príkladom výstupu je obrázok 6.1. Vľavo je výstup vygenerovaný modelom, ktorý sa učil z datasetu skíc. Vpravo bol pou-

žitý dataset farebných obrázkov. Obe výstupy zobrazujú skladbu Turkish March od Mozarta, pričom ide o 16 dvadsaťpäť sekundových úsekov, ktorých odstup je jedna sekunda.

Literatúra

[1] Thierry Dutoit. *A Short Introduction to Text-to-Speech Synthesis*. Online. Dec. 1999. URL: http://tcts.fpms.ac.be/synthesis/introtts_old.html.

[2] Fifth Generation Computer Comporation. *Speaker independent connected speech recognition*. Online. URL: http://www.fifthgen.com/speaker-independent-connected-s-r.htm.

[3] *Image Recognition*. TensorFlow. Nov. 2017. URL: https://www.tensorflow.org/tutorials/image_recognition.

[4] Google Cloud Platform. *Cloud vision API*. Online. URL: https://cloud.google.com/vision/.

[5] Ryan Kiros Kelvin Xu Jimmy Lei Ba. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*. Tech. spr. Universite de Montréal, University of Toronto, apr. 2016. URL: https://arxiv.org/pdf/1502.03044.pdf.

[6] Matthew Field. "Facebook tests face recognition technology". In: *The Telegraph* (okt. 2017). URL: http://www.telegraph.co.uk/technology/2017/10/02/forgot-password-facebook-tests-face-recognition-technology-unlock/.

[7] Xinchen Yan Scott Reed Zeynep Akata. *Generative Adversarial Text to Image Synthesis*. Tech. spr. University of Michigan, Ann Arbor, MI, USA, jún 2016. URL: https://arxiv.org/pdf/1605.05396.pdf.

[8] Soumith Chintala Alec Radford Luke Metz. *Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks*. Tech. spr. indico Research, Facebook AI Research, jan. 2016. URL: https://arxiv.org/pdf/1511.06434.pdf.

[9] Homer H. Chen Yi-Hsan Yang. *Music Emotion Reckognition*. Boca Raton: CRC Press, 2011. ISBN: 978-1-4398-5046-6.

[10] George Tzanetakis Tao Mitsunori Ogihara. *Music data mining*. Boca Raton: CRC Press, 2012. ISBN: 978-1-4398-3552-4.

[11] Jiří Pospíchal Vladimír Kvasnička Ľubica Beňušková. *Úvod do teórie neurónových sietí*. Iris, 1997. ISBN: 8088778301.

[12] Carl Doersch. *Tutorial on Variational Autoencoders*. Tech. spr. Carnegie Mellon / UC Berkeley, aug. 2016. URL: https://arxiv.org/pdf/1606.05908.pdf.

[13] Jean Pouget-Abadie Ian J. Goodfellow. *Generative Adversarial Nets*. Tech. spr. Departement d'informatique et de recherche opérationnellé Universite de Montréal, jún 2014. URL: https://arxiv.org/pdf/1406.2661.pdf.

[14] Hongsheng Li Han Zhang Tao Xu. *StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks*. Tech. spr. Rutgers University, Lehigh University, The Chinese University of Hong Kong, aug. 2017. URL: https://arxiv.org/pdf/1612.03242.pdf.

[15] Simon Osindero Mehdi Mirza. *Conditional Generative Adversarial Nets*. Tech. spr. Departement d'informatique et de recherche opérationnellé Universite de Montréal, Flickr / Yahoo Inc. San Francisco, nov. 2014. URL: https://arxiv.org/pdf/1411.1784.pdf.