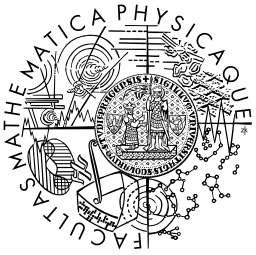


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

**BAKALÁŘSKÁ PRÁCE**

Gabriela Dvořáková

# Použití DNN pro analýzu trojúhelníkových sítí v geometrické morfometrii

Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Josef Pelikán

Studijní program: Informatika

Studijní obor: IPSS

Praha 2018

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Rada by som vyjadrila svoju vďačnosť predovšetkým vedúcemu práce RNDr. Josefovi Pelikánovi za jeho cenné rady, nadšenie a promptné odpovede na moje dotazy aj v čase nad rámec svojich pracovných povinností.

Veľké poďakovanie patrí tiež rodičom a sestre, ktorí boli svedkami všetkých ťažkých chvíľ, ktoré so sebou celé štúdium prinieslo. Napriek tomu ma po všetkých stránkach podporovali a som za nich nekonečne vďačná.

Ďalej by som sa chcela poďakovať laboratóriu Katedry antropologie a genetiky člověka Přírodovědecké fakulty UK za poskytnutie modelov naskenovaných tvári.

V neposlednej rade ďakujem organizácii MetaCentrum za prístup k výpočtovým a úložným zdrojom, ktoré sú vo vlastníctve skupín a projektov prispievajúcich k Národní Gridové Infrastruktuře. MetaCentrum poskytuje svoje služby v rámci programu „Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042).

Název práce: Použití DNN pro analýzu trojúhelníkových sítí v geometrické morfometrii

Autor: Gabriela Dvořáková

Katedra: Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Josef Pelikán, Katedra softwaru a výuky informatiky

Abstrakt: Témou tejto práce je aplikácia hlbokého učenia na rozpoznávanie trojrozmerných objektov. Hlboké učenie už bolo úspešne použité na rozpoznávanie 3D dát, avšak väčšina existujúcich prác volila reprezentáciu 3D objektov pomocou sekvencie 2D obrázkov alebo diskretizáciou do binárnych voxelov. Hlavným cieľom je navrhnúť alternatívne mapovanie 3D dát na vstupy NN. Predstavíme tri reprezentácie: Prosté uloženie súradníc vrcholov do poľa, projekcia do 2D mriežky a množina povrchových lomených kriviek vedúcich jeho významnými časťami. Všetky popísané reprezentácie sú testované na úlohe klasifikácie pohlavia pomocou NN a CNN na 3D faciálnych modeloch. Analyzovali sme vplyv relativizácie súradníc a vytvorenia modifikovanej datovej množiny extrakciou oblasti nosa z pôvodných trojuholníkových sietí. Experimentálne výsledky preukázali kvalitu prístupu s lomenými krivkami pomocou CNN úspešnosťou 84,2%.

Klíčová slova: trojúhelníková síť, model obličeje, hluboké učení, geometrická morfometrie

Title: Using DNN for triangular network analysis in geometric morphometry

Author: Gabriela Dvořáková

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Josef Pelikán, Department of Software and Computer Science Education

Abstract: The aim of this thesis is to use deep learning for the task of 3D object recognition. Deep learning has been succesfully used for three dimensional data recognition. However, most of the published work chose to represent 3D objects as a set of projected 2D pixel images or in the form of binary voxels. The main goal is to propose an alternative mapping of 3D data to the NN input. Three data representations are introduced: Treating vertex coordinates as a 1D array, projection to a 2D grid and a set of surface oblique lines crossing the significant parts of an object. All of the proposed data representations are tested for the gender classification task using NN and CNN on 3D facial models. We analyzed the impact of coordinate relativization and a new modified dataset created by extracting a nose area from original triangle meshes. Experimental results confirmed the quality of the oblique lines approach with achieved classification accuracies of 84,2% using CNN.

Keywords: triangle mesh, facial model, deep learning, geometric morphometry

# Obsah

# Úvod

V posledných desiatich rokoch je strojové učenie svedkom revolúcie v hlbokých neurónových sieťach (DNNs). Neurónové siete v súčasnosti poskytujú najlepšie riešenia mnohých problémov od rozpoznávania obrazu, reči, spracovania prirodzeného jazyka, automatického prekladu, až po aplikácie v oblasti robotiky, virtuálnej reality, medicíny alebo financií.

Konvolučné neurónové siete (CNNs) sú biologicky inšpirovanou triedou hlbokého učenia s najväčším zastúpením v súčasných aplikáciách, a to predovšetkým v rozpoznávaní 2D obrazu. Úspešne zapĺňajú medzeru medzi tradičnými technikami strojového učenia, ktoré vyžadujú typický krok manuálnej extrakcie príznakov z dát. CNNs sú efektívne v pochopení komplexného obsahu obrázkov a ukázali sa byť silným nástrojom nielen na rozpoznávacie, ale aj generatívne úlohy.

Zatiaľ čo rozpoznávanie obrazu na základe 2D obrázkov zaznamenalo veľký pokrok a pokračuje v napredovaní, spracovanie 3D dát stále mierne zaostáva. A síce sa množstvo informácií vo forme verejne dostupných 3D modelov neustále zvyšuje, v porovnaní so všadeprítomnými 2D obrázkami je stále nedostačujúce. Ďalším dôvodom je výhodná povaha obrázkov, pretože priestorové vzťahy medzi obrazovými prvkami sú zakódované v štruktúre samotného obrazu indexmi pixelov v matici. Point cloudy a trojuholníkové siete nedisponujú pravidelnou štruktúrou, preto nie je rovnako triviálne rozhodnúť sa, v akej forme budú prezentované neurónovej sieti.

Materiál tvoriaci datovú množinu má aplikáciu v obore antropológie a obsahuje modely ľudských tvári naskenovaných povrchovým 3D skenerom. Je k dispozícii vo forme vyčistených trojuholníkových sietí.

Väčšina existujúcich prístupov v rozpoznávaní 3D objektov môže byť rozdelená do dvoch hlavných tried. Jedna z nich je motivovaná úspechom CNN a typicky transformuje trojrozmerné dáta do množiny vyrendrovaných obrázkov z viacerých kamier. Druhá kategória ponecháva trojdimenzionalitu a diskretizuje 3D geometriu do pravidelnej mriežky voxelov. Takáto objemová reprezentácia má niekoľko výhod. Jednou z nich je napríklad silná expresivita dát, vďaka ktorej nevyžaduje k učeniu invarianty objektu z viacerých uhlov a neurónovej sieti postačuje menej trénovacích dát. Pridaná tretia dimenzia však so sebou prináša aj nevýhody vo forme vysokých nárokov na pamäťovú a výpočtovú réžiu.

## Cieľ práce

V tejto práci navrhneme a porovnáme tri alternatívne reprezentácie 3D faciálnych modelov. Prvým prístupom je uloženie súradníc vrcholov do poľa „bez ladu a skladu". Takáto reprezentácia neobsahuje žiaden manuálne vytvorený popis príznakov objektu a preto slúži prevažne ako referencia pre porovnanie s ďalšími metódami, u ktorých očakávame lepšie výsledky. Ďalšia reprezentácia je projekcia 3D dát do 2D mriežky, pre ktorú sme navrhli jednoduchý algoritmus mapovania vrcholov. Poslednou reprezentáciou je množina povrchových lomených čiar vedená cez významné časti klasifikovaného objektu. Prevedieme analýzu vplyvu ďalšieho predspracovania na výsledok klasifikácie, napríklad relativizácie súradníc

a selekcie oblasti nosa z pôvodných trojuholníkových sietí. Pre otestovanie kvality navrhnutých reprezentácii ich aplikujeme na úlohu rozpoznávania pohlavia pomocou NN a CNN.

## Štruktúra práce

Práca je rozdelená do piatich kapitol. V kapitole 1 si prostredníctvom biologickej motivácie popíšeme matematický model neurónu a jeho prepojenie do viacvrstvovej neurónovej siete. Vysvetlíme si proces učenia neurónových sietí a poslednú časť kapitoly venujeme konvolučným neurónovým sieťam.

Kapitola 2 popisuje označenú datovú množinu 3D povrchových faciálnych modelov, ktorá bola základom všetkých experimentov v tejto práci.

Výberu frameworku na trénovanie neurónových sietí je venovaná kapitola 3. Ďalej si preberieme popis algoritmov implementovaných v pomocných programoch a skriptoch.

V kapitole 4 si najprv povieme niečo o existujúcich reprezentáciach 3D geometrie a potom predstavíme nami navrhnuté prístupy.

Teoretické poznatky z kapitoly 1 a predspracované dáta popísané v kapitole 4 využijeme v kapitole 5. Vysvetlíme si postupy pri výbere architektúr sietí a prediskutujeme výsledky experimentov.

# 1. Umelé neurónové siete

## 1.1 Úvod

Umelá neurónová sieť je výpočetný systém inšpirovaný biologickým neurónovým systémom, akým je mozog cicavcov. Je vytvorený tak, aby simuloval aktivitu neurónových vrstiev v mozgovej kôre pomocou matematického modelu jej neurofyziologickej štruktúry. Disponuje schopnosťou sa učiť, generalizovať a spracovávať informácie a riešiť komplexné problémy. Nachádza širokú aplikáciu v rozpoznávaní a klasifikácií vzorov do tried, predikovaní časových radov a hľadaní závislostí medzi vstupmi a výstupmi v digitálnej reprezentácii jazyka, zvukov, obrazov a iných dát.

Simulácia komplikovanej molekulárnej a biologickej mašinérie skutočnej siete neurónov je stále značne idealizovaná. Jednak preto, že naše vedomosti o neurónoch a ľudskom mozgu nie sú kompletné. Navyše kvôli limitom výpočtovej sily je robustná sieť mimo náš technologický dosah. Preto je snaha docieliť čo najvyššiu efektivitu a zároveň udržiavať veľkosť siete čo najmenšiu.

Pretože konštrukcia neurónových sietí je založená na neurobiológii a jej termínoch, začneme túto kapitolu stručným priblížením fungovania jej biologickej predlohy. Popíšeme si model perceptrónu, ktorý síce dokáže aproximovať iba lineárnu funkciu, ale má nezanedbateľnú dôležitosť z hľadiska histórie a teórie strojového učenia. Je základom pre moderné dopredné viacvrstvové siete, ktoré si následne vysvetlíme. Na záver si priblížime najčastejší postup učenia neurónových sietí a to metódu spätného učenia a odvodíme si ju matematicky.

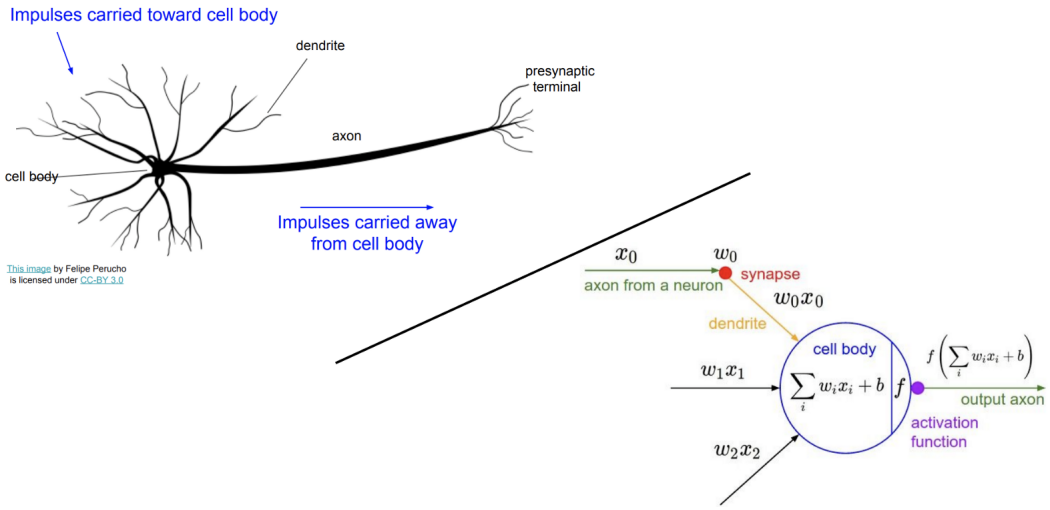
## 1.2 Model neurónu

Prvý model umelých neurónových sietí pomocou analýzy biologického nervového systému vytvorili neurofyziológ Warren McCulloch a matematik Walter Pitts (McCulloch a Pitts, 1943). Tým položili základ nového výskumného smeru umlých neurónových sietí (ANN). Umelý model predstavuje iba zjednodušenie komplexnej fyzickej štruktúry biologického modelu, ale ich spoločný základ tvorí neurón, ktorý si popíšeme v tejto podkapitole.

### 1.2.1 Biologický neurón

Základná funkčná a stavebná jednotka nervovej sústavy človeka je bunka zvaná *neurón* (Barrett a kol., 2009). V ľudskom nervovom systéme je odhadom 100 biliónov neurónov. Neuróny nachádzajúce sa v mozgovej kôre majú na starosti myslenie, vnímanie a pamäť. Tie sú vzájomne prepojené pomocou spojov (*synapsie*), kde dochádza k prenosu informácií pomocou elektrochemických impulzov. Neurón pozostáva z tela (*somy*) a zo spojov (*dendritov*), omocou ktorých prijíma informácie z iných neurónov alebo vonkajších receptorov. Príklad biologického neurónu popisuje obrázok 1.1.

Silu pôsobenia impulzu určuje *váha synapsie*, ktorú si môžeme predstaviť ako reálne číslo vyjadrujúce intenzitu elektrochemického signálu. Pokiaľ suma impulzov vedúca od ostatných neurónov prekročí istú hranicu nazývanú *prah excitácie*,




Obrázek 1.1: Porovnanie biologického a umelého neurónu. Obrázok prevzatý z (Li a kol., 2017)

tak neurón „vypáli" - vygeneruje výstupný impulz do ďalších neurónov v sieti. Impulz sa ďalej šíri dlhým, úzkym výbežkom nervového vlákna (*axónom*) k druhému neurónu. Axón sa na konci bohato rozvetvuje. Schopnosť učenia môže byť popísaná termínom *synaptická plasticita*, čo znamená že váhy jednotlivých synaptických spojení sa menia vplyvom individuálnych skúseností a neurónovej aktivity. Avšak kompletný obrázok o tom, ako sa mozog učí, vedcom stále chýba.

### 1.2.2 Umelý neurón

V nasledujúcej časti sme vychádzali predovšetkým z knihy Goodfellow a kol. (2016), ktorú odporúčame čitateľovi pre zoznámenie sa s teóriou hlbokého učenia. Umelý neurón vznikol zjednodušením svojej biologickej predlohy popísanej v predošlej sekcii. Skladá sa z uzlov (*neuróny*), vstupných signálov (*dendrity*), hrán (*synapsie*) asociovaných s váhami, biasu (*externý vstup z receptorov*) a skalárneho výstupu (*axón*).

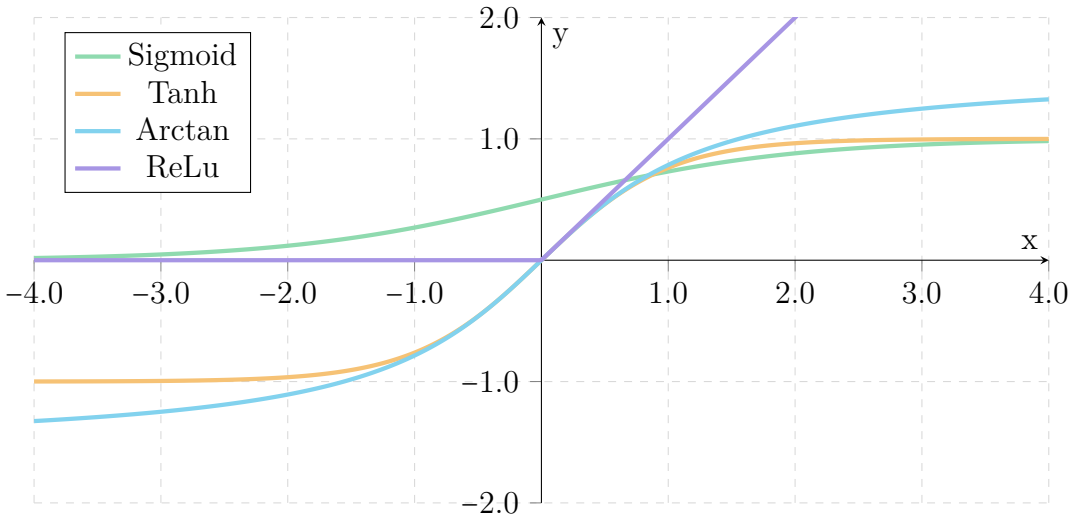
Do neurónu vedie vektorový vstup $x = (x_1, x_2, \ldots, x_n)$. Vstupu sú pridelené váhy $w = (w_1, w_2, \ldots, w_n)$, t.ž. $w_i \in \mathbb{R}$. Komponenty vstupu $x_i$ sú vynásobené príslušnými váhami $w_i$ a následne sčítané. Ich vynásobením získame tzv. *vážený súčet*. K výsledku je navyše pripočítaný vstup $x_0$ s konštantnou hodnotou 1 a váhou $w_0$, tzv. *bias* $b$, ktorý ovplyvňuje aktivačnú funkciu neurónu. Celkovému súčtu $z$ hovoríme **vnútorný potenciál**:

$$z = x_0 w_0 + \sum_{i=1}^{n} w_i x_i = b + \mathbf{W}\mathbf{x}, \tag{1.1}$$

kde $\mathbf{W}$ je matica váh. Výstup z neurónu $y$ je potom definovaný vzťahom:

$$y = f(z) = f\left(x_0 w_0 + \sum_{i=1}^{n} w_i x_i\right) = f(b + \mathbf{W}\mathbf{x}) \tag{1.2}$$

kde funkcia $f : \mathbb{R} \to \mathbb{R}$ je tzv. *aktivačná funkcia*, ktorá nesie podobný účel ako prahovanie v biologickom neuróne. Na obrázku 1.2 sú znázornené najpoužívanejšie typy aktivačných funkcií.




Obrázek 1.2: Aktivačné funkcie, ktoré sa v praxi vyskytujú najčastejšie. **Logistický sigmoid** sa často využíva pre binárne klasifikačné úlohy a má matematickú formu $f_{sigmoid}(x) = \frac{1}{1+e^{-x}}$. Reálne čísla premieta do intervalu $(0, 1)$. Alternatívou k logistickej sigmoide je **hyperbolický tangens** $f_{tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. Výstupné hodnoty patria do intervalu $(-1, 1)$. Ďalšou funkciou je **arkus tangens** $f_{atan}(x) = arctan(x)$ s hodnotami v intervale $(-\frac{\pi}{2}, \frac{\pi}{2})$. **ReLu** (Rectified Linear Unit) počíta funkciu $f_{ReLu}(x) = \max(0,x)$, ktorá všetky hodnoty menšie ako 0 nahradí hodnotou 0. (0 je prahová hodnota).

V nasledujúcom texte budeme termínom *neurón* a *neurónová sieť* myslieť jej matematický model.

## 1.3 Topológia neurónovej siete

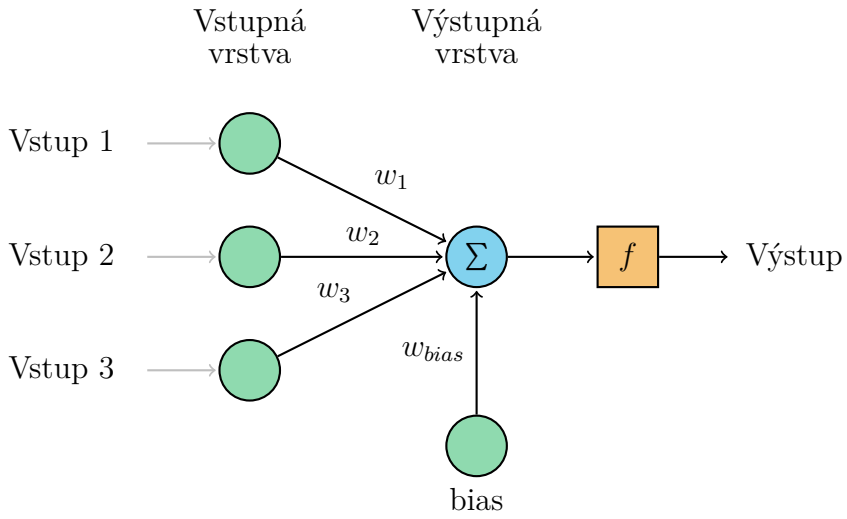
Vhodným prepojením neurónov, ktoré sme si popísali, vznikne neurónová sieť. Zadefinujeme si ju formálne:

**Definice 1** (Umelá neurónová sieť)**.** ***Neurónová sieť*** *je usporiadaná trojica (N, C, w), kde:*

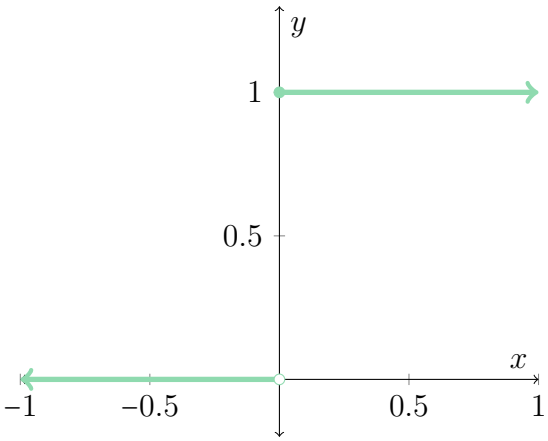
- $\boldsymbol{N}$ *je konečná, neprázdna množina neurónov*
- $\boldsymbol{C} \subseteq N \times N$ *je neprázdna množina* $\{(i, j) | i, j \in \mathbb{N}\}$*, ktorej prvky sú orientované spojenia medzi neurónmi*
- $\boldsymbol{w} : C \to \mathbb{R}$ *je váhová funkcia, kde* $w(i, j)$ *je váha spojenia medzi neurónom* $i$ *a neurónom* $j$*, skrátene* $w_{i,j}$

### 1.3.1 Jednovrstvový perceptrón

Najjednoduchší model neurónovej siete je *perceptrón* (Rosenblatt, 1957), ktorý je zobrazený na obrázku 1.3. Pôvodne bol navrhnutý ako model zrakovej sústavy. Základom Rosenblattovho perceptrónu je vstupná vrstva zložená z $n$ vstupov $x = (x_1, x_2, \ldots, x_n)$ a jedného výstupného neurónu. Každý vstupný neurón $x_i$




Obrázek 1.3: Rosenblattov jednovrstvový perceptrón s tromi vstupnými neurónmi a jedným výstupným neurónom. Smer propagácie signálu je naznačený šípkami. Dáta môžu nadobúdať binárne, celočíselné alebo reálne hodnoty. Váhy $w_{bias}, w_1, w_2, w_3$ vstupujúce do sčítacieho neurónu $\sum$ sú trénovateľným parametrom siete.




Obrázek 1.4: Ostrá nelinearita s prahom $\boldsymbol{\theta} = 0$

je s ním prepojený váhou $w_i$, $i = 1, 2, \ldots, n$. Výstupný sčítací neurón počíta lineárnu kombináciu vstupov a synaptických váh. Berie do úvahy aj *bias*, ktorý nepochádza zo vstupných neurónov, ale z okolitého sveta. Na výslednú sumu je aplikovaná skoková aktivačná funkcia $f$, ktorá je ilustrovaná na obrázku 1.4 (ostrá nelinearita). Môže nadobúdať hodnotu 0 alebo 1, pričom konštantu $\boldsymbol{\theta}$ nazývame *aktivačný prah neurónu*:

$$f(z) = \begin{cases} 1, & \text{ak } z \geq \boldsymbol{\theta} \\ 0, & \text{ak } z < \boldsymbol{\theta} \end{cases}, \tag{1.3}$$

Trénovací algoritmus perceptrónu môže byť aplikovaný iba na jednu vrstvu. V opačnom prípade by bola potrebná nejaká odozva od jednotlivých vrstiev, napríklad metódou *spätného učenia*, ktorú si popíšeme neskôr. Hodnota, ktorá vstupuje do sčítacieho neurónu, je zhodná s vnútorným potenciálom (1.1).

Rosenblatt v *konvergenčnom teoréme perceptrónu* (Rosenblatt, 1962) dokázal, že ak môže byť datová množina rozdelená do dvoch *lineárne separovateľných tried*, tak algoritmus učenia perceptrónu v konečnom počte krokov konverguje a aproximuje pozíciu rozhodovacej roviny medzi dvomi triedami vo forme nadroviny. Ide o tzv. *lineárny klasifikátor*. Výpočtová schopnosť jednoduchých perceptrónov sa tým ukázala byť značne obmedzená, pretože reálne problémy väčšinou lineárny charakter nemajú. Typickým príkladom problému, ktorý nedokáže riešiť, je XOR problém (Minsky a Papert, 1969). Kvôli tomuto zisteniu sa výskum nasledujúcich dvadsať rokov spomalil. Riešením tohoto problému je spojiť perceptróny do niekoľkých vzájomne prepojených vrstiev, čím vznikne tzv. *viacvrstvový perceptrón*.

### 1.3.2 Viacvrstvové dopredné neurónové siete

*Viacvrstvová neurónová sieť* (angl. feed-forward neural network) je charakteristická tým, že signál počas aktivácie prúdi smerom od vstupu k výstupu. Táto vlastnosť zabezpečuje, že všetky výpočty sú prevedené sekvenčne. Štruktúra siete je organizovaná do acyklického grafu, ktorého uzly sú tvorené neurónmi a hrany sú ohodnotené váhami. Váhy predstavujú pamäť neurónovej siete. Existuje špeciálny model neurónových sietí, tzv. *rekurentná neurónová sieť*, ktorá môže obsahovať cykly vedúce signál aj spätne do neurónov, z ktorých predtým vychádzal. Týmito sieťami sa v tejto práci hlbšie zaoberať nebudeme.

Prvá vrstva, tzv. *vstupná*, má za úlohu sprostredkovať vstupné vzory zvyšku siete. Posledná vrstva, tzv. *výstupná*, obsahuje $k$ neurónov, kde $k$ je počet tried klasifikácie. Vrstvy medzi nimi nazývame *skryté vrstvy*. Tie aplikujú aktivačnú funkciu na vážený súčet a výsledok pošlú výpočtovým jednotkám v ďalšej vrstve. Tento proces sa opakuje až po výstupnú vrstvu. Obvykle je každý uzol vrstvy $i$ spojený s každým uzlom vrstvy $i + 1$. Takejto neurónovej sieti hovoríme, že je **plne prepojená** (v anglickej literatúre *fully connected*). Trénovanie neurónovej siete s minimálne jednou skrytou vrstvou sa preto volá **hlboké učenie** (angl. *deep learning*). Aktivačná funkcia $f$ neurónu môže byť ľubovoľná nelineárna diferencovateľná funkcia.

Medzi vrstvami neurónov existujú iba dopredné spojenia a každý výstup sa nimi po aplikovaní aktivačnej funkcie propaguje ďalej. Takýto spôsob učenia sa nazýva **dopredná propagácia** (angl. feed-forward propagation).

## 1.4 Proces učenia neurónovej siete

Najdôležitejšou vlastnosťou neurónovej siete je iteratívny tréningový proces. V každej iterácii je sieti prezentovaný jeden príklad z množiny dát, na základe ktorého sa učí a upravuje parametre (váhy) tak, aby predikovala správne výsledky. Po predložení všetkých vzorov trénovacej množiny neurónovej sieti sa celý proces zvykne opakovať. V termínoch strojového učenia tomuto procesu hovoríme **epocha** učenia. Neurónová sieť nie je schopná učenia bez správne zvolenej trénovacej množiny. Dôvodom môže byť nedostatočné množstvo trénovacích dát. Siete nekonvergujú ani v prípade, že vstup neobsahuje dostatok informácií potrebných k určeniu výsledku. Preto by v ideálnom prípade mala mať sieť dostatočne veľkú a rozmanitú vzorku, z ktorej sa môže učiť. V tejto podkapitole si popíšeme, ako pri výbere trénovacej množiny postupovať.

Ďalej si popíšeme algoritmus **spätnej propagácie**, vďaka ktorému sa v posledných rokoch stali neurónové siete populárnym nástrojom na klasifikáciu a predikciu.

### 1.4.1 Trénovacia, testovacia a validačná množina

**Trénovacia množina** $T$ je množina usporiadaných dvojíc, pomocou ktorých chceme natrénovať neurónovú sieť, t.ž.:

$$T = \{(x_i, t_i)\}_{i=1}^{N} \tag{1.4}$$

kde $\mathbf{x}_i = [x_1, x_2, \ldots, x_n]^T$ je $i-tý$ vektor vstupu, $t_i$ je skalár s číselnou hodnotou označujúcou triedu, do ktorej patrí $i-tý$ príklad a jeho kardinalita musí byť rovná počtu tried. $N$ je počet príkladov v trénovacej množine. Označením $y_i$ budeme rozumieť reálny výstup siete s predikovanými hodnotami na danom vzore $i$.

Dáta sa musia voliť tak, aby zastupovali čo najširšiu variáciu vzorov. Cieľom je, aby natrénovaná sieť mala schopnosť **generalizovať**, inak povedané, aby dosahovala dobré výsledky aj pre neznáme vzory. Naopak nežiadúci jav, ktorému sa snažíme vyhnúť, je **preučenie** (*overfitting*). Dochádza k nemu, keď sieť stráca schopnosť generalizovať, pretože učenie trvalo príliš dlho a naučila sa podrobné detaily konkrétnej trénovacej množiny. Môže k nemu dochádzať aj keď je trénovacia množina príliš malá a nedostatočne pokrýva priestor všetkých možných kombinácií vzorov.

Medzi tréningovými metódami obvykle rozlišujeme dve základné kategórie. **Učenie s učiteľom** (*supervised learning*) je najčastejšou formou strojového učenia. Vyžaduje anotovanú tréningovú množinu. Každý vstup má priradené označenie s požadovanou výstupnou hodnotou, a tak s ňou môže byť výstup siete s predikovaným výsledkom priamo porovnaný. **Učenie bez učiteľa** (*unsupervised learning*) nemá k dispozícii žiadne označenie s prislúchajúcou triedou. Prebieha tak, že sieť má na vstupe dáta, v ktorých sa snaží rozpoznať vzory, nájsť spoločné vlastnosti a generovať pre vzory triedy. Výhodou oproti učeniu s učiteľom je, že získať takéto dáta je omnoho jednoduchšie. Tomuto typu učenia sa ale v tejto práci venovať nebudeme, pretože sme po celý čas pracovali s dobre anotovanou datovou množinou.

V prípade, že máme niekoľko z hľadiska výkonnosti konkurujúcich si architektúr (líšiacich sa napríklad počtom skrytých vrstiev, počtom neurónov v skrytých vrstvách, atď.), ktoré vznikli na základe trénovacej množiny, tak na zvolenie optimálnej architektúry použijeme **validačnú množinu**. Využíva sa aj ako koncová podmienka pri algoritme spätného šírenia chýb. Ak sa začne prediktívna úspešnosť zlepšovať na tréningovej množine, ale zhoršovať na validačnej množine (dochádza k preučeniu), je dobré ukončiť trénovanie.

**Testovacia množina** je množina vzorov, ktoré sa neúčastnili učenia a preto je vhodným kandidátom na overenie, ako dobre je náš model natrénovaný.

### 1.4.2 Algoritmus spätného šírenia chyby

Najpoužívanejšia metóda trénovania viacvrstvových neurónových sietí je metóda **spätného šírenia chyby** (v strojovom učení je častejšie využívaný anglický

termín *back-propagation*). Cieľom učiaceho procesu je nájsť takú kombináciu váh $\mathbf{W}$, aby sa skutočný výstupný vektor $\mathbf{y}$ čo najmenej líšil od cieľových hodnôt $\mathbf{t}$ pre všetky vzory z množiny. Na to si potrebujeme zadefinovať kritérium, akým budeme kvantifikovať úspešnosť aproximácie.

Uvážme trénovaciu množinu $T$ definovanú v 1.4, ktorú použijeme na učenie neurónovej siete. Nech $y_j$ je výsledná predikovaná hodnota neurónu $j$ vo výstupnej vrstve a $t_{pj}$ je očakávaná výstupná hodnota vzoru s indexom $p$. *Chybový signál* $e_{pj}$ generovaný výstupným neurónom $j$ definujeme ako

$$e_{pj} = t_{pj} - y_j \tag{1.5}$$

Obvykle použitá metóda na výpočet miery výkonnosti je metóda **najmenších štvorcov** (MSE). **Parciálna chyba** vhľadom ku $p$-tej trénovacej vzorke je teda sumou súčtov druhých mocnín chýb na všetkých výstupných neurónoch vynásobená koeficientom

$$E_p(\mathbf{W}) = \frac{1}{2}\sum_{j=1}^{m}(e_{pj})^2 = \frac{1}{2}\sum_{j=1}^{m}(t_{pj} - y_j)^2 = \frac{1}{2}\sum_{j=1}^{m}(t_{pj} - f(\mathbf{W}\mathbf{x_p}))^2, \tag{1.6}$$

kde $m$ je počet neurónov vo výstupnej vrstve. Pre jednoduchosť vysvetľovania sme vynechali člen bias vo vstupe aktivačnej funkcie $f$. Potom *chybovú funkciu* $E(\mathbf{W}) : \mathbf{W} \to \mathbb{R}$, definujeme ako súčet parciálnych chýb spomedzi $N$ vzorov z trénovacej množiny:

$$E(\mathbf{W}) = \sum_{p=1}^{N} E_p(\mathbf{W}) \tag{1.7}$$

Teraz by malo byť zrejmé, že napriek tomu, že názov *spätná propagácia* môže mylne napovedať, že sa signál šíri po sieti opačným smerom (ako to je v rekurentných neurónových sieťach), v skutočnosti sa spätne šíri iba chyba. Vlastnosti dopredných neurónových sietí ostávajú zachované. Trénovací algoritmus spätnej propagácie pozostáva z dvoch fáz:

- **Dopredná fáza**, v ktorej je do vstupnej vrstvy privádzaný vektor vstupných signálov. Vstup do každého neurónu sa násobí váhami, na výstup sa aplikuje aktivačná funkcia (1.2) a propaguje sa smerom k výstupnej vrstve. Výsledok je porovnaný s požadovanou hodnotou a spočíta sa celková chyba.

- **Spätné šírenie chýb** (Rumelhart a kol., 1986) prepočítava chybu spätne do prechádzajúcich vrstiev smerom k vstupnej vrstve. Cieľom je minimalizovať chybovú funkciu $E(\mathbf{W})$ korekciou váh neurónov.

Prvú fázu sme si priblížili v predošlej podkapitole, teraz si stručne vysvetlíme základnú variantu optimalizačnej procedúry v druhej časti algoritmu. Cieľom je nájsť $\hat{\mathbf{W}}$ t.ž.

$$E(\hat{\mathbf{W}}) = \min E(\mathbf{W}),$$

teda aby bola hodnota chybovej funkcie čo najmenšia pre danú datovú množinu.

Nutná podmienka k nájdeniu lokálneho optima $\mathbf{W}$ je, aby bol *gradient* [1] chybovej funkcie $\partial E/\partial w$ nula, $l$ je počet váh vo vektore $\mathbf{W}$:

$$\nabla E(\mathbf{W}) = \frac{\partial E(\mathbf{W})}{\partial w} = \left[\frac{\partial E(\mathbf{W})}{\partial w_1}, \frac{\partial E(\mathbf{W})}{\partial w_2}, \ldots, \frac{\partial E(\mathbf{W})}{\partial w_l}\right]^T = 0 \tag{1.8}$$

K minimalizácii chybovej funkcie a vyriešeniu rovníc v 1.8 sa obvykle používa iteratívna metóda **gradientného zostupu** (angl. *gradient descent*). Iteratívna znamená, že sa váhy nemenia naraz, ale inkrementálne. Metódu gradientného zostupu využíva aj knižnica na strojové učenie *Tensorflow* (Abadi a kol., 2015), ktorú sme využívali v tejto práci, preto je dobré vysvetliť si ho podrobnejšie.

V gradientnom zostupe je zmena váh $\mathbf{W}$ nepriamo úmerná parciálnej derivácii:

$$\Delta\mathbf{W} = -\eta\nabla E(\mathbf{W}) \tag{1.9}$$

Symbol $\eta$ si vysvetlíme nižšie. V iterácii $t = 0$ sú hodnoty váh $\mathbf{W}^{(0)}$ inicializované náhodnými malými číslami s obvykle veľkou chybou. Je treba pripomenúť, že neurónová sieť je citlivá na počiatočné nastavenie váh a ovplyvňuje sa tým konečný výsledok učenia. Ďalej je zostrojený gradient a zmena je aktualizovaná v smere najväčšieho klesania, teda v smere negatívneho gradientu. Ak sa snažíme aproximovať konvexnú funkciu, nájdenie lokálneho minima ešte neznamená, že ide zároveň aj o globálne minimum. Ak sa lokálne minimum zhoduje s globálnym, tak bolo nájdené optimálne riešenie. Celý proces prebieha iteratívne, kde sa v každom kroku iterácie posunieme o krok k menším hodnotám chybovej funkcie, opravenej sieti sa privedie ďalší vstupný vzor a vypočítajú sa nové hodnoty váh na základe gradientu chybovej funkcie:

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} + \eta\Delta\mathbf{W}^{(t)} \tag{1.10}$$

Skalárny symbol $\eta \in (0, 1)$ je optimalizačný parameter označujúci veľkosť kroku, ktorý v strojovom učení nazývame **rýchlosť učenia** (*learning rate*). Rýchlosť konvergencie je závislá na výbere $\eta$. Ak je krok príliš veľký, tak môžeme minúť bod, kde je hodnota funkcie najnižšia. S malou veľkosťou kroku je malá zmena v hodnotách váh, tým pádom je aj znižovanie chyby veľmi malé. Dochádza k častému počítaniu gradientu a preto je nájdené minimum presnejšie, ale učenie pomalé.

Pre zmenu váhy $w_{ji}$ vedúcu z neurónu $i$ do neurónu $j$ (nezabudnime, že váhy sú vedené iba medzi neurónmi v rozdielnych, za sebou idúcich vrstvách) platí vzťah:

$$\Delta w_{ji}^{(t)} = -\eta\frac{\partial E}{\partial w_{ji}} = -\eta\sum_{p=1}^{N}\frac{\partial E_p}{\partial w_{ji}} \tag{1.11}$$

Použitím jednoduchého reťazového pravidla o derivácii zloženej funkcie vypočítame parciálnu deriváciu. Jednotlivé derivácie si odvodíme nižšie. Pripomeňme si, že $z$ označuje vnútorný potenciál neurónu (1.1) a $y = f(z)$. Zároveň vidíme, prečo je diferencovateľnosť aktivačnej funkcie $f$ nutnou podmienkou pri jej voľbe:

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial y_j}\frac{\partial y_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ji}} = \frac{\partial E_p}{\partial y_j}f'(z(j))y_i \tag{1.12}$$

[1] Gradient je vektor určujúci smer najväčšieho spádu chybovej funkcie.

kde derivácia $y_j$ podľa $z_j$:

$$\frac{\partial y_j}{\partial z_j} = f'(z(j)),$$

pokračujeme derivovaním $z_j$ podľa $w_{ji}$ a dostávame:

$$\frac{\partial z_j}{\partial w_{ji}} = y_i$$

Pri derivovaní $\partial E_p/\partial y_j$ potrebujeme vedieť chybu na výstupe neurónu $j$. Preto musíme odlišovať dva prípady:

1. **Neurón $j$ je vo výstupnej vrstve**
   Ak je neurón $j$ vo výstupnej vrstve, tak na vypočítanie chyby v rovnici 1.7 sa do $y_j$ priamo dosadí výstupná hodnota neurónu. Pokračujeme deriváciou $E_p$ (1.6) podľa $y_j$ a dostaneme:

$$\frac{\partial E_p}{\partial y_j} = \frac{\partial E_p}{\partial e_j}\frac{\partial e_j}{\partial y_j} = -e_j = -(t_j - y_j)$$

2. **Neurón $j$ je v skrytej vrstve** V tomto prípade nepoznáme požadovaný výstup z neurónu $j$. Potrebujeme zistiť, ako sa chyba spôsobená v neuróne $y_j$ propagovala do ďalšej ($k$-tej) vrstvy. Preto musíme chybu vypočítať rekurzívne ďalším aplikovaním reťazového pravidla:

$$\frac{\partial E_p}{\partial y_j} = \sum_{k \epsilon j\rightarrow} \frac{\partial E_p}{\partial y_k}\frac{\partial y_k}{\partial z_k}\frac{\partial z_k}{\partial y_j} = \sum_{k \epsilon j\rightarrow} -\frac{\partial E_p}{\partial y_k}\frac{\partial y_k}{\partial z_k} w_{kj}$$

Zhrňme si priebeh algoritmu:

1. Inicializuj váhy $\mathbf{W}^{(0)}$ na náhodné malé hodnoty.
2. Vyber vzor z tréningovej množiny a polož ho do vstupnej vrstvy.
3. Aplikuj doprednú propagáciu a získaj predikované hodnoty $\hat{y}(\mathbf{w})$.
4. Vypočítaj chybový signál $E(\mathbf{w})$ vzhľadom na očakávaný výstup $y(\mathbf{w})$.
5. Pomocou spätnej propagácie vypočítaj gradient chybovej funkcie.
6. Aktualizuj váhy.
7. Ak je splnená koncová podmienka (splnený počet povolených epoch, celková chyba je menšia než konštantne zvolená odchýlka), skonči. Inak pokračuj bodom 2.

## 1.5 Konvolučné neurónové siete

V tejto sekcii sa zameriame na špeciálnu triedu viacvrstvových dopredných neurónových sietí, ktorú nazývame konvolučné neurónové siete (*convolutional neural networks* (CNN)). Štruktúra konvolučnej neurónovej siete je navrhnutá tak, aby dokázala rozpoznať dvojdimenzionálne tvary invariantne voči niektorým transformáciám a deformáciám príznakov. Ak sa obrázok otočí, posunie o zopár pixelov, zošikmí alebo zmení veľkosť, klasifikácia by sa nemala zmeniť. Konvolučné neurónové siete sú jedným z najdôležitejších príkladov úspešného prenesenia vedomostí získaných na základe pozorovania živých organizmov do strojového učenia.

Teoretickým základom bola práca dvoch neurofyziológov, ktorí skúmali komplexný zrakový systém mačky. Hubel a Wiesel (1968) zistili, že niektoré neuróny hľadajú v zrakovom stimule konkrétne vlastnosti, ako sú napríklad hrany rôznych orientácií (horizontálne, vertikálne, diagonálne). Navyše, všetky takéto neuróny boli usporiadané do stĺpcovej štruktúry. Táto myšlienka je základom konvolučných neurónových sietí.

Ďalšie zásluhy na úspechu moderných konvolučných neurónových sietí majú Lecun a Bottou. Navrhli a detailne popísali prvú neurónovú architektúru CNN zvanú **LeNet 5** použitú na rozpoznávanie ručne písaných znakov, ktorá dosiahla úspešnosť až 99,2% (LeCun a kol., 1998).

Najpopulárnejšie CNN je nepochybne v spracovaní obrazu, v ktorom dosahuje špičkové výsledky. Ich použitie sa ale ukázalo byť natoľko široké, že našli uplatnenie v oboroch ako rozpoznávanie reči, spracovanie prirodzeného jazyka alebo rozpoznávanie objektov z videa. Dnes zaznamenáva explóziu rôznych aplikácií: Pomocou CNN bol porazený šampión v Go (Silver a kol., 2016), Google klasifikoval a označil milióny obrázkov na internete, aby užívateľ dostával lepšie vyhľadávacie výsledky, Facebook pomocou CNN automaticky detekuje tváre (Taigman a kol., 2014).

Klasická dopredná neurónová sieť obsahuje spojenie medzi každými dvomi neurónmi susedných vrstiev. Konvolučná sieť má oveľa redšie neurónové spojenia, tým pádom menší počet synaptických váh, menšie nároky na pamäť siete a efektívnejšie výpočty. To je dosiahnuté princípom tzv. *zdieľania váh*. Ďalšou výhodou je menšia náročnosť architektúry na počet trénovacích parametrov. Naopak konvolučné siete zvyknú mať viac vrstiev. ako klasické siete. V tejto podkapitole si povieme viac aj o výbere architektúry CNN.

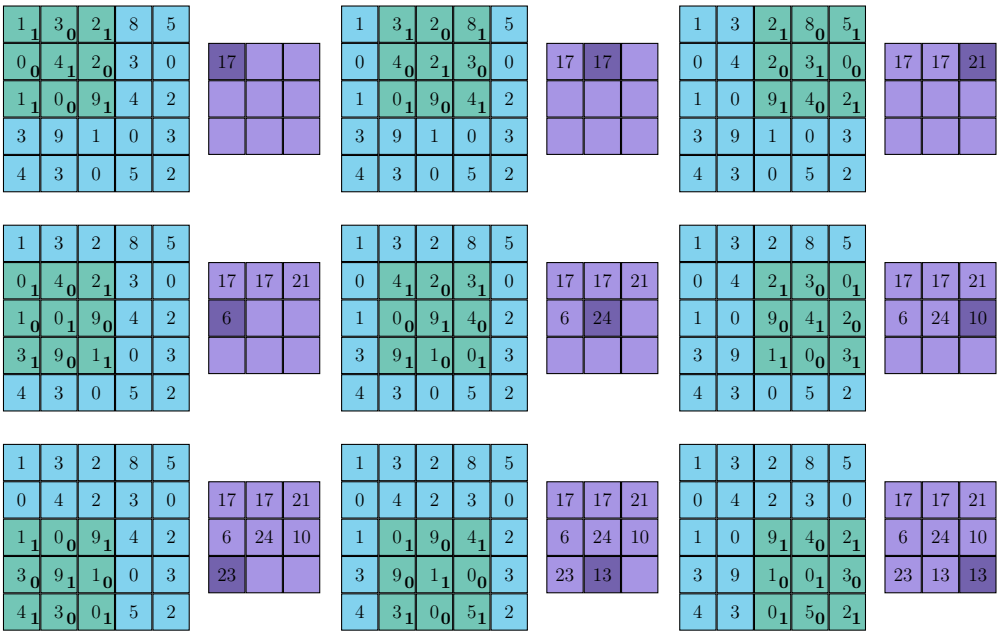
### 1.5.1 Konvolúcia

Prvá otázka, ktorú by sme si mali položiť je, prečo sa tieto siete volajú *konvolučné*. Konvolúcia je matematický koncept, ktorý sa veľmi často využíva v digitálnom spracovaní obrazu.

Majme signály $x$ (*vstupný signál*) a $w$ (tzv. *filter* alebo *kernel*). $x[t]$ a $w[t]$ sú ich zložky na pozícii $t$. Keď implementujeme konvolúciu v počítači, vstup je diskrétny, preto budeme predpokladať, že $t \in \mathbb{Z}$ a zadefinujeme pre nás relevantnú diskrétnu konvolúciu.

**Definice 2** (Diskrétna konvolúcia)**.** *Konvolúcia dvoch diskrétnych signálov $x[t]$ a $w[t]$ je operácia $*$, pre ktorú platí:*

$$(x * w)[t] = \sum_{\tau} x[t-\tau]w[\tau]$$



Obrázek 1.5: Priebeh výpočtu diskrétnej konvolúcie.

Konvolučné siete prijímajú na vstupe obrazové dáta, pričom každý obrázok je dvojrozmerné pole s hodnotami intenzít pixelov v danom bode. Dvojrozmernú konvolúciu potom definujeme:

$$y[s,t] = (x * w)[s,t] = \sum_{\sigma,\tau} x[s-\tau, t-\tau]w[\sigma,\tau],$$

kde výstup $y$ nazývame *mapa príznakov*. Schématické znázornenie diskrétnej konvolúcie je na obrázku 1.5.

Konvolúcia znižuje rozmery obrázka. V konvolučných neurónových sieťach je ale často žiadúce ponechať pôvodnú veľkosť. K tomu sa využíva metóda doplnenia nulami, v angličtine **zero-padding**. Na obrázku 1.6 je znázornená operácia $Pad(\mathbf{X}, n)$, ktorá doplní okraje matice $\mathbf{X}$ o $n$ vrstiev núl. Zero-padding nielen kontroluje veľkosť výstupu konvolúcie, ale zároveň zvyšuje aj presnosť, pretože konvolučný filter tak detekuje vlastnosti aj na hraničných bodoch obrazovej matice.

Obrázok reprezentujeme štruktúrou, ktorú v strojovom učení nazývame **tensor**. Tensor má rozmery $šírka \times výška \times počet\ kanálov$. Kanály sa používajú na prístup k rôznym pohľadom na dáta. Napríklad v prípade farebných obrázkov (RGB) obrázok obsahuje tri separátne farebné kanály, stereo zvuk obsahuje dva zvukové kanály (ľavý a pravý). Obrázok 1.7 ilustruje tensory s jednou, dvomi a tromi dimenziami.

### 1.5.2 Architektúra konvolučných neurónových sietí

Konvolučné neurónové siete využívajú na trénovanie, ako väčšina neurónových sietí, algoritmus spätného učenia. Hlavný rozdiel v porovnaní s klasickými neurónovými sieťami však spočíva v architektúre. Štruktúra CNN obvykle pozostáva z troch rozdielnych typov vrstiev: *konvolučná*, *pooling* alebo *aktivačná*.

| | | | | |
|---|---|---|---|---|
| 1 | 3 | 2 | 8 | 5 |
| 0 | 4 | 2 | 3 | 0 |
| 1 | 0 | 9 | 4 | 2 |
| 3 | 9 | 1 | 0 | 3 |
| 4 | 3 | 0 | 5 | 2 |

(a) Pôvodná matica $\mathbf{X}$.

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 3 | 2 | 8 | 5 | 0 |
| 0 | 0 | 4 | 2 | 3 | 0 | 0 |
| 0 | 1 | 0 | 9 | 4 | 2 | 0 |
| 0 | 3 | 9 | 1 | 0 | 3 | 0 |
| 0 | 4 | 3 | 0 | 5 | 2 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

(b) Nová matica $Pad(\mathbf{X}, 1)$.

Obrázek 1.6: Doplnenie vstupnej matice $5 \times 5$ nulami na veľkosť $7 \times 7$ (*zero-padding*). Na obrázku (a) je pôvodná matica, na ktorú aplikujeme operátor $Pad(\mathbf{X}, \mathbf{n})$, ktorý oblepí maticu $n$ vrstvami núl.

Niekedy sa aktivačná vrstva zahŕňa pod konvolučnú. Okrem týchto základných vrstiev existuje ešte *plne prepojená vrstva* (angl. fully connected layer), umiestená tesne pred výstupnou vrstvou. Výsledkom konvolučných a pooling vrstiev sú vysoko-úrovňové príznaky (v prípade detekcie tváre napríklad oči, nos), ktoré vstupujú do plne prepojenej vrstvy. Tá má za úlohu na základe týchto príznakov zaradiť objekt do tried. Dáta sú najskôr transformované do jednodimenzionálneho vektora, aby mohol byť vypočítaný konečný výsledok. Príklad architektúry konvolučnej neurónovej siete je na obrázku 1.8.

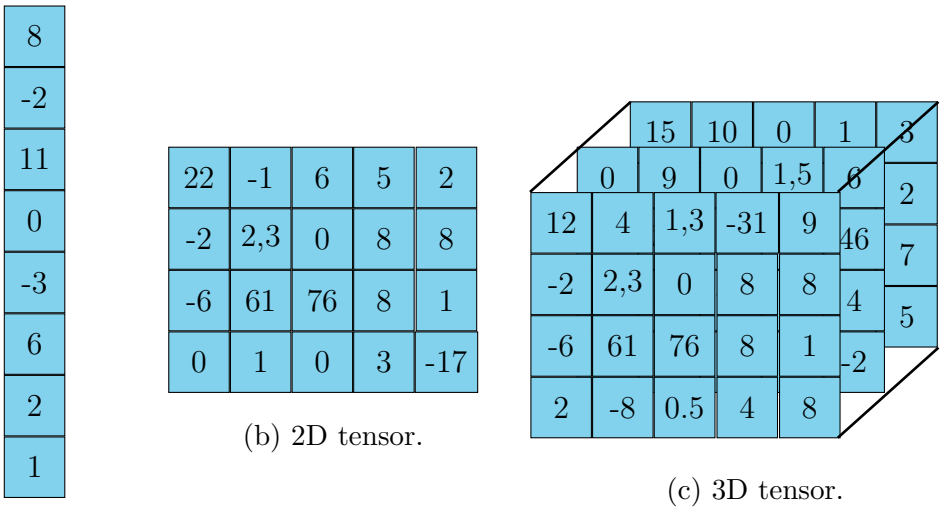
# Konvolučná vrstva

Konvolučná vrstva pozostáva z niekoľkých rôznych filtrov, ktoré majú definovanú veľkosť a príslušnú množinu váh. Tomuto konceptu hovoríme aj **filtrová banka**. Výstupom z tejto vrstvy je tensor príznakových máp. Hlavnou myšlienkou konvolučných neurónových sietí je, že neuróny v konvolučnej vrstve pracujú s malou časťou obrázku, ktorú budeme nazývať **recepčné pole**.

Recepčné pole je neformálne definované ako oblasť vo vstupnom priestore, na ktorom aktuálny filter konvolúcie detekuje charakteristické príznaky (črty) a ktorý ovplyvňuje výslednú príznakovú mapu. Pod príznakom si môžeme predstaviť napríklad roh, hranu alebo kruh. Čím je konvolučná vrstva hlbšie (ďalej od vstupnej vrstvy), tým bývaju tieto črty komplexnejšie. V prvej vrstve filtre väčšinou detekujú nízkoúrovňové príznaky, ako napríklad spomínané hrany.

V klasickej doprednej sieti s plne prepojenými vrstvami existuje spojenie medzi každým neurónom susednej skrytej vrstvy a každým neurónom vstupu. Žiadna priestorová štruktúra sa neberie do úvahy, môže byť iba odvodená učením. V tom prípade hovoríme o *globálnom recepčnom poli*.

Tu prichádza výhoda konvolučných neurónových sietí. V CNN sú spojenia oveľa redšie. Každý neurón skrytej vrstvy je spojený iba s istou oblasťou vstupného obrázka. Túto oblasť voláme *lokálne recepčné pole*. Lokálne recepčné pole je definované svojou stredovou lokáciou a rozmermi. Jeho rozmery korešpondujú s filtrom v konvolučnej vrstve, ktorého váhy sú trénovacie parametre neurónovej siete. Každé lokálne recepčné pole je spojené z jedným skrytým neurónom

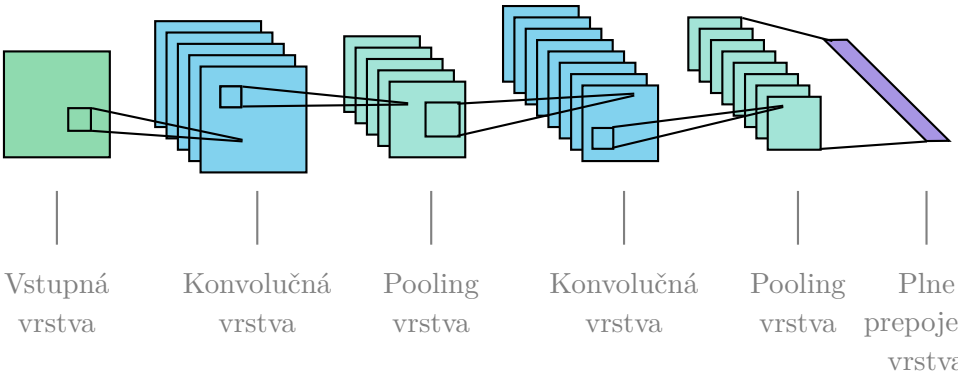



(a) 1D tensor.

(b) 2D tensor.

(c) 3D tensor.

Obrázek 1.7: Tensory reprezentujú $N$-dimenzionálne vektory. Na obrázku (a) je vektor, ktorý reprezentuje tensor [8], (b) je 2D tensor $[4, 5]$, alebo matica 4 riadkov a 5 stĺpcov a (c) je 3D tensor [4, 5, 3] - množina 3 matíc rozmerov $4 \times 5$.



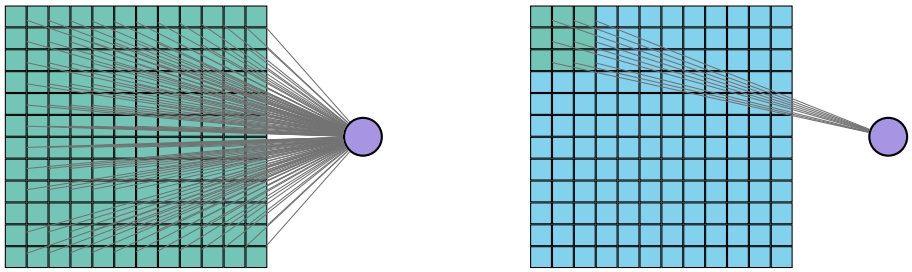

Obrázek 1.8: Príklad konvolučnej neurónovej siete.

v nasledujúcej vrstve. Obrázok 1.9 ilustruje rozdiel medzi globálnym a lokálnym recepčným poľom.

Recepčné pole sa postupne po krokoch posúva cez celý obrázok a detekuje výrazné črty. Parameter, ktorý určuje vzdialenosť medzi dvomi za sebou idúcimi pozíciami stredov konvolučného filtru sa označuje ako **stride**. Pre stride 1 sa počíta konvolúcia pre okolie každého pixelu, pre stride $n$ pre každý $n$-tý pixel. Obr. (1.10) ukazuje, ako ovplyvní výsledok konvolúcie, ak je recepčné pole posúvané po jednom a po dvoch krokoch.

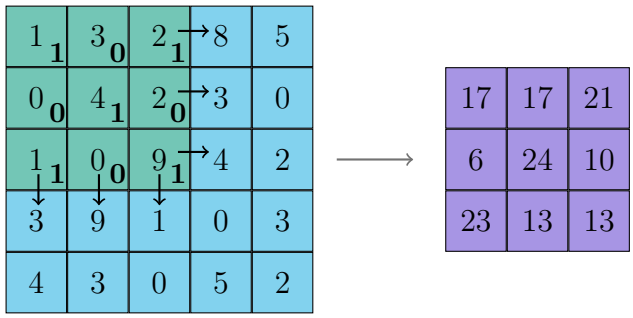
Výstupom jedného lokálneho recepčného poľa je príznaková mapa. Celá procedúra sa opakuje pre každý filter vo vrstve. V prípade, že je na vstupe do konvolučnej vrstvy niekoľko príznakových máp, tak sa filter aplikuje na každú z týchto máp zvlášť a výsledné mapy vyprodukované z totožných filtrov sa sčítajú (viz obrázok 1.11). V prvej konvolučnej vrstve nastane taká situácia v prípade, že má vstupná datová množina viac kanálov.

Všetky neuróny konvolučnej vrstvy zdieľajú tú istú množinu váh daného filtra

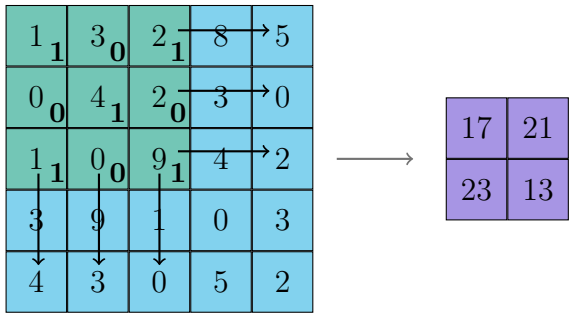


(a) Globálne recepčné pole. (b) Lokálne recepčné pole.

Obrázek 1.9: Príklad globálneho a lokálneho recepčného poľa. Neuróny vstupnej vrstvy prepojené so všetkými neurónmi skrytej vrstvy tvoria globálne recepčné pole (a). V konvolučnej vrstve sú neuróny skrytej vrstvy spojené iba s neurónmi tvoriacimi lokálne recepčné pole (b).




(a) Konvolúcia so stride 1.

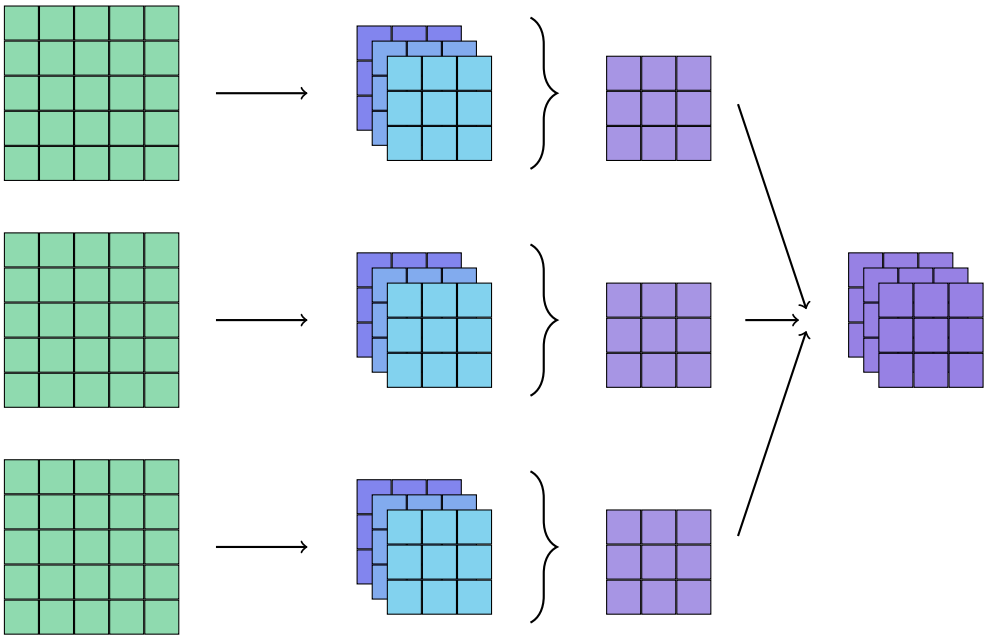



(b) Konvolúcia so stride 2.

Obrázek 1.10: Vplyv hodnoty parametru stride na výstup konvolúcie. Na vstupný obrázok veľkosti $5 \times 5$ sa aplikuje konvolučný filter $3 \times 3$ s hodnotami označenými hrubými číslami v pravej spodnej časti buniek. Šípky označujú najbližšiu pozíciu filtru po posune. (a) pre stride 1 sa pozícia ikrementuje v oboch súradniciach o 1 a výstup konvolúcie má rozmer $3 \times 3$. (b) pre stride 2 sa filter posúva o 2 pozície, výstupný obrázok je zmenšený na $2 \times 2$.

a sú prepojené iba s lokálnou oblasťou obrázka. Preto je počet trénovacích parametrov siete dramaticky zredukovaný v porovnaní s plne prepojenými vrstvami. Zdieľanie parametrov je zároveň dôvod, prečo je CNN odolná voči transformáciám a deformáciam, ktoré sme si spomínali v úvode.

Tým sme si ukázali, že konvolučná vrstva sa líši od plne prepojenej vrstvy dvomi zásadnými vlastnosťami:



Obrázek 1.11: Príklad konvolúcie na viackanálovom vstupe. Vstup každého z troch kanálov je dvojdimenzionálna matica. Konvolučná vrstva obsahuje 3 filtre o rozmeroch $3 \times 3$ pre každý kanál. Po aplikovaní 3 filtrov na prvý kanál vzniknú 3 príznakové mapy, ktoré sa sčítajú do 1 príznakovej mapy. Analogicky sa spracujú zvyšné dva kanály a výstupom je zoskupenie 3 máp. V takejto forme tvoria vstup do ďalšej vrstvy.

1. **Riedke neurónové spojenia**: Nie každý vstupný neurón je spojený s každým výstupným neurónom.
2. **Zdieľanie váh**: Všetky neuróny jednej príznakovej mapy zdieľajú tie isté váhy.

## Aktivačná vrstva

Aktivačná vrstva konvolučnej neurónovej siete pozostáva z aktivačnej funkcie $f$, ktorá aplikuje nelinearitu na výstupné príznakové mapy z konvolučnej vrstvy. Výstup aktivačnej vrstvy voláme *aktivačná mapa*. Najčastejšie používaná aktivačná funkcia v CNN je **ReLU** (Rectified Linear Units), ktorá všetky záporné hodnoty zobrazí do nuly a zhora nie je ohraničená. Obrázok 1.2 ilustruje graf funkcie ReLU a matematicky ju zapíšeme ako:

$$f(x) = max(0, x) \tag{1.13}$$

Pripomeňme si, že aktivačná funkcia sa v spätnej propagácii derivuje. V tom spočíva výhoda ReLU, ktorej hodnota aj derivácia sa spočíta veľmi ľahko. Jediný problém je, že nie je diferencovateľná v nule.

V klasifikačných úlohách, kde je tried viac, sa používa aktivačná funkcia **softmax**. Softmax sa správa ako „zjemnená ReLU", pretože je diferencovateľná

v každom bode. Zobrazuje hodnoty do intervalu $\langle 0, 1 \rangle$.

$$f(x) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_j}} \qquad \forall j \in 1..K \tag{1.14}$$

### Pooling vrstva

Na začiatku kapitoly sme si povedali, že konvolučné neurónové siete sú invariantné voči transláciám a deformáciám. Túto špeciálnu vlastnosť zabezpečuje práve *pooling vrstva*.

Tie oblasti v obrázku, kde boli detekované charakteristické príznaky, sa stávajú menej dôležité pri ďalšej detekcii. Ďalšie konvolučné vrstvy viac zaujíma pozícia relatívna voči ostatným príznakom, aby dokázali rozpoznať iné vysoko-úrovňové príznaky. To navádza k tomu zmenšiť rozmery príznakových máp, čo je hlavným účelom pooling vrstvy. Okrem toho, že tým eliminujeme citlivosť voči skresleniu a posunutiu, má aj iné výhody: počet parametrov, ktoré treba trénovať, je redukovaný, tým pádom je zvýšená výpočtová rýchlosť a menšia pravdepodobnosť preučenia.

Komprimácia príznakových máp prebieha podobne ako konvolúcia výpočtom jednej hodnoty pre každé lokálne recepčné pole. Najčastejší druh poolingu je tzv. *max pooling*, kedy sa z recepčného poľa vyberie maximálna hodnota. Ďalšou alternatívou je napríklad priemerovanie cez recepčné pole.

Je dobré si uvedomiť, že zmenšiť rozmer sa dá aj priamo v konvolučnej vrstve a to zvýšením hodnoty parametru *stride*. Takýmto spôsobom taktiež znížime počet výpočtov a parametrov siete. Preto môže byť niekedy preferovanou voľbou nahradiť pooling vrstvu konvolučnou so zvýšeným stride bez straty na presnosti (Springenberg a kol., 2014).

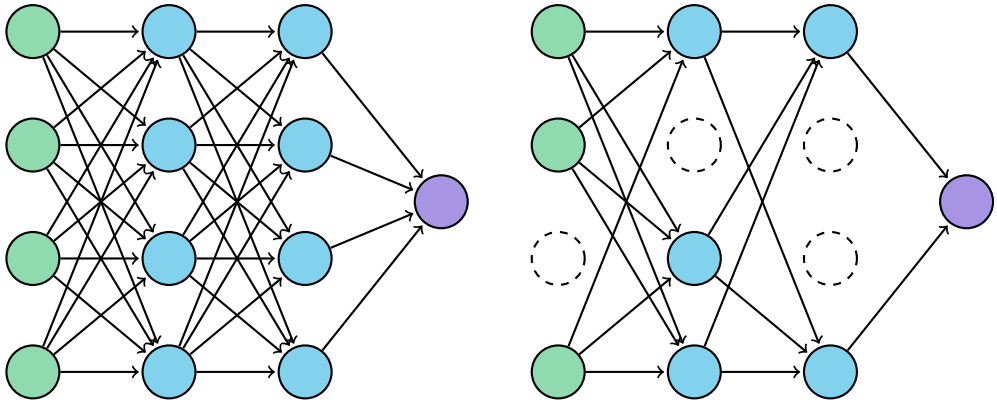
## 1.6 Použité metódy na trénovanie a testovanie

### 1.6.1 Dropout

Dropout je pravdepodobne najčastejšie používaná metóda proti pretrénovaniu siete. Aplikuje sa na plne prepojených vrstvách ako v CNN tak klasických NN a okrem zabraňovaniu preučenia takisto znižuje trénovací čas. Základnou myšlienkou dropoutu je, že každému neurónu sa pridelí istá pravdepodobnosť $p$, s akou bude dočasne v aktuálnej iterácii deaktivovaný. Inak povedané, niektoré neuróny aj so spojeniami budú náhodne „odhodené" zo siete. Trénovanie bude potom ďalej prebiehať klasickým spôsobom, s tým rozdielom, že má sieť v každej iterácii inú architektúru s rozdielnou kombináciou deaktivovaných neurónov. Cieľom je zabrániť prílišnému adaptovaniu neurónov na váhy predošlých vrstiev takým spôsobom, že pri výpočtoch doprednej a spätnej propagácie budú dropout neuróny ignorované. Metóda dropout je ilustrovaná na obrázku 1.12

### 1.6.2 K-Fold Cross Validation

Krížová validácia, častejšie nazývaná v anglickej literatúre *K-fold cross validation*, je postup, ktorý určuje presnosť, s akou klasifikátor natrénovaný na tré-



(a) Plne prepojená neurónová sieť. (b) Po aplikovaní dropoutu.

Obrázek 1.12: Model dropoutu aplikovaný na štandardnú plne prepojenú neurónovú sieť. (a) je plne prepojená neurónová sieť, (b) je tá istá sieť po aplikovaní metódy dropout, kde prerušovaná čiara označuje neuróny, ktoré boli zo siete vyhodené.

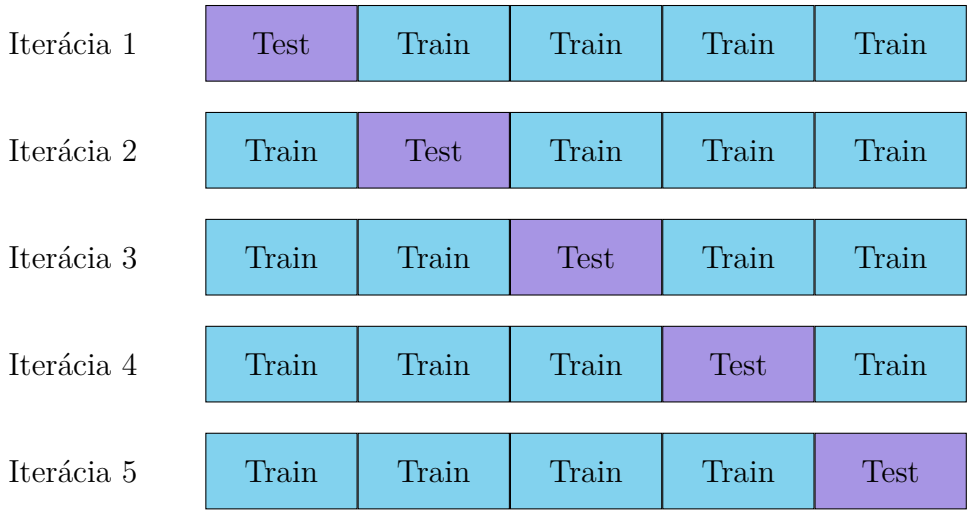
novacej množine pracuje na testovacej množine. Výhodou krížovej validácie je, že všetky dáta sú použité aj na trénovanie, aj na testovanie, a preto sa využíva často v prípade malej trénovacej množiny.

V metóde K-fold cross validation je celý dataset rozdelený na $K$ častí. V každej iterácií sa vyberie $K-1$ podmnožín, ktoré budú spolu použité ako *tréningová množina* a zvyšná 1 časť ako *testovacia množina*. Proces cross validácie bude trvať $K$ iterácií a vždy vynechá inú z $K$ podmnožín. Nakoniec je každá podmnožina použitá práve raz na testovanie a práve $K-1$ krát na trénovanie. Na konci sa spočíta priemer zo všetkých testovacích chýb. $K$ môže nadobúdať ľubovoľnú hodnotu, ale v praxi sa štandardne používa $K = 5$ alebo $K = 10$. 5-násobnú krížovú validáciu vizualizuje obrázok 1.13.

Tento jednoduchý postup garantuje, že medzi $K$ testovacími množinami nie je žiaden prienik a preto nedochádza k negatívnemu skresleniu. Zároveň niekoľko testovacích chýb umožňuje vypočítať priemernú a štandardnú odchýlku a podáva tak lepší obraz o skutočnej presnosti klasifikátora.

### 1.6.3 Shuffling

Neurónová sieť je silným nástrojom práve pre svoju schopnosť nachádzať komplexné vzťahy medzi vstupnými a výstupnými dátami. Problém však nastáva, keď nájde vzťah aj v poradí, v akom sú jej dáta v trénovacom procese predstavované. Najčastejšie sa tomuto problému zabraňuje prinesením randomizácie do poradia vstupu. Náhodné premiešanie tréningovej množiny po každej epoche, alebo *shuffling*, dokáže minimalizovať vplyv poradia dát na výstup.




Obrázek 1.13: Príklad rozdelenia datasetu na trénovaciu a testovaciu množinu pri K-fold cross validation s $K = 5$.

# 2. Dataset

V prechádzajúcej kapitole sme si predstavili neurónové siete z teoretického hľadiska. Predtým, než sa dostaneme k hlavnej experimentálnej časti tejto práce, si najprv popíšeme súbor materiálov a metód nevyhnutných k realizácii tohoto projektu.

Všetky dáta použité v tejto práci sú 3D faciálne modely naskenovaných ľudských tvárí, ktoré nám boli poskytnuté v spolupráci s 3D laboratóriom Katedry antropologie a genetiky člověka na Přírodovědeckej Fakulte Karlovej Univerzity[1].

3D skenery sú zariadenia, ktoré zachytávajú tvary a textúry fyzických objektov tak, že jednotlivé nasnímané body ukladajú do digitálnej podoby do tzv. **mraku bodov**. Mrak bodov je zhluk trojdimenzionálnych bodov v priestore. Nakoniec sa na mrak 3D bodov aplikuje algoritmus na generovanie polygonálnej siete. **Polygónová sieť** pozostáva zo súboru vrcholových bodov n-uholníkov pospájaných hranami do polygónov (obvykle trojuholníkov) tak, že popisujú tvar trojrozmerného objektu.

Nám bola poskytnutá kolekcia vyčistených[2] trojuholníkových sietí vo formáte **OBJ**, ktorý si popíšeme v ďalšej sekcii. Obsahuje 298 naskenovaných tvári zdravých jedincov vo vekovom intervale 14 – 83 rokov. Súčasťou dát je súbor `labels.csv`, v ktorom sú každému modelu priradené negeometrické atribúty, konkrétne:

1. **Pohlavie** - M (muž), F(žena)
2. **Váha** v kilogramoch
3. **Výška** v centimetroch
4. **Vek** v rokoch

Úlohou tejto práce je popísanú množinu 3D faciálnych modelov použitím rôznych prístupov predspracovania mapovať na neurónové siete. Úspešnosť potom vyhodnocujeme na úlohe klasifikácie pohlavia[3]. Príklady niekoľkých naskenovaných tvári z našej datovej množiny môžeme vidieť na obrázku 2.1.

Dôležitou vlastnosťou datasetu je, že je *topologicky unifikovaný*. To znamená, že všetky vzorky majú rovnaký počet a očíslovanie vrcholov (ak vrchol s indexom $i$ označuje špičku nosa v sieti $s \in D$, $D$ je celý dataset, tak index $i$ označuje špičku nosa pre $\forall x \in D$). Každá sieť má 15003 vrcholov a 29713 trojuholníkov.

## Wavefront OBJ (.obj)

OBJ formát je voľne dostupný formát vyvinutý firmou Wavefront Technologies. Je určený na reprezentáciu polygonálnych objektov v ASCII podobe definovaním vrcholov a ich prepojením do stien.

[1] https://www.natur.cuni.cz/biologie/antropologie/pracoviste-katedry/3dlab_old

[2] Trojuholníková sieť je *vyčistená*, ak pre každú hranu a každý vrchol platí, že ohraničuje trojuholník a ak je každý trojuholník súčasťou siete.

[3] Úloha klasifikácie pohlavia bola zvolená pre dobrú dokumentáciu vo fyzickej antropológii a jednoduchému vyhodnocovaniu výsledkov.

Každý neprázdny riadok začína kľúčovým slovom a ďalej nasledujú informácie konkrétne pre dané kľúčové slovo. Na začiatku súboru vo formáte OBJ obvykle býva nepovinný komentár. Riadok obsahujúci komentár sa začína s mriežkou (#). Riadok začínajúci kľučovým slovom $v$ obsahuje súradnice geometrického vrcholu v priestore určené reálnymi číslami.

Okrem súradníc geometrických vrcholov môžu byť v OBJ súbore definované textúrové vrcholy ($vt$), normály vrcholov ($vn$) a vrcholy parametrického priestoru ($vp$).

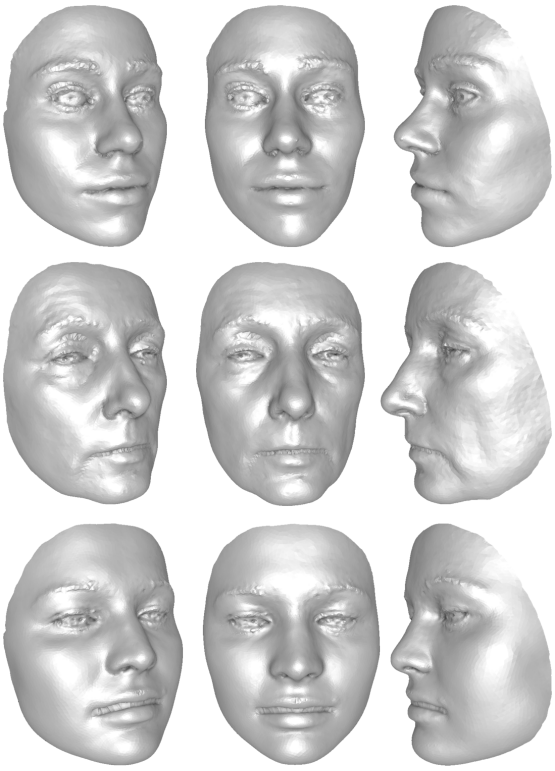
Kľúčové slovo $f$ označuje, že sa jedná o definíciu polygónu a za ním nasledujú celé čísla označujúce index v zozname vrcholov. Vrcholy sa indexujú od čísla 1 a sú uložené v poradí proti smeru hodinových ručičiek. Nasledujúci príklad demonštruje vytvorenie jednoduchého trojuholníka:

```
# subor Wavefront OBJ
v 25.12419 88.99742 4.161045
v 21.3773 89.44572 4.550673
v 17.40979 89.96128 4.78232
f 1 2 3
```

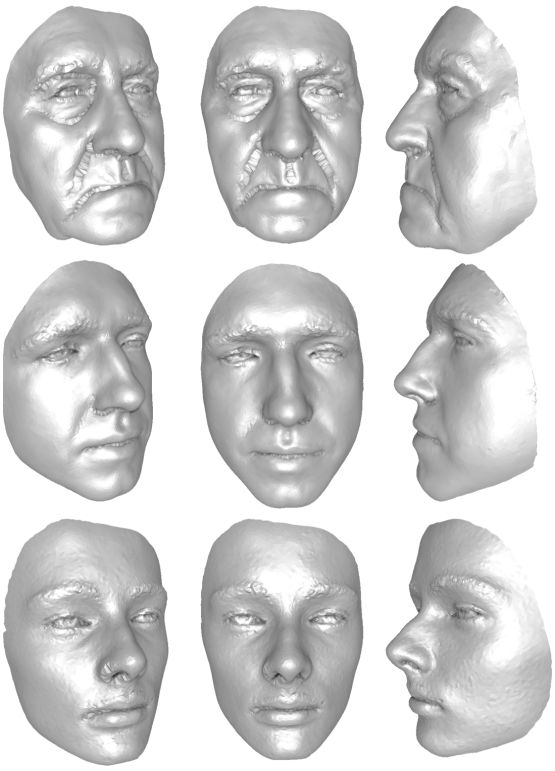
Podrobný popis formátu OBJ je dostupný v dokumente Wavefront's Advanced Visualizer Manual [ 4 ].

## 2.1 Vytvorenie datovej množiny

Datová množina, ktorá nám bola poskytnutá, obsahuje 15003 vrcholov. Pri použití klasickej neurónovej siete (viz 4.2.1) vynásobíme vrcholy počtom dimenzií $15003 \times 3$ a dostaneme 45009 neurónov vo vstupnej vrstve neurónovej siete. Motiváciu zmenšiť vstup sme chceli zlúčiť s cieľom najsť konkrétnu oblasť tváre, na ktorej je potenciálne detekovateľný pohlavný dimorfizmus. Z toho dôvodu sme sa rozhodli pripraviť ešte jednu datovú množinu, ktorá pozostáva iba z vybranej časti každej trojuholníkovej siete v množine. Zvolili sme oblasť nosu, pričom výstupný výrez ilustruje 2.2. Výrez nosu bol použitý pri reprezentácii bez predspracovania a projekcii do mriežky, v ďalšom prístupe s lomenými krivkami sme už operovali nad celým modelom tváre.



(a) Modely tvárí žien.



(b) Modely tvárí mužov.

Obrázek 2.1: Príklady 3D snímkov ľudských tvárí, ktoré boli zhotovené špeciálnym optickým 3D skenerom umiesteným na Katedře antropologie a genetiky člověka UK. Výstupom je model s polygonálnou trojuholníkovou sieťou vo formáte OBJ. Naša datová množina obsahuje: (a) 165 žien, (b) 133 mužov.

(a) (b)

Obrázek 2.2: Trojuholníková sieť vytvorená vystrihnutím oblasti nosu z pôvodného 3D modelu tváre. Obsahuje 3164 vrcholov.

# 3. Softwarové nástroje a hardware

Na to, aby sme mohli navrhnúť čo najlepší model hlbokej siete, je nevyhnutné zvoliť ten správny framework hodiaci sa pre danú úlohu. Preto si v prvej časti tejto kapitoly popíšeme niektoré voľne dostupné nástroje, ktoré v dnešných dňoch získavajú najviac pozornosti.

Ďalšia časť tvorí programátorskú dokumentáciu k pomocným programom, ktoré sme vytvorili pre manipuláciu s trojuholníkovými sieťami, skripty na spracovanie dát do podoby priateľnej CNN a samotné trénovanie neurónových sietí.

V poslednej sekcii tejto kapitoly si preberieme nutné hardwarové požiadavky na úspešný trénovací proces. Neurónové siete si musia pamätať tisícky až milióny trénovacích parametrov. Preto potrebujeme hardware, ktorý dokáže uchovávať veľké množstvo dát a zároveň prevedie výpočty v efektívnom čase.

## 3.1 Framework

S rapídnym nárastom úspechu hlbokého učenia v posledných rokoch prirodzene narástol aj zoznam nástrojov umožňujúcich implementovať jeho modely. Niektoré poskytujú úplne nové prístupy pre jednoduchší a efektívnejší vývoj hlbokých sietí, iné sa vylepšujú v miestach, kde konkurenčné nástroje zaostávajú. Obrovské množstvo nástrojových balíčkov a softwarových knižníc poskytuje rozhranie pre návrh, trénovanie a testovanie hlbokých neurónových sietí v obrovskej škále vysokoúrovňových programovacích jazykov. Tak sa pre používateľa stáva nemalým problémom zistiť, v čom sa rôzne platformy líšia. V skutočnosti má každý nástroj svoje prednosti v rozdielnych fázach návrhu a procesu trénovania neurónových sietí. Takisto každý užívateľ vyžaduje pre návrh svojej siete inú úroveň abstrakcie. Preto je výber frameworku veľmi ovplyvnený aj konkrétne riešeným problémom. Za jeden z najdôležitejších faktorov sa považuje efektívna podpora trénovania siete na GPU.

V tejto podkapitole sa bližšie pozrieme na populárne frameworky Caffe, Torch, Theano, Microsoft Cognitive Toolkit a Tensorflow a porovnáme ich podľa viacerých kritérií (viz. Tabuľka 3.1).

### 3.1.1 Caffe

Caffe *(Convolution Architecture For Feature Extraction)* je jedna z prvých knižníc pre umelú inteligenciu, ktorá bola využívaná nie len výskumnými tímami a vedcami, ale aj širšou mainstreamovou verejnosťou. Bola vyvinutá na univerzite v Berkeley (Jia a kol., 2014). Súčasne je udržiavaná vo vývojovom centre BVLC (Berkeley Vision and Learning Center). Vznikla predovšetkým za účelom implementácie konvolučných neurónových sietí na klasifikáciu obrazu a počítačové videnie. Naopak sa nehodí na trénovanie na dátach ako zvuk, text alebo časové rady. Vysoký level abstrakcie poskytuje API (Application programming interface) v jazyku Python, MATLAB, C++ a CLI nad nízkoúrovňovým back-

endom v efektívnom C++ a CUDA [1].

Veľkým plusom Caffe je, že je stále udržiavaný a doplňovaný o rôzne rozšírenia vďaka aktívnej komunite na *Githube* [1], hlavne *Caffe Model Zoo* [2]. Model Zoo je množina predtrénovaných referenčných modelov pre rôzne úlohy a dáta. Obľúbené architektúry neurónových sietí GoogleNet a AlexNet sú tiež súčasťou ModelZoo. Ďalšou výhodou je veľmi jednoduchá implementácia nových vrstiev a funkcií. Prostredie Caffe sa vydáva pod licenciou BSD2, a preto je voľne dostupné.

### 3.1.2 Torch

Torch (Collobert a kol., 2011) je jedna z najstaršie používaných multiplatformových knižníc určená na strojové učenie, ktorá poskytuje širokú podporu algoritmov pre hlboké neurónové siete. Spomedzi všetkých spomenutých nástrojov sa jedná o framework, s pravdepodobne najjednoduchším použitím. Široké využitie našiel vo výskumných laboratóriách veľkých technologických firiem ako je Facebook Artificial Intelligence Research, Google DeepMind alebo aj Twitter.

Implementácia knižnice Torch je založená na jazyku C/CUDA, ktorá tvorí rýchly a efektívny základ pre výpočty CPU aj GPU (optimalizácia výpočtov pre lineárnu algebru, optimalizačné úlohy, strojové učenie). Uživateľom poskytuje rozhranie skriptovacom jazyku LuaJIT (Lua Just In Time). LuaJIT je jednoduchý, rýchly, ale rozhodne nie je tak rozšírený a známy ako napríklad Python, ktorý tvorí API väčšiny frameworkov popísaných v tejto kapitole.

Svoju popularitu si stále drží vďaka kvalitnej dokumentácii a širokej komunite programátorov, ktorá poskytuje množstvo stiahnuteľných balíčkov rozširujúcich hlavné funkcie knižnice.

[1] CUDA je rozhranie od firmy NVIDIA určené na programovanie grafických kariet

| **Framework** | **Podpora GPU** | **Jazyk API** | **Jazyk back-endu** | **Platforma** |
|---|---|---|---|---|
| Caffe | Áno | Python, MATLAB, C++, CLI | C++, CUDA | Ubuntu, macOS, Windows experimentálne |
| Torch | Áno | Lua | C, CUDA | Linux, macOS, Windows, Android, iOS |
| Theano | Áno | Python | Python | Linux, macOS, Windows |
| CNTK | Áno | Python, C++, C#, CLI | C++ | Windows, Linux |
| Tensorflow | Áno | Python, C++, Java, Go | C++, Python, CUDA | Linux, macOS, Windows, Android, iOS |

Tabulka 3.1: Porovnanie obľúbených frameworkov

### 3.1.3 Theano

Theano (Bergstra a kol., 2010) je knižnica vyvinutá výskumným tímom na fakulte MILA (University of Montreal Institute for Learning Algorithms), na ktorej čele stojí Yoshua Bengio, jeden z priekopníkov moderného hlbokého učenia. Tento framework je voľne dostupný pod licenciou BSD. Jedná sa o všeobecný matematický nástroj na definíciu a vyhodnocovanie matematických výrazov a funkcií v programovacom jazyku Python so zameraním na strojové učenie. Ešte pred samotným výpočtom preloží Theano matematické výrazy do rýchlejšieho C++ (alebo CUDA v prípade GPU). Kvôli tomu sú mnohé algoritmy využívané v strojovom učení rýchlejšie, než u iných zmienených frameworkov. Podobne ako vyššie spomenuté nástroje, aj Theano má veľkú podporu prispievateľov na Githube.

Theano pracuje nad knižnicou Numpy, čo je modul Pythonu pre efektívne operácie s multidimenzionálnymi poľami (tensormi). Spolu v kombinácii so SciPy tak tvorí veľmi silný nástroj v oblasti spracovania obrazu. Rovnaké knižnice využíva aj Tensorflow, ktorý sa považuje za veľkého konkurenta Theana.

Jedným z problémov Theana je, že mnohé algoritmy hlbokého učenia si musí uživateľ implementovať sám, čo sa pokladá za komplikovanú úlohu. Preto sa považuje skôr za výskumnú platformu než knižnicu hlbokého učenia. Navyše podľa najnovších informácií, ktoré uviedla MILA v septembri 2017, po najnovšej verzii Theana sa na jeho vývoji už nebude ďalej pracovať.

### 3.1.4 Microsoft Cognitive Toolkit

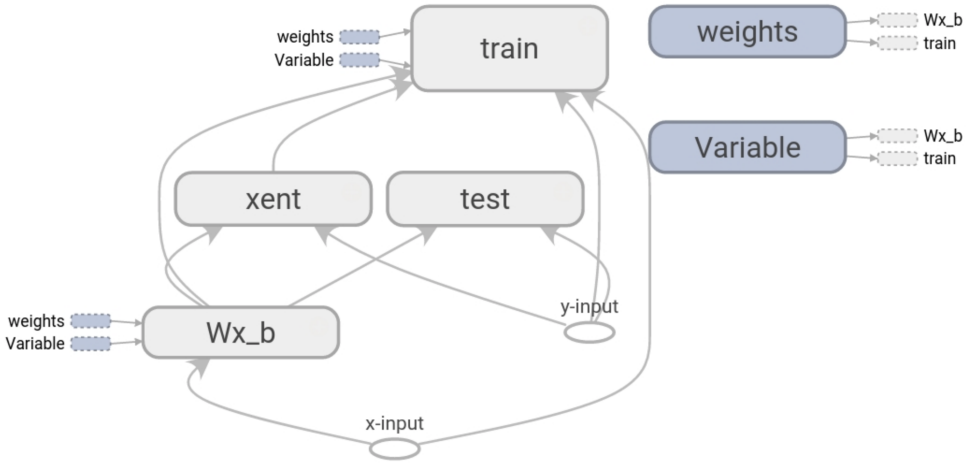
Microsoft Cognitive Toolkit, kedysi známy ako CNTK (Yu a kol., 2014), je open-sourcový sofwarový balíček, ktorý vznikol zjednotením nástrojov od Microsoftu. Bol vydaný pod licenciou MIT na Githube [3]. Jedná sa o obecnú knižnicu, je však zameraná na návrh a trénovanie hlbokých neurónových sietí. Základ je napísaný v jazyku C++ pre platformy Windows, Linux a výhodou je možnosť výpočtu na paralelných grafických kartách.

Neurónovú sieť popisuje ako postupnosť výpočtových krokov, ktoré prevádza sieť v priebehu trénovania. Tieto kroky sú usporiadané do orientovaného grafu. Každý vnútorný uzol reprezentuje maticovú operáciu nad jeho synovskými uzlami a listové uzly reprezentujú vstupnú hodnotu alebo parameter. Hrany medzi uzlami reprezentujú datový tok medzi operáciami. Táto reprezentácia umožňuje uživateľovi ľahko kombinovať uzly do iných obľúbených modelov ako dopredné hlboké neurónové siete, konvolučné neurónové siete a rekurentné neurónové siete.

Microsoft Cognitive Toolkit podporuje API v jazykoch Python a C++. Pôvodne bol vyvinutý na rozpoznávanie reči, v ktorom dosiahol špičkové výsledky, ale svojou podporou CNN a RNN je vhodným kandidátom aj na klasifikáciu obrazu.

### 3.1.5 TensorFlow

TensorFlow (Abadi a kol., 2015) je knižnica na numerické výpočty vyvinutá tímom Google Brain. V roku 2015 bola voľne sprístupnená pre výskumné aj komerčné účely pod licenciou Apache 2.0 ako druhá generácia internej knižnice od Googlu *DistBelief*. DistBelief bola úzko zameraná na hlboké siete, zatiaľ čo




Obrázek 3.1: Príklad jednoduchého datového toku (Rodolfo Bonnin, 2016).

TensorFlow predstavili ako nástroj na široké použitie, ktorý dosahoval v niektorých prípadoch až dvojnásobnej rýchlosti. Aktuálne patrí nepochybne medzi najviac používané frameworky hlbokého učenia.

TensorFlow sa často považuje za veľmi podobný Theano: názov TensorFlow napovedá, že sa knižnica tiež zaoberá efektívnou prácou s tensormi, ktoré „prúdia" grafom od uzla k uzlu. Na prácu s tensormi využíva rovnaké knižnice NumPy a SciPy.

Ako bolo spomenuté, všetky výpočty sú reprezentované orientovanými grafmi, kde uzly označujú matematické operácie, zatiaľ čo po hranách grafu prúdia tensory (viz Obrázok 3.1). Vďaka tejto reprezentácii využívaefektivitu GPU a okrem toho podporuje aj napríklad Android a iOS. Výhodou je flexibilná architektúra umožňujúca paralelné využitie CPU alebo GPU a jednoduchá prenositeľnosť na iný hardware bez zmeny kódu. TensorFlow je implementovaný vo vysoko optimalizovanom C++ v kombinácii s CUDA. Hlavný a prvý jazyk rozhrania s klientskou podporou je Python. Sú dostupné aj experimentálne rozhrania v C++, Jave a Go, komunita na Github dokonca vytvorila aj rozhranie v C#, Ruby, Haskell, Julia, Rust a Scala.

K popularite Tensorflow prispieva aj modul *TensorBoard* slúžiaci na vizualizáciu výpočtového grafu a prúdenia dát v grafe.

## 3.2 Výber frameworku

Pre túto bakalársku prácu sme sa celkom jednoznačne rozhodli zvoliť voľne dostupný framework TensorFlow od firmy Google. Bolo potrebné zvážiť niekoľko kritérií, ako sú napríklad dostupné jazyky rozhrania, rýchlosť alebo prenositeľnosť na úrovni hardware.

V prvom rade sme chceli pracovať v programovacom jazyku, ktorý je dobre známy a často používaný, ale zároveň sme jazyk nepovažovali za rozhodujúci faktor. A síce Python nebol spočiatku náš preferovaný jazyk, nakoniec po krátkom prieskume sa svojou jednoduchosťou a faktom, že je to zároveň skriptovací jazyk, ukázal ako vhodná voľba.

Jedným z najdôležitejších faktorov pre nás bola aktívna komunita vývojárov a uživateľov, kvalita dokumentácie a dostupnosť materiálov vo forme tutoriálov. V porovnaní s inými voľne dostupnými knižnicami na hlboké učenie sa TensorFlow zdal byť najlepšie zdokumentovaným frameworkom.

Keďže sa jedná skôr o experimentálnu prácu, tak rýchlosť bola až druhoradým, no nemenej dôležitým kritériom. Tensorflow podporuje paralelné výpočty na GPU narozdiel od jemu konkurujúceho Theano. Taktiež sme požadovali framework fungujúci pre Windows aj MacOS, čo TensorFlow spĺňa.

Nástroj na vizualizáciu TensorBoard sa ukázal byť iba ďalším plusom, pretože uľahčuje ako porozumenie datovým tokom v sieti, tak aj rýchlejšie nachádzanie chýb pri debugovaní (ako napríklad neprepojené tensory).

## 3.3 Pomocné programy

V tejto sekcii si popíšeme implementáciu programov, ktoré sme si vytvorili za účelom predspracovania surových dát do neurónovej siete. Ďalej priblížime štruktúru skriptov slúžiacich na samotné trénovanie klasických a konvolučných neurónových sietí. Podrobný návod, ako spúšťať tieto nástroje a ako postupovať pri práci s nimi, je uvedený v softwarovom manuáli v prílohe A.

### 3.3.1 MeshEditor

Na extrakciu podmnožiny meshe sme vytvorili pomocný program *MeshEditor*. MeshEditor je napísaný v programovacom jazyku C# a je založený na projekte 068shader z repository pre predmet Počítačová grafika I (NPGR003[2]), vytvorený vyučujúcim a vedúcim tejto práce RNDr. Josefom Pelikánom. Pôvodný projekt obsahuje základné nástroje pre zobrazovanie súborov vo formáte OBJ, ako otáčanie a približovanie 3D modelov, zobrazenie súradnicových osí a rôzne režimy rendrovania.

My sme do programu doplnili funkcie predovšetkým na uľahčenie práce pri orezávaní trojuholníkových sietí a upravili sme uživateľské rozhranie pre pohodlnejšie ovládanie. Teraz si popíšeme, o aké funkcie sme rozšírili pôvodný program:

1. Možnosť importovať a exportovať súbory vo formáte OBJ a PLY.
2. Označovanie množiny vrcholov a trojuholníkov
3. Exportovať označené podmnožiny trojuholníkovej siete do nového súboru.
4. Exportovať označené podmnožiny trojuholníkovej siete pre viaceré topologicky unifikované modely
5. Návrat o niekoľko editačných krokov späť pomocou funkcie *Undo*
6. Záznam označených indexov vrcholov do textového súboru.

[2] http://cgg.mff.cuni.cz/lectures/npgr003.cz.php

### Označovanie časti meshe

Všetky z troch nástrojov na označovanie vrcholov Point selection, Brush selection a Triangle selection ovláda uživateľ jednoduchým kliknutím na obrazovku. Keďže obrazovka je 2D a mesh je 3D, musia sa aj súradnice kliknutého bodu previesť na 3D, tzv. *world coordintes*. Ďalším postupom je nájsť trojuholník patrici kliknutému vrcholu. Metódy, ktoré to umožňujú, boli súčasťou pôvodného programu. Ostatná práca so získanými vrcholmi je zabezpečená nami vytvorenou triedou **Selection** a pomocnou triedou na výpočty **SelectionHelper**.

Trieda Selection obsahuje verejnú datovú položku **List<int> SelectionPointers**, do ktorej sú priebežne ukladané a odoberané indexy vrcholov, ktoré sú označené.

Dôležitá je sekcia Selection handlers, ktorá obsahuje metódy, ktoré tento zoznam vypĺňajú podľa toho, aký nástroj bol zvolený:

- **SelectTriangle(index :** int**)**
- **SelectPoint(uv :** int**)**
- **SelectPoint(uv :** Vector2d, **index :** int**)**

Algoritmus na zistenie, ktorý z vrcholov trojuholníka bol zvolený, je jednoduchý a využíva už predom zistené barycentrické súradnice kliknutého bodu na obrazovke voči trojuholníku - zistí, ktorý vrchol má od neho najmenšiu Euklidovskú vzdialenosť.

Samostatnou kategóriou je sekcia Brush selection. Tá iteratívne prechádza všetky susedné body od daného stredu a označuje vlnou všetky body od neho vzdialené do určenej vzdialenosti.

## Undo a Redo

Funkcie Undo a Redo fungujú na princípe návrhového vzoru Command. Je vytvorená abstraktná trieda **Command** s abstraktnou metódou **Execute()**, ktorú dedí abstraktná trieda **UndoableCommand** ktorá jej dodáva rozhranie metódy **Undo()**.

Pre jednotlivé operácie spúšťajúce selekciu vrcholov sú vytvorené samostatné triedy **PointSelectionCommand**, **TriangleSelectionCommand** a **BrushSelectionCommand**, ktoré dedia rozhranie UndoableCommand. Pri každom volaní **Execute()** si najprv uložia aktuálny stav označených vrcholov a potom prevedú označenie nových vrcholov. Každá command sa uloží na zásobník tak, že pri volaní **Undo()** sa vyberie posledne prevedený príkaz a inštancii **Selection** vrátia uložený stav predošlých označených bodov. **Redo()** má taktiež vlastný zásobník, kde si ukladá commands, na ktorých bolo zavolané Undo().

## Export

Export meshe prebieha podľa zvoleného formátu v triede **StanfordPly** alebo **WavefrontObj**. O zavolanie správnej metódy v týchto triedach sa stará delegát **WriteBrepHandler**. Tam prebehne zápis do vybraného súboru. Na ex-

port z množiny unifikovaných modelov je vytvorená samostatná trieda **ExportBatch**. Keďže sa jedná o časovo náročnú operáciu, ktoré by viedli k zmrazeniu uživateľského okna, sú naprogramované asynchrónne. Inštancia triedy **Progress** umožňuje výpis progresu asynchrónnej metódy do formulára a tým informuje používateľa, aký súbor sa práve exportuje. Inštancia triedy **CancellationToken** zaznamenáva v hlavnom vlákne žiadosť o ukončenie práce, ktorú odošle vláknam z Task Factory pracujúcim na zápise.

### 3.3.2 3DTo2DProjector

Pre potreby mapovania trojdimenzionálnych trojuholníkových sietí do dvojdimenzionálnej mriežky sme vytvorili konzolovú aplikáciu *3DTo2DProjector*. Prioritným cieľom implementovaného algoritmu bolo vytvoriť také uloženie, aby vrcholy, ktoré sú v trojuholníkovej sieti prepojené hranou, ležali „blízko" pri sebe aj vo výslednej projekcii. Výsledná projekcia je matica, ktorej jednotlivé položky obsahujú prirodzené čísla reprezentujúce index vrcholu (číslovaný od 1) alebo 0 v prípade, že je políčko prázdne. Program 3DTo2DProjector je napísaný v jazyku Java 9.

#### Architektúra programu

Program sa skladá z dvoch hlavných častí. Prvou je trieda **Parser**, ktorá číta a spracúva súbory vo formáte OBJ obsahujúce trojuholníkovú sieť. Zisťuje informácie o susednosti jednotlivých vrcholov a ich počte. Druhú hlavnú časť tvorí trieda **Grid**, ktorá ukladá všetky vrcholy meshe do obdĺžnikovej mriežky.

#### Inicializácia programu

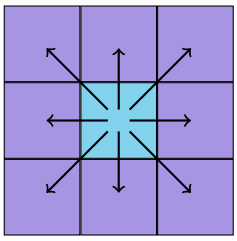
Program očakáva na príkazovej riadke okrem voliteľných options cestu k súboru vo formáte OBJ. Po spustení programu predá tento súbor triede `Parser`, ktorá ho prečíta a spracuje. Pre každý vrchol siete vytvorí inštanciu triedy `Vertex`. Vertex obsahuje nasledujúce datové položky:

1. Index v poli vrcholov (číslovaný od 0)
2. Unikátny celočíselný identifikátor (číslovaný od 1)
3. Zoznam vrcholov, ktoré sú ním hranou prepojené v trojuholníkovej sieti
4. Bunku v mriežke, kde bol uložený (ak ešte uložený nebol, tak neobsahuje žiadnu bunku)

Zoznam s inštanciami Vertex predá triede `Grid`, ktorá má na starosti nájsť pre každý vrchol takú pozíciu v mriežke, aby susedila s čo najväčším počtom vrcholov, ktoré sú s ním hranou prepojené v trojuholníkovej sieti.

Základom triedy `Grid` je matica (mriežka), ktorej každá bunka obsahuje inštanciu triedy `Cell`. `Cell` obsahuje nasledujúce datové položky:

1. Súradnicu $x$ označujúcu pozíciu v mriežke
2. Súradnicu $y$ označujúcu pozíciu v mriežke



Obrázek 3.2: 8-okolie pixelu tvoria pixely, ktoré sú spojené hranami a rohami.

3. Inštanciu triedy `Vertex`, teda vrchol, ktorý je v bunke uložený (ak neobsahuje žiadny vrchol, tak je táto položka prázdna)

Po inicializovaní mriežky prázdnymi bunkami sa ešte pripraví pomocná mapa `occupiedNeighbouringCellsCounter`, ktorá každému vrcholu priradí počet vrcholov, pre ktoré platí:

- sú s ním hranou prepojené v trojuholníkovej sieti
- sú už umiestnené v mriežke

V procese inicializácie má každý vrchol priradený počet 0.

### Susednosť buniek v mriežke

Pojem susednosť definujeme v mriežke ako 8-okolie[3]: 8-okolie bunky je množina takých buniek, ktoré sa dotýkajú jednej z jeho hrán alebo rohov. Tieto bunky sú spojené horizontálne, vertikálne a diagonálne (viz obrázok 3.2).

### Algoritmus ukladania vrcholov

Budovanie mriežky je realizované iteratívne v dvoch hlavných krokoch, a to *výber ďalšieho vrcholu na uloženie* a *nájdenie vhodnej pozície* v mriežke až kým nie je zoznam `occupiedNeighbouringCellsCounter` prázdny.

(a) **Výber ďalšieho vrcholu na uloženie** prebieha na základe jedného kritéria. Vyberie sa vrchol $v$ s najvyšším počtom susediacich vrcholov v zozname `occupiedNeighbouringCellsCounter`, ktoré sú už uložené v mriežke.

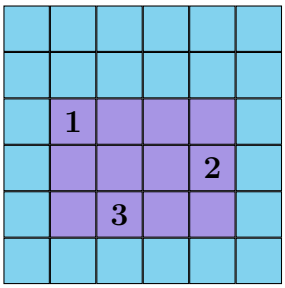
(b) **Nájdenie vhodnej pozície** využíva práve túto vedomosť o uložených susediacich vrcholoch.

   (1) Okolo susedných vrcholov vrcholu $v$, ktoré už sú uložené v mriežke, sa vytvorí minimálny ohraničujúci obdĺžnik (bounding box), ako ilustruje obrázok 3.3.

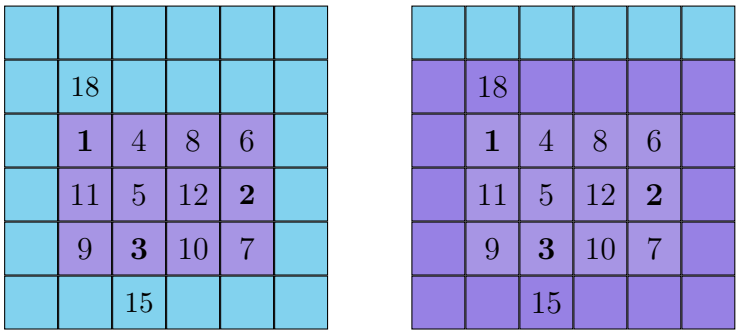
   (2) Nájde sa zoznam voľných políčok v ohraničujúcom obdĺžniku.

   (3) Ak je zoznam prázdny, zväčší sa obdĺžnik o 1 bunku z každej strany a vráti sa na (2). Túto situáciu demonštruje obrázok 3.4).

[3] https://en.wikipedia.org/wiki/Pixel_connectivity#8-connected




Obrázek 3.3: Hľadanie bunky mriežky, do ktorej sa uloží vrchol $v$. Vrcholy $1, 2, 3$ susedia s $v$ v trojuholníkovej sieti. Najprv sa vytvorí *ohraničujúci obdĺžnik* okolo vrcholov, ktoré 1) susedia s $v$ v trojuholníkovej sieti, 2) sú už uložené v mriežke. Pri výbere bunky ďalej zohľadňuje už iba voľné políčka ohraničujúceho obdĺžnika.




Obrázek 3.4: Ohraničujúci obdĺžnik vrcholov $1, 2, 3$ neobsahuje žiadne voľné políčka. Preto sa obdĺžnik rozšíri o 1 z každej strany a voľné políčka sa hľadajú v novom obdĺžniku.

(4) Trieda `Score` priradí každej voľnej bunke skóre podľa toho, aká je jeho vzdialenosť od buniek všetkých susedných vrcholov. Vzdialenosť $d$ medzi dvomi bunkami $a$ a $b$ sa vypočíta vzťahom:

$$d = \max(|a.X - b.X|, |a.Y - b.Y|)$$

(5) Vrchol sa uloží do bunky $c$, ktorá je ohodnotená najnižším skóre

(6) Bunka $c$ sa vymaže zo všetkých zoznamov voľných susedných buniek. Aktualizuje sa zoznam `occupiedNeighbouringCellsCounter` tak, že každý vrchol, ktorý susedí s $v$, zvýši počítadlo o 1 a odstráni sa z neho záznam o $v$ (už v algoritme v časti (a) nebude uvažovaný)

Prvý vrchol, ktorý bude uložený, sa zvolí ako medián indexov z poľa vrcholov a uloží sa do stredu mriežky. Ďalej už algoritmus pokračuje tak, ako sme popísali vyššie.

### 3.3.3 Trénovacie skripty

Za experimentálnou časťou tejto práce stoja skripty v jazyku Python. Nachádzajú sa na priloženom CD v adresári *trainingScripts*, ktorý obsahuje pre každú

datovú reprezentáciu samostatné adresáre: *1dArray*, *gridProjection* a *obliqueLines*. Skripty s rovnakým názvom v odlišných priečinkoch sa drobne líšia v implementácii vzhľadom na potreby danej reprezentácie, ale rozhranie a funkčnosť základných metód sa prevažne zhoduje:

## config

Súbory `config.py` sa líšia v prípadoch, či sa jedná o klasickú alebo konvolučnú neurónovú sieť. Trénovacie parametre sa predajú ako argumenty konštruktoru triedy Configuration. Parametre, ktoré sme v trénovacom procese nemenili, sú repezentované ako konštanty na začiatku skriptu.

## nn1dArray, cnnGridProjection, cnnObliqueLines

Skripty s týmito názvami sú hlavné skripty, ktoré spúšťajú celý proces načítania vstupných súborov, trénovania a testovania. Operujú nad aktuálnymi nastaveniami v súbore `config.py`. Základné metódy, ktoré obsahujú, sú nasledujúce:

- **run()** je metóda, ktorá sa zavolá hneď po spustení programu. Má na starosti volanie funkcií z `parse.py`, ktoré spracujú vstupné súbory. Vytvorí výstupný súbor, do ktorého sa uložia výsledky a následne spustí cross-validáciu.
- **cross_validation(data_set, configuration, result_writer)**:
  - **data_set**: pole s anotovanou datovou množinou
  - **configuration**: inštancia triedy Configuration obsahujúca parametre neurónovej siete
  - **result_writer**: výstupný súbor

  Spúšťa opakovane trénovanie neurónovej siete a testovania pomocou K-fold cross validácie. Výsledky si priebežne ukladá a nakoniec zapíše priemernú úspešnosť do výstupneho súboru.

## parse

`parse.py` je program, ktorý slúži na prácu so vstupnými datovými súbormi. Číta a spracováva ich do polí a následne vie s týmito poľami pracovať a transformovať ich do rôznych foriem. Tvoria ho štyri metódy:

- **create_feature_label_set(features_file, labels_file)**:
  - **features_file** obsahuje cestu k textovému súboru s datasetom
  - **labels_file** obsahuje cestu k súboru vo formáte CSV, ktorý obsahuje anotáciu datasetu

  Táto funkcia v prvej časti číta súbor `labels_file`, pričom jeden riadok v súbore obsahuje označenia jedného príkladu z datovej množiny. Z každého riadku vyberie požadovaný stĺpec, a jeho informáciu zakóduje do *one-hot* reprezentácie do poľa `label_set`. My sme navrhovali klasifikátor pohlavia,

preto sme vyberali informáciu z druhého stĺpca v súbore `labels.csv` (M - male, F - female). Zmenením indexu stĺpca `column` sa dá táto časť kódu pre budúce použitie ľahko modifikovať:

```
# index stlpca
column = 1

if line[column] == 'M':
label = [0, 1]
elif line[column] == 'F':
label = [1, 0]
```

V druhej časti číta súbor s datasetom a ukladá súradnice vrcholov do zoznamu. Každý príklad datovej množiny je v samostatnom zozname a všetky príklady sú potom uložené za sebou do jedného vnoreného zoznamu `feature_set`.

Funkcia vracia na výstupe zoznamy `feature_set` a `label_set`.

- **create_data_set(feature_set, label_set)**:
  - **feature_set** zoznam s datovou množinou vytvorený funkciou `create_feature_label_set`
  - **label_set** zoznam s anotáciou datovej množiny vytvorený funkciou `create_feature_label_set`

  Táto funkcia spojí 2 vstupné zoznamy do jedného zoznamu `data_set` tak, že na koniec každého poľa so súradnicami jedného príkladu datovej množiny pripojí jeho označenie $[0, 1]$ v prípade muža alebo $[1, 0]$ v prípade ženy. Takáto reprezentácia sa hodí pre udržanie správneho priradenia príkladu s príslušným označením pri premiešavaní dát a náhodnom výbere dát do $K$ podmnožín v *K-fold cross validácii*.

- **feature_label_from_dataset(data_set)**:
  - **data_set** zoznam s anotovanou datovou množinou vytvorený funkciou `create_data_set`

  Vezme `data_set` a rozdelí ho na `feature_set` a `label_set`.

- **shuffle_data(feature_set, label_set)**:
  - **feature_set** zoznam s datovou množinou vytvorený funkciou `create_feature_label_set`
  - **label_set** zoznam s anotáciou datovej množiny vytvorený funkciou `create_feature_label_set`

  Náhodne premieša prvky v datovej množine.

## k_fold

Program `k_fold.py` obsahuje jedinú metódu **create_k_folds(data_set, k)**, ktorá rozdelí dataset do $k$ náhodných disjunktných podmnožín. Každá z podmnožín je potom využitá práve raz ako testovacia množina v K-fold cross validácii.

### train

`train.py` je skript, v ktorom prebieha vytvorenie modelu neurónovej siete so špecifikovanými parametrami a následne trénovanie. Obsahuje dve hlavné metódy:

- **create_model(configuration)**:
  - **configuration**: inštancia triedy Configuration obsahujúca požadované parametre neurónovej siete

  Vytvorí model neurónovej siete.

- **train_neural_network(train, train_labels, test, test_labels, configuration, shuffle)**:
  - **train**: trénovacia množina
  - **train_labels**: označenie trénovacej množiny
  - **test**: testovacia množina
  - **test_labels**: označenie testovacej množiny
  - **configuration**: inštancia triedy Configuration obsahujúca požadované parametre neurónovej siete
  - **shuffle**: boolovská premenná

  V tejto funkcii prebieha trénovanie neurónovej siete so zadaným počtom epoch a ako výstup vráti úspešnosť siete na trénovacej množine. Ak shuffle=True, tak je trénovacia množina v každej epoche náhodne premiešaná. Ak shuffle=False, tak sú dáta v trénovacej množine predstavované neurónovej sieti v každej epoche v rovnakom poradí.

## 3.4 Hardware

Výpočty pri učení hlbokých neurónových sietí sú extrémne náročné na hardware po časovej aj pamäťovej stránke. V jednom trénovacom cykle obvykle prebehnú tisíce násobení matíc a všetky vypočítané trénovacie váhy musia byť uložené do pamäti. Osobné počítače často nemajú dostatok operačnej pamäti a moderné CPU nie sú určené na tak náročné výpočty.

Navyše, nami vybraný framework TensorFlow podporuje výpočty algoritmov na grafickej karte, čo môže viesť aj k niekoľkonásobnému zrýchleniu, preto je dobré jej potenciál efektívne využiť. GPU je špecializovaný procesor na grafickej karte, ktorý slúži na spracovania a vykresľovanie obrazových dát. Ako už vieme, obrazové dáta sú vlastne matice a všetky operácie s obrázkami (ako napríklad rotácia, zoomovanie) sú implementované transformáciami matíc. Teda je celkom prirodzené, že sa pre výkonné grafické procesory našlo aj iné využitie než zobrazovanie grafických informácií. Algoritmy hlbokého učenia využívajú práve podobné maticové operácie. TensorFlow je konkrétne kompatibilný s NVIDIA GPU.

Implementácia návrhu neurónovej siete, krátke testy, ladenie konfigurácie, ale tiež pomocné skripty a programy boli prevedené na notebooku MacBook Pro

| | |
|---|---|
| **CPU** | 2x 8-core Intel Xeon E5-2650v2 2.60GHz |
| **RAM** | 64GB |
| **Disk** | 2x 1TB 10k SATA III, 2x480GB SSD |
| **Net** | Ethernet 1Gb/s, Infiniband QDR |
| **GPU** | 2x NVIDIA Tesla K20 5GB (aka Kepler) |
| **Vlastník** | CESNET |

Tabulka 3.2: Cluster strojov doom.metacentrum.cz (Brno) obsahuje 29 uzlov a 464 CPU, pričom každý uzol má hardwarovú špecifikáciu popísanú v tabuľke.

(2016, 8GB RAM), macOS High Sierra s procesorom Intel Core i5 2,9 GHz a integrovanou grafikou Intel Iris Plus Graphics 550.

Pre náročné výpočty sme využili služby MetaCentra [4], ktorý disponuje našimi požiadavkami na hardware. MetaCentrum je virtuálna organizácia českej Národnej Gridovej Iniciatívy MetaCentrum NGI, ktorá poskytuje výpočtové servery a datové úložisko pre všetkých akademických pracovníkov a študentov inštitúcií, ktoré sú členmi združenia CESNET. Použitím grafickej karty NVIDIA K20 s podporou CUDA a cuDNN sme mohli dosiahnuť lepšie časy učenia. V porovnaní s využitím iba klasického CPU sa dalo učenie urýchliť z jednotiek dní na hodiny. Špecifiká hardwaru poskytnuté MetaCentrom sú uvedené v tabuľke 3.2.

[4] https://metavo.metacentrum.cz

# 4. Reprezentácia 3D dát na trénovanie

Klasifikácia 2D obrazu pomocou konvolučných sietí sa dá dnes bezpochyby označiť za štandardnú úlohu hlbokého učenia dosahujúcu špičkové výsledky. Avšak ani najpokročilejší nástroj alebo architektúra siete zatiaľ nestačí, ak je datová množina nevhodne zvolená a nedostatočne popisuje charakteristické vlastnosti vzorov.

S pokrokom v 3D technológiách sa klasický obrázok rozrástol o informáciu o hĺbke. To so sebou prinieslo množstvo otázok ohľadom toho, ako čo najlepšie mapovať robustné 3D dáta do formy na vstup neurónovej siete. Odpoveď nie je tak jednoznačná, ako je to v prípade dvojdimenzionálneho priestoru s obrázkami. Avšak oblasti výskumu, ako napríklad medicína, robotika, či autonómne riadenie, si kladú vysoké nároky na kvalitu rozpoznávania 3D obrazu a preto je vhodná reprezentácia rozhodujúca.

V tejto kapitole prevedieme prieskum najčastejších reprezentácií 3D dát v existujúcich súvisiacich prácach. Rozdelíme si ich do skupín na také, ktoré zachovávajú trojdimenzionalitu (*volumetrická reprezentácia*) a tie, ktoré reprezentujú 3D objekt ako množinu 2D obrázkov (*pixelová reprezentácia*). Následne predstavíme riešenie tohoto problému nami navrhnutými reprezentáciami.

## 4.1 Súvisiace práce

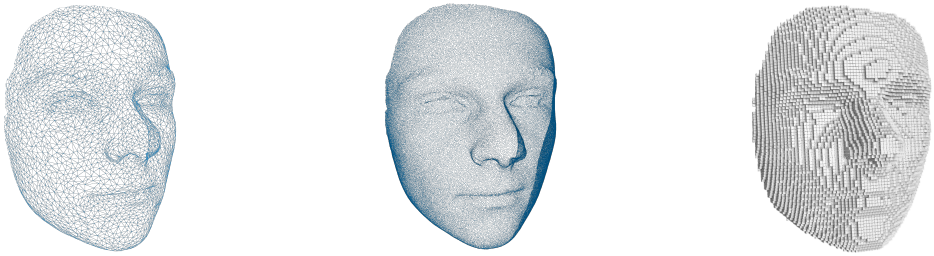
Výskum sa už dlho zaoberá otázkou, ako predspracovať 3D dáta, aby boli vo forme vhodnej pre hlboké učenie. V závislosti na dátovej reprezentácii popisujúcej 3D modely sme identifikovali dve hlavné triedy líšiace sa tým, či pracujú na natívnej 3D dimenzii, alebo mapujú problém na 2D CNN.

### 4.1.1 Volumetrická reprezentácia

Volumetrická reprezentácia umožňuje vykonávať rozpoznávanie obrazu priamo na 3D modeli, ktorý zachováva informáciu o geometrii tváre.

Výhoda trénovania na 3D dátach je napríklad tá, že nie je potrebné sieti poskytnúť pohľad na objekty z rôznych perspektív, keďže priestorová informácia je obsiahnutá v tretej dimenzii, dôsledkom čoho jej postačuje menej dát na trénovanie. Naopak nevýhodou je, že explicitná informácia o hĺbke má svoju cenu, ktorou je neprekvapivo nižšia výpočetná rýchlosť a vyššia pamäťová náročnosť.

Najčastejšie používanou reprezentácia 3D objektov je jednoduchá **trojuholníková sieť**, ktorú sme si definovali v kapitole 2. Ďalšou možnosťou je nereprezentovať objekt pomocou povrchov, ale pomocou volumetrických primitív. Jednou takou reprezentáciou je tzv. **mrak bodov**, ktorý je viac známy anglickým termínom *point cloud*. Je to množina trojdimenzionálnych bodov, ktoré modelujú povrch snímaného objektu v priestore. Každý bod môže okrem súradníc $XYZ$ obsahovať napríklad atribút farby ($RGB$), priehľadnosť, lom svetla alebo materiál. Problémom je, že v prípade point-cloudov nie je jasné, ako na nich aplikovať



(a) Trojuholníková sieť. (b) Point cloud. (c) Mriežka voxelov.

Obrázek 4.1: Rôzne reprezentácie 3D modelov, ktoré môžu byť použité na trénovanie 3D CNN. Väčšina existujúcich prístupov volí (c) mriežku voxelov, pretože je organizovaná do štruktúry kvádru, ktorá sa mapuje na vstup siete jednoduchšie, ako (a) trojuholníková sieť alebo (b) point cloud, ktoré nemajú pravidelnú štruktúru.

nástroje hlbokého učenia. Klasické konvolučné neurónové siete vychádzajú z výhodnej pravidelnej štruktúry obrázkov, kde sú priestorové vzťahy priamo zakódované usporiadaním pixelov v matici. Bodom v point cloude aj trojuholníkovej sieti pravidelná štruktúra chýba. Prácou PointNet Qi a kol. (2016a) bola vyvinutá snaha prekonať tento problém. Predstavili neurónovú sieť, ktorá prijíma na vstupe priamo point cloud.

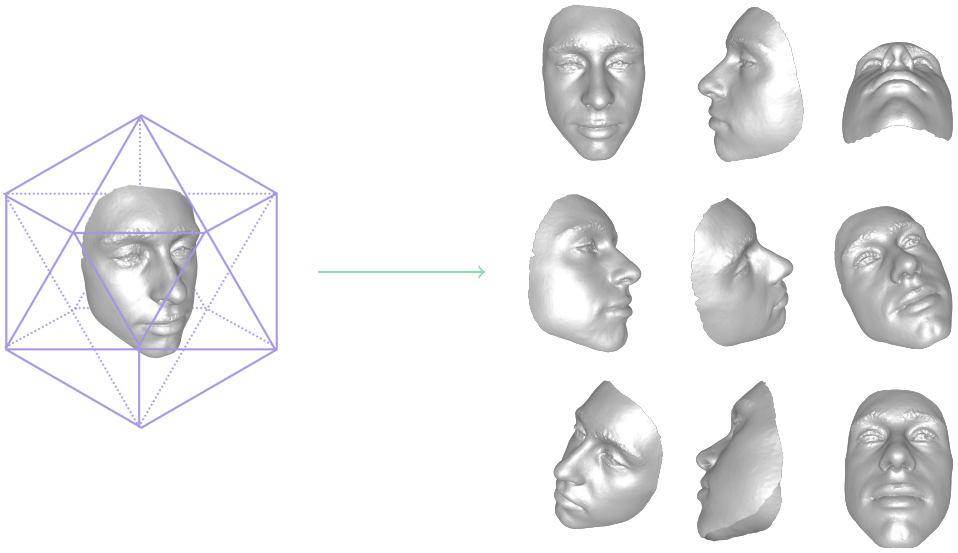
Intuitívnym riešením je konvertovať point cloud do nejakej štruktúrovanej ohraničenej formy. To je najčastejšie **voxel grid**, teda pravidelná mriežka trojdimenzionálneho priestoru. *Voxel* (volumetric pixel) je jednotka grafickej informácie, ktorá definuje bod v trojdimenzionálnej mriežke. Jednoducho povedané, predstavuje analógiu v trojdimenzionálnom svete k pixelu definujúcemu 2D bod na obrázku.

**Binárna mriežka voxelov** diskretizuje 3D objekt do binárnych voxelov. Obsahujú hodnotu 1, ak sa aspoň jeden bod objektu nachádza vo vnútri voxelu a 0, ak je voxel prázdny. Pri takejto reprezentácii dochádza k zhusťovaniu pôvodne riedkych dát, a preto sa stráca veľa informácie z pôvodného modelu. Vstupom do CNN sú potom namiesto 2D pixelov 3D voxely a aplikuje sa na nich 3D objemová konvolučná neurónová sieť (Phong (1973), Hegde a Zadeh (2016), Qi a kol. (2016b)).

Wu a kol. (2014) so svojím návrhom *3DShapeNets* boli prví, ktorí aplikovali na trojdimenzionálne dáta konvolučnú sieť. Podobným prístupom dosiahli ešte lepšie výsledky Maturana a Scherer (2015) predstavením *VoxNetu*. Navrhli hlbokú konvolučnú neurónovú sieť s objemovou reprezentáciou pomocou binárnej mriežky voxelov. Tu narazili na problém, že reprezentácia založená na voxelovej mriežke má vysoké pamäťové nároky a aby sa dala neurónova sieť natrénovať v rozumnom čase, je nevyhnutné výrazne limitovať rozlíšenie mriežky, čo má za následok ďalšiu stratu informácie.

### 4.1.2 Pixelová reprezentácia

Pixelová reprezentácia slúži na to, aby aj klasifikácia 3D dát benefitovala z úspešnej CNN. Hľadá spôsob, akým konvertovať trojdimenzionálnu úlohu na dvojdimenzionálnu. Jedným z prísutpom je reprezentácia 3D objektu množinou projek-



Obrázek 4.2: Príklad Multi-View renderingu.

tovaných 2D obrázkov, tzv. metóda **Multi-View** (Su a kol. (2015), Qi a kol. (2016b), **?**). Multi-view je založená na predpoklade, že ak sú si dva 3D modely podobné, tak vyzerajú podobne aj zo všetkých perspektív. Na získanie vstupných záberov sa z trojuholníkovej siete vyrendruje niekoľko projekcií pomocou viacerých kamier, ktoré sa väčšinou umiestnia do rohov mnohostena (viz. obrázok 4.2). Na generáciu rendrov sa často používa Phongov osvetľovací model (Phong, 1973).

Aby sa zachovala informácia o hĺbke, alternatívnym riešením je zakódovať ju do štvrtej dimenzie, podobne ako farbu pixelov vyrendrovaných obrázkov v multi-view reprezentácii. Tým získame **hĺbkovú mapu** (Gupta a kol., 2014), ktorá simuluje relatívnu vzdialenosť od pozorovateľa k častiam objektu. Potom hovoríme, že ide o tzv. 2.5D model (dva a pol dimenzionálny).

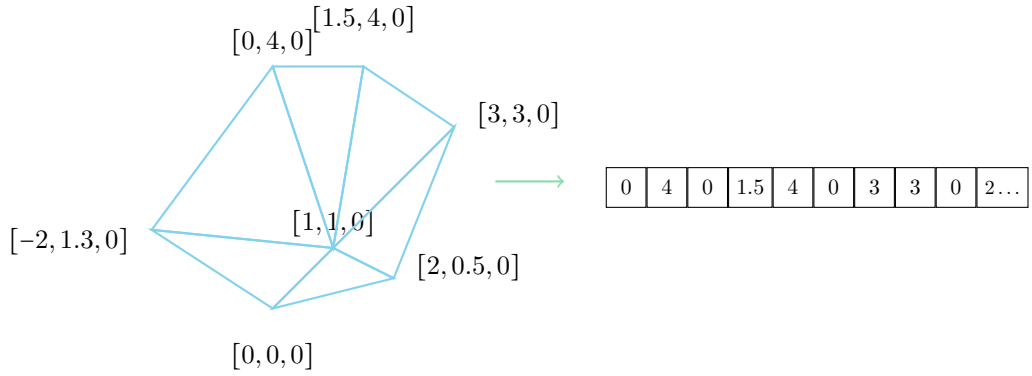
## 4.2 Použitá reprezentácia

V našej práci sme sa rozhodli o alternatívnejší prístup k reprezentáciám 3D objektov aby sme sa vyhli problémom, ktoré prichádzajú s prebytočnou objemnosťou dát. Hĺbku a topológiu siete sme sa pokúsili zakódovať do konečnej reprezentácie rôznymi technikami predspracovania dát.

Nakoniec sme navrhli tri rôzne spôsoby, ako reprezentovať 3D geometriu za účelom aplikácie na úlohu rozpoznávania pomocou hlbokého učenia. V tejto podkapitole si predstavíme každú z navrhnutých reprezentácií. Na prípravu datovej množiny sme vytvorili sadu skriptov a návod, ako ich spustiť a používať, je podrobne popísaný v prílohe A. Pripravené predspracované datasety sa nachádzajú v priečinku *preparedData* a ich základné vlastnosti zhrňuje tabuľka 4.1.

### 4.2.1 Uloženie súradníc do poľa

Prvý spôsob, akým sme reprezentovali množinu trojuholníkových sietí, je veľmi jednoduchý a vznikol bez ďalšieho predspracovania. Slúži najmä ako referenčná




Obrázek 4.3: Trojdimenzionálne súradnice položené do poľa „bez ladu a skladu".

metóda pre porovnanie s ostatnými reprezentáciami, od ktorých očakávame lepšie výsledky. Na obrázku 4.3 sa dá vypozorovať, ako sme každú 3D súradnicu $x, y, z$ kládli do samostatného políčka jednodimenzionálneho poľa. Na takýto model sme potom použili klasickú hlbokú neurónovú sieť. Naším predpokladom bolo, že ak k dátam budeme pristupovať primitívnym spôsobom „bez ladu a skladu", tak model nebude natoľko expresívny, aby dostatočne popísal vzor vyskytujúci sa v modeloch. Ďalej sme očakávali, že výsledok bude slúžiť ako spodná hranica s priestorom na vylepšenie ostatnými prístupmi.

NN sme trénovali na datovej množine vygenerovanej z vybranej oblasti nosa. Každý príklad z množiny má 3164 vrcholov, každý vrchol má 3 súradnice, preto bude mať výsledné pole dĺžku 9492 vrcholov.

| **Reprezentácia** | **Formát** | **Súbor** | **Neurónová sieť** |
|---|---|---|---|
| Súradnice v poli | 9492 | nose1dArrayDataset.txt | NN |
| Relativizácia súradníc | 9492 | nose1dArrayRelativized-Dataset.txt | NN |
| Projekcia 3D na 2D | $75 \times 87 \times 3$ | projectionsDataset.txt | Multichannel CNN |
| Povrch. lomené krivky 3 nulové vrcholy | $1 \times 973 \times 3$ | linesDataset3.txt | Multichannel CNN |
| Povrch. lomené krivky 43 nulových vrcholov | $1 \times 1293 \times 3$ | linesDataset43.txt | Multichannel CNN |
| Povrch. lomené krivky 56 nulových vrcholov | $1 \times 1387 \times 3$ | linesDataset56.txt | Multichannel CNN |
| Povrch. lomené krivky 74 nulových vrcholov | $1 \times 1541 \times 3$ | linesDataset74.txt | Multichannel CNN |

Tabulka 4.1: Prehľad navrhnutých reprezentácii 3D modelov.

**Relativizácia súradníc**

Táto reprezentácia je vytvorená dodatočným krokom v predspracovaní dát.

Zaujímalo nás, aký efekt bude mať posun bodov do iného súradnicového rámca na presnosť neurónovej siete. Relativizácia sa prevedie definovaním nového počiatku súradnicového systému $xyz$. Za nový bod počiatku $(0, 0, 0)$ definujeme vrchol $v$ a od ostatných súradníc odčítame pôvodné súradnice vrcholu $v$. My sme zvolili za nový počiatok špičku nosa a ostatné súradnice sme upravili s ohľadom na jeho súradnice. Následne sme takto upravenú trojuholníkovú sieť uložili do poľa analogicky ako bolo popísané vyššie.

### 4.2.2 Projekcia 3D objektu na 2D mriežku

Pri návrhu vhodnejšej reprezentácie sme sa zamýšľali, ako môžeme zakódovať informáciu o hĺbke a geometrii tváre bez použitia tretej dimenzie. K tomuto účelu sme vytvorili pomocný program *3DTo2DProjector* (viz 3.3.2), ktorý hľadá mapovanie vrcholov v 3D priestore do 2D mriežky. Hlavná vlastnosť, ktorú musí mapovanie spĺňať, je aby vrcholy, ktoré ležia blízko seba v trojuholníkovej sieti, ležali blízko seba aj v 2D mriežke. To vykazuje dobrý predpoklad pre učenie konvolučných neurónových sietí, pretože neuróny budú získavať príznaky iba zo svojich recepčných polí. Tento experiment sme opäť prevádzali na datovej množine s modelmi nosov.

Výstupom z programu je mriežka, ktorej položky obsahujú indexy vrcholov. Na obrázku 4.4 je vidieť, že výsledné mapovanie je asymetrické a nezapĺňa celú plochu pbdĺžnika. Vstupné tensory v Tensorflow vyžadujú tvar pravidelnej matice, preto sme na prázdne políčka obdĺždnika doplnili vrchol s indexom 0, ktorý odkazuje na vrchol so súradnicami $0, 0, 0$. Domnievame sa, že takýto prístup spôsobí v dátach minimum šumu. Za každý index $i$ v 2D mriežke dosadíme súradnice $x, y, z$ príslušného vrcholu $i$. Pretože je naša datová množina topologicky unifikovaná, vrcholy v sieťach s rovnakými indexami sú ekvivalentné a preto je vytvorenie mapovania jednorázový proces.

Na takto pripravené dáta sa potom pustí viackanálová konvolučná neurónová sieť s rozmermi $h \times w \times 3$. Symbol $h$ a $w$ označujú výšku a šírku mriežky. Hĺbka 3 znamená, že sieť bude prjímať zdroj z 3 kanálov. Každý kanál bude veľkosti $h \times w$, prvý kanál obsahuje súradnice $x$, druhý $y$ a tretí $z$.

### 4.2.3 Povrchové lomené krivky

Ako poslednú reprezentáciu sme vytvorili pokrytie všetkých významných častí ľudskej tváre povrchovými lomenými čiarami. Vytvorili sme 9 čiar, ako ilustruje Obrázok 4.5. Z toho 2 horizontálne a vertikálne tvoria hlavnú os tváre, 3 vedľajšie horizontálne lomené čiary pokrývajú čelo, oblasť medzi perami a nosom a bradu. 2 vedľajšie vertikálne čiary prechádzajú cez lícne kosti až ku brade. V bode prieniku s hlavnou osou sa priečia 2 diagonálne čiary tak, aby sa vyhli očiam. Lesklé povrchy odrážajú a rozptyľujú svetlo zo skenera do nekontrolovateľných smerov, preto oblasť oka obsahuje veľa šumu a na úlohu rozpoznávania nie je vhodná.

Možnosť extrakcie lomených čiar sme implementovali do programu *MeshEditor*. Každá čiara je jednodimenzionálna postupnosť indexov vrcholov a každá

Obrázek 4.4: Vizuálne spracovaný výstup z mapovacieho algoritmu. Políčka mriežky, ktoré obsahujú vrchol siete, sú zafarbené čiernou.

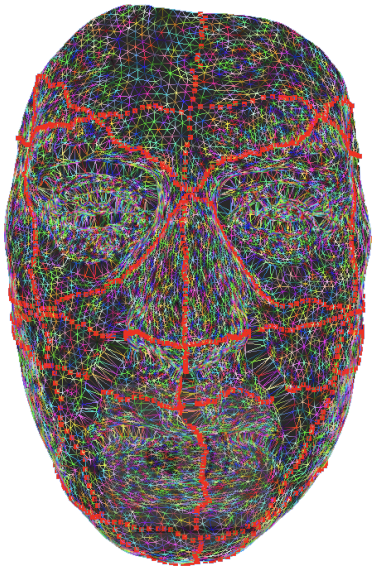
trojuholníková sieť je reprezentovaná množinou týchto čiar usporiadaných za sebou do 1D poľa. Pole bude mať dĺžku $w$, kde $w$ je súčet dĺžok všetkých povrhových lomených čiar. Za každý index sa potom dosadí súradnica príslušného vrcholu v sieti rovnakým spôsobom, ako v prípade 2D mriežky. Multikanálová konvolučná neurónová sieť bude mať rozmery $1 \times w \times 3$.

Pri navrhovaní tejto reprezentácii sme narazili na jeden problém. Pri prechode konvolučného filtra 1D poľom nastane moment, keď sa ocitne na rozmedzí dvoch susedných čiar naraz. Jednotlivé čiary však spolu nesúvisia a preto by filter medzi nimi nemal hľadať žiaden príznak. Z tohoto dôvodu sme každé dve čiary oddelili niekoľkými nulovými vrcholmi, t.j. vrcholmi so súradnicami $0, 0, 0$. Názorná ukážka je na obrázku 4.6. Vytvorili sme niekoľko datových množín, aby sme otestovali, ako sa bude CNN správať, ak jej predložime prekrývajúce krivky (3 oddeľujúce nuly), minimálne prekrývajúce krivky (43 a 56 núl) a krivky, ktoré sa v žiadnom recepčnom poli konvolučnej vrstvy neprekrývajú (74 núl).

## Výpočet počtu núl

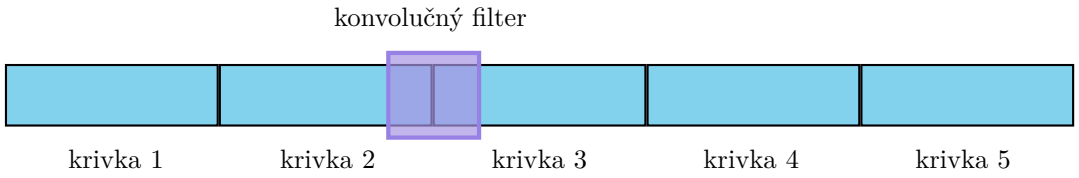
Vyššie sme spomenuli, že ak má pripravená datová množina 74 oddeľujúcich núl, tak nedochádza k prekrytiu dvoch kriviek v jednom recepčnom poli konvolúcie. Je nutné povedať, že k tomuto číslu sme dospeli vzhľadom na architektúru ModelNet3, ktorú si podrobne popíšeme v nasledujúcej kapitole. Architektúra ModelNet3 (viz 5.1) pozostáva z piatich konvolučných vrstiev, medzi ktorými sa nachádzajú pooling vrstvy, ktoré zmenšujú rozmer obrázkov. Musíme zistiť, koľko núl medzi krivkami na vstupe je potrebných, aby aj v poslednej konvolučnej vrstve detekoval filter v každom bode posunu príznaky z práve jednej krivky (alebo iba z núl). To sa dá ľahko vypočítať postupovaním smerom od poslednej konvolučnej vrstvy nahor smerom k prvej konvolučnej vrstve.

Filter poslednej konvolučnej vrstvy má rozmer $1 \times 3$. Z tohoto faktu a z ob-

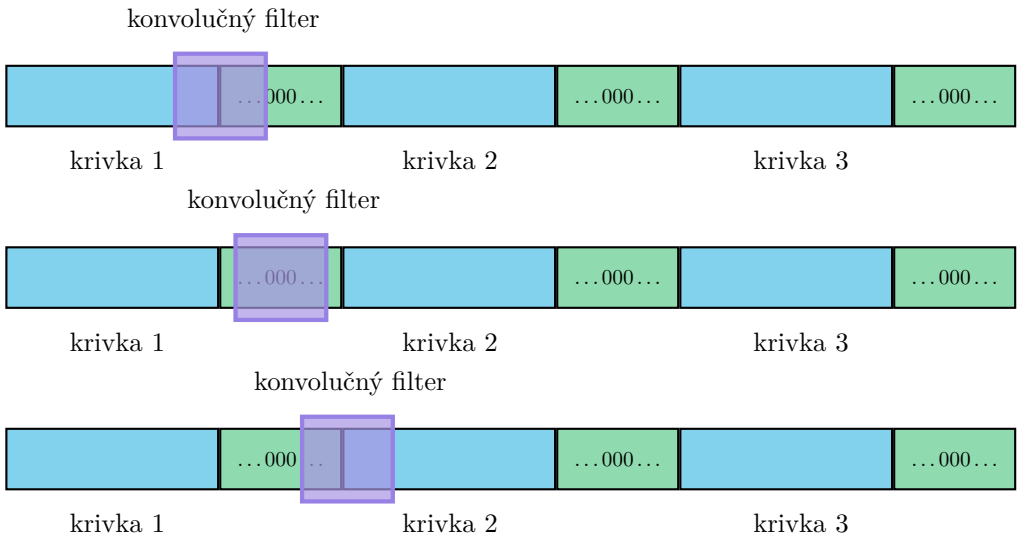


Obrázek 4.5: Snímka z programu *MeshEditor*. Manuálne sme si označili povrchové lomené krivky, ktoré zachytávajú topológiu tváre.

rázku 4.6 vyplýva, že medzi krivkami musia byť aspoň 2 nuly. Filter predposlednej konvolučnej vrstvy má tiež rozmer $1 \times 3$. Aby na výstupe dal požadované 2 nuly, musí sa podľa princípu konvolúcie vyskytnúť aspoň 2 krát na pozícii, kde bude detekovať samé nuly (aby po vynásobení tromi váhami filtra vyšla nula). Preto musia byť v tejto vrstve medzi krivkami aspoň 4 nuly. Vrstva nad ňou je totožná a opäť sa pýtame, aký je minimálny počet núl, aby sme vo výstupe dostali 4 nuly. Odpoveď je 6. Takýmto spôsobom sme postupovali vyššie až do prvej konvolučnej vrstvy. Pooling vrstva a konvolučná vrstva so stride 2 zníži rozmer obrázka na polovicu, preto je vtedy potrebné nuly zdvojnásobiť. Napokon sme dostali výsledok 74 oddeľujúcich núl. Obrázok 4.7 pre lepšie pochopenie tento postup ilustruje.

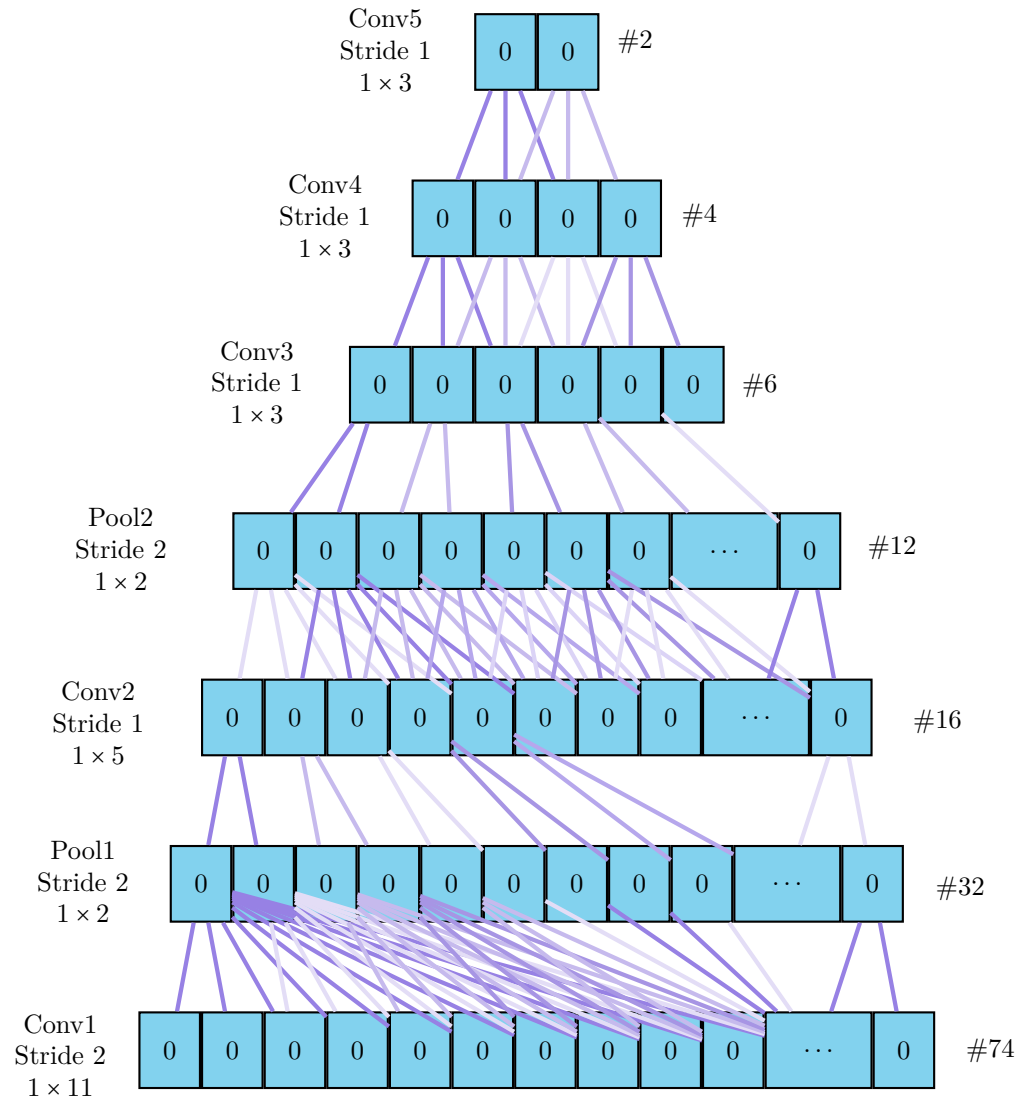



(a) Konvolučný filter na rozmedzí dvoch kriviek.




(b) Pridanie sekvencie núl medzi každé dve krivky.

Obrázek 4.6: Obrázok ilustruje pohyb konvolučného filtra 1D poľom tvoreného z lomených kriviek. Na obrázku (a) dochádza k situácii, kedy 1D filter detekuje príznaky v oboch krivkách naraz. Krivky spolu ale nesúvisia a preto by v tejto časti filter nemal hľadať žiadne príznaky. Riešenie ukazuje obrázok (b), kde medzi každé dve krivky pridáme takú postupnosť nulových vrcholov, aby recepčné pole tvorila práve jedna krivka.




Obrázek 4.7: Priebeh výpočtu počtu oddeľujúcich núl medzi krivkami pre architektúru CNN ModelNet3. Pretože vieme, že v poslednej konvolučnej vrstve potrebujeme 2 nuly, celý výpočet prebieha smerom od poslednej k prvej konvolučnej vrstve. V prvej vrstve predpokladáme, že filter môže začať na párnom indexe, preto sa pripočíta 1.

# 5. Experimenty

Hlavným cieľom tejto kapitoly je porovnať kvalitu navrhnutých reprezentácií 3D objektov podľa úspešnosti neurónových sietí. Využijeme vyššie popísané teoretické poznatky a navrhneme, implementujeme a natrénujeme vlastné neurónové siete. Všetky datové množiny, ktoré sme použili v experimentoch, sú popísané v časti 4.2. Ďalej sa v tejto kapitole budeme držať názvoslovia jednotlivých reprezentácií uvedeného v tabuľke 4.1. Neurónové siete sú naprogramované v jazyku Python pomocou frameworku Tensorflow.

Popis architektúr a výsledkov experimentov si rozčleníme do dvoch častí, podľa toho či sa jedná o klasickú alebo konvolučnú neurónovú sieť. Na záver získané výsledky prediskutujeme a navzájom medzi sebou porovnáme.

## Popis úlohy

Navrhnuté neurónové siete sú aplikované na úlohu **rozpoznávania pohlavia** modelov naskenovaných tvárí. Každú triedu klasifikácie sme reprezentovali tzv. *one hot*, inak zvaným 1-out-of-K binárnym vektorom, kde $K$ je počet tried klasifikácie. V našom prípade je $K = 2$, žena je reprezentovaná vektorom $(1, 0)$ muž $(0, 1)$.

V ideálnej situácii sú dáta rozdelené na trénovacie a testovacie podmnožiny. My sme 9/10 datovej množiny použili na trénovanie a 1/10 na testovanie siete. V trénovacej množine máme k dispozícii 298 3D modelov ľudských tvárí. Nie je to ideálne veľká množina, preto čiastočným riešením proti skresleniu výsledkov nevhodným rozdelením množiny je K-fold cross validácia, ktorú sme si popísali v kapitole 1 v časti 1.6.2. Bola aplikovaná 10-fold cross validation a experiment bol opakovaný 10 krát (celkovo 100 opakovaní experimentu pre každú neurónovú sieť). V každom opakovaní sa vytvorilo náhodné rozdelenie tréningovej množiny na desatiny a v každej epoche bola množina náhodne premiešaná.

## 5.1 Klasická neurónová sieť

Experimentálnu časť tejto práce sme začali návrhom klasickej hlbokej neurónovej siete pre datasety `nose1dArrayDataset.txt` a `nose1dArrayRelativizedDataset.txt`, kde bola klasickej neurónovej siete predstavená jednoduchá postupnosť súradníc vrcholov. Predpokladali sme, že výsledky poskytnú vhodnú spodnú hranicu pre porovnanie a priestor na vylepšenie „premyslenejšou" reprezentáciou.

Nájsť správnu konfiguráciu parametrov pre hlboké neurónové siete je nepraktické a časovo náročné, ale zásadné pre úspešné učenie. Aby sme si boli istí, že sme prehľadali dostatočne široký úsek parametrického priestoru, vytvorili sme si zoznam všetkých možných kombinácii, o ktorých sme predpokladali potenciálny úspech.

Nasledujúce parametre NN sme ponechali konštantné pre všetky architektúry:

- vstupná vrstva obsahuje 9492 neurónov s hodnotami súradníc
- 50 učiacich epoch

- rýchlosť učenia 0,001
- každá skrytá vrstva využíva aktivačnú funkciu *ReLU*
- výstup poslednej skrytej vrstvy využíva funkciu *Softmax* pre získanie predikovanej príslušnosti do triedy
- prvej skrytej vrstve sa najprv predá 100 tréningových príkladov, čo je veľkosť jednej dávky (batch size)

Ostatné premenné parametre sme hľadali stratégiou pokus - omyl. Pri výbere optimálnej štruktúry plne prepojených hlbokých vrstiev sa musia tradične spraviť dve rozhodnutia. Koľko skrytých vrstiev bude mať neurónová sieť a počet neurónov v týchto vrstvách.

- Pre lineárne neseparovateľné dáta potrebujeme aspoň jednu **skytú vrstvu**. Dve alebo viac skrytých vrstiev sa dokážu naučiť komplexnejšie reprezentácie. Vo väčšine článkoch a programátorských fórach sa ľudia vyjadrovali, že jedna až dve vrstvy sú postačujúce pre väčšinu úloh, preto sme testovali neurónovú sieť s 1 a 2 skrytými vrstvami.
- Na optimalizovanie **počtu neurónov** v skrytých vrstvách neexistuje žiadne univerzálne pravidlo. Pre každú úlohu a dataset sa optimálne parametre líšia, a preto pre tento problém v strojom učení stále neexistuje jedno správne riešenie. Existujú isté základné pravidlá, ako napríklad: „veľkosť skrytej vrstvy by mala byť niekde medzi počtom vstupných a počtom výstupných neurónov" (Blum, 1992) alebo „počet neurónov v skrytej vrstve by nemal prekračovať dvojnásobok počtu neurónov vo vstupnej vrstve"(Linoff a Berry, 2011). Na základe toho sme sa rozhodli vyskúšať všetky kombinácie počtu vrstiev a nasledujúcej postupnosti počtu neurónov v skrytej vrstve:

$$\{2, 16, 64, 128, 256, 512, 640, 768, 896\} \tag{5.1}$$

### 5.1.1 Výsledky

Výsledky z navrhnutých architektúr sú v Tabuľke (5.1) a Tabuľke (5.2) pre oba datasety. Z dôvodu prehľadnosti sú v tabuľkách zastúpené iba najlepšie výsledky, výpis všetkých experimentov je v priloženom CD v adresári *results*.

Modelom s nízkym počtom neurónov sa darilo výrazne horšie, ako tým, ktoré mali v každej vrstve aspoň 64 vrcholov. 2 skryté vrstvy sa ukázali byť najvhodnejšou voľbou. Maximálny výsledok na testovacej množine **68**,**99**% bol dosiahnutý na datasete `noseDataset.txt` neurónovou sieťou s id 7 s 2 plne prepojenými skrytými vrstvami s príslušným počtom neurónov 512, 512. Rozdiely medzi jednotlivými úspešnejšími sieťami však boli tak malé, že tvrdenie, že jeden model je lepší ako druhý, nemusí byť objektívne.

Ako sa ukázalo, neurónová sieť nie je invariantná voči posunu. Relativizácia súradníc (dataset `noseDatasetNormalized.txt`), neposkytla sieti žiadnu užitočnú informáciu. Priemerne dosahovala nižšie výsledky a dokonca bol zaznamenaný pokles najlepšieho výkonu siete na **62**,**60**% na modeli s 2 skrytými vrstvami

| ID | # skrytých vrstiev | # neurónov vo vrstvách | Úspešnosť na test. množ. |
|---|---|---|---|
| 1 | 1 | 512 | 65,69% |
| 2 | 2 | 256, 2 | 52,24% |
| 3 | 2 | 256, 16 | 65,37% |
| 4 | 2 | 512, 64 | 66,15% |
| 5 | 2 | 768, 128 | 67,82% |
| 6 | 2 | 896, 256 | 67,28% |
| 7 | 2 | 512, 512 | **68,99**% |
| 8 | 2 | 768, 640 | 67,89% |
| 9 | 2 | 768, 768 | 68,04% |
| 10 | 2 | 896, 896 | 66,23% |

Tabulka 5.1: Tabuľka úspešností neurónových sietí natrénovaných na datasete `noseDataset.txt`.

| ID | # skrytých vrstiev | # neurónov vo vrstvách | Úspešnosť na test. množ. |
|---|---|---|---|
| 1 | 1 | 512 | 61,57% |
| 2 | 2 | 256, 2 | 51,52% |
| 3 | 2 | 128, 16 | 60,01% |
| 4 | 2 | 896, 64 | 60,86% |
| 5 | 2 | 512, 128 | 62,37% |
| 6 | 2 | 640, 256 | **62,60**% |
| 7 | 2 | 896, 512 | 62,24% |
| 8 | 2 | 640, 640 | 61,94% |
| 9 | 2 | 768, 768 | 62,04% |
| 10 | 2 | 896, 896 | 62,34% |

Tabulka 5.2: Tabuľka úspešností neurónových sietí natrénovaných na datasete `nose1dArrayRelativizedDataset.txt`.

a príslušnými počtami neurónov 640, 256. Výsledky nepotvrdili našu hypotézu, že relativizáciou sa úspešnosť NN zlepší.

Ako sme očakávali, tento experiment nepriniesol žiadne dramatické výsledky. Klasické neurónové siete neberú do úvahy lokálne črty dát a preto je obvyklý postup v procese predspracovania najprv extrahovať zaujímavé príznaky, ako napríklad hrany, rohy alebo oblúky. Navyše, neurónové sieťe sú navrhnuté na princípe ľudského myslenia. Ak by sme dali človekovi 3D model ľudskej tváre, pravdopodobne by si ho obzrel z rôznych uhlov a vzápäti by vedel povedať, či sa jedná o muža alebo ženu. Podobne si môžeme odôvodniť výsledok tohoto experimentu. Ak by sme dali človekovi zoznam čísel bez popisu, čo znamenajú, bolo by oveľa menej pravdepodobné, že by našiel správnu odpoveď.

## 5.2 Konvolučná neurónová sieť

Konvolučné neurónové siete, narozdiel od klasických, berú do úvahy priestorové vzťahy v dátach. Predpokladáme, že vzhľadom na charakter vstupných dát sú konvolučné siete silnejším nástrojom pre túto úlohu.

Pre túto úlohu sme navrhli tri architektúry, na ktorej sme trénovali CNNs. Pre zjednodušenie sme do obrázku zaznačili konvolučné filtre so štvorcovými rozmermi, ale treba pripomenúť, že v prípade datasetu `obliqueLinesDataset.txt` sú čiary uložené za sebou v 1D poli s výškou 1. Preto sú filtre tiež 1D s rozmermi $1 \times w$, kde $w$ je šírka filtra. Všetky datasety sú spracované tak, ako sme popísali v častiach 4.2.2 a 4.2.3 a použijú sa na učenie viackanálových konvolučných neurónových sietí.

Konvolučné siete sme trénovali 50 a 100 epoch, najperspektívnejšiu architektúru sme nechali trénovať dokonca 200 epoch. Pri fixnom počte epoch môže dojsť k preučeniu, preto bola zaradená do všetkých konvolučných sietí počas trénovania metóda *dropout*, ktorá zabraňuje sieti adaptovať sa na trénovaciu množinu. Každý neurón s pravdepodobnosťou $p = 0.8$ ponechá v sieti, a s pravdepodobnosťou 0.2 neurón a spojenia vedúce smerom dnu a von z neurónu odstráni. Tak vznikne náhodne vygenerovaná riedšia verzia siete, ktorá navyše urýchli učenie. Dropout vrstva je uložená v každej architektúre tesne pred výstupnou vrstvu.

Vstupom do vrstiev konvolučných sietí sú 3 kanály so súradnicami $x, y, z$, každý má rozmer $75 \times 86$, $1 \times 973$, $1 \times 973$, $1 \times 1293$, $1 \times 1387$ alebo $1 \times 1541$ (podľa datovej množiny z Tabuľky 4.1).

Ďalej si popíšeme jednotlivé použité architektúry CNNs. Ich štruktúru vizualizuje obrázok 5.1.

(a) ModelNet1:

- Prvá skrytá vrstva je konvolučná a obsahuje 32 filtrov o veľkosti $3 \times 3$, všetky s posunom o 1 pixel a paddingom 'SAME', ktorý zachováva pôvodnú veľkosť obrázka. Aktivačná funkcia je ReLU.
- Druhá vrstva je max pooling vrstva, ktorá dá na výstup iba maximálne hodnoty príznakových máp v okne o veľkosti $2 \times 2$ s posunom o 2. Táto vrstva zredukuje veľkosť každej príznakovej mapy na polovicu.
- Príznakové mapy z pooling vrstvy potom vstupujú do druhej konvolučnej vrstvy, ktorá má 64 filtrov o veľkosti $3 \times 3$, posun o 1 a padding 'SAME'. Aktivačná funkcia je ReLU.
- Nasleduje max pooling vrstva, ktorá zredukuje rozmery 64 príznakových máp na polovičnú veľkosť s filtrom $2 \times 2$ a posunom o 2 pixely.
- Výstupy pooling vrstvy sa zlúčia do plne prepojenej vrstvy, ktorá sa skladá z 1024 neurónov.
- Na plne prepojenú vrstvu sa aplikuje technika dropout s pravdepodobnosťou $p = 0{,}8$.
- Výstup sa pomocou aktivačnej funkcie Softmax transformuje na výsledné hodnoty.

(b) ModelNet2 je architektúra analogická k (a) s tým rozdielom, že veľkosť konvolučných filtrov je $5 \times 5$.

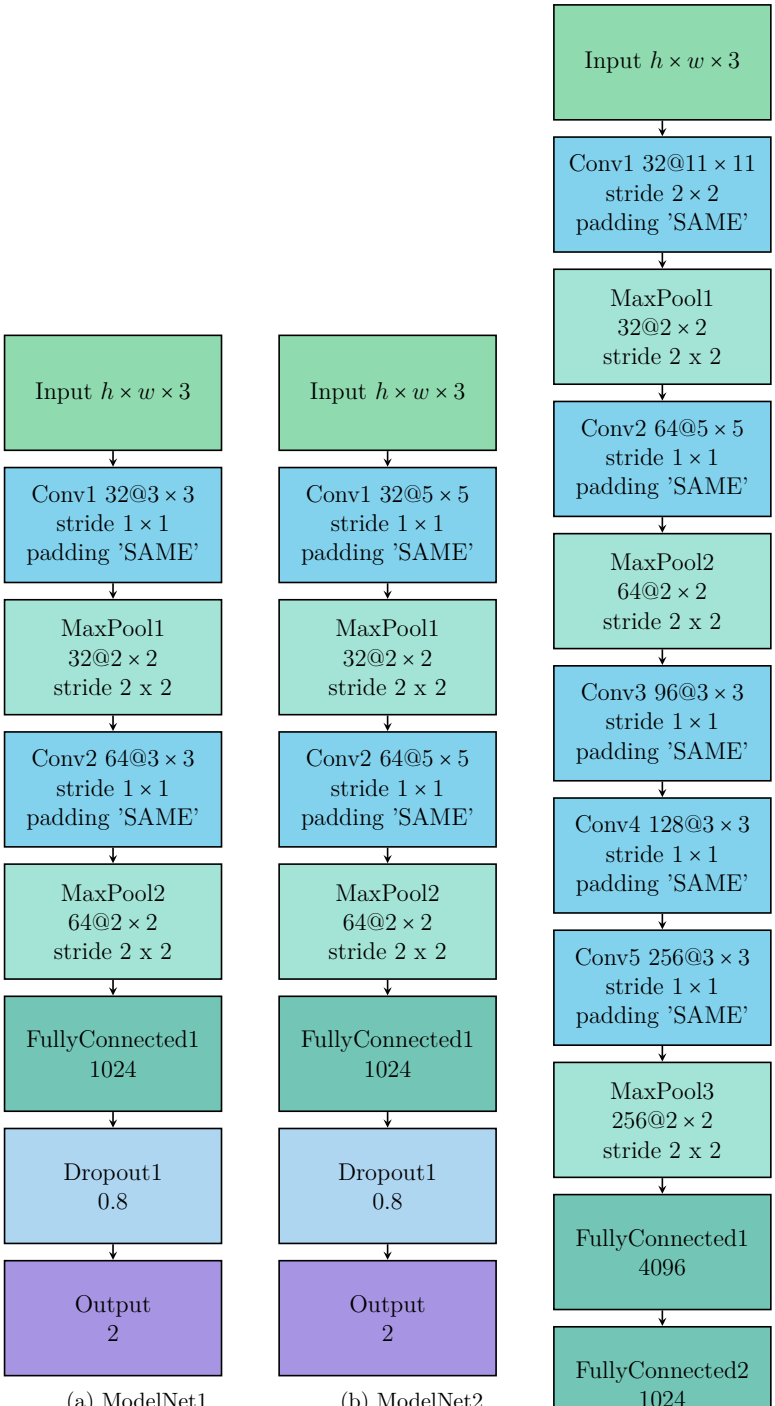
(c) ModelNet3:

- Prvá vrstva je konvolučná a obsahuje 32 filtrov o veľkosti 11×11, všetky s posunom o 2 pixely a paddingom 'SAME', ktorý zachováva pôvodnú veľkosť obrázka. Aktivačná funkcia je ReLU.
- Druhá vrstva je max pooling vrstva, ktorá dá na výstup iba maximálne hodnoty príznakových máp v okne o veľkosti $2 \times 2$ s posunom o 2. Táto vrstva zredukuje veľkosť každej príznakovej mapy na polovicu.
- Príznakové mapy z pooling vrstvy potom vstupujú do druhej konvolučnej vrstvy, ktorá má 64 filtrov $5 \times 5$, posun o 1 a padding 'SAME'. Aktivačná funkcia je ReLU.
- Výstupy druhej konvolučnej vrstvy vstupujú do max pooling vrstvy, ktoré zredukujú rozmery 64 príznakových máp na polovičnú veľkosť s filtrom $2 \times 2$ a posunom o 2 pixely.
- Nasleduje tretia konvolučná vrstva s 96 filtrami o veľkosti $3 \times 3$, posun o 1 a padding 'SAME'. Aktivačná funkcia je ReLU.
- 96 príznakových máp z tretej konvolučnej vrstvy vstupuje do štvrtej konvolučnej vrstvy s 128 filtrami o veľkosti $3 \times 3$, posun o 1 a padding 'SAME'. Aktivačná funkcia je ReLU.
- Piata a posledná konvolučná vrstva obsahuje 256 filtrov o veľkosti 3×3, posun o 1 a padding 'SAME'. Aktivačná funkcia je ReLU.
- Nasleduje max pooling vrstva číslo 3, ktorá zredukuje rozmery 256 príznakových máp na polovičnú veľkosť s filtrom $2 \times 2$ a posunom o 2 kroky.
- Výstupy pooling vrstvy sa zlúčia do plne prepojenej vrstvy, ktorá sa skladá z 4096 neurónov.
- Druhá prepojená vrstva pozostáv z 1024 neurónov.
- Na plne prepojenú vrstvu sa aplikuje metóda dropout s pravdepodobnosťou $p = 0{,}8$.
- Výstup sa pomocou aktivačnej funkcie Softmax transformuje na výsledné hodnoty.

### 5.2.1 Výsledky

Výsledky experimentov pre 3D projekcie do mriežky a povrchové lomené čiary sú v Tabuľke 5.3 a Tabuľke 5.4.

Ak porovnáme rôzne architektúry s použitím rovnakých vstupných datových množín, v oboch prípadoch projekcie do mriežky aj lomených kriviek prekonala komplexnejšia architektúra ModelNet3 jednoduchšie architektúry ModelNet1 a ModelNet2. ModelNet3 obsahovala zo všetkých navrhnutých architektúr najviac vrstiev a bola aj časovo najnáročnejšia. Zväčšenie veľkosti filtru v architektúre ModelNet1 na $5 \times 5$ v ModelNet2 nespôsobilo takmer žiadne vylepšenie výkonnosti siete.




(a) ModelNet1 (b) ModelNet2 (c) ModelNet3

Obrázek 5.1: Štruktúra troch modelov konvolučných neurónových sietí, na ktorých boli prevádzané experimenty v tejto práci. Všetky datové množiny pozostávajú z troch kanálov, súradníc $x, y, z$, preto musíme na vstupe definovať rozmer tensora $h \times w \times 3$, kde $h$ je výška, $w$ šírka a 3 je hĺbka, resp. počet kanálov.

| Model | # epoch | Úspešnosť na test. množ. |
|---|---|---|
| ModelNet1 | 50 | 69,07% |
| ModelNet2 | 50 | 69,90% |
| ModelNet2 | 100 | 76,79% |
| ModelNet3 | 50 | 69,59% |
| ModelNet3 | 100 | **77**,**66**% |

Tabulka 5.3: Tabuľka úspešností neurónových sietí natrénovaných na datasete s projekciami 3D objektov do 2D mriežky, ktorý sa nachádza v súbore `2dMap-Dataset.txt`.

Model CNN natrénovaný na datách s projekciou 3D modelov do 2D mriežky dosiahol najväčšiu výkonnosť **77**,**60**% na architektúre ModelNet3 so 100 epochami. Tento výsledok si odôvodňujeme aj z biologického hľadiska, keďže sme pracovali s množinou nosov. Je možné, že anatomické rozdiely medzi mužmi a ženami v geometrii nosa nedokazujú pohlavný dimorfizmus.

Najvyšší zaznamenaný výsledok u povrchových lomených kriviek bol zároveň najlepším výsledkom vôbec. Jednotlivé experimenty na povrchových krivkách sa líšili počtom oddeľujúcich núl medzi krivkami. Dataset so 74 nulami bol vypočítaný tak, aby sa pri konvolúcii nikdy nedetekovali príznaky z dvoch susedných čiar naraz. Pri 100 epochách dávali najlepšie výsledky neurónové siete trénované na architektúre ModelNet3 s 3, 56 a 74 nulami s úspešnosťou 80,00%, 80,93% a 80,90%. Tieto neurónové siete sme potom nechali trénovať 200 epoch a napokon bol dosiahnutý maximálny výsledok **84**,**20**% na datasete s 56 oddeľujúcimi nulami. V porovnaní s reprezentáciou s 3 separujúcimi nulami sa potvrdilo, že sa neurónka učí lepšie, ak sa v jednom recepčnom poli neprekrývajú dve krivky. Pokiaľ sa krivky prekrývajú veľmi málo, ako v prípade 56 núl, tak to CNN nevadí. Obrázok 5.2 ukazuje zmenu testovacej úspešnosti tejto CNN so zvyšujúcim sa počtom epoch.

## 5.3 Diskusia

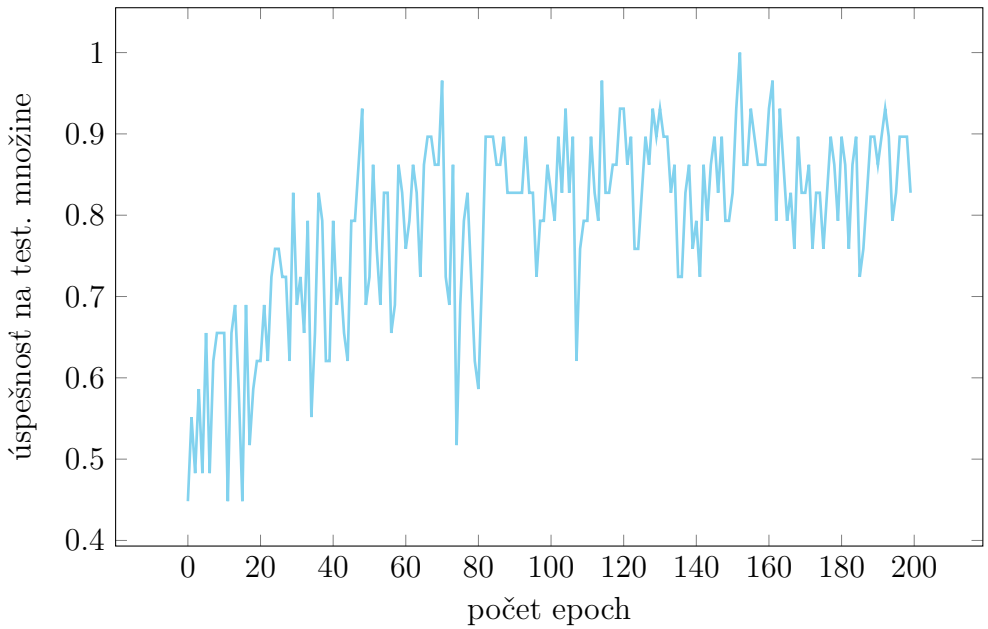
Cieľom našich experimentov nebolo nájsť najvýkonnejšiu architektúru, ale navrhnúť nové metódy predspracovania 3D dát do neurónových sietí, zistiť, či sú dostatočne expresívne, porovnať ich medzi sebou a medzi existujúcimi prístupmi.

Z konečných výsledkov sa nám podarilo ukázať, že konvolučné neurónové siete sú pre daný typ problému jednoznačne lepšou voľbou. Klasická neurónová sieť nie je dostatočne chytrá na to, aby vedela rozpoznať príznaky z jednoduchého poľa súradníc. Na jej vylepšenie by mohlo pomôcť napríklad nájsť parametrický popis objektov a ten použiť ako vstup.

V pokusoch s konvolučnými sieťami bol zaznamenaný výrazný pokrok v správnej klasifikácii testovacej množiny, kde sa nám podarilo získať presnosť **84**,**20**%. Povrchové lomené krivky preukázali najväčšiu schopnosť extrahovať topologické vlastnosti modelov tvárí. Záleží na tom, aby bol medzi krivkami dostatočný počet oddeľujúcich núl, aby pri konvolúcii v jednom recepčnom poli bolo ich prekrytie minimálne. Domnievame sa, že spôsob, akým boli CNN prestavené dáta vo forme

| Model | # sep. núl | # epoch | Úspešnosť na test. množ. |
|---|---|---|---|
| ModelNet1 | 3 | 50 | 66,17% |
| ModelNet2 | 3 | 50 | 66,17% |
| ModelNet3 | 3 | 50 | 75,17% |
| ModelNet3 | 3 | 100 | 80,00% |
| ModelNet3 | 3 | 200 | 75,86% |
| ModelNet3 | 43 | 100 | 79,31% |
| ModelNet3 | 56 | 100 | 80,93% |
| ModelNet3 | 56 | 200 | **84,20**% |
| ModelNet3 | 74 | 100 | 80,90% |
| ModelNet3 | 74 | 200 | 83,83% |

Tabulka 5.4: Tabuľka úspešností neurónových sietí natrénovaných na datasetoch s povrchovými lomenými krivkami. Jednotlivé datasety sa líšia počtom separujúcich núl medzi krivkami.




Obrázek 5.2: Graf úspešnosti na testovacej množine získaný počas trénovania CNN na datovej množine `linesDataset56.txt` po dobu 200 epoch.

lomených kriviek, jej boli predané aj hlavné vlastnosti objektu a hľadanie príznakov bolo pre ňu viac priamočiare. V reprezentácii s 2D mriežkou mohlo dôjsť k tomu, že filter detekoval vzory v oblastiach, kde bolo redundantné množstvo informácie, ktoré mohlo spôsobiť šum a sťaženie rozpoznávania.

Pre zhrnutie výsledkov všetkých experimentov, výkon rástol vzostupne pri reprezentáciách *relativizácia súradníc* → *súradnice v poli* → *projekcia do 2D mriežky* → *povrchové lomené krivky*.

Navyše, proces učenia klasických sietí bol výrazne rýchlejší a preto bolo prevedené komplexnejšie prehľadanie parametrického priestoru NN než u CNN. Z toho dôvodu si myslíme, že výsledky konvolučných sietí by ešte narástli skúšaním rôznych architektúr a ladením hyperparametrov. Napriek tomu úspešnosť 84,20% naznačuje, že nami navrhnutá reprezentácia je dostatočne expresívna pre konvolučnú neurónovú sieť a zároveň podstatne menej pamäťovo a časovo náročná, ako existujúce prístupy s volumetrickou 3D CNN.

# Závěr

Hlavným cieľom tejto práce bola aplikácia hlbokého učenia na rozpoznávanie trojrozmerných objektov. Neurónové siete sú rozsiahlo využívané predovšetkým na úlohy súvisiace s 2D obrazom, kde je mapovanie pixelov na vstup siete triviálne. Pri 3D dátach stále nie je jednoznačné, ako ich reprezentovať najlepšie. Našou prácou sme si kládli otázku, ako predať neurónovej sieti významné informácie.

Po analýze existujúcich prístupov sme predstavili tri vlastné reprezentácie 3D geometrie, a to uloženie súradníc do poľa bez predspracovania, projekciu do 2D mriežky a množinu povrchových lomených kriviek. Pri poslednej zmienenej reprezentácii sme analyzovali, nakoľko ovplyvní dĺžka pridaného segmentu nulových vrcholov medzi dvomi krivkami konečný výsledok. Bola vytvorená interaktívna aplikácia v C# na editáciu trojuholníkových sietí, konzolová aplikácia v Jave na vytvorenie projekcie do 2D mriežky a sada skriptov na predspracovanie dát a trénovanie NN. Materiálom k otestovaniu navrhnutých reprezentácií bola množina 3D povrchových modelov ľudských tvárí vo forme trojuholníkových sietí. Za kritériom kvality reprezentácií sme považovali úspešnosť učenia NN a CNN na úlohe klasifikácie pohlavia. Dodatočným krokom pri predspracovaní dát sme analyzovali vplyv relativizácie súradníc vrcholov a vytvorenia modifikovanej datovej množiny selekciou oblasti nosa.

Klasické neurónové siete sme trénovali v širkom rámci parametrického priestoru. Pre konvolučné neurónové siete sme navrhli tri architektúry: jednoduché ModelNet1, ModelNet2 a komplexnejšiu ModelNet3. Presnosť klasifikátoru sme získali pomocou metódy K-fold cross validácie pre $K = 10$.

Úspešnosť NN bola napriek dôslednému testovaniu rôznych kombinácií parametrov limitovaná. Na základe výsledkov považujeme množinu lomených povrchových kriviek za najviac perspektívnu reprezentáciu. Experimenty sme prevádzali s rôznym počtom oddeľujúcich núl a ukázalo sa, že s minimálnym až nulovým prekrytím núl v recepčnom poli konvolúcie je sieť najviac schopná učenia. Najúspešnejšiu CNN sme trénovali po dobu 200 epoch s úspešnosťou 84,2% na architektúre ModelNet3.

Výsledky ukázali, že CNN sú pre daný typ problému jednoznačne lepšou voľbou. Takisto sme došli k záveru, že na oblasti nosa nie sú detekovateľné rozdiely pohlavia. Porovnaním výsledkov s relativizovaním súradníc došlo k zhoršeniu, NN teda nie je invariantná voči posunu. Domnievame sa, že úspešnosť 84,20% by mohla byť ešte vylepšená ladením parametrov architektúry.

## Možnosti ďalšieho rozšírenia práce

Navrhnuté reprezentácie trojrozmerných dát ponúkajú priestor pre ďalšie zlepšenie a výskum. Problém, s ktorým sme sa v práci potýkali, je pomerne malá veľkosť datovej množiny. Možným rozšírením by bola augmentácia dát na základe bezstratových permutácií. Takýmto spôsobom by sa dala veľkosť datovej množiny niekoľkonásobne zväčšiť.

Úspešnosť CNN na projekcií vrcholov má potenciál na vylepšenie. Jednou z možností je úprava umiestňovcieho algoritmu, kde by sa za hlavné kritérium

poradia umiestňovania vrcholov použila skutočná vzdialenosť namiesto relatívnej medzi susediacimi vrcholmi. Výsledok algoritmu by sa líšil aj iným prístupom k výberu prvého ukladaného vrcholu. Ďalšou možnosťou je extrakcia inej časti tváre okrem nosa, ktorá by mohla vykazovať dimorfizmus.

V budúcnosti by bolo taktiež zaujímavé otestovať, ako sa úspešné reprezentácie 3D dát správajú pri iných úlohách. Vďaka označenej datovej množine je možné vyskúšať napríklad odhad veku, BMI alebo predpoveď starnutia.

# Seznam použité literatury


Caffe github. http://github.com/BVLC/caffe.

Caffe model zoo. http://tutorial.caffe.berkeleyvision.org/model_zoo.html.

Microsoft cognitive toolkit github. https://github.com/Microsoft/CNTK.

(2016). Advanced visualizer manual. URL https://www.cs.utah.edu/~boulos/cs3505/obj_spec.pdf. Appendix B1. Wavefront Technology. n.d. Web.

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. a Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Barrett, K., Barman, S., Boitano, S. a Brooks, H. (2009). *Ganong's Review of Medical Physiology, 23rd Edition*. LANGE Basic Science Series. Mcgraw-hill. ISBN 9780071605670. URL https://books.google.cz/books?id=fnEc16_ArFUC.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D. a Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

Blum, A. (1992). *Neural Networks in C++: An Object-Oriented Framework for Building Connectionist Systems*. Number zv. 1 in Wiley professional computing. John Wiley & Sons. ISBN 9780471552017. URL https://books.google.cz/books?id=9H0pAQAAMAAJ.

Collobert, R., Kavukcuoglu, K. a Farabet, C. (2011). Torch7: A matlab-like environment for machine learning.

Goodfellow, I., Bengio, Y. a Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Gupta, S., Girshick, R. B., Arbelaez, P. a Malik, J. (2014). Learning rich features from RGB-D images for object detection and segmentation. *CoRR*, **abs/1407.5736**.

Hegde, V. a Zadeh, R. (2016). Fusionnet: 3d object classification using multiple data representations. *CoRR*, **abs/1607.05695**.

Hubel, D. a Wiesel, T. (1968). Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology*, **195**, 215–243. ISSN 0022-3751.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R. B., Guadarrama, S. a Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *CoRR*, **abs/1408.5093**. URL `http://arxiv.org/abs/1408.5093`.

LeCun, Y., Bottou, L., Bengio, Y. a Haffner, R. (1998). Gradient-based learning applied to document recognition. *Procedings of the IEEE*, **86**, 2278?2324.

Li, F., Johnson, J. a Yeung, S. (2017). Side-by-side illustrations of biological and artificial neurons. `http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture4.pdf`.

Linoff, G. S. a Berry, M. J. A. (2011). *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management*. Wiley Publishing, 3rd edition. ISBN 0470650931, 9780470650936.

Maturana, D. a Scherer, S. (2015). Voxnet: A 3d convolutional neural network for real-time object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

McCulloch, W. a Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical bio-physics*, **5**(4).

Minsky, M. a Papert, S. (1969). *Perceptrons*. MIT Press, Cambridge MA, USA.

Phong, B. T. (1973). *Illumination for Computer-generated Images*. PhD thesis. AAI7402100.

Qi, C. R., Su, H., Mo, K. a Guibas, L. J. (2016a). Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, **abs/1612.00593**.

Qi, C. R., Su, H., Niessner, M., Dai, A., Yan, M. a Guibas, L. J. (2016b). Volumetric and multi-view cnns for object classification on 3d data. *CoRR*, **abs/1604.03265**.

Rodolfo Bonnin (2016). Building machine learning projects with tensorflow. URL `https://www.safaribooksonline.com/library/view/building-machine-learning/9781786466587/ch01s02.html`. [Online; accessed April 3, 2018].

Rosenblatt, F. (1957). *The Perceptron: A Perceiving and Recognizing Automaton*. First edition, first issue. Cornell Aeronautical Laboratory, New York. ISBN 85-460-1.

Rosenblatt, F. (1962). *Principles of Neurodynamics*. Spartan, New York.

Rumelhart, D. E., Hinton, G. E. a Williams, R. (1986). Learning internal representations by error propagation. *Explorations in the microstructure of cognition*, **1**.

Silver, D., Huang, A., Maddison, J., Guez, A., Laurent, S., G., D., Schrittwieser, J., Antonoglou, I., Panneershelvam, V. a Lanctot, M. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, **7587**, 484?489.

Springenberg, J. T., Dosovitskiy, A., Brox, T. a Riedmiller, M. A. (2014). Striving for simplicity: The all convolutional net. *CoRR*, **abs/1412.6806**. URL `http://arxiv.org/abs/1412.6806`.

Su, H., Maji, S., Kalogerakis, E. a Learned-Miller, E. G. (2015). Multi-view convolutional neural networks for 3d shape recognition. *CoRR*, **abs/1505.00880**.

Taigman, Y., Yang, M. a Ranzato, M. (2014). Deepface: Closing the gap to human-level performance in face verification.

Wu, Z., Song, S., Khosla, A., Tang, X. a Xiao, J. (2014). 3d shapenets for 2.5d object recognition and next-best-view prediction. *CoRR*, **abs/1406.5670**.

Yu, D., Eversole, A., Seltzer, M., Yao, K., Kuchaiev, O., Zhang, Y., Seide, F., Huang, Z., Guenter, B., Wang, H., Droppo, J., Zweig, G., Rossbach, C., Gao, J., Stolcke, A., Currey, J., Slaney, M., Chen, G., Agarwal, A., Basoglu, C., Padmilac, M., Kamenev, A., Ivanov, V., Cypher, S., Parthasarathi, H., Mitra, B., Peng, B. a Huang, X. (2014). An introduction to computational networks and the computational network toolkit. Technical report.

# Seznam obrázků

# Seznam tabulek

# A. Softwarový manuál

Nasledujúca kapitola tvorí uživateľskú dokumentáciu k programom a skriptom, ktoré sme vytvorili za účelom tejtom práce. Popis algoritmov implementovaných v pomocných programoch sa nachádza v kapitole 3. Popíšeme si podrobný návod, ako pripraviť jednotlivé datasety a spustiť trénovanie klasickej a konvolučnej neurónovej siete.

## A.1 Obsah priloženého CD

Všetky zdrojové kódy a potrebné dáta sú na priloženom CD. V tomto manuáli budeme predpokladať, že je obsah koreňového adresára CD nakopírovaný na disk do adresára *thesis*. Štruktúra CD je nasledujúca:

- *koreňový adresár*
  - *3DTo2DProjector* - program na vytvorenie mriežky
    - *libs* - adresár s externými knižnicami
      - *commons-cli-1.4*
      - *commons-lang3-3.7*
    - *src* - zdrojové súbory
    - *tests* - testovacie súbory
    - `build.xml` - zostavujúci súbor
  - *MeshEditor* - interaktívny editor trojuholníkových sietí
    - `meshEditor.exe` - spustiteľný súbor
    - `...` potrebné knižnice na spustenie
    - *project* - adresár s projektom pre Microsoft Visual Studio
  - *preparedData* - predspracované datové množiny
    - *lines* - adresár s krivkami vo forme postupnosti indexov vrcholov
    - `2dGrid.txt`
    - `labels.csv`
    - `linesDataset3.txt`
    - `linesDataset43.txt`
    - `linesDataset56.txt`
    - `linesDataset74.txt`
    - `nose1dArrayDataset.txt`
    - `nose1dArrayRelativizedDataset.txt`
    - `projectionsDataset.txt`
  - *preprocessingScripts* - obsahuje skripty, ktoré služia na predspracovanie datovej množiny zo surových dát
    - `prepareGrids.py`
    - `prepare1dArray.py`

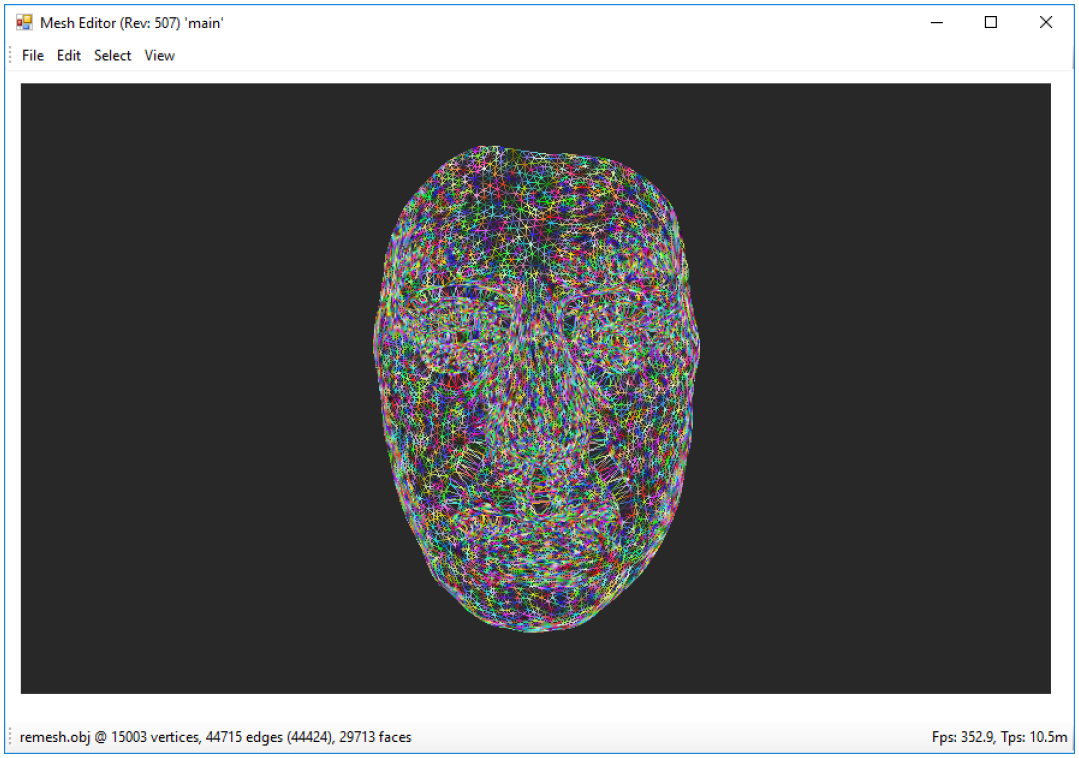
    - `prepareLines.py`
    - `relativize.py`
  - *results* - obsahuje tabuľky s výsledkami experimentov
    - `1dArrayRelativizedResults.pdf`
    - `1dArrayResults.pdf`
    - `gridProjectionResult.pdf`
    - `obliqueLinesResult.pdf`
  - *trainingScripts* - obsahuje adresáre podľa príslušnej datovej reprezentácie so skriptami, ktoré slúžia na trénovanie a testovanie neurónových sietí
    - *1dArray*
    - *gridProjection*
    - *obliqueLines*
  - `DNNForTriangularNetworkAnalysisInGeometricMorphometry.pdf` - text bakalárskej práce vo formáte PDF
  - `readme.txt` - text k doprovodnému CD

Z dôvodu ochrany osobných údajov sme na doprovodné CD nemohli priložiť datovú množinu s trojuholníkovými sieťami naskenovaných tvári. Dáta môžu byť dodané kontaktovaním autora práce alebo laboratória Katedry antropologie a genetiky člověka PřF UK. Ďalej budeme predpokladať, že datovú množinu máte uloženú v priečinku *rawData* v koreňovom adresári s nasledujúcou štruktúrou:

- *rawData*
  - *faces* - adresár s trojuholníkovými sieťami ľudských tvárí
    - `0.obj`
    - `1.obj`
    - `2.obj`
    - `...`
    - `298.obj`
  - *noses* - adresár s trojuholníkovými sieťami selektovanej oblasti nosa
    - `0.obj`
    - `1.obj`
    - `2.obj`
    - `...`
    - `298.obj`

# Požiadavky k inštalácii

Na spustenie skriptov slúžiacich na prípravu datasetov a na samotné trénovanie neurónovej siete je nutné mať nainštalovaný *Python 3*, najlepšie verziu 3.6.3 a vyššie.




Obrázek A.1: Screenshot GUI prostredia programu MeshEditor.

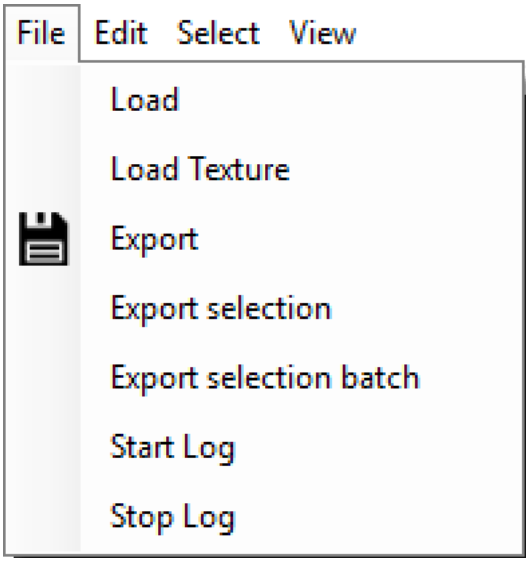
## A.2 MeshEditor

Začnime v priečinku *MeshEditor*. **MeshEditor** (viz A.1) je aplikácia napísaná v jazyku C#, ktorá obsahuje skupinu nástrojov na prácu s mnohouholníkovými sieťami. Uživateľ si do MeshEditoru môže importovať trojuholníkové siete vo formátoch PLY a OBJ, využívať rôzne zobrazovacie režimy. Ďalej umožňuje označiť podčasť meshe (množinu vrcholov), ktoré dokáže exportovať do súboru vo formáte PLY alebo OBJ. Okrem toho vie vyexportovať aj jednoduchý textový súbor s indexami označených vrcholov.

Na inštaláciu stačí otvoriť spustiteľný súbor *meshEditor.exe*, ak sú splnené nasledujúce podmienky:

1. Operačný systém Windows
2. Nainštalovaný framework *.NET*
3. Spustiteľný súbor je spúšťaný z priečinku, kde sa nachádzajú aj externé knižnice *OpenTK.dll* a *OpenTK.GLControl.dll*.

MeshEditor je naprogramovaný ako okenná aplikácia Windows Forms Application. Práca s programom je veľmi jednoduchá a intuitívna, ovláda sa priamo pomocou ovládacích prvkov formulára. Vstup je ovládaný myšou alebo klávesnicou a výstup je realizovaný pomocou okna, resp. do súboru v prípade exportu. Po spustení programu sa uživateľovi zobrazí okno s plátnom, na ktoré sa vykresľuje trojuholníková sieť a hlavné menu. Menu programu obsahuje 4 položky File, Edit, Select a View, ktorých fungovanie si nižšie vysvetlíme.



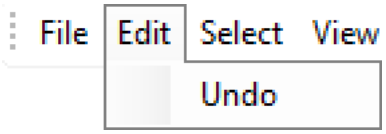

Obrázek A.2: Položka Menu *File* umožňuje ovládať funkcie programu, ktoré súvisia so vstupnými a výstupnými súbormi.

Lišta v spodnej časti programu s textom informuje o názve načítaného súboru, v pravej časti sa nachádza informácia o *fps* (frames per second). V prípade, že prebieha ukladanie súboru na disk, tak sa v pravom rohu zobrazí aktuálny progres ukladania, ktoré prebieha na pozadí. Stlačením tlačidla Stop, označeného symbolom ■, je možné ukladanie zastaviť a ukončiť.

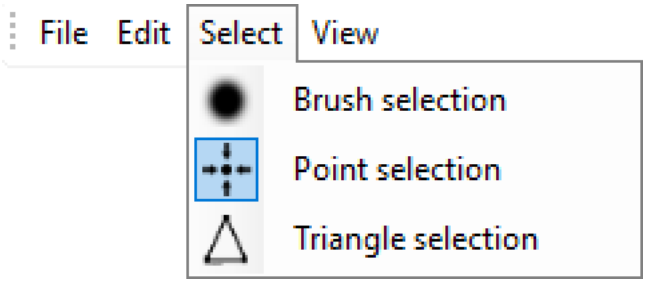
## File

Položka *File* obsahuje po rozkliknutí ďalších 7 podpoložiek, ktoré umožňujú importovať meshe do programu alebo ich exportovať s rôznymi nastaveniami. Na obrázku A.2 vidíme ponuku menu po rozkliknutí *File*.

- **Load** otvorí dialógové okno, kde si uživateľ zvolí ľubovoľný súbor vo formáte OBJ alebo PLY, ktorý následne načíta do okna programu.
- **Load texture** načíta textúru vo formáte BMP, GIF, JPG, PNG alebo TIF.
- **Export** umožňuje uložiť celú trojuholníkovú sieť do súboru s požadovaným formárom - OBJ alebo PLY do cieľového priečinku.
- **Export selection** má rovnakú funkciu ako Export, ale uloží iba požadovanú vybranú oblasť. Vybrané vrcholy sú graficky znázornené červenou farbou.
- **Export selection batch** je funkcia naprogramovaná špecificky pre prácu *s topologicky unifikovanými* modelmi (viz. 2). To znamená, že funguje rovnako ako Export selection, ale exportuje vybrané vrcholy z viacerých sietí naraz. Najprv zobrazí dialóg na výber cieľového priečinku a výstupného formátu. Potom sa zobrazí ďalší dialóg pre načítanie priečinka, ktorý bude rekurzívne prehľadaný a každá nájdená trojuholníková sieť (súbor s príponou .obj alebo .ply) bude spracovaná analogicky ako prvý súbor a uložená




Obrázek A.3: Stlačením Undo po rozkliknutí *Edit* sa aplikácia vráti do stavu pred vykonaním poslednej operácie.




Obrázek A.4: Pre označovanie vrcholov sú určené 3 nástroje: 1 - Brush selection, 2 - Point Selection, 3 - Triangle selection.

do cieľového priečinku. Túto funkciu sme využili pre export oblasti nosov zo všetkých modelov tvárí v datasete. Selekciu vrcholov stačilo urobiť pre jeden model, ostatné boli vybrané ekvivalentne.

- Po stlačení **Start Log** sa vytvorí nový súbor s názvom $log[0-9]+.txt$. Index každého vrcholu, ktorý sa zvolí pomocou nástrojov v políčku menu Select, sa zapíše do súboru. Túto funkcionalitu sme naprogramovali pre prípravu datasetu s povrchovými lomenými krivkami.
- **Stop Log** slúži na korektné ukončenie záznamu označených vrcholov. Výberom položky Stop Log sa prestanú zapisovať označené vrcholy do súboru, na nový zápis je potrebné opäť stlačiť Start Log.

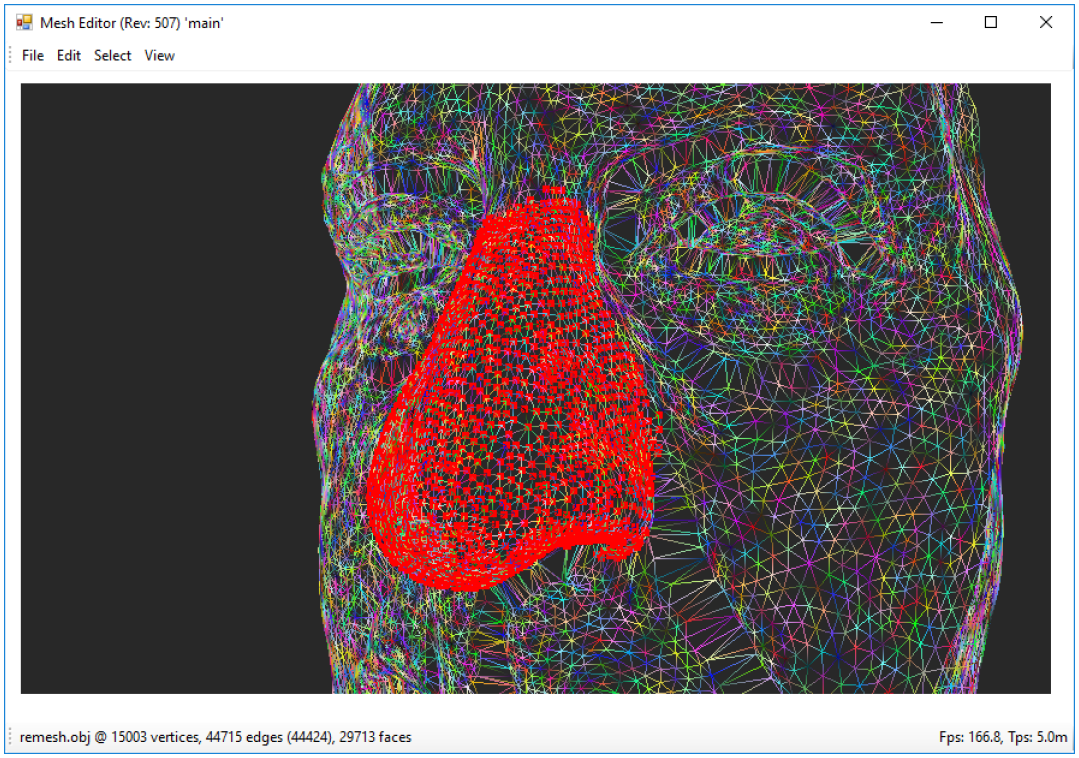
## Edit

V položke *Edit* sa nachádza príkaz **Undo** (viz A.3). Pokiaľ na načítanej meshi prebehli nejaké editácie, je možné sa vrátiť o 5 krokov späť do stavu pred ich vykonaním.

## Select

Funkcia *Select* slúži na označovanie množiny vrcholov 3D objektu. Uživateľ si môže podľa potreby zvoliť z 3 nástrojov na označovanie, ktoré vidíme na obrázku A.4.

- **Brush selection** umožňuje výber väčšieho množstva vrcholov naraz pomocou nástroja pripomínajúceho Brush v programe Photoshop.
- **Point selection** slúži na presnú selekciu práve jedného vrcholu.
- **Triangle selection** označí všetky vrcholy daného trojuholníka.




Obrázek A.5: Označené vrcholy (červená farba) pomocou nástrojov v položke menu Select.

Všetky tieto nástroje sa ovládajú pomocou pravého tlačidla myši. Pri **odznačovaní** vrcholov sa postupuje rovnako, je potrebné len navyše stlačiť na klávesnici CTRL. Označené vrcholy sú zafarbené červenou farbou (obrázok A.5).

### View

*View* umožňuje nastaviť si rôzne zobrazenia 3D modela. Ich funkcia sa ovláda jednoduchým zaškrtnutím vybraného stavu, ako je znázornené na obrázku A.6. Všetky zobrazenia sú súčasťou pôvodného programu, ktorý sme rozširovali.

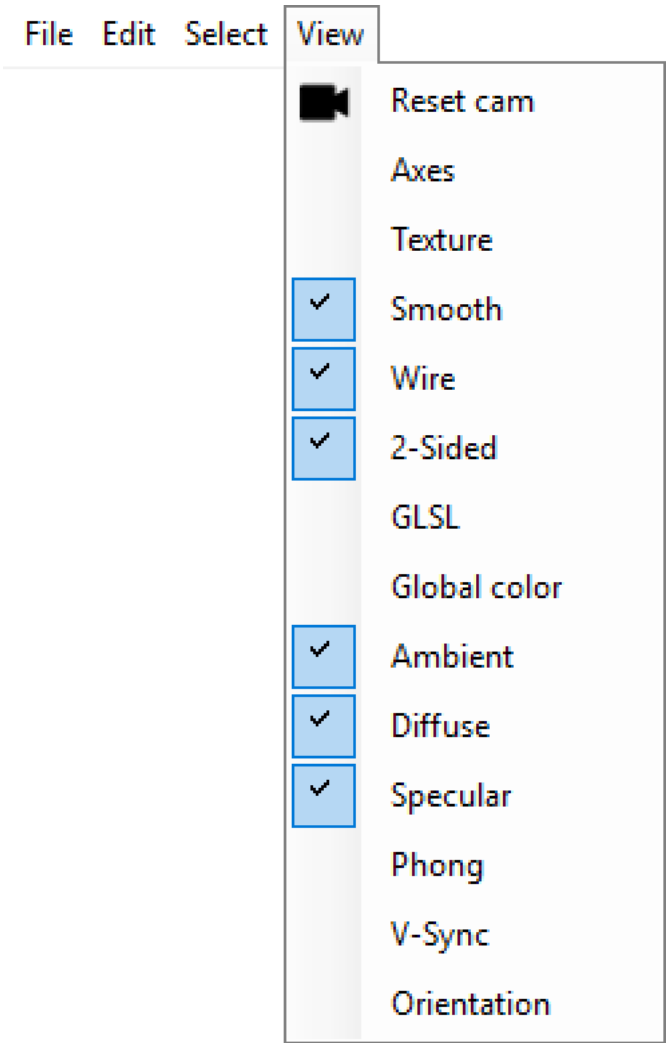
## A.3 3DTo2DProjector

**3DTo2DProjector** je pomocný program, ktorá vytvorí projekciu trojuholníkovej siete do 2D mriežky. Jedná sa o konzolovú aplikáciu a na celý proces inštalácie a spúšťania postačí príkazová riadka. Spustenie vyžaduje akýkoľvek operačný systém s nainštalovanou Javou 9. Ako nástroj na preklad zdrojových kódov je použitý Apache Ant[1], ktorý je potrebné mať nainštalovaný pre nasledujúce kroky. Na otestovanie aplikácie sú v adresári *3DTo2DProjector/tests* trojuholníkové siete nenáročné na hardware, ktoré môžu byť použité na testovanie programu.

1. Ako prvý krok je nutné zmeniť working directory na adresár, kde je súbor `build.xml`.

```
1 $ cd thesis/3DTo2DProjector
```

[1] Ant je dostupný na http://ant.apache.org/




Obrázek A.6: Stlačením Undo po rozkliknutí *Edit* sa aplikácia vráti do stavu pred vykonaním poslednej operácie.

| Option | Voliteľný | Argument |
|---|---|---|
| -d,--distance <arg> | ✓ | názov súboru |
| -h,--histogram <arg> | ✓ | názov súboru |
| -i,--input <arg> | X | názov súboru |
| -n,--neighbours <arg> | ✓ | názov súboru |
| -o,--output <arg> | ✓ | názov súboru |

Tabulka A.1: Zoznam argumentov, ktoré prijíma program 3DTo2DProjector na príkazovom riadku. -i je jediný argument, ktorý je povinný.

2. Spustíme príkaz, ktorým sa vytvorí v koreňovom adresári projektu priečinok *out*, do ktorého sa vygenerujú skompilované súbory s príponou *.class*.

```
$ ant -f build.xml
```

3. Na spustenie vygenerovaných *.class* súborov využijeme utilitu *java*. Musíme špecifikovať, kde má hľadať *.class* súbory a externé knižnice použité v programe. Jednotlivé cesty oddelíme od seba separátorom bez medzery "**:**"(Unixové operačné systémy) alebo "**;**"(Windows).

```
$ java -cp out/production/3DTo2DProjector/:libs/commons-lang3-3.7/commons
    -lang3-3.7.jar:libs/commons-cli-1.4/commons-cli-1.4.jar cz.cuni.mff.java.
    projector.Main [ARGS]
```

Tabuľka A.1 popisuje argumenty príkazovej riadky, ktoré očakáva namiesto ARGS vo vyššie popísanom príkaze. Vstupom programu je trojuholníková sieť vo formáte OBJ, výstupom je výsledok umiestňovacieho algoritmu, a to obdĺžniková mriežka vo formátovanej textovej podobe, ktorej jednotlivé položky obsahujú indexy vrcholov. 0 na výstupe značí prázdnu pozíciu, kde nie je umiestnený žiaden vrchol. Ak sa na príkazovej riadke špecifikuje option -o a ako argument uvedie meno výstupného súboru, tak sa doň zapíše výstup. V opačnom prípade sa ako výstup použije konzola.

Očakáva sa, že vstupom programu budú veľké trojuholníkové siete (v rádoch tisícov vrcholov). Kontrola úspešnosti umiestnenia preto nie je triviálna. Z toho dôvodu je možné vygenerovať pomocné súbory, ktoré hodnotia umiestnenie vrcholov. Hodnotenie je aktivované optionami príkazovej riadky.

## -i,--input

Option -i je jediný povinný option, bez ktorého sa program nespustí. Ako argument obsahuje cestu k vstupnému súboru. Vstupný súbor je trojuholníková sieť vo formáte OBJ, pre ktorú chceme vytvoriť projekciu do mriežky.

## -n,--neighbours

Súbor `neighbours.txt` sa vygeneruje, ak je na príkazovom riadku zadaná option -n. Obsahuje jednoduchý výpis, kde sa pre každý vrchol vytvorí zoznam indexov susedných vrcholov v 3D sieti.

### -h,--histogram

Histogram vypíše počet všetkých vrcholov, ktoré sa nepodarilo umiestniť vedľa svojho suseda. Okrem toho vypíše histogram vzdialeností medzi zle umiestnenými vrcholmi s počtom párov vrcholov, ktoré sú v takejto vzdialenosti umiestnené.

### -d,--distance

Výpis konkrétnych dvojíc indexov vrcholov, ktoré neboli umiestnené priamo vedľa seba, a ich vzdialenosť.

### Príklad konfigurácie príkazovej riadky

Nasledujúce argumenty príkazovej riadky vytvoria projekciu trojuholníkovej siete `test0.obj`, zapíšu ju do súboru `output.txt` a vygenerujú súbory na meranie úspešnosti uloženia `neighbours.txt`, `distance.txt` a `histogram.txt`.

```
1 –i /tests/test0.obj –o output.txt –n neighbours.txt –d distance.txt –h histogram.txt
```

Dosadením do príkazu spustíme program:

```
1 $ java –cp out/production/3DTo2DProjector/:libs/commons–lang3–3.7/commons–lang3
    –3.7.jar:libs/commons–cli–1.4/commons–cli–1.4.jar cz.cuni.mff.java.projector.Main
    –i /tests/test0.obj –o output.txt –n neighbours.txt –d distance.txt –h histogram.txt
```

## A.4 Spracovanie vstupných dát

Samotné naplnenie dát do tensorov sa deje priamo v trénovacom procese, na starosti to majú skripty s názvom `parse.py`. Avšak niektoré datové reprezentácie potrebujú ešte ďalšie úpravy, ktoré si krok za krokom prejdeme v nasledujúcich podkapitolách.

### A.4.1 Rozprestrenie súradníc do poľa

Na získanie prvého datasetu potrebujeme sieti predstaviť súradnice vrcholov každého modelu tváre v datovej množine s oblasťou nosu. Vstupné trojuholníkové siete vo formáte OBJ musíme spojiť do jedného súboru a súradnice usporiadať do jednotnej formy:

- Každá trojuholníková sieť bude na novom riadku súboru.
- Vrcholy budú od seba oddelené čiarkou.
- Súradnice jedného vrcholu budú oddelené medzerou, teda vo forme $xyz$.

Skript, ktorý takto spracuje dáta, je v priečinku *preprocessingScripts* a má názov `prepare1DArray.py`. Skript má jeden povinný argument, a to *cestu ku priečinku* s jednotlivými trojuholníkovými sieťami. Trojuholníkové siete musia mať číselný názov (napr. `0.obj, 1.obj`...), aby boli vo výslednom súbore v správnom poradí a teda aby k nim boli priradené správne označenia pri príprave tensorov. Skript spustíme:

```
$ cd thesis/preprocessingScripts
$ python prepare1DArray.py thesis/rawData/noses
```

Dataset, ktorý je vytvorený týmto skriptom, je uložený v priečinku, *thesis/preparedData*. Názov výstupného súboru je `noseCoordinatesDataset.txt`.

### A.4.2 Relativizácia súradníc

Pri relativizácii súradníc potrebujeme najprv zvoliť jeden vrchol, voči ktorému budeme relativizovať sieť. My sme si zvolili vrchol $v$ s indexom 2471 na špičke nosa, ktorého index sme získali jednoduchým použitím MeshEditoru. Ďalší krok v relativizácii je odčítanie $v$ od všetkých ostatných vrcholov v sieti, aby platilo platiť $v = (0, 0, 0)$. Takto modifikujeme každú trojuholníkovú sieť. Pre upravenie datovej množiny spustíme skript `relativize.py` s cestou k pôvodnej datovej množine a indexu vrcholu $v$ ako argumenty. Tento experiment bol prevádzaný rovnako na datovej množine s oblasťou nosu, takže spustenie skriptu bude vyzerať následovne:

```
$ cd thesis/preprocessingScripts
$ python relativize.py thesis/rawData/noses 2471
```

V priečinku s pôvodnými dátami sa vytvorí adresár *relativized*, kde budú uložené všetky trojuholníkové siete v pôvodnom formáte OBJ s relativizovanými súradnicami. Tento adresár sa potom pre prípravu jednotného datasetu predá skriptu `prepare1DArray.py` tak, ako bolo popísané v predošlej časti.

### A.4.3 Projekcia 3D objektu na 2D mriežku

Projekcia na 2D mriežku pracuje na datovej množine s oblasťou nosa. Príprava datasetu na reprezentáciu pomocou projekcie na mriežku prebieha v dvoch hlavných krokoch:

- **Vytvorenie mriežky** indexov vrcholov v programe 3DTo2DProjector (viz A.3). Tento krok stačí spraviť pre jednu ľubovoľnú trojuholníkovú sieť. Takto vytvorenú mriežku môžete nájsť v adresári *preparedData* v súbore `2dGrid.txt`.
- **Dosadenie súradníc** za indexy vrcholov v mriežke pre každú trojuholníkovú sieť v množine. Na to je pripravený skript `prepareGrids.py`, ktorý požaduje dva argumenty: Cestu k vytvorenej 2D mriežke a cestu k trojuholníkovým sieťam. Vytvorený dataset `projectionsDataset.txt` je uložený v adresári *preparedData*.

  ```
  $ cd thesis/preprocessingScripts
  $ python prepareGrids.py /thesis/preparedData/2dGrid.txt thesis/rawData/noses
  ```

  Nami vytvorená mriežka na datovej množine s nosami má rozmery $87 \times 75$, ktoré sú nastavené defaultne v kóde na tieto hodnoty a na tomto mieste môžu byť zmenené.

  ```
  cols, rows = 87, 75
  ```

### A.4.4 Povrchové lomené krivky

Príprava reprezentácie modelov tvárí pomocou povrchových lomených kriviek opäť vyžaduje dva predspracovacie kroky:

- **Získanie lomených kriviek** v podobe postupností indexov vrcholov sa vykoná pomocou programu MeshEditor a jeho funkcií *Start Log* a *Stop Log*. Tento proces je popísaný v sekcii A.2. My sme extrahovali 9 čiar, každý do samostatného súboru, ktoré nájdete v adresári *preparedData/lines*. Jednotlivé krivky obsahujú na každom riadku index vrcholu. V ďalšom kroku ich musíme zjednotiť a dosadiť za ne súradnice vrcholov.

- **Dosadenie súradníc** a vytvorenie datasetu je zabezpečené zavolaním jedného skriptu, a to `prepareLines.py` s dvomi argumentami. Prvý argument je cesta k priečinku s lomenými čiarami a druhý argument sú trojuholníkové siete, ktorých indexy vrcholov korešpondujú s tými, ktoré sme získali pomocou lomených čiar. V tomto prípade sme používali pôvodný dataset s celými naskenovanými tvárami, ktorý je v adresári *rawData/faces*.

```
$ cd thesis/preprocessingScripts
$ python prepareLines.py /thesis/preparedData/lines thesis/rawData/faces
```

Na začiatku tohoto kódu sa nastaví hodnota premennej `zero_filling` defaultne na 43, čo znamená, že medzi každé dve krivky sa vloží postupnosť 43 nulových vrcholov. Tento parameter je možné upravovať a rôzne testovať, ale je potom potrebné zohľadiť zmenený rozmer vstupu. Taktiež v kóde:

```
for i in range(1, n_lines + 1):
temp = "log" + str(i)
input = open(os.path.join(line_path, temp + ".txt"), mode='r')
for index in input:
        count += 1
        index_list.append(int(index))
        # add zeros to separate lines
        if not i == n_lines:
                for j in range(0, zero_filling):
                        index_list.append(0)
        input.close()
```

prebieha iterovanie cez súbory s krivkami, ktorých počet špecifikuje premenná `n_lines`. Defaultné nastavenie korešponduje s datovými reprezentáciami popísanými v kapitole 2. Vstupné súbory s čiarami musia mať názov v tvare `log[0-9]+.txt`. Výstup sa uloží do adresára *preparedData* do súboru s názvom `linesDataset43.txt`.

## A.5 Trénovanie neurónovej siete

V adresári *trainingScripts* sa nachádzajú skripty v jazyku Python, ktoré sa podieľajú na trénovaní neurónových sietí. Sú rozdelené do troch skupín, vzhľadom na prislúchajúcu datovú reprezentáciu: *1dArray*, *gridProjection* a *obliqueLines*.

Každá skupina obsahuje skripty s rovnakými názvami, ktoré sa líšia svojou implementáciou. Všetky neurónové siete môžu byť priamo spustené z ich pracovného adresára na príkazovej riadke nasledujúcim volaním:

```
$ cd thesis/trainingScripts/1dArray
$ python3 nn1dArray.py
```

```
$ cd thesis/trainingScripts/gridProjection
$ python3 cnnGridProjection.py
```

```
$ cd thesis/trainingScripts/obliqueLines
$ python3 cnnObliqueLines.py
```

Základné fungovanie jednotlivých skriptov sme si popísali v 3.3.3. V tejto podkapitole si ukážeme tie časti skriptov, ktoré modifikujú ako architektúru neurónovej siete, tak vstupné a výstupné súbory.

### A.5.1 Klasická neurónová sieť (1dArray)

#### config.py

V súbore **config** môžu byť nastavené nasledujúce parametre:

- **FEATURES_FILE**: cesta k vstupnému súboru
- **LABELS_FILE**: cesta k súboru s označením datovej množiny
- **OUTPUT_FILE**: názov výstupného súboru
- **ACTIVATION_FUNCTION**: typ aktivačnej funkcie ktorá je aplikovaná na výstup každého neurónu
- **K**: parameter $K$ v K-fold cross validácii
- **LEARNING_RATE**: rýchlosť učiaceho kroku
- **N_BATCH**: počet tréningových vzoriek propagovaných v jednej tréningovej iterácii
- **N_EPOCHS**: koľko krát je sieti predstavený celý dataset
- **N_CLASSES**: počet výstupných tried

#### nn1dArray.py

- Konštanta `MAX_N_LAYERS` určuje inkluzívnu hornú hranicu počtu skrytých vrstiev, pre ktoré sa budú prevádzať experimenty (pre `MAX_N_LAYERS = 3` to budú $1, 2$ a $3$ skryté vrstvy).
- `WIDTH_OF_HIDDEN_NODES` má defaultne priradenú sekvenciu počtov skrytých neurónov, ktorých kombinácie budú otestované.

### A.5.2 Konvolučná neurónová sieť (projectionGrid, obliqueLines)

#### config

V súbore `config.py` môžu byť nastavené nasledujúce konštanty:

- **FEATURES_FILE**: cesta k vstupnému súboru
- **LABELS_FILE**: cesta k súboru s označením datovej množiny
- **OUTPUT_FILE**: názov výstupného súboru
- **ACTIVATION_FUNCTION**: typ aktivačnej funkcie ktorá je aplikovaná na výstup každého neurónu
- **K**: parameter $K$ v K-fold cross validácii
- **KEEP_RATE**: pravdepodobnosť, s akou je neurón ponechaný pri dropoute
- **LEARNING_RATE**: rýchlosť učiaceho kroku
- **N_BATCH**: počet tréningových vzoriek propagovaných v jednej tréningovej iterácii
- **N_EPOCHS**: koľko krát je sieti predstavený celý dataset
- **N_CLASSES**: počet výstupných tried
- **N_ROWS**: výška vstupného obrázku
- **N_COLUMNS**: šírka vstupného obrázku
- **N_DEPTH**: počet kanálov vstupného obrázku

Pri zmene rozmerov vstupného obrázka (projekcie, lomených kriviek) je potrebné aktualizovať aj premenné `N_ROWS` a `N_COLUMNS`. CNN pre reprezentáciu s lomenými krivkami je defaultne je nastavený vstupný súbor so 43 deliacimi nulami medzi jednotlivými krivkami. Pri zmene vstupného súboru (konštanta `FEATURES_FILE`) je potrebné upraviť `N_COLUMNS` podľa tabuľky 4.1.

#### cnnGridProjection, cnnObliqueLines

Pre konvolučné neurónové siete sú pripravené tri architektúry, ktoré je možné obmeniť na začiatku týchto skriptov v zozname importov:

ModelNet1:

```
1 import trainModelNet1 as train
```

ModelNet2:

```
1 import trainModelNet2 as train
```

ModelNet3:

```
1 import trainModelNet3 as train
```