Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra Informatiky a kvantitativních metod

Strojové učení v počítačové grafice

Autor: Adam Ouhrabka

Aplikovaná informatika

Vedoucí práce: Ing. Bruno Ježek, PhD.

Hradec Králové duben 2019

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 30.4.2019 Adam Ouhrabka

Poděkování:
Děkuji vedoucímu práce Ing. Bruno Ježkovi, Ph.D. za pomoc, přístup, ochotu a pozitivní motivaci při odborném vedení této bakalářské práce.

Anotace

Tato práce obsahuje návrh metody pro kombinaci dopředných hlubokých neuronových sítí implementovaných za účelem generování distribuce objektových tříd ve dvojrozměrné grafické scéně. Ve vztahu k tomuto záměru jsou v kontextu počítačové grafiky představena vybraná teoretická východiska. Konkrétně se jedná o oblast strojového učení, neuronových sítí a procedurálního generování obsahu. V praktické části je navrhovaná metoda aplikována na generování jednoduché scény pokoje.

Annotation

Title: Machine learning in computer graphics

The Bachelor Thesis proposes a method of combining multiple deep forward neural networks in order to generate a distribution of objects in a 2D scene. In relation to this intention, the main concepts of machine learning, neural networks and procedural content generation, are presented here. Additionaly these concepts are put into the context of computer graphics and used in a practical example of generating indoor scene.

Obsah

Seznam zkratek a použitých symbolů

PCG	Procedurální generování obsahu
PCGML	Procedurální generování obsahu s pomocí strojového učení
GAN	General adversarial neural networks
CNN	Konvoluční neuronové sítě
NN	Neuronová síť
RNN	Rekurentní neuronové sítě
EDPCG	Procedurální generování obsahu na základě uživatelského chování
VS	Význam scény

1 Úvod

Strojové učení a počítačová grafika jsou oblasti, které v posledních dvou dekádách zaznamenávají rapidní vývoj. V oblasti strojového učení lze identifikovat tři příčiny tohoto progresu. Architektury a metody strojového učení se rozvíjely už od druhé poloviny 20. století, nicméně teprve dostatečný hardwarový výkon posledních let umožnil tyto metody naplno využít, a to i v praxi. Kromě toho byly vymyšleny nové sofistikované modely. Hardwarový výkon je doplňován softwarem na bázi vyšších programovacích jazyků, který umožňuje relativně jednoduše implementovat i složité architektury. Během vývoje oboru byly rovněž vytvořeny datové sady, které jsou s výhodou využívány při trénování a testování modelů. (Abadi et al. 2016).

Podobně i počítačová grafika těží z vývoje nových grafických subsystému, a především z rostoucího zájmu o vizuální prezentaci a interakci ve všech oblastech nasazení počítačových technologií. Mezi obory strojového učení a počítačové grafiky existuje podstatný průnik. Jedná se například o techniky počítačového vidění či zpracování obrazu. Jádrem teoretické části textu je představení tohoto průniku a specifické podoblasti, jíž lze označit jako „Generování grafického obsahu s pomocí metod strojového učení“ - v zahraniční literatuře se jedná o pojem „Procedural content generation via machine learning“, známý pod zkratkou PCGML.

Bakalářské práce je členěna následovně: V první části textu je v obecných rysech představena oblast strojového učení s explikací metod, které jsou využívány v praktické části. Výčet aplikací založených na těchto metodách (zejména na neuronových sítích) je pak striktně omezen na oblast počítačové grafiky. Dále je v kapitolách tři a čtyři vytvořena stručná rešerše postupů a metod, které jsou používány či diskutovány v souvislosti s procedurálním generováním grafického obsahu. Nejprve je tato oblast představena obecně a poté je text zaměřen na aplikace vybraných technik PCGML.

Implementačním otázkám se věnuje pátá kapitola, popisující typické technologie a prostředí, které jsou vhodné pro implementaci metod strojového učení. Tyto technologie jsou využity v praktické části bakalářské práce.

V šesté kapitole je prezentován návrh a implementace metody, která za využití architektur dopředných neuronových sítí predikuje distribuci objektů ve dvojrozměrné scéně na základě naučených dat. Softwarová implementace navržené metody je otestována v kapitole sedm a vyplynulé výsledky jsou zhodnoceny v závěru, kde jsou shrnuty i poznatky z teoretické části práce a jsou nastíněny možná rozšíření zkoumané oblasti.

2 Strojové učení v obecných rysech

Z formálních definic strojového učení je vybráno tradiční znění Toma Mitchella, který definuje strojové učení následujícím způsobem. *„Stroj (počítačový program) se učí plnit třídu zadání T ze zkušenosti E a s úspěšností P v tom případě, když se úspěšnost plnění zadaných úkolů T zlepšuje s využitím znalostí nabytých ze zkušenosti E.“* (Mitchell 1997). K základním praktickým úlohám, pro něž jsou algoritmy strojového učení používány patří klasifikace, predikce, expertní systémy, popřípadě kontrola agenta autonomního systému.

Algoritmy strojového učení se podle způsobu učení dělí na dvě skupiny. Za prvé je to skupina tzv. „učení s učitelem“ angl. „supervised“. Cílem algoritmů této skupiny je aproximace neznámé komplexní funkce $y = f*(x)$ (Goodfellow et al. 2016). S menší nebo větší úspěšností se tato aproximace děje za pomocí předkládání vstupních a výstupních vektorů hledané funkce. Celý aparát takového optimalizačního procesu je často parametrizovatelný takzvanými hyperparametry, přičemž celková úspěšnost nalezení příslušné funkce je na těchto parametrech přímo závislá (Jordan a Mitchell 2015). Druhou skupinou jsou algoritmy, které se takzvaně „učí bez učitele“ angl. „unsupervised“. Ghahramani charakterizuje tento druh učení jako algoritmické hledání souvislostí v datech, která mají velice složitá rozdělení (Ghahramani 2004).

Jednoduché funkce lze v zásadě aproximovat i jinými čistě programově-algoritmickými metodami, to je nicméně často neefektivní, popřípadě u komplexnějších funkcí vyloženě nereálné. Následuje výčet hlavních argumentů pro upřednostňování algoritmů strojového učení, namísto implementace jiného procedurálně či obecně programově laděného systému.

Mitchellem definovaná zkušenost E je v oblasti strojového učení reprezentována tréninkovými daty. Pro trénovací sadu je důležité to, jaké formy nabývá a v jakém pořadí vstupují jednotlivá data do algoritmu. Především je pak podstatné *„…jak dobře data reprezentují distribuci příkladů, přes které musí být výsledný systém evaluován úspěšností P.“* (Mitchell 1997). V takto definované tréninkové sadě, lze pak s pomocí vhodných algoritmů strojového učení hledat obecné vztahy a korelace, které by tradiční analýze zůstaly skryté. To znamená, že by se jen těžko hledala přiměřená algoritmizovatelná logika a mapování vstupu funkce na její výstup. Tak zní první argument. Vzhledem k záměru této práce lze dodat, že tyto skryté vztahy mohou

být později užity ke generování nových vzorků při zachování obecné charakteristiky a distribuce trénovací sady.

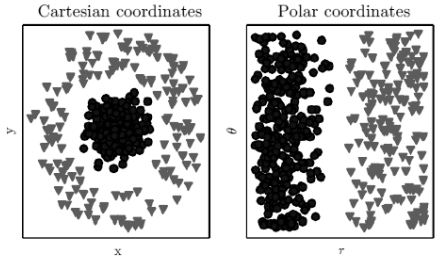
Mezi další důvody pro preferenci algoritmů strojového učení patří dle Nilssona enormní velikost datové sady, která znemožňuje syntézu do tradičního programu. Dalším argumentem je dynamická proměnlivost dat, na kterou musí být systém schopný v ideálním případě reagovat. A konečně jsou některé úlohy definovatelné pouze pomocí předkládání četných příkladů (Nilsson 1998). Většinu z těchto jedinečných náležitostí lze demonstrovat na současném projektu Alterego, který mapuje neuromuskulární signály kolem úst na omezený slovník pojmů s úspěšností 92% (Kapur et al. 2018). Lze si jen stěží představit, že by tato funkce byla nalezena a prováděna programovou cestou.

2.1 Rozlišení diskriminativních a generativních modelů

Kromě rozlišení strojového učení dle způsobu učení, lze zavést ještě dělení na diskriminativní a generativní modely. Zatímco u diskriminativních modelů jde o mapování vstupů na konkrétní výstup, kterým může být konkrétní třída (klasifikace) či skalární hodnota (regrese), generativní přístup modeluje pravděpodobnostní relace mezi proměnnými daného modelu (Jebara 1996). Pro N proměnných lze v těchto modelech nalézt úplnou sdruženou pravděpodobnost ve formě $p(x_1, \dots, x_n)$. Existuje-li vyjádření takové distribuce, lze na základě této formule odvozovat další hodnoty proměnných například pomocí Bayesovských pravidel. Goodfellow definuje generativní modely jako *„...modely, které se na základě tréninkové distribuce dat tvořených množinou p_{data} naučí odhad takové distribuce reprezentovat“*. (Goodfellow et al. 2014) Generativní modely existují v několika formách. Může se jednat o modely hlubokého učení či takzvané grafické modely potažmo bayesovské sítě se směrovými relacemi mezi proměnnými, Markovovy modely, popřípadě Markovova nahodilostní pole (Jebara 2004). Na příkladu klasifikace rastrových obrázků jednociferných čísel demonstruje Revow zásadní praktický důsledek při použití generativních modelů, totiž ten, že najdeme-li generativní model pro daný systém, našli jsme i model, který je schopen generovat nové vzorky (Revow et al. 1996).

Diskriminativní modely se v jádru svého fungovaní snaží aproximovat ideální hranici, která žádoucím způsobem odděluje vícedimenzionální datovou distribuci.

Příklad takového oddělení je znázorněn na Obr. 1. Mezi diskriminativní algoritmy patří typicky neuronové sítě, support vector machines, lineární regresní algoritmy a další (Jebara 1996). Primární užití diskriminativních algoritmů lze vidět v datové klasifikaci, popřípadě regresi.



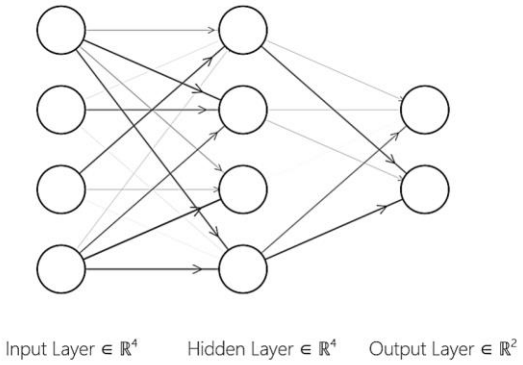
Obr. 1: Demonstrace typického vymezení hranice při klasifikování dat. Převzato z (Goodfellow et al. 2016).

2.2 Metody strojového učení s omezením na neuronové sítě

V následujících kapitolách budou postupně představeny konkrétní metody strojového učení s omezením na formy neuronových sítí. Nejprve je popsána standardní architektura umělé neuronové sítě (ANN), poté budou popsány konvoluční neuronové sítě (CNN). Závěrem je ve stručnosti zmíněna další v současnosti používaná architektura anglicky známá jako Generative adversarial networks (GAN). U každého modelu jsou uvedeny příklady jejich aplikací v oboru počítačové grafiky.

2.2.1 Hluboké dopředné neuronové sítě

Hluboké neuronové sítě jsou jednou ze základních dosud používaných metod strojového učení. Je to aparát, který má charakter funkce pracující s reálnými čísly a lze jej charakterizovat jako „*masivně paralelní procesor*“ (Olej a Hájek 2010). Typické schéma hluboké neuronové sítě, ze kterého vycházejí i komplikovanější architektury je ukázáno na Obr. 2.



Obr. 2: Schéma neuronové sítě. Vygenerováno pomocí nástroje (Nail 2019).

Ze schématu výše vyplývají typické znaky modelu neuronové sítě. Na grafu vícevrstevnaté dopředné neuronové sítě lze vždy identifikovat vstupní vrstvu (Input Layer), výstupní vrstvu (Output Layer) a několik vrstev skrytých (Hidden Layer). Každá z těchto vrstev může mít různý počet neuronů.

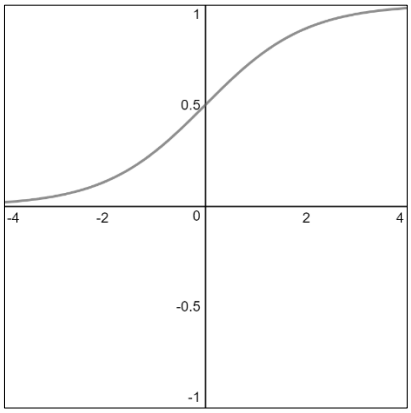
Neuron potažmo perceptron funguje na principu funkce na jejímž vstupu jsou vektory hodnot x a w z předcházející vrstvy neuronové sítě. Na výstupu je poté jedna skalární hodnota. Propojení neuronů je ve schématu reprezentováno hranami grafu, které zároveň zastupují váhy navázané na jednotlivé neurony. Obvykle je každý neuron napojen na všechny neurony vrstvy následující. Pro každý neuron je při dopředném šíření „signálu" spočten součet všech součinů navázaných vah a příslušných skalárních hodnot neuronů předcházející vrstvy. Tento součet poté vstupuje do aktivační funkce Φ, která tvoří skalární výstupní hodnotu neuronu v definičním oboru Φ. Parametr θ je v podstatě neuron inicializovaný pro každou vrstvu neuronové sítě, obvykle má hodnotu jedna a je propojen s neurony příslušné vrstvy. θ přidává prahovou hodnotu neuronu angl. bias. *„Je-li hodnota vstupního signálu neuronu nižší, než hodnota prahová, je na výstupu z neuronu signál odpovídající pasivnímu stavu neuronu"* (Olej a Hájek 2010). Formálně je výstup jednoho neuronu vyjádřen v Rov. 1, kde je počet neuronů předcházející vrstvy roven n. Na výstupu neuronu je tedy skalární hodnota, která zpětně figuruje jako vstupní parametr pro další napojené neurony.

$$y = \Phi(\sum_1^n x_i w_i + \theta) \qquad \text{Rov. 1}$$

Aktivačních funkcí existuje celá řada a mají zásadní charakter na fungování neuronové sítě. Tyto funkce mohou mít za důsledek lineární i nelineární charakter

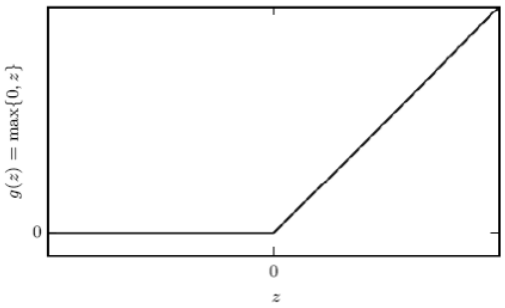
klasifikačních oblastí. Příkladem nelineární aktivační funkce je sigmoida (Olej a Hájek 2010). Její předpis vystihuje Rov 2., graf je znázorněn na Obr. 3.

$$f(y_i) = \frac{1}{1+e^{y_i}} \qquad \text{Rov. 2}$$



Obr. 3: Průběh funkce sigmoid.

Aktuálním trendem v oblasti hlubokého učení je však používání jiné aktivační funkce, která se anglicky označuje jako „rectified linear unit“ zkráceně ReLU, kde $f(x) = \max\{0, x\}$ (Goodfellow et al. 2016). Obvykle se tato funkce implementuje pro skryté vrstvy. Graf ReLU je ukázán na Obr. 4. Myšlenka neuronu v informatice původně vychází z fungování biologického neuronu. Dle Glorota je to však teprve užití aktivační funkce ReLU, díky, které se podařilo „umělý“ a organický neuron vzájemně přiblížit, co do principu fungování. V této souvislosti diskutuje především pojem „řídkosti aktivace“, při které v rámci šíření signálu zůstane mnoho neuronových jednotek na svém výstupu nulových. Neurony jsou pak takzvaně pasivní. Je to podstatný a pozitivní důsledek fungování neuronové sítě při používání ReLU. (Glorot et al. 2011). Aplikace této aktivační funkce má rovněž za důsledek rychlejší trénování (Bhandare et al. 2016).



Obr. 4: Průběh funkce ReLU. Převzato z (Goodfellow et al. 2016).

Podstatnou komponentou neuronové sítě je aktivační funkce poslední vrstvy. Zde záleží na tom, jaký charakter výstupu je žádoucí. Obecně se však užívá již prezentovaná funkce sigmoid. Výstup neuronové sítě klasifikuje její vstup do konkrétní proměnné. Je-li tato proměnná diskrétní, to znamená, že představuje například příslušnost vstupních dat do n tříd, měla by výstupní vrstva vyjadřovat vektor pravděpodobnosti příslušnosti vstupního objektu do těchto tříd. Pro tento konkrétní účel se využívá funkce softmax odvozená od sigmoidy (Goodfellow et al. 2016).

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \qquad \text{Rov. 3}$$

Důsledkem užití funkce softmax je, že na výstupu neuronové sítě bude vektor v_1, který bude vyjadřovat pravděpodobnosti pro jednotlivé třídy určené pořadím v rámci vektoru a díky normalizační složce v čitateli bude složkový součet v_1 roven jedné.

2.2.2 Učení neuronových sítí

Podstatu učení neuronové sítě shrnuje Olej, který jej charakterizuje jako proces, ve kterém *„...dochází k modifikaci synaptických vah za účelem dosažení co možná největší shody mezi skutečným a požadovaným výstupem.“* (Olej a Hájek 2010). V případě „učení s učitelem“ probíhá učení postupným předkládáním m dvojic $(x \rightarrow y)$ do neuronové sítě. Na začátku tohoto procesu jsou všechny váhy iniciovány jako náhodné hodnoty blízké nule. Vstupní data poté postupně procházejí transformacemi jednotlivých vrstev neuronové sítě až na její výstupní vrstvu. Ve chvíli, kdy tímto způsobem projde celá datová sada, hovoříme o takzvané jedné epoše učení (Olej a Hájek 2010). K modifikaci vah nemusí však dojít až po dokončení celé epochy, ale váhy lze adaptovat už po projití m vstupních dat – anglicky je m nazýváno jako „batch-size“. Hodnota m má přímý vliv na kvalitu a rychlost učení (Keskar et al. 2016). Po projití m dat je tedy na podmnožině dat t, které prošly neuronovou sítí, spočítán výstup chybové funkce E_t. Jde o rozdíl výstupních hodnot vyprodukovaných neuronovou sítí $\hat{y}_k(t)$ a správných hodnot y_k, vyplynulých ze vstupních dat. Chybových funkcí existuje v oboru strojového učení celá řada. Rov 5. představuje předpis jedné z nich, a to konkrétně „střední kvadratické chyby“. Použité proměnné odpovídají zde

zavedené symbolice. Tento konkrétní tvar je použitelný v případě, že na výstupní vrstvě je jeden neuron (Olej a Hájek 2010).

$$E_t = \sum_{k=1}^{m} [\hat{y}_k(t) - y_k]^2 \qquad \text{Rov. 4}$$

Samotný algoritmus učení spočívá v minimalizaci vybrané chybové funkce pomocí adaptace funkčně závislých vah w. Vzhledem k počtu parametrů vah je nevhodné, ba nemožné hledat (lokální) minimum E_t analyticky, například za pomocí druhé derivace funkce. Místo toho se obvykle ve strojovém učení aproximuje lokální minimum iterativně, kdy se parametry ladí v záporném směru nalezeného gradientu E_t. Znamená to, že ve chvíli, kdy je spočítána hodnota chybové funkce, je nutné zpětně pro všechny váhy vypočítat vhodné Δw. V kontextu neuronových sítí se jedná o metodu nazvanou „metoda zpětného šíření chyby“ angl. „backpropagation“. Rov. 5 formálně vyjadřuje hledanou hodnotu, kde η figuruje jako rychlost učení. Tento hyperparametr definuje velikost postupu ve směru gradientu. Rov. 5 lze nadále rozvinout a najít potřebné vztahy pro korekci vektorů vah v závislosti na E_t a užitých aktivačních funkcích. (Olej a Hájek 2010). Jsou-li všechny hyperparametry nastaveny vhodným způsobem, pak systém neuronové sítě konverguje po několika epochách k lokálnímu minimu chybové funkce, což znamená, že adaptoval všechny své parametry (synaptické váhy) pro replikaci $x \rightarrow y$ datové sady a v ideálním případě bude použitelný i na data mimo tuto sadu.

$$\Delta w_{ij} = -\eta \frac{\partial E_t}{\partial w_{ij}} \qquad \text{Rov. 5}$$

Podobně jako existuje několik chybových funkcí, existují i různé varianty učících algoritmů. Zde byl popsán pouze základní princip metody zpětného šíření chyby pomocí postupného sestupu po gradientu chybové funkce.

2.2.3 Konvoluční neuronové sítě

Mezi nejúspěšnější architektury aplikované v počítačové grafice posledních let patří hluboké konvoluční neuronové sítě (CNN), které se pravidelně umísťují na předních příčkách soutěže ILSVRC[1] (Goodfellow et al. 2016).

[1] Large Scale Visual Recognition Challenge - soutěž v rozpoznávání objektu na rastrovém obrázku.

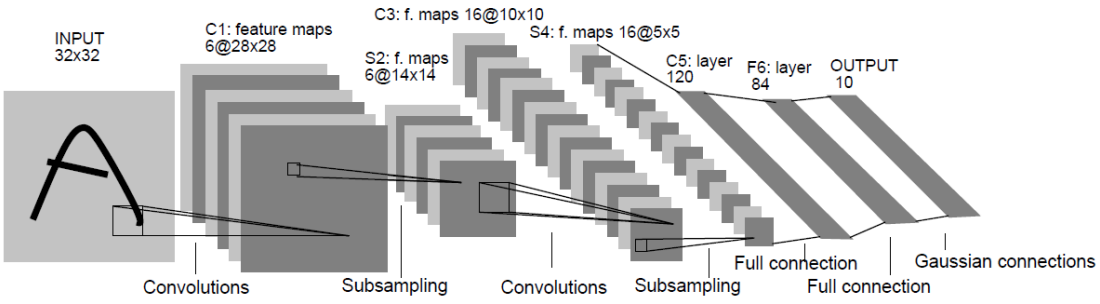
CNN jsou propojením dvou silných konceptů dopředné neuronové sítě a principu konvoluce. Přestože aplikace z poslední doby užívají CNN například pro rozpoznání řeči či práci s jazykem, používají se obvykle pro obrazová nebo časová data. Formálně jsou definovány jako *„...sítě, které namísto maticového násobení užívají alespoň v jedné vrstvě konvoluci.“* (Goodfellow et al. 2016).

Máme-li dvoudimenzionální pole dat I aplikujeme konvoluční operaci $(*)$ s využitím konvolučního jádra K následovně:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n)$$

Rov. 6 operace konvoluce pro pole I (**dim**=$\mathbb{R}^2$) a jádro K (**dim**=$\mathbb{R}^2$).

Z formálního vyjádření vyplývá, že se jedná o vzájemné pronásobení matice vzoru I a matice jádra K, přičemž výsledek tohoto násobení je agregován do skalární hodnoty a uložen do výsledné matice S. Motivací k provedení této operace je agregace specifických znaků (informací) vstupních dat. Při konvoluci se redukuje velikost vstupních dat, ale agregovaná informace pro účely klasifikace je zachována. Díky této redukci může CNN efektivně klasifikovat i barevné obrázky většího rozlišení. Rovněž je pomocí konvoluce do určité míry zajištěno, že varianty dat jedné třídy budou do neuronové sítě vstupovat jako invariantní co do velikosti, občasných posunů a jiných distorzí (LeCun et al. 1998). Důvodem této skutečnosti je to, že se ve fázi učení naleznou taková jádra (tj. jejich váhy), která ve spojení s příslušnými vstupy vyselektují typické znaky, podle kterých bude možné určit třídu. Kromě těchto jader figurují ve výsledné klasifikaci ještě váhy klasických vrstev, které jsou napojeny na výstup konvoluční části modelu.



Obr. 5: Konvoluční architektura LaNet-5.

Na obrázku výše je zobrazena architektura konvoluční sítě, která byla svými autory nazvána jako LeNet-5 (LeCun et al. 1998). Kromě klasických neuronových vrstev, vystupují v těchto architekturách ještě další jednotky v podobě konvolučních a „agregačních“ angl. „subsampling“ vrstev.

Vrstva, které se říká konvoluční, aplikuje operaci konvoluce na předchozí vrstvu modelu. Efektivita CNN spočívá mimo jiné v tom, že se konvoluce aplikuje hned několikrát. To znamená, že jednotlivá jádra vyústí v procesu zpětného šíření chyby do navzájem různých konfigurací. Důsledkem toho je „selekce“ odlišných znaků. Počet konvolučních jader figuruje v softwarových knihovnách pro strojové učení obvykle jako hyperparametr. S řetězovitou aplikací konvolučních vrstev se tak uvnitř modelu navyšuje počet konfigurací konvolučních jader a jejich produktů. Zároveň se snižuje velikost matice těchto produktů, což je přirozený důsledek konvoluce, při níž se výsledná velikost $S = \frac{(W-K+2P)}{S} + 1$, kde W = velikost vstupu, K = velikost jádra, P = odsazení, S = velikost kroku.

V posloupnosti vrstev CNN následuje obvykle za konvoluční vrstvou transformace, která na její výstup aplikuje aktivační funkci, čímž je dosaženo nelinearity výstupu. Třetí v tomto typickém CNN tripletu je takzvaná subsampling či pooling vrstva. Funkce této vrstvy se může lišit. Obecně jde ale o další redukci velikosti konvolučního výstupu na základě okolí. Hodnota výstupu tak může být například maximální numerická hodnota prvku uvnitř okolí či průměrná hodnota prvků. Účelem této operace je, aby se informace konvolučních produktů staly ještě více agregovanými a invariantními (Goodfellow et al. 2016). V některých případech je za tímto tripletem přítomna ještě tzv. „Dropout vrstva“, která při šíření dopředného „signálu“ nahodile deaktivuje některé „neurony“. Dle Krizhevského zamezuje tato technika jevu takzvaného přeučení a obecně přispívá k přesnosti modelu (Krizhevsky et al. 2012).

Konvoluční sítě mají své aplikace především v počítačové grafice, konkrétně počítačovém vidění. Prezentovaný model LaNet-5 byl svými autory v roce 1998 použit ke strojovému čtení psaného textu. Chyba se v rámci testovací sady pohybovala v řádech desetin procent (LeCun et al. 1998).

Současné aplikace shrnuje ve svém článku Bhandare. Většina aplikací se týká dvourozměrných rastrových obrázků o třech barevných kanálech. CNN byly úspěšně použity pro rozpoznání tváře a klasifikaci objektů obecně. Z implementovaných řešení lze poukázat na klasifikátor ImageNET, jehož vstupní datová sada obsahuje patnáct

milionů obrázků patřících do 22 000 kategorií. Přesnost tohoto modelu byla 62.5 %. (Krizhevsky et al. 2012). Dalším odvětvím aplikace je takzvaný „scene labeling“, v němž jde o identifikaci a zaměření objektů ve scéně.

Vývoj CNN nadále spočívá v zacházení s hyperparametry a ladění architektur modelu. Na základě Bhandarova souhrného článku lze konstatovat, že existuje mnoho variant CNN, plnící cíle klasifikace, segmentace, ale i další. (Bhandare et al. 2016).

2.2.4 Generative adversarial neural networks

Ve stručnosti jsou zde zmíněny i aktuálně populární generativní modely uvedené Ianem Goodfellowem nesoucí anglický název Generative Adversarial Neural Networks zkráceně GAN. Základním principem jejich fungování je princip „maximální pravděpodobnosti“ tzn. *„...nalezení takových parametrů pro model, aby maximalizovaly pravděpodobnost tréninkových dat“*.

Goodfellow navrhl generativní model skládající se ze dvou komponent – generátoru a diskriminátoru. Generátor v každé iteraci učení ladí své váhy tak, aby transformoval prvotně iniciovaný šum na vzorky, které napodobují distribuci tréninkových dat. V tom mu pomáhá diskriminátor, který tyto vzorky průběžně klasifikuje na 1 nebo 0. Vzniklá chyba této klasifikace se projeví v příští iteraci generování. Obě komponenty tak stojí proti sobě a finální naučení redistribuce plyne ze vzniklé tenze mezi nimi (Goodfellow 2016).

Většina aplikovaných GAN architektur se užívá při generování obrazových dat a uplatňuje v sobě konvoluční mechaniku. Hovoří se pak o tzv. DCGAN (Goodfellow 2016). Tyto modely byly kromě generování nových obrázků založených na naučených datech použity například na obarvení obrázků v infračerveném spektru (Suarez et al. 2017). Obecně lze pomocí GAN realizovat transformace obrázků například transformovat fotografii denní scény na scénu noční (Isola et al. 2016).

3 Procedurální generování

V oboru počítačové grafiky existují v podstatě tři způsoby pro definici obsahu modelované scény. Ať už uvažujeme dvojrozměrné, popřípadě trojrozměrné zobrazení, tak vlastnosti vykreslovaných objektů této scény musí být vždy předem zaneseny v příslušných datových strukturách, které posléze figurují v rámci vizualizačního řetězce. Zachycení geometrie a umístění objektu v souřadnicích scény může být definováno buď staticky, míněno „ručním“ modelováním objektů v příslušném softwaru anebo algoritmicky, kdy se vzhled a transformace objektů odvíjejí od sekvence příkazů daného algoritmu. Třetím způsobem může být modelování pomocí skenování reálného světa. Podobné rozdělení se týká například i textur a dalších elementů, které scénu spoluvytvářejí. Druhý ze zmíněných přístupů lze označit termínem procedurální generování obsahu angl. procedural content generation (PCG).

Zkratka PCG bývá používána především v oblasti počítačových her, kde najdou tyto algoritmy široké uplatnění. Tyto algoritmy jsou však použitelné i v rámci budování virtuálních prostředí, simulací, vizualizací, popřípadě v oblasti dekorativního a technického designu. Nutno zmínit, že PCG se ve své definici neomezuje jen na generování grafického obsahu jako takového, nýbrž se tento pojem používá i pro algoritmické generování jakéhokoliv obsahu. Tím může být například obsah textový, animovaný, popřípadě hudební. Yannakis zahrnuje do oblasti PCG i generování personalizovaného obsahu, které se uplatňuje například v rozhraních pro e-commerce a uživatelských rozhraních obecně (Yannakakis a Togelius 2011).

Univerzální vlastností většiny algoritmů pro PCG je, že ve svém těle pracují s faktorem nahodilosti, který je nezbytný pro zajištění diverzity obsahu. Základní metodou pro naplnění tohoto kritéria je například implementace pseudonáhodných čísel v podobě šumu (Hendrikx et al. 2013). Mezi další podstatné vlastnosti PCG algoritmů patří rychlost, spolehlivost, kontrolovatelnost a uvěřitelnost (Togelius et al. 2016). Smith ve svém článku užívá v kontextu PCG výstižný pojem „řízená nahodilost“ (Smith 2015). Nahodilost má v PCG efekt spíše na dílčí části negenerovaného obsahu než na samotný celek, který musí být uvěřitelný, popřípadě hratelný. Demonstrováno na příkladu stromu, u kterého může být distribuce a geometrická variabilita jednotlivých listů náhodná do té míry, do kdy se objekt jeví

stále jako objekt dané třídy, v tomto konkrétním případě – jako strom. Termínem „řízená“ je vystihnuto kritérium pro smysluplnost, ze kterého vyplývá další podstatná vlastnost PCG algoritmů, kterou je přítomnost určitých omezení, které v jádře algoritmu zajišťují, že obsah bude funkční a uvěřitelný.

3.1 Klasifikace PCG algoritmů

Algoritmy PCG se dělí na parametrizovatelné a na ty, které ke svému fungování žádnou parametrizaci nepotřebují. I minimální parametrizace jako například definice počtu iterací spadá do skupiny parametrizovatelných PCG algoritmů, které nabízejí designérovy kontrolu nad finální podobou generovaného obsahu (Michelon De Carli et al. 2011).

Další dělení definující PCG algoritmy se týká toho, zdali má generování zpětnou vazbu na uživatelské interakce nebo takovou vazbu nemá, popřípadě zdali se obsah generuje při běhu aplikace (online) nebo slouží pouze jako podpůrný nástroj designu při vývoji (offline) (Togelius et al. 2010). Jedná-li se o algoritmus, který přímo reaguje na interakci s uživatelem a v rámci běhu aplikace dotváří „personalizovaný obsah“, označuje ho Yannakakis zkratkou EDPCG, kterou lze přeložit jako „Generování obsahu na základě uživatelského chování“ (Yannakakis a Togelius 2011).

3.2 Tradiční metody procedurálního generování

Tradiční PCG algoritmy jsou založeny na poznatcích z mnoha oblastí matematiky a informatiky. Pro generování grafického obsahu se tradičně používají evoluční algoritmy, Lindenmayerovy systémy, fraktály, a další specifickým způsobem aplikované postupy, které pro každý svůj běh negenerují jiný monotematický obsah. K tomu lze podotknout, že *„Všechny tyto metody mají společné to, že tyto algoritmy, parametry, omezení a cíle, které se podílí na vytváření obsahu, jsou v podstatě ručně definované svými tvůrci.“* (Summerville et al. 2017). Příkladem takového produktu generování je Obr. 6, kde je vyobrazena bonsai vygenerovaná pomocí systému založeném na Lindenmayerově systému a programové nadstavbě (Boudon et al. 2003).

Obr. 6: Vlevo reference, dle které Boudon postupoval. Vpravo nagenerovaný produkt. Převzato z (Boudon et al. 2003).

Stěžejní proměnou PCG algoritmů je podoba reprezentace obecných genotypů, na základě kterých vzniká vždy originální entita (fenotyp). Takové genotypy mohou reprezentovat jak obecné geometrie objektů, tak například obecné vztahy mezi nimi. Tato biologická paralela vychází z často aplikované metody PCG, kterou jsou genetické algoritmy.

3.3 Motivace, účel a vize používání PCG

Klíčovou součástí moderních grafických aplikací, popřípadě počítačových her jsou namodelované objekty plnící buď vizuální či interaktivní funkci. Nevyužije-li se v rámci produkce PCG metod je všechen obsah modelován ručně. Taková praxe výrazně zvyšuje cenu produktu a prodlužuje dobu jeho vývoje. Na druhé straně je možné potenciál PCG použít pouze tam, kde prvek náhody dotváří funkci herního mechanismu nebo vytváří diverzitu, která nenarušuje účel obsahu (nebrání například v postupu či jinak nemate uživatele). Generovaný obsah tedy může být na základě tohoto kritéria rozdělen na funkční a čistě kosmetický (Togelius et al. 2010).

Vize a cíle současného bádání PCG v grafické oblasti definuje ve svém článku Julian Togelius. Jako první vize je nadneseně představena univerzální metoda na vytvoření komplexních virtuálních světů určitého žánru. I v případě, že by se jednalo pouze o statický grafický obsah takového světa dá se společně s Togeliusem konstatovat, že podobný algoritmus je nerealizovatelný i v rámci vzdálené budoucnosti. Komplexita takového úkolu je enormní. Problémů, které stojí v cestě takovému zadání je hned několik a tvoří primární úkoly v rámci výzkumu PCG (Togelius et al. 2013).

Generovaný obsah se potýká s originalitou, nedostatečnou diverzitou a kontextuální smysluplností. To vše jsou znaky kreativity v rámci ručního modelování. Klíčová otázka zní, zdali se v případě PCG dá kreativita algoritmizovat a v případě generování prostřednictvím strojového učení stochasticky modelovat i v rámci komplexních celků. Kritéria výpočetní kreativity, která významně korelují s požadavky na PCG definoval ve své práci Pereira. Hlavní znaky spatřuje v uchování modelu znalostí o dané třídě, schopnost redistribuce znalosti do nových spojení, schopnost kriticky zhodnotit produkt (validace), uvážení kontextu, do kterého je tvorba zasazena a užití konvergentních i divergentních postupů při kreaci (Pereira 2008). Všechny tyto podmínky, které Pereira postuluje pro svůj model („Creative general problem solver“) jsou realizovatelné a byly realizovány prostřednictvím generativních modelů strojového učení například za použití GAN architektur neuronových sítí v oblasti rastrových obrázků. Jako konkrétní příklad lze uvést GAN model pro generování lidských tváří ve vysokém rozlišení (Karras et al. 2018).

Togelius ve svém článku nadále vybízí k syntéze metod, které byly vynalezeny v rámci generování jednotlivých objektů jako například terénu, budov či interiéru, do komplexního parametrizovatelného generátoru. Interakci mezi jednotlivými generátory podtrhuje ve svém článku i Hendrikx (Hendrikx et al. 2013). Výstupy jednotlivých generátorů by tedy měly být vzájemně kompatibilní a ve vztahu, který dodává generovanému celku přidanou hodnotu.

Tradiční metody PCG jsou často aplikovány ad hoc ke konkrétnímu použití. Znovupoužitelnost je realizovatelná pouze principiálně a jen stěží se dají stejné algoritmy použít napříč různými aplikacemi. I to je nedostatek, na který Togelius poukazuje (Togelius et al. 2013). Uvážíme-li Togeliusův nárok na komplexní procedurální generátor, je nutné do něj zahrnout i dynamickou stránku scénu, to znamená například animace. Procedurální animování je další oblastí, v rámci které probíhá aktivní výzkum (Togelius et al. 2013).

Jako obecný trend v rámci PCG lze tedy primárně identifikovat směřování k univerzálním, navzájem propojeným generátorům. Na základě provedené rešerše je možné konstatovat, že se výzkumníci v oblasti PCG často uchylují k metodám strojového učení, jež má v návaznosti na současné cíle PCG velký potenciál. Umožňuje totiž nalezení modelu složitých datových distribucí. Takovým modelem není obvykle funkce $y = f(x)$ tradičního diskriminativního modelu, nýbrž je vhodnější najít

generativní pravděpodobnostní datový model, který v rámci svého učení algoritmicky hledá parametry pro $p(y|x)$, kde x jsou vstupní data. Takovými parametry mohou být například proměnné normálního rozdělení $\mathrm{N}(\mu, \sigma)$. Pro generování nových dat založených na této distribuci lze pak uplatnit vztah $y = \mu + \sigma\epsilon$, kde $\epsilon \in N(0,1)$. Tento konkrétní postup je zde popsán na základě Kingmem popsané generativní architektury neuronových sítí známé jako variační autoenkodér (Kingma a Welling 2013).

4 Procedurální generování prostřednictvím strojového učení

Tradiční PCG přistupuje k tvorbě obsahu s konkrétní referencí ve vztahu k cíli. Tato reference však není součástí samotného algoritmu, ale figuruje pouze jako inspirace toho, kdo algoritmus navrhuje. Znamená to, že primární část návrhu takového algoritmu tvoří hledání pravidel, abstrakcí a omezení, která pomohou aproximovat žádoucí obsah. Toto hledání figuruje i v rámci procedurálního generování prostřednictvím strojového učení, avšak s tím podstatným rozdílem, že toto hledání neprobíhá na straně vývojáře, nýbrž na straně výpočetního stroje. Ten s pomocí obecně laděných algoritmů strojového učení a vhodně zpracovaných vstupních dat vytvoří model, pomocí kterého je možné generovat nový obsah. Tak ve zkratce definuje PCGML Summerville (Summerville et al. 2017).

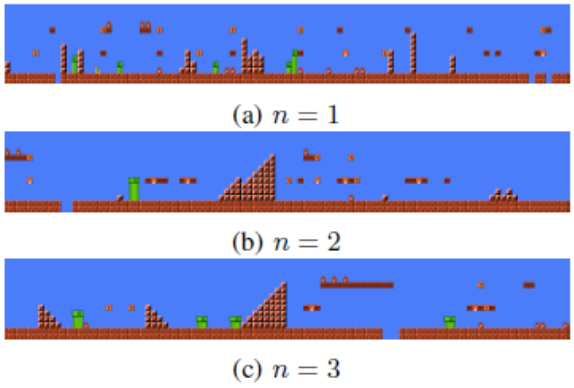
Definice strojového učení od Toma Mitchella, uvedená v první kapitole, je aplikovatelná rovněž pro PCGML, kdy zadáním T je generování konkrétní třídy obsahu na základě zkušenosti E, která je tvořena množinou ručně modelovaných obsahů (v případě učení s učitelem). Měření úspěšnosti P může být vzhledem k žádoucí variabilitě výsledků značně komplikované a může v konečné fázi spočívat v subjektivním posouzení omezeného množství vygenerovaného obsahu. Příkladem implementace takové metriky jsou dotazníky vyplněné testery, na jejichž základě byly evaluovány hratelnosti procedurálně vygenerovaných levelů a tedy celého algoritmu (Roberts a Chen 2015). Pro posouzení většího vzorku dat je třeba implementace algoritmů, jejichž funkcí je evaluace generovaného obsahu na základě příslušných kritérií.

Ne všechny metody strojového učení, které se jinak výborně hodí pro klasifikaci či predikci, jsou vhodné pro generování nového obsahu. Odpovídá tomu i zmíněné rozdělení metod strojového učení na diskriminativní a generativní modely. Z generativních modelů strojového učení vyzdvihuje Summerville jako vhodné zejména, n-gramy, Markovovy modely, současné generativní modely hlubokého učení (GAN) a specifické architektury rekurentních neuronových sítí (Summerville et al. 2017).

4.1 Současné experimenty PCGML

Následující kapitola prezentuje vybrané aplikace a metody v oblasti PCGML. Některé uvedené příklady posloužily jako motivace k sepsání této bakalářské práce a vytvoření metody užité v její praktické části. V kontextu strojového učení se všechny tyto aplikace musely po svém způsobu vypořádat s problémem malé datové sady a s požadavkem na funkční použitelnost vygenerovaných vzorků.

Silným nástrojem pro generování jednoduchého obsahu se ukázala být nejjednodušší varianta Markovova řetězce ve spojení s n-gramy. Markovovův řetězec je graf, jehož vrcholy reprezentují určitý stavový prostor. Mezi vrcholy existují ohodnocené hrany, a ty vyjadřují pravděpodobnost přechodu mezi jednotlivými stavy systému (Gagnius 2017). N-gram je struktura, která vyjadřuje jaké y bude následovat v případě, že pole x délky n obsahuje nějaké konkrétní hodnoty. Technika tohoto typu se dá dobře využít pro generování jednoduchých levelů viz Obr. 7. V tomto experimentu byla vstupní data nejprve indexována a rozřezána na sloupce. Poté byl na základě n-gramů vytvořen pravděpodobnostní model, který iterativně predikoval další sloupec podmíněně k n posledním sloupcům (Summerville et al. 2017).

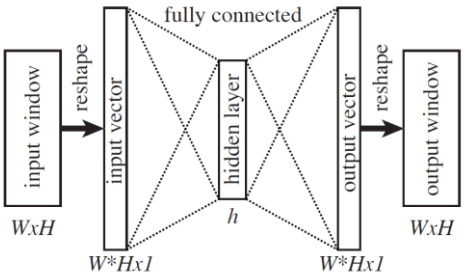


Obr. 7: Výsledné vygenerované levely pro jednotlivá n, vstupní data byla rozdělena na sloupce a za pomocí n-gramů transformována do Markovova řetězce. Převzato z (Summerville et al. 2017).

Komplexnější alternativou pro dosažení obdobného cíle je užití hlubokých neuronových sítí, a to konkrétně rekurentních architektur. V tomto přístupu se už generování neodehrává na úrovni sloupců, nýbrž na úrovni jednotlivých políček třinácti kategorií. Kromě těchto políček byla do algoritmu zahrnuta i informace o pohybu hráče uvnitř vstupních dat (levelů). Účelem tohoto kroku byla reálná hratelnost vygenerovaných úrovní. Generování levelů probíhalo tedy iterativně v prostoru $[n_x, n_y]$ od pomyslného $[y_0, x_0]$ k $[y_n, x_0]$ zase zpátky dolů od $[y_n, x_1]$

k $[y_0, x_1]$. To vše s paměťovou stopou na předtím vygenerovaná pole o délce zhruba 200 polí (Summerville a Mateas 2016).

Binárně zakódovaná vstupní datová sada použitá Summervillem byla východiskem i pro Jainův tým, který však namísto rekurentních neuronových sítí využil architektury klasického autoenkoderu (Jain et al. 2016). Jedná se o neuronovou síť, ve které se počet vstupních neuronů rovná počtu výstupních. Skryté vrstvy jsou nadefinovány tak, aby nejprve došlo ke „komprimaci" (zhuštění) informace a její následné „dekomprimaci". Toho je dosaženo postupným snižováním počtu neuronů v jednotlivých vrstvách směrem do středu h na přijatelné minimum a jejich následnému zvyšování směrem od středu viz Obr. 8. Tyto sítě byly aplikovány především v oblasti zpracování obrazu (škálování obrazu, odšumění a další) (Goodfellow et al. 2016). Obecně tato síť umožňuje funkčně mapovat transformaci vstupů na výstupy o stejné dimenzi.



Obr. 8: Schéma autoenkodéru. Převzato z (Jain et al. 2016).

Aby Jain dosáhl požadované nahodilosti, zaznamenal distribuci hodnot vnitřní autoenkodérové vrstvy h napříč trénovanými daty. Ve fázi generování poté reinterpretoval tuto vrstvu jako vstupní a tuto distribuci varioval za pomocí šumu (Jain et al. 2016). Binárně kódované úrovně jsou ukázány na Obr. 9.

Obr. 9: Generování binárních levelů při aplikaci náhodného šumu s rozptylem 0,01; 0,1; 0,3; 0,6. Převzato z (Jain et al. 2016).

Jako čtvrtý příklad, který lze označit za výsledek v PCGML, je zde uveden Fisherův pokus o syntézu žánrových scén do modelu schopného tyto scény generovat v četných variacích. Konkrétně se jedná o scénu stolu obklopeném objekty. K učení algoritmu byly použito 130 manuálně vymodelovaných prostředí interiéru, ve kterých bylo dohromady znovu použito 1723 kategorizovaných objektů. Variace objektů bylo dosaženo implementací takzvaných kontextuálních kategorií. Variování tedy neprobíhá pouze na základě příslušnosti objektu k určité kategorii, ale je založené na pozičním kontextu objektu. Druhým modelem pro finální generování je model okurence, který je založen na bayesovských sítích, a který řeší pravděpodobnost možné přítomnosti objektu vzhledem k již přítomným. V rámci modelu okurence byla rovněž implementována umělá omezení, za účelem zavedení informace o relaci „x je na y" to znamená „y podpírá x". Poslední složkou je poziční model, který řeší finální umístění a natočení objektu. Za tímto účelem jsou použity „Gaussian mixture models" (Fisher et al. 2012).

Obr. 10 Výsledky generování a variování scény se stolem za pomocí Fisherova algoritmu. Převzato z (Fisher et al. 2012).

V souvislosti s PCGML lze rovněž zmínit experiment Giacomella a jeho týmu, který s využitím GAN vytvořil model pro generování herních úrovní hry DOOM. V tomto případě však byla tréninková data o velikosti 1088 vzorků zakódována jako rastrové obrázky (128x128) půdorysů jednotlivých úrovní. V rámci každé úrovně existuje celkem šest obrázků, z nichž každý reprezentuje jinou informaci, tj. například o topologii, výškovém profilu či rozmístění předmětů uvnitř úrovně. Autoři vytvořili kromě standardního modelu i podmíněný model, který za pomoci sedmi parametrů vygenerované vzorky parametrizoval (Giacomello et al. 2018). Podobným způsobem byly GAN architektury implementovány za účelem procedurálního generování terénu (Beckham a Pal 2017).

V oblasti PCGML existuje ještě několik dalších experimentů, které se k tomuto pojmu samy hlásí, popřípadě by tam mohly být zařazeny. V této kapitole šlo o vymezení

hlavních principů, které byly inspirací pro metodu prezentovanou v praktické části této práce.

5 Technologie pro implementaci algoritmů strojového učení

Většina používaných knihoven pro účely implementace algoritmů strojového učení je dnes navázána na programovací jazyk Python. Jmenovitě to jsou například knihovny PyTorch, Scikit-learn, Tensorflow a v této práci užívaný Keras. Vzhledem k jednostrannému zaměření této práce na dopředné neuronové sítě jsou postupně ve stručnosti představeny ty technologie, se kterými se pracuje v praktické části a které jsou obecně základem pro aplikační práci s neuronovými sítěmi.

5.1 Python

Python je opensource interpretovaný, multiplatformní jazyk s čistou syntaxí evokující pseudokód. Obecně je dnes tento jazyk používán především ve vědecké a technologické komunitě. Důvody pro jeho užívání vystihuje Oliphant. V první řadě je to obrovské množství jak nativních, tak dodatečných knihoven, které řeší komplexní úlohy vyplynulé z vědeckých potřeb. Jmenovitě se jedná například o vizualizační nástroje, knihovny pro statistiku, strojové učení a mnohé další. Za druhé lze jazyk užívat jak procedurálním, tak i objektovým způsobem. A konečně Python umožňuje živou interakci s běžícím kódem a má velice silnou odbornou komunitu uživatelů (Oliphant 2007).

5.2 NumPy

NumPy je dle Oliphanta podstatným důvodem akademické oblíbenosti Pythonu (Oliphant 2007). Je to knihovna, která umožňuje za pomocí vyššího jazyku, kterým Python je, efektivně implementovat numerické operace (především na maticích a tensorech obecně). Za touto efektivitou stojí tři složky. V první řadě je to vektorizace veškerých výpočtů, za pomocí operací implementovaných v jazyce C. Za druhé efektivní správa proměnných v paměti a za třetí minimalizace počtu operací (Van Der Walt et al. 2011). V souvislosti se strojovým učením se knihovna NumPy využívá při předzpracování dat a je rovněž nativní součástí většiny knihoven pro strojové učení, které závisí na efektivní práci s tensory.

Základní strukturou (objektem) této knihovny je NumPy pole (ndarray). Van der Walt konstatuje, že: *„...NumPy pole je pouze příhodný způsob popisu jednoho nebo více bloků paměti za účelem jednoduché manipulace s reprezentovanými čísly.“*. Každá ndarray struktura si drží ukazatel na první byte v poli, datový typ prvků, svůj tvar (shape), počet kroků pro skok na další element (strides) a příznaky, zdali je možné s polem manipulovat.

Důležitým parametrem je především onen počet kroků. Van der Walt tak hovoří o „krokovém paměťovém modelu“, který umožňuje intepretaci paměti na více způsobů bez nutnosti kopírovat data. Toho se využívá především při manipulaci s tvarem (shape) multidimenzionálního pole, kdy dochází pouze k přepočítání potřebných kroků. (Van Der Walt et al. 2011).

5.3 TensorFlow

Tensorflow je knihovna pro podporu strojového učení vyvíjená společností Google. Primárně funguje v Pythonovském kontextu, ale existují i alternativy a API pro jiné jazyky. Umožňuje poměrně komplikovanou distribuci výpočtů na výpočetní jednotky, popřípadě jejich paralelizaci či naopak synchronizaci. Obecně je tato knihovna uzpůsobena pro experimentální a dobře kontrolovatelný vývoj nejen algoritmů strojového učení. Myšlenka této knihovny je založena na toku operací uspořádaných v grafu. Tyto operace jsou výlučně prováděny na tensorech, tj. vstupem i výstupem je tensor (v Pythonu se jedná o ndarray). Interpretováno jinak: Vrcholy tohoto grafu reprezentují matematické operace, které vlastní či mění stav výpočtu. Po hranách pak „proudí“ jednotlivé tensory. Kromě jiného obsahuje tensorflow nástroje pro diferenciaci libovolné chybové funkce (Abadi et al. 2016).

Omezíme-li se na modely hlubokého učení, umožňuje Tensorflow v podstatě dva přístupy k implementaci. Odpovídá tomu i rozdělení oficiální dokumentace na „High level APIs“ a „Low Level APIs“. První „High level funkcionalita“ se překrývá se specifikací knihovny Keras a bude pojednána v následující kapitole. Je to právě ono „Low Level API“, které s pomocí nativních struktur umožňuje značnou kontrolu nad budovanými modely. V praktické části této práce se však s tímto API nepracuje, proto nebude nadále popisováno.

Za poslední zmínku z knihovny TensorFlow stojí nástroj TensorBoard, který slouží k vizualizaci jednotlivých fází učení modelů a mnohým dalším vizualizačním

a optimalizačním potřebám vývojáře. Umí rovněž vizualizovat celý nadefinovaný komputační graf.

5.4 Keras

Keras je již zmiňované „High level API“ primárně určené pro jednoduché budování modelů hlubokého učení. S jeho pomocí je možné implementovat a parametrizovat většinu základních architektur (CNN, RNN, LSTM, ANN, GAN) včetně jejich variant pro praktické a aplikační potřeby.

Uživatelská přívětivost je podstatnou zásadou této knihovny, proto je možné nadefinovat funkční modely velice rychle. Keras je nativní nadstavbou nad TensorFlow, popřípadě jinými alternativami knihoven strojového učení jako je například Theano. Pro trénování implementovaných modelů lze s výhodou využít grafického hardwaru, kde mohou výpočty na tensorech probíhat paralelně a celý proces je několikanásobně rychlejší.

Základní definovatelnou strukturou je v Kerasu Model. Ten slouží jako placeholder pro různé druhy vrstev. Jednotlivé vrstvy se dají do modelu přidávat buď sekvenčně jednu za druhou, nebo je v případě komplikovanějších modelů možné definovat souslednost vrstev do grafu. Veškeré vrstvy jsou plně parametrizovatelné a diferencovatelné. Keras podporuje standardní plně propojené vrstvy, konvoluční triplety, rekurentní vrstvy aj. Keras rovněž obsahuje nativní nástroje pro datový preprocessing, škálu callbacků volatelných při trénování, nástroje pro načítání standardních datových sad, vizualizační nástroje a další utility. Pochopitelně jsou přítomné parametrizovatelné funkce pro trénování a evaluaci nadefinovaných modelů.

6 Návrh a implementace

Na základě popsaných experimentů PCGML v kapitole 4.1 lze konstatovat následující obecné rysy algoritmů pro generování logicky uspořádaných prostorů. Společným jmenovatelem představených experimentů je replikace naučených relací mezi přítomnými objekty. Příkladem může být naučená poziční relace mezi židlí a stolem. K dosažení tohoto cíle je třeba účelným způsobem předzpracovat vstupní data. To znamená, že vybranému modelu strojového učení musí být připraveny vhodně zakódované partikulární části žádoucího obsahu. Ve fázi generování jsou pak v rámci „nahodilé“ redistribuce těchto částí, zachovány naučené relace, ve kterých tyto části figurovaly v rámci učení. Dále si lze všimnout, že generování probíhá obvykle sekvenčně, kdy stav aktuálního prostoru vystupuje jako vstup pro další krok algoritmu.

Cílem následující kapitoly je demonstrovat metodu, která vychází z poznatků nabytých z prezentovaných PCGML experimentů a poznání v oblasti neuronových sítí. Tato metoda účelným způsobem kombinuje výstupy neuronových sítí pro logickou distribuci objektů do interiéru pokoje. Implementace vychází z technologií představených v kapitole 5. S daty bylo primárně pracováno jako s csv soubory a pro maticové vizualizace byl využita knihovna matplotlib (Hunter et al. 2019).

6.1 Reprezentace scény

Na nejvyšší úrovni jsou vstupní data pro algoritmus definována jako scény $S = \{s_1, s_2, \dots, s_n\}$. Scéna je diskrétním prostorem n^2 možných pozic $P = (x, z)$, kde $x, z \in \langle 0, n \rangle$ a $P \in s_{1\dots n}$. Na jednotlivých pozicích se může vyskytovat žádný nebo jeden objekt.

$$s = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Rov. 7: Příklad scény s definované v diskrétním prostoru 3^2, kde každá pozice reprezentuje jeden objekt z konkrétní třídy objektů.

Dohromady vytvářejí objekty určitou množinu O. Tato množina vytváří smysluplnost neboli význam scény ve dvou aspektech. Za prvé je pro tento význam podstatná přítomnost nebo nepřítomnost objektů ve scéně. Například vyskytuje-li se

ve scéně 5 postelí, je její význam jiný, než vyskytuje-li se v ní 5 stolů obklopených židlemi. Za druhé je význam scény v menší míře spoluvytvářen i vzájemnými pozičními relacemi v diskrétním prostoru scény. Ukázáno na příkladu: Nachází-li se ve scéně dvacet židlí a pět stolů, je význam scény jiný, když se všech pět židlí nachází u stolu, než když tomu tak není. Prezentovaný algoritmus řeší druhý aspekt významu scény, totiž umístění objektu na nejpravděpodobnější místo vzhledem k aktuálnímu kontextu prostoru scény. Přítomnost jednotlivých objektových tříd ve scéně je řízena uživatelsky.

Každý objekt přítomný ve scéně sebou nese vektor své pozice, rotace a velikosti. Podstatnou abstrakcí prezentovaného modelu je, že z těchto tří vlastností objektu je pro zjednodušení úlohy uvažován pouze poziční dvoudimenzionální vektor $P = (x, z)$. Vektory rotace a velikosti uvažovány nejsou. Kromě geometrických vlastností objektu, je každý objekt ve scéně kategorizován třídou, kterou představuje.

Z uvedeného lze názorně představit obecný cíl: Za předpokladu výchozí scény s_0, která je určena množinou objektů O, umístit uživatelsky vybraný objekt na logickou pozici. Tím vzniká nová scéna $s_.$ Tento proces bude iterativní, to znamená, že v jednom kroku algoritmu bude scéna doplněna právě o jeden objekt. Načež se tato nová scéna stane východiskem pro další krok generování. Fáze, které povedou k naplnění tohoto cíle, jsou předmětem následující kapitoly.

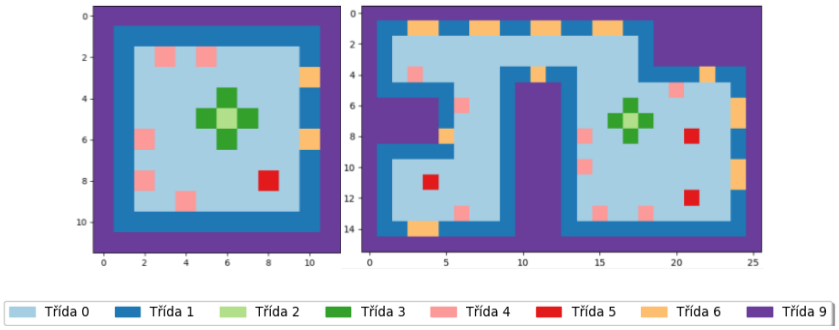
6.2 Popis algoritmu a modelové situace

Celý algoritmus sestává ze tří fází. Za prvé je to fáze předzpracování dat. V druhé fázi se natrénují neuronové sítě a v poslední fázi je volána metoda generování, která účelným způsobem kombinuje výstupy neuronových sítí pro finální predikci umístění objektu do scény. Modelovou situací pro otestování navrženého principu je umísťování nábytku do prostoru pokoje. Aplikační užití tohoto principu se však neomezuje na tento konkrétní případ a lze ho obecně aplikovat i pro jiná zadání. Všechny tři fáze jsou blíže popsány v následujících kapitolách.

6.2.1 Charakteristika vstupních dat

Pro všechny testované případy byla užita vstupní datová sada čítající 23 ručně nadefinovaných scén, které reprezentují pokoj. Matice dat (a jejich vizualizace) lze nalézt v příloze bakalářské práce. Tvar pokoje byl v 18 případech čtvercový a v pěti

případech byl tvarem nepravidelný n-úhelník s vodorovnými nebo svislými hranami. Příklady obou tvarů pokoje jsou vyobrazeny na obrázku Obr. 11. Tabulka 1 představuje použité objekty a logiku, se kterou byly předměty do scén umísťovány. Tyto údaje jsou podstatné pro popis jednotlivých fází a následnou evaluaci vygenerovaných vzorků. Obr. 12 představuje prostor matice, ve které jsou zobrazeny a dodrženy všechny sledované vlastnosti.



Obr. 11: Příklady scén (pokojů) z datové sady. Vlevo čtvercový tvar. Vpravo nepravidelný n-úhelník.

Tabulka 1: Názorný popis objekty přítomné ve vstupních datech. Objekt typu SS je takový objekt, který pouze vystupuje ve scénách a nehodláme ho predikovat.

Id	Interpretace	Typ	Logika umístění
0	Prázdný prostor	SS	Reprezentuje prázdný **vnitřní** prostor místnosti.
1	Stěna	SS	Stěna je vždy umístěna po obvodu scény.
2	Stůl	O	Stůl je umístěn náhodně uvnitř prostoru scény včetně krajních oblastí.
3	Židle	O	Židle je objekt, na kterém sledujeme hned několik hypotéz, a platí pro něj několik speciálních pravidel. Tato pravidla jsou rozvedena v kapitole 7.
4	Skříň	O	Skříň je umístěna vždy v blízkém kontaktu se stěnou.
5	Objekt ve volném prostoru	O	Objekt volného prostoru, je abstraktní objekt, který je v rámci scén vždy umístěn do prázdného prostoru.
6	Okno	O	Objekt umístěný ve zdi. Je dbáno na to, aby se před ním nevyskytoval objekt třídy **4**.
9	Prostor za zdí	SS	Abstraktní objekt, který simuluje neznámý prázdný **vnější** prostor. Slouží jako protiklad k objektu **0** a pomáhá při predikci prostorů s nepravidelným tvarem.

$$S_1 = \begin{pmatrix} 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\ 9 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 9 \\ 9 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 9 \\ 9 & 1 & 0 & 0 & 3 & 0 & 0 & 3 & 2 & 0 & 1 & 9 \\ 9 & 1 & 0 & 3 & 2 & 3 & 0 & 0 & 3 & 0 & 1 & 9 \\ 9 & 1 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 1 & 9 \\ 9 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 9 \\ 9 & 1 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 9 \\ 9 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 6 & 9 \\ 9 & 1 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 9 \\ 9 & 1 & 1 & 1 & 1 & 1 & 6 & 1 & 6 & 1 & 1 & 9 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \end{pmatrix}$$

Obr. 12: Matice, ve které jsou splněny všechny nadefinované vlastnosti viz Tabulka 1.

6.2.2 Vektorizace dat

Fáze vektorizace je operačně nejnáročnějším krokem, ve kterém dochází ke zpracování dat ze vstupních příkladů a vytvoření dvojic $(x \rightarrow y)$ vyjadřujících vztah mezi zadaným vstupním vektorem a očekávaným výstupem. Na konci fáze vektorizace představují tyto dvojice vstupně-výstupní datovou sadu pro neuronové sítě.

Jako parametr pro tuto fázi figurují čtyři pole. První pole D reprezentuje data všech vstupních scén. Pro naučení lokálních relací jednotlivých tříd objektů umístěných ve scénách je důležité, aby se scéna analyzovala po částech. Z toho důvodu je zavedeno druhé pole C, které definuje velikosti „okének“ (jader), pomocí kterých se budou procházet scény. Pomyslné okénko se ve vstupní matici pohybuje po řádcích zleva doprava směrem dolů vždy o jeden krok viz diagram D_1 a postupně generuje vstupní vektory. Pohyb lze připodobnit k posunům konvolučního jádra. S operací konvoluce nemá však tato metoda nic společného – podobnost je možné vidět pouze ve směru pohybu.

$$D_1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 3 & 0 & 0 \\ 1 & 0 & 2 & 0 & 0 \\ 1 & 4 & 0 & 0 & 0 \end{pmatrix}$$

Třetí pole S reprezentuje speciální symboly SS viz Tabulka 1. Jsou to třídy, pro které není potřeba predikovat polohu – například podlaha či zeď. Čtvrté pole O ukládá všechny třídy objektů v datech včetně SS.

```
for každé jádro v C do:
      for každý příklad v D do:
            V = x_vektory_po_projití_jádrem_C()
            for každý vektor v poli V do:
                  for každý objekt z O, který se nachází ve V do:
                     vytvor_kombinace_z_pritomnych_objektu_stejne_tridy()
                     pro_tyto_kombinace_vytvor_vektory_x()
                     pro_tyto_kombinace_vytvor_vektory_y()
```

Pseudokód výše vyjadřuje iterativní práci se vstupními daty. Každá vstupní scéna je pro každé jádro transformována („rozřezána“) na vektory velikosti příslušného jádra.

Ze vzniklých vektorů V se za pomocí specifických úprav ustanovují výsledné dvojice $(x \rightarrow y)$. Názorně je operace vektorizace představena na následujícím příkladu uspořádání scény.

$$V_1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 4 \\ 1 & 0 & 0 \end{pmatrix}$$

Cílem této názorné ukázky je generovat vektory x, y pro třídu 4 a velikost jádra (3×3). Předpokládá se, že ve vstupních datech jsou jen tři třídy objektů, tedy $O = [0,1,4]$. Prvky jádra jsou pro názornost označeny písmeny, na které bude odkazováno v následujícím textu.

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

Jako vstup pro neuronové sítě se obecně hodí normalizované hodnoty. V prezentovaném algoritmu spočívá tato normalizace v zakódování vstupní matice V_1 na vektor tvaru x_1, kde každý jedničkový příznak ve trojici označuje přítomnost objektu dané kategorie. Například šestá trojice z x_1 je vektor $[0_0, 0_1, \mathbf{1}_2]_f$, ten označuje přítomnost objektu třídy čtyři na pozici f. Pozice příznaku v rámci trojice se odvíjí od pořadí dané objektové třídy v poli $O = [0, 1, \mathbf{4}]$. Čtyřka je v poli O na pozici indexované dvojkou, proto na té stejné pozici figuruje i v x_1. Obecně je počet prvků ve vektoru v dán vztahem $\mathrm{P} = n_v \times n_O$. Podrobně rozepsaný vektor x má podobu

$$x_1 = ([0,1,0]_a, [0,1,0]_b, [0,1,0]_c, [0,1,0]_d, [1,0,0]_e, [0,0,\mathbf{1}]_f, [0,1,0]_g, [1,0,0]_h, [1,0,0]_i)$$

a jeho zkrácený zápis je následující

$$x_1 = (0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,0,0,\mathbf{1},0,1,0,1,0,0,1,0,0)$$

V tomto příkladu se generují učící dvojice pro třídu 4. Proto je nutné vektor x_1 transformovat na x_1', kde se na pozici objektu 4 vloží 0.

$$x_1' = (0,1,0,\ 0,1,0,\ 0,1,0,\ 0,1,0,\ 1,0,0,\ 0,0,\mathbf{0},\ 0,1,0,\ 1,0,0,\ 1,0,0)$$

Vstupní vektor x_1' simuluje nepřítomnost objektu 4 na pozici f. Tato skutečnost je ve vektorech x_1 a x_1' zvýrazněna tučně. Jako odpověď pro x_1' funguje výstupní vektor y_1, který na pozici f tento objekt „vkládá“. V tomto smyslu se iteruje přes všechny jaderné vektory a vytváří se dvojice $(x \rightarrow y)$.

$$f(x_1') = y_1 = (0,0,0,0,0,1,0,0,0)$$

Při takovém postupu je přirozeně vygenerováno mnoho redundantních dvojic, neboť vektory vyprodukované jádry se často opakují. Redundance dat pro neuronovou síť je obecně diskutovaným tématem. Článek Davida Medlera ji popisuje jako pozitivní jev (Medler a Dawson 1994). V případě prezentovaných experimentů byly redundance zachovány s hypotézou posílení často opakovaných vzorků. Nicméně v případě větší datové sady by bylo pravděpodobně žádoucí je odstranit.

Popsaný příklad názorně ukazuje postup pro vytvoření jedné učící dvojice pro jeden vektor jedné objektové třídy (4) jádra (3×3). Ve finálním algoritmu se pracuje celkem se čtyřmi jádry $[(3 \times 3), (4 \times 4), (5 \times 5), (6 \times 6)]$ a pěti predikovatelnými třídami objektů, viz Tabulka 1. Při tomto počtu vzniká na výstupu fáze předzpracování celkem 4×5 dvojic csv souborů. Každá dvojice obsahuje vektory x, y pro jednu konkrétní třídu a jádro. Tato data budou v další fázi vstupem pro 4×5 neuronových sítí. Obecný vztah pro výsledný počet potřebných sítí je tedy $n_O \times n_C$, to znamená počet jader krát počet unikátních objektových kategorií, které bude možné predikovat.

V případě, že se v jádře nachází více objektů jedné kategorie, je vhodné vytvořit co nejvíce kombinací učících dvojic v rámci tohoto výřezu. Z důvodu tzv. „kombinační

exploze“ je však třeba tento počet limitovat pro případ, že by se uvnitř jednoho jádra vyskytovalo mnoho objektů jedné kategorie. To se pro větší jádra děje často. Ukázáno na příkladu: pro situaci P se v rámci vektorizace vytvoří dva umělé výřezy scény x_1, x_2, na které se aplikují výše zmíněné operace. Tím získáme $x_1{}'$, x_2', y_1, y_2.

$$P = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \mathbf{4} & \mathbf{4} \\ 1 & 0 & 0 \end{pmatrix} \; x_1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & \mathbf{4} \\ 1 & 0 & 0 \end{pmatrix} \; x_2 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \mathbf{4} & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

6.2.3 Natrénování modelů

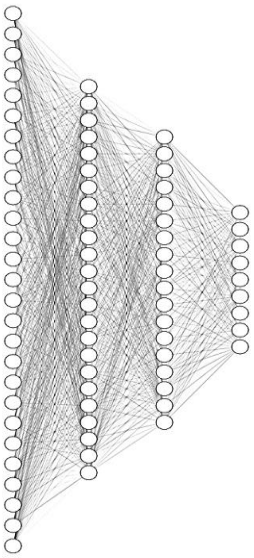
Pro natrénování všech modelů byla využita architektura dopředné neuronové sítě. Veškeré modely byly definovány ve frameworku Keras na bázi Tensorflow s podporou učení pomocí grafické karty. Konkrétně se jednalo o model s čipem NVIDIA GeForce GTX 970. Experimentováno bylo s několika různými tvary a parametry. Tabulka 2 ukazuje šest vybraných variant topologií a trénování. Dimenze vstupní vrstvy L_0 pro vektor v odpovídá vztahu $n_{L_1} = n_v \times n_O$, a je tedy pro každé jádro jiná. Počty neuronů ve skrytých vrstvách L_1, L_2, L_3 byly spoluurčeny vybranými koeficienty a odpovídají násobku příslušného koeficientu s počtem neuronů vstupní vrstvy L_0. Výstupní vrstva L_4 má velikost jádra, pro které je neuronová síť trénována. Aktivační funkce za skrytými vrstvami (L_1, L_2, L_3) je ReLU. Před výstupní vrstvu je za účelem pravděpodobnostního výstupu implementována funkce softmax. Jako chybová funkce byla zvolena „categorical_cross_entropy“ implementovaná v knihovně Keras.

Testovací sada byla vytvořena vyjmutím 20 % učících dvojic z konce datové sady. Úspěšnost neuronové sítě je vyhodnocena jak na trénovacích, tak i testovacích datech. Hodnota úspěšnosti představuje průměr napříč všemi dvaceti neuronovými sítěmi. V tabulce je rovněž uveden učící čas všech sítí při 50 epochách učení.

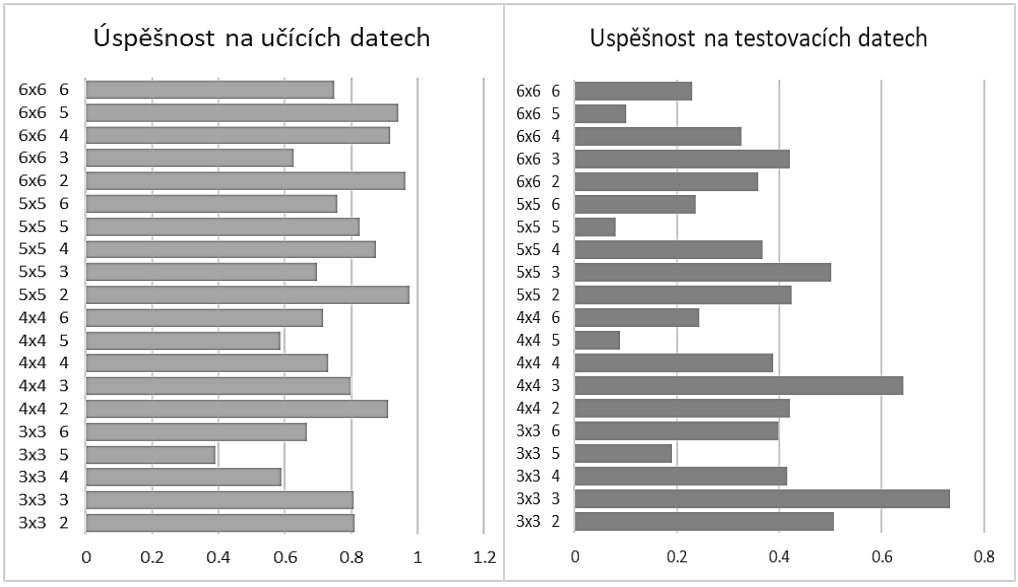
Tabulka 2: Statistiky několika konfigurací neuronových sítí.

Id	Batch size	Koef. K_1	Koef. K_2	Koef. K_3	Prům. úspěšnost na učících datech	Prům. úspěšnost na testovacích datech	Čas trénování (min)
1	64	0.9	0.7	0	0.762	0.351	8
2	128	0.9	0.7	0	0.767	0.353	8
3	512	0.9	0.7	0	0.730	0.351	3
4	128	0.9	0.7	0.5	0.766	0.346	10
5	512	0.9	0.7	0.5	0.731	0.346	4
6	512	1	1	0.7	0.739	0.347	4

Výsledná topologie neuronové sítě se po testování několika konfigurací ustálila na dvou skrytých vrstvách L_1 a L_2 při projití 128 vektorů před laděním vah viz Tabulka 2 - druhý řádek. Počet neuronů ve skrytých vrstvách se směrem k výstupní vrstvě snižuje do pyramidového tvaru s koeficientem $0.9 \times K_1$ pro L_1 a $0.7 \times K_2$ pro L_2. Vizualizaci této konkrétní architektury lze vidět na Obr. 13. Úspěšnost pro všech 20 neuronových sítí na Obr. 14.



Obr. 13: Vizualizace neuronové sítě pro jádro 3×3 , předpokládaje 3 objekty podobně jako v příkladu z kapitoly 0. Počet neuronů odpovídá $L_0 = 27$, $L_1 = 24$, $L_2 = 18$, $L_3 = 9$. Vizualizováno pomocí nástroje dostupném na (Nail 2019).

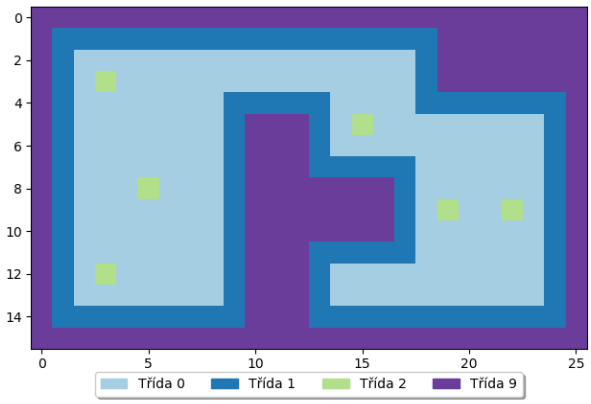


Obr. 14.: Statistiky úspěšnosti pro jednotlivé neuronové sítě. Identifikátor 6x6 6. Odpovídá jádru velikosti 6×6 v kombinaci s objektovou třídou 6.

Na základě Tabulky 2 a grafů z Obr. 14, lze konstatovat, že průměrná úspěšnost predikcí jednotlivých konfigurací je vyšší než 70 % a nižší než 80 %. Z Obr. 14 plyne, že problematicky šla natrénovat pouze objektová třída 5. Jedná se o „objekt volného prostoru“, u kterého se příliš nepodařilo zachytit žádnou obecnou závislost na jiných objektech. Průměrná úspěšnost na testovacích datech je nízká kolem 35 %. Tento údaj obvykle vypovídá o finální efektivitě neuronové sítě. Nicméně vzhledem ke způsobu používání neuronových sítí v navrhované metodě a vyplynulým výsledkům se nezdá, že by byl podstatný v kontextu této implementace.

6.2.4 Generování scén

Generování scén probíhá pomocí uspořádaného volání natrénovaných neuronových sítí. V každém kroku generování je do scény přidán jeden objekt. Ten je volen uživatelsky. Podobně jako ve fázi vektorizace figuruje i v této fázi pole jader C, které určuje, jakým způsobem se bude prostor procházet. Každý krok generování má výchozí scénu s_0. V první iteraci je tato scéna zvolena uživatelsky a v dalších iteracích je výchozí scénou výstup předešlého kroku generování. Výchozí scéna může být definována například po vzoru Obr. 15.



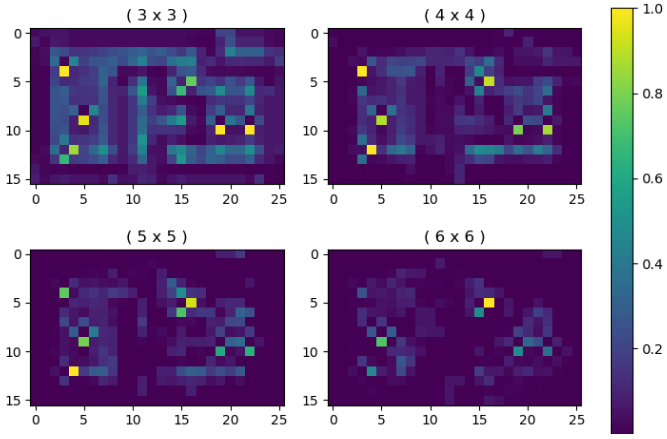
Obr. 15: Příklad výchozí ohraničené scény s_0 se šesti stoly.

Postup generování shrnuje následující pseudokód:

```
vychozi_prostor = M
For každé jádro v C do:
      J = projdi_výchozí_prostor_a_vrať_pole_pro_predikci()
      For každé pole v J do:
            N = nacti_spravnou_neuronovu_sit()
            P = N.predikuj(pole)
            zanes_predikci_do_souctove_matice_pro_jadro(P)
MP = utvor_finalni_soucinovou_matici_z_jadernych_souctovych_matic()
K = vyber_kandidata_ze_soucinove_matice(MP)
vychozi_prostor = umisti_do_prostoru(K)
```

V generování jde tedy o postupné projití výchozí matice pomocí jader. Pro všechna tato jádra se musí načíst správná neuronová síť, která pro daný výsek velikosti jádra predikuje pozici objektu požadované třídy. Načte se tedy ta síť, která odpovídá predikované třídě objektu a aktuální velikosti jádra.

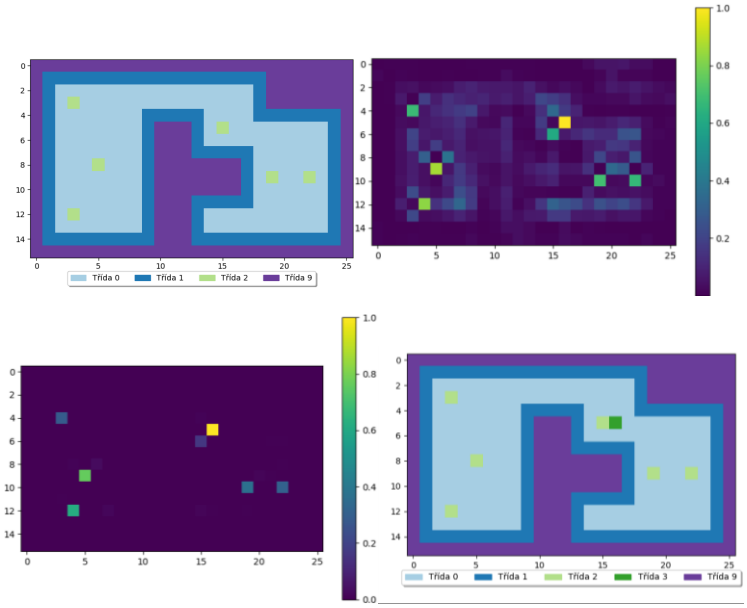
Všechny predikce se pak agregují do součtových pravděpodobnostních matic pro každou velikost jádra. Jde o jednoduchou funkci iterativního přičítání dílčích predikovaných pravděpodobností do prostoru s_0 prvek po prvku. Pro uvažovaná jádra $[(3 \times 3), (4 \times 4), (5 \times 5), (6 \times 6)]$ to pro s_0 budou čtyři matice MS o velikosti výchozí scény s_0, viz například Obr. 16.



Obr. 16: Součtové matice pravděpodobnosti výskytu pro jádra (3 × 3), (4 × 4), (5 × 5), (6 × 6). Konkrétní predikce odpovídá první iteraci na výchozí scéně z Obr. 15 pro objekt třídy 3 - židle.

Pro finální predikci se osvědčila matice, která je nadále označována jako finální součinová matice pravděpodobnosti výskytu - MP_{soucin}. Tato matice se vytvoří postupným vynásobením součtových prediktivních matic MS. Po dokončení této operace vzniká výsledný obraz o tom, kde by se objekt dané třídy měl nacházet. Nejedná se o násobení matic v klasickém smyslu, nýbrž ve smyslu Hadamardova produktu ($\circ$), to znamená násobení po prvcích.

Kromě MP_{soucin} se v průběhu generování utváří ještě finální součtová pravděpodobnostní matice MP_{soucet}. Děje se to podobným způsobem jako u součinové matice s tím rozdílem, že namísto Hadamardova produktu je aplikován součet.



Obr. 17: Postupně: Výchozí prostor M, MP_{soucet}, MP_{soucin}, výsledná predikce objektu třídy 3 = max(MP_{soucin}).

Co se týče nejpravděpodobnějších pozic, tak MP_{soucin} a MP_{soucet} ve větší míře korelují. Nicméně v průběhu testování produkovala MP_{soucin} méně chyb než MP_{soucet}, a proto se výsledná predikce řídí součinovou maticí MP_{soucin}. Součtová matice nefiguruje při generování, ale její vizualizace lépe ukazuje možné oblasti výskytu objektu.

Výběr finální pozice predikovaného objektu je možné určit různými způsoby. Jako nejjistější se ukázalo vybrat nejpravděpodobnější místo na MP_{soucin} - max(MP_{soucin}). Při tomto postupu by však v rámci stejných výchozích prostorů nedocházelo k žádoucí variabilitě obsahu. Proto je možné aplikovat například náhodný vážený výběr z uspořádaných pravděpodobností MP_{soucin} či zvolit jinou techniku výběru. Záleží rovněž na aplikačním záměru, neboť někdy může být žádoucí vybrat právě maximum z MP_{soucin}.

6.2.5 Parametrizace algoritmu

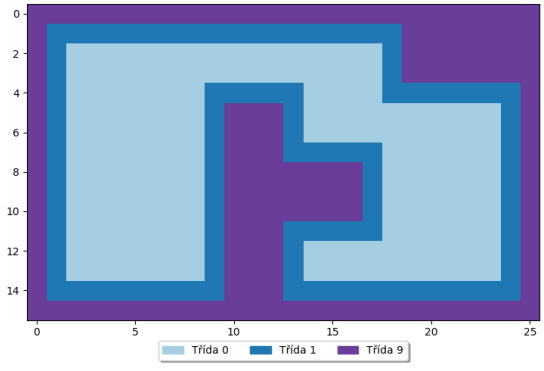
Na závěr této kapitoly lze podotknout, že algoritmus je v mnoha částech svého fungování parametrizovatelný. Pro lepší výsledky se dá primárně experimentovat s tvary a počty jader, architekturou neuronové sítě, se způsobem výběru finálního kandidáta, popřípadě se způsobem agregace do matic pravděpodobnosti výskytu. Příkladem může být zavedení minimální prahové hodnoty pro prvky MS při utváření MP_{soucin}, popřípadě MP_{soucet}. Tato konkrétní úprava se nicméně příliš neosvědčila.

7 Výsledky a testování

Postup měření úspěšnosti navrženého algoritmu je proveden následujícím způsobem. Uvnitř datové sady, která byla představena v předchozí kapitole, jsou dodržována určitá relační pravidla mezi objekty, viz Tabulka 1. V případě, že budou tyto vlastnosti nalezeny i v algoritmicky vygenerovaných datech, bude potvrzeno, že se algoritmus naučil tuto logiku replikovat. Vzhledem k stochastické povaze inicializace neuronových sítí je pravděpodobné, že na konci každého procesu učení může algoritmus produkovat odlišné výsledky. To samé se týká generovaných scén při zapnutí náhodného váženého výběru viz kapitola 6.2.4. Prezentované výsledky vyplývají z dvaceti naučených neuronových sítí, které sledují architekturu vybranou v kapitole 6.2.3.

7.1 Jednotkové testování

V první fázi testování bylo ověřováno, zda dokáže algoritmus replikovat konkrétní vlastnosti v jednoduchém prostoru, který není rušen kontextem ostatních objektů. Tento prostor je definovaný jako matice M_1 (Obr. 18), po jejímž obvodu se nachází objekt třídy 1, to znamená zeď. Není-li uvedeno jinak, byla pro všechny predikované objekty vybrána jejich nejpravděpodobnější pozice ze součinové matice pravděpodobnosti, to znamená $\max(MP_{soucin})$. Všechny výsledky testů jsou verifikovatelné s pomocí skriptů přiložených v příloze bakalářské práce popřípadě v repozitáři vytvořeném za tímto účelem (Ouhrabka 2019).



Obr. 18: Vizualizace výchozího prostoru M_1.

7.1.1 Test obsazeného souseda

V tomto testu sledujeme, že se algoritmus naučil obklopit stůl čtyřmi židlemi, za předpokladu, že tento stůl je umístěn ve středu místnosti. Tuto vlastnost formálně vyjadřuje matice X_1.

$$X_1 = \begin{pmatrix} 0 & 3 & 0 \\ 3 & \mathbf{2} & 3 \\ 0 & 3 & 0 \end{pmatrix}$$

Pro účely verifikace „obsazeného souseda“ byly na prostoru M_1 určeny všechny vnitřní pozice scény, kde by se stůl mohl vyskytovat. Poté bylo algoritmu řečeno, aby postupně přidal čtyři židle. Ze všech 34 vnitřních pozic matice M_1 doplnil algoritmus v rozsahu čtyř iterací židle kolem stolů po vzoru X_1.

Tabulka 3: Statistiky testování obsazeného souseda.

Celkový počet predikcí	**Správné umístění objektu**	**Špatné umístění objektu**
34	34	0

7.1.2 Test volného souseda

V tomto testu jsou sledovány vlastnosti vyobrazené v maticích X_{2a}, X_{2b} a X_{2c}. Jedná se o pravidlo, které určuje, že v případě umístění stolu jednu pozici od stěny, nebude na tuto pozici přidána židle, která by tam standardně měla být dle pravidla X_1 (obsazený soused).

$$X_{2a} = \begin{pmatrix} 0 & 3 & 0 \\ 3 & 2 & 3 \\ 0 & \mathbf{0} & 0 \\ 1 & 1 & 1 \end{pmatrix} \quad X_{2b} = \begin{pmatrix} 1 & 0 & 3 & 0 \\ 1 & \mathbf{0} & 2 & 3 \\ 1 & 0 & 3 & 0 \end{pmatrix} \quad X_{2c} = \begin{pmatrix} 1 & 0 & 3 & 0 \\ 1 & \mathbf{0} & 2 & 3 \\ 1 & 0 & \mathbf{0} & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Pro test vlastnosti X_2 byly stoly umístěny ob jednu pozici od obvodového zdiva. Celkem na M_1 vzniklo padesát testovacích pozic.

Tabulka 4: Počet chyb ve všech 50 testovacích pozicích vzhledem k pravidlům X_{2a}, X_{2b}, X_{2c} po 2., 3. a 4. iteraci.

Případ	**Chyby po 2. iteraci**	**Chyby po 3. iteraci**	**Chyby po 4. iteraci**	**Pravidlo dodrženo**	**Počet predikcí**
U stěny (X_{2a}, X_{2b})	0	3	30	7krát z 43 případů	43
V rohu (X_{2c})	0	3	3	1krát ze 7 případů	7
Chybovost %	0 %	22 %	88 %		
Úspěšnost %	100 %	88 %	22 %		

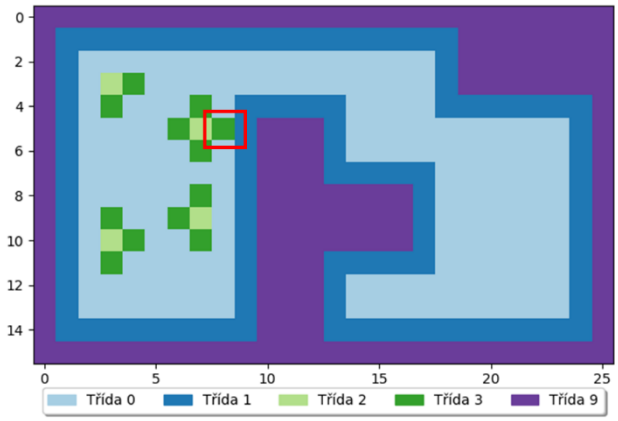
Tabulka 4 ukazuje počty chyb napříč padesáti pozicemi po jednotlivých iteracích. V rozsahu tří iterací neumístil algoritmus židle kolem stolu po vzoru X_2 v šesti případech. Při čtvrté iteraci došlo obvykle k tomu, že nejpravděpodobnější pozice byla predikována ke zdi po vzoru X_1 (obsazený soused), což není nelogické, ale vzhledem k testu volného souseda je to chyba. Tento jev nastal ve třiceti třech případech z celkových padesáti. V osmi případech byl výsledek v souladu s pravidlem volného souseda i po čtvrté iteraci a algoritmus preferoval umístit čtvrtou židli raději do volného prostoru než ke stěně.

7.1.3 Opakovaný test sousednosti

Pro účely tohoto testu byl vytvořen algoritmus, který náhodně umísťoval čtyři stoly do prostoru M_1. Celkem bylo v M_1 předvybráno dvanáct pozic, na kterých se v rámci náhodného výběru mohl stůl objevit. Tyto pozice byly voleny tak, aby ověřily varianty vlastností X_1 a X_2 viz testy volného a obsazeného souseda. Stoly tak mohly být umístěny jak ke krajním oblastem prostoru, tak do vnitřních částí. Celkem byl algoritmus spuštěn dvacetkrát a bylo sledováno v jaké iteraci se poprvé odkloní od pravidel X_1 a zejména X_2.

Mají-li ve scéně být čtyři stoly, měl by je algoritmus obkroužit dohromady šestnácti židlemi. Toto číslo kolísá, neboť jsou-li židle v krajních oblastech, je okolo stolu žádoucí jedna nebo dvě židle. Průměrně umístil algoritmus správně 10 židlí a poté se odklonil od pravidla X_2 k X_1 , čímž udělal chybu. Minimální počet správně umístěných židlí byl čtyři a maximální třináct. Příklad této chyby je znázorněn

na Obr. 19. V ideálním případě by měl algoritmus umístit židli volně do prostoru. Tato skutečnost však nebyla naučena a pravděpodobnost umístění židle po vzoru X_1 byla větší nežli umístit židli podle X_2. Pravidlo X_1 nebylo ve dvaceti běžích testu porušeno ani jednou.



Obr. 19: Chyba algoritmu pro náhodně vybrané pozice stolů. Chyba ve vztahu k X_2 nastala ve 12. iteraci algoritmu a je zvýrazněna červeně.

7.1.4 Testování blízkého souseda na objektu skříně

Test formulovaný v této kapitole testuje, že se objekt třídy 4, to znamená skříň, bude stavět ke stěně viz pravidlo X_{3a}, X_{3b}. Testováno bylo patnáct běhů algoritmu po deseti iteracích, při kterých se do prostoru scény vkládal objekt této třídy. Potenciálně správných pozic je po obvodu scény M_1 87 z celkových 416. V rámci testování byl povolen vážený náhodný výběr ze seřazené posloupnosti predikcí z MP_1.

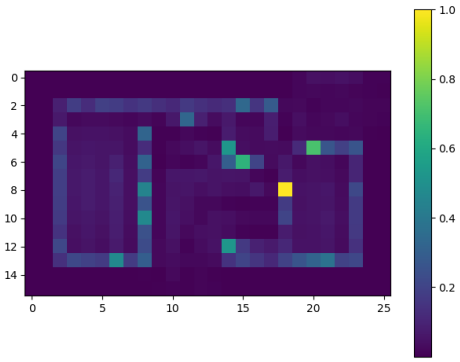
$$X_{3a} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & \mathbf{4} & 0 \\ 1 & 1 & \mathbf{1} & 1 \end{pmatrix} X_{3b} = \begin{pmatrix} 1 & 0 & 0 \\ \mathbf{1} & \mathbf{4} & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

Tabulka 5: Statistiky testování blízkého souseda na objektu skříně.

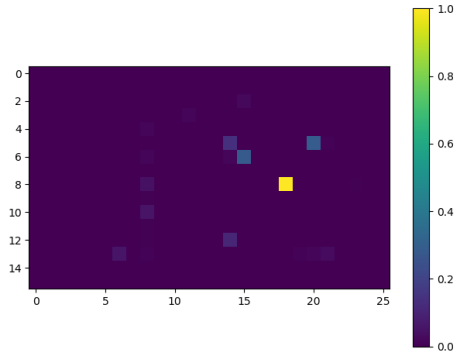
Celkový počet predikcí	**Správné umístění objektu**	**Špatné umístění objektu**
150	148	2

Z tabulky výše vyplývá, že chybovost navrženého testu je méně než 2 %. V jednom případě ze 150 byla skříň umístěna do vnitřního prostoru na roh. Ve druhém chybovém případě byla skříň umístěna do prostoru bez kontaktu se zdí.

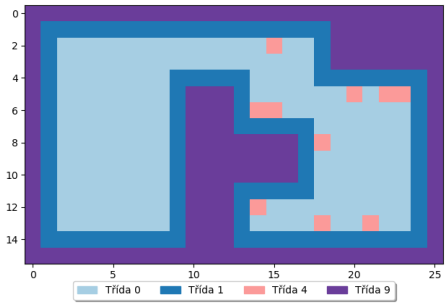
Následující obrázek (Obr. 20) ukazuje příklad součtové matice pravděpodobnosti výskytu třídy čtyři před první predikcí. Pravděpodobnosti jsou správně rozloženy kolem stěny M_1. Predikce objektu (Obr. 22) se pak odvíjí od součinové matice viz Obr. 21.



Obr. 20: Součtová matice pravděpodobnosti pro predikci výskytu objektu třídy 4 v první iteraci.



Obr. 21: Součinová matice pravděpodobnosti pro predikci výskytu objektu třídy 4 v první iteraci.



Obr. 22: Příklad výsledné scény po 10 iteracích.

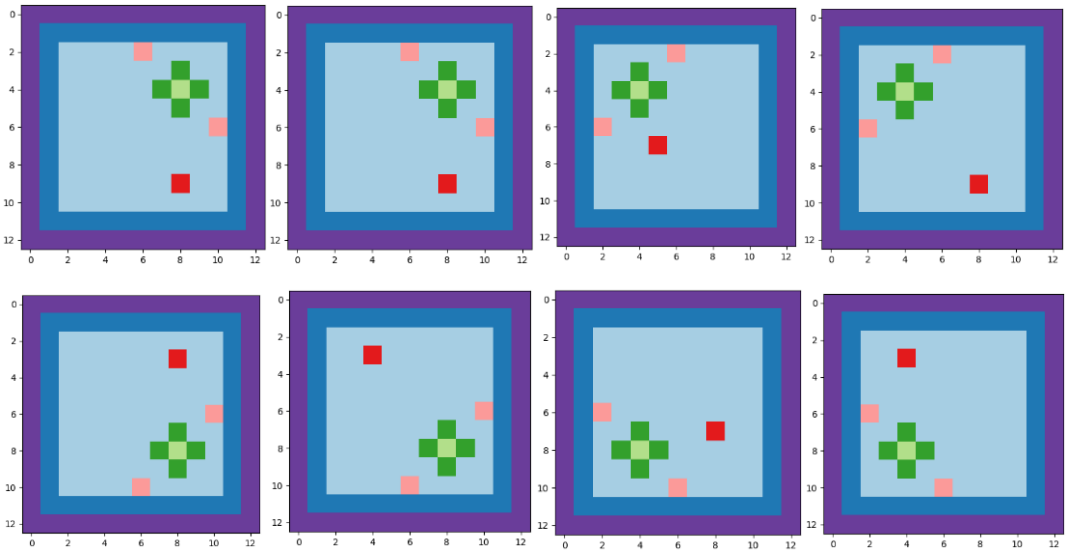
V rámci tohoto testování se replikace vlastnosti X_3 potvrdila a napříč vygenerovanými scénami spolehlivě nedocházelo k odklonům od tohoto pravidla.

7.1.5 Testování prostorovosti

Hypotéza testuje vlastnost vyobrazenou v matici X_4, která představuje skutečnost, že objekt třídy 5, to znamená objekt popsatelný jako „objekt volného prostoru", by se uvnitř M_1 měl objevit na místě, kolem kterého existuje rozpoznatelný volný prostor. Řečeno jinak, existuje-li v M_1 shluk objektů, měl by být objekt třídy 5 predikován mimo tento shluk.

$$X_4 = \begin{pmatrix} 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & \mathbf{0} & 5 & \mathbf{0} & \mathbf{0} \\ 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Tento konkrétní test je založen spíše na subjektivním posouzení a není proto příliš vypovídající. Objektová třída pět měla rovněž velice nízkou úspěšnost při trénování viz Obr. 14. V testu byl povolen náhodný výběr z nejpravděpodobnějších predikovaných pozic. Na Obr. 23 jsou scény, ve kterých se v rozích čtvercové scény nachází shluk objektů. Červeně je označena predikce testovaného objektu. Dá se konstatovat, že „objekt volného prostoru" je většinou umísťován dvě a více pozic od shluku objektu. V žádném z testovaných případů pak bezprostředně nesousedí s ostatními objekty.



Obr. 23: Predikce objektu třídy 5 pro čtyři ručně nadefinované shluky objektů.

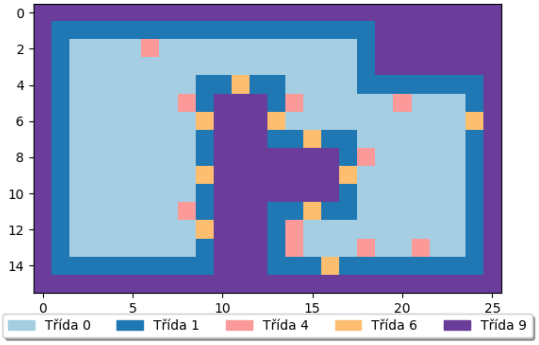
7.1.6 Testování zasazení objektu mezi jiné

Reprezentační matice X_{5a} a X_{5b} představují skutečnost, že objekt třídy 6, který zastupuje okno, by měl vždy nahradit objekt třídy jedna, tedy zeď. Současně je sledováno, že okno nebude umístěno za objektem, který by hypoteticky bránil průchodu světla. Primárně se jedná o objekt skříně 4. V testu proběhlo deset běhů algoritmu, každý s dvaceti iteracemi. Nejprve bylo vygenerováno deset objektů třídy čtyři a poté deset objektů třídy šest, o které primárně šlo.

$$X_{5a} = \begin{pmatrix} 1 & 4 & 0 \\ \mathbf{6} & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \quad X_{5b} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 4 & 0 \\ 1 & 1 & \mathbf{6} \end{pmatrix}$$

Celkový počet predikcí	**Správné umístění objektu**	**Špatné umístění objektu**
100	91	9

Z celkového počtu sta iterací bylo 91 oken umístěno správně a 9 špatně. Ze špatně umístěných případů stála čtyřikrát před oknem skříň a třikrát bylo okno nevhodně zasazeno do rohu pokoje. Chybovost tak odpovídá zaokrouhleně 10 %. Jedna z dobře vygenerovaných scén je znázorněna na Obr. 24.



Obr. 24: Příklad výsledné scény po 10 iteracích, ve kterých se přidávala objektová třída 6

7.2 Integrační testování

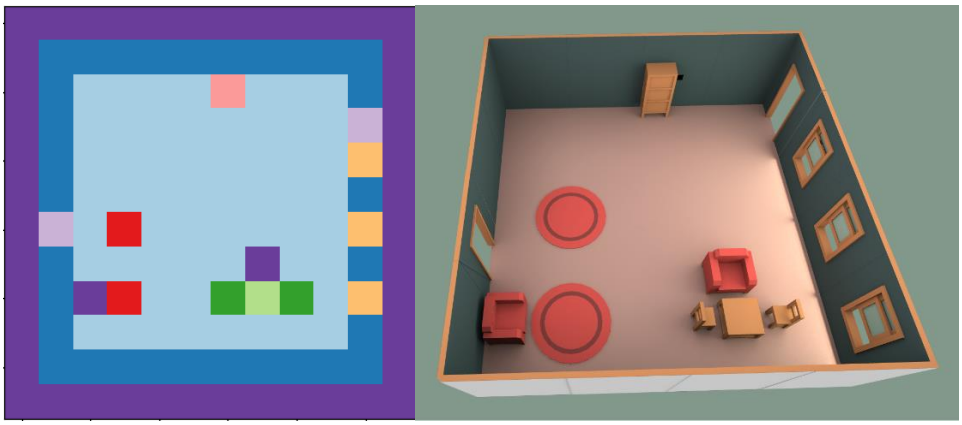
Z oblasti tradičního softwarového vývoje je převzat pojem integračního testu. Ten je zde používán ve smyslu otestování, jak dobře jsou sledované vlastnosti replikovatelné v případě, že do generovaného prostoru vstupuje větší variabilita objektů. Nejprve jsou představeny vybrané vygenerované produkty a poté

je poukázáno na omezení a problémy, které byly během generování objeveny. Produkty jsou vizualizovány s pomocí herního enginu Unity3D. Rotace objektů, které algoritmus neřeší, byly upraveny ručně s pomocí editoru. Modely objektů jsou převzaty z volně dostupné kolekce (Kenney 2019). Finální umístění objektů bylo přizpůsobeno geometrickým vlastnostem modelů z této kolekce.

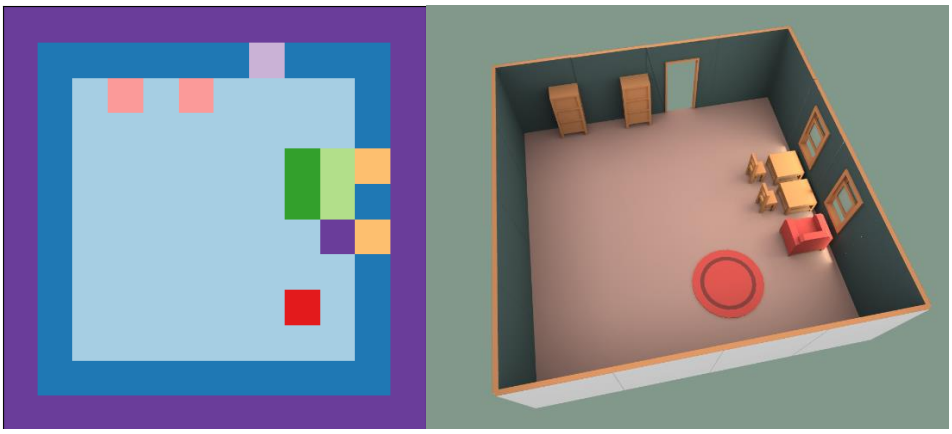
Pro pestrost vygenerovaných scén byly do algoritmů učení zařazeny dva další objekty a jádro velikosti 7 × 7. Logiku umístění přidaných objektů ukazuje následující Tabulka 6. Obrázky Obr. 25, Obr. 26 a Obr. 27 vyobrazují vygenerované matice a vizualizované scény.

Tabulka 6: Dodatečné objekty a jejich logika umístění.

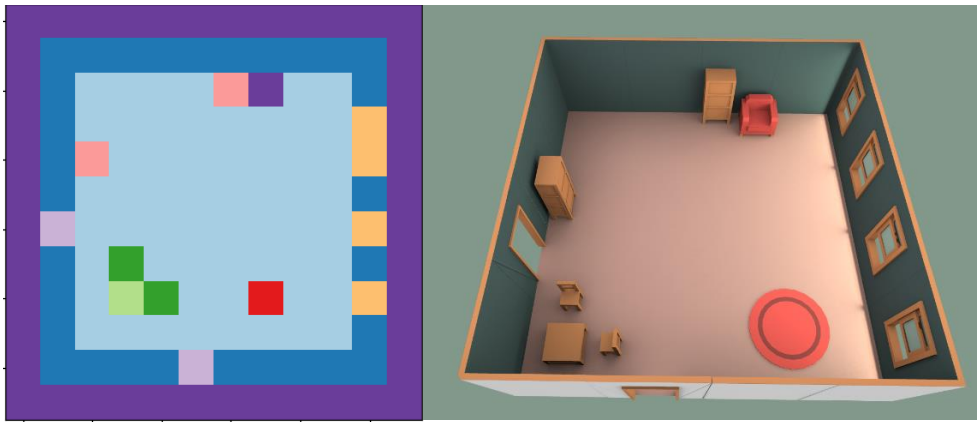
Id	Interpretace	Typ	Logika umístění
7	Dveře	O	Dveře jsou vsazené do stěny. Je dbáno na to, aby před nimi nestál žádný objekt.
8	Pohovka	O	Pohovka je primárně umístěna u stěny, nicméně může být umístěna i u stolu. V datech se v jeho blízkosti vyskytoval občasně i objekt třídy 5, který byl pro účel intepretace zobrazen jako koberec.



Obr. 25: Výsledná scéna při generování objektů v pořadí (okno, dveře, dveře, okno, pohovka, pohovka, stůl, židle, židle, koberec, koberec, skříň).



Obr. 26: Výsledná scéna při generování objektů v pořadí (stůl, stůl, židle, židle, dveře, pohovka, skříň, skříň, koberec, okno, okno).



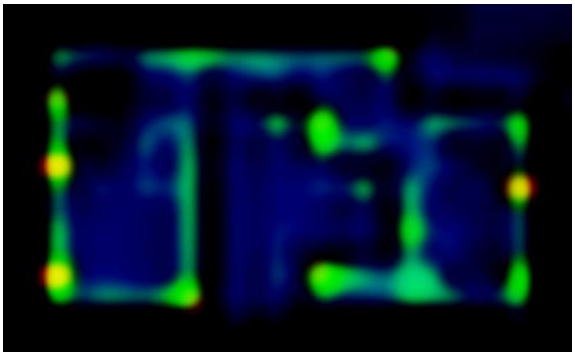
Obr. 27: Výsledná scéna při generování objektů v pořadí (stůl, židle, židle, dveře, okno, okno, okno, okno, koberec, skříň, skříň, pohovka).

7.3 Vize a závěrečné hodnocení algoritmu

V rámci jednotkových testů se potvrdilo, že algoritmus dokázal do určité míry replikovat vybrané vlastnosti. Ve spojení s integračním testováním lze konstatovat, že vlastnosti X_1, X_5 a X_3 jsou replikovatelné v limitu uvedené 10 % chybovosti. Na úskalí vlastnosti X_2 bylo poukázáno v kapitolách 7.1.2 a 7.1.3. Vlastnost X_4 se bezpečně replikovat nepodařilo. Co se týče nejpravděpodobnější pozice predikovaného objektu, je algoritmus použitelný. Pro generování rozmanitého obsahu je třeba zavést prvek „řízené nahodilosti“. V integračním testování byl tímto prvkem algoritmus, který postupně iteroval přes všechny hodnoty od té nejvyšší směrem k nejnižší. Vyšší pravděpodobnostní hodnoty měly úměrně vyšší váhu pro finální výběr. Obvykle se tak volilo ze tří nejvyšších pozic, které dodržovaly naučená pravidla. V principu mohly však být zvoleny i pozice méně pravděpodobné, a tedy vadné vzhledem k nastavené logice. Zde lze vidět prostor pro zlepšení.

Velkou roli v úspěšnosti algoritmu hraje rovněž pořadí, ve kterém se objekty vkládají do scény. V kapitole 6.2 byl tento aspekt identifikován jako první podstatný bod významu scény. Jde o predikci třídy objektu, který by se měl do scény vložit podmíněně k již přítomným. Za tímto účelem by se pravděpodobně dala navrhnout další neuronová síť. Tento aspekt byl v rámci prezentovaného algoritmu vynechán a třída objektu je volena uživatelsky. Na úskalí důležitosti pořadí vkládaných objektů, lze poukázat pomocí následujícího příkladu. V případě, že jsou do scény vloženy objekty stůl, židle a židle (v tomto pořadí), nelze už v další iteraci vložit druhý stůl, neboť jeho nejpravděpodobnější pozice jsou v blízkosti prvního vloženého stolu (u židlí). To je vzhledem k fungování algoritmu logické, nicméně nežádoucí. Preferovaná pozice druhého stolu by byla ve volném prostoru. Důsledkem tohoto je, že se nejprve musí do scény umísit stoly a teprve poté objekty, které mají ke stolu blízkou relaci, tedy židle. Obecně má však pořadí objektů, které může být pokaždé různé, pozitivní vliv na rozmanitost vygenerovaných scén.

Data i nové vzorky mají jako podklad diskrétní dvoudimenzionální mřížku. Oba aspekty, to znamená diskrétnost a dvoudimenzionalita, limitují použitelnost. Pro překonání těchto omezení je možné předběžně vytyčit následující návrhy. Diskrétnost lze překonat vhodnou transformací prediktivních matic na spojité mapy (Obr. 28), u kterých může finální predikce probíhat na úrovni pixelu a příslušně transformovaných souřadnic scény.



Obr. 28: Příklad „heat mapy“ pro predikci objektu třídy 4 (skříň) pro prostor M_1 Mapa byla vytvořena úpravami obrazu vizualizace diskrétní predikce.

Dvoudimenzionální limit je principiálně překonatelný dodáním dalších pozic do učících dvojic. Na straně interpretace generátoru by pak tyto pozice byly interpretovány do třídimenzionálního tensoru. Výsledek takové úpravy nelze předem odhadovat.

8 Závěr

V teoretické části textu byla představena oblast strojového učení, na kterou volně navázal popis oblasti, kde se metody strojového učení prakticky aplikují na generování obsahu. Na základě poznaného aktuálního stavu a výsledků v oboru strojového učení generativních modelů je možné konstatovat, že generování je a bude obecně doménou komplexnějších generativních architektur typu GAN či variačních autoenkoderů v kombinaci s konvolučními a jinými architekturami. Úspěšné aplikace těchto architektur byly sice primárně zaměřeny na rastrové obrázky, nicméně principy lze bezesporu uplatnit i pro data jiného charakteru.

Dá se předpokládat, že oblast procedurálního generování pomocí strojového učení se bude i nadále rozvíjet. Příspěvky z akademické obce figurují spíše jako demonstrace principu nežli za metody, které by byly masově aplikovány při vývoji současných aplikací pracující s grafickým obsahem. To samé platí i o navrhované metodě, prezentované v kapitole šest. Přesto má PCGML velký potenciál například pro vytváření prototypů či generování upozaděného obsahu.

V rámci praktické části byla implementována originální nadstavba, která umožnila použít klasické modely neuronových sítí diskriminativního charakteru pro úkol generování jednoduché scény pokoje. Konkrétně se jedná o postupné umístění objektů jednotlivých definovaných tříd do půdorysu scény podle předem určených pravidel. Výsledky tohoto generování byly testovány a dosažené výsledky okomentovány.

Z pěti vybraných vlastností zanesených ve vstupních datech se ve vygenerovaných vzorcích podařilo bezpečně replikovat tři z nich. Jmenovitě se v limitu maximálně 10 % chybovosti podařilo naučit blízkou sousednost stolu a židle či skříně a stěny. Rovněž byla naučena negativně formulovaná vlastnost, že skříň nestojí před oknem. Čtvrtá vlastnost určující nepřítomnost objektu v daném kontextu (židle u stěny) funguje do určitého počtu iterací viz kapitola 7.1.2. Pátou vlastnost charakterizující umístění objektů do volného prostoru se bezpečně otestovat nepodařilo. Na úspěšnost algoritmu má kromě způsobu procházení prostoru v rámci učení vliv i pořadí generovaných objektů a metoda pro jejich výběr ze seřazeného vektoru možných pravděpodobností výskytu. S přihlédnutím k výsledkům

z integračního testování, lze říci, že navržená metoda je použitelná případně se dá považovat za východisko k hlubšímu rozpracování, otestování či optimalizaci.

Jádrem prezentované implementace je účelná kombinace pravděpodobnostních výstupů několika neuronových sítí. Každá tato síť je namodelována tak, aby jejím výstupem byl pro tuto kombinaci hodnotný příspěvek. V principu je možné uvažovat o tom, že by se této myšlenky kombinace dalo využít, popřípadě ji rozšířit, i pro jiné úkoly. Konkrétním příkladem použití může být predikce nejvhodnějšího umístění pro objekt vzhledem k prostorovému kontextu. Nadneseně řečeno jde tedy o nalezení nejlepších odpovědí na otázku „kam to dát“ podmíněně k dříve naučeným datům. V souvislosti s tímto případem užití se dají dobře využít i součtové pravděpodobnostní mapy. Navržený algoritmus je rovněž parametrizovatelný a nabízí několik míst, kde lze experimentovat za účelem zlepšení výsledných produktů. V textu byla konstatována omezení a úskalí, která se prezentovaného způsobu generování týkají. Pro tyto problémy byly ve stručnosti nastíněny návrhy řešení.

Seznam použité literatury

[1] ABADI, Martın, Paul BARHAM, Jianmin CHEN, Zhifeng CHEN, Andy DAVIS, Jeffrey DEAN, Matthieu DEVIN, Sanjay GHEMAWAT, Geoffrey IRVING, Michael ISARD, Manjunath KUDLUR, Josh LEVENBERG, Rajat MONGA, Sherry MOORE, Derek G MURRAY, Benoit STEINER, Paul TUCKER, Vijay VASUDEVAN, Pete WARDEN, Martin WICKE, Yuan YU a Xiaoqiang ZHENG, 2016. TensorFlow: A system for large-scale machine learning. In: *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation* [online]. Savannah,: USENIX Association, s. 21. ISBN 978-1-931971-33-1. Dostupné z: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[2] BECKHAM, Christopher a Christopher PAL, 2017. A step towards procedural terrain generation with GANs. *arXiv:1707.03383 [cs, stat]* [online]. [vid. 2019-03-15]. Dostupné z: http://arxiv.org/abs/1707.03383

[3] BHANDARE, Ashwin, Maithili BHIDE, Pranav GOKHALE a Rohan CHANDAVARKAR, 2016. Applications of Convolutional Neural Networks. *International Journal of Computer Science and Information Technologies,* [online]. **7**(2). ISSN 0975-9646. Dostupné z: https://pdfs.semanticscholar.org/89db/184cb8b1397a9aa35693f8dc03e1e728992a.pdf

[4] BOUDON, Frederic, Przemyslaw PRUSINKIEWICZ, Pavol FEDERL, Christophe GODIN a Radoslaw KARWOWSKI, 2003. Interactive Design of Bonsai Tree Models. *Computer Graphics Forum* [online]. **22**(3), 591–599. ISSN 0167-7055, 1467-8659. Dostupné z: doi:10.1111/1467-8659.t01-2-00707

[5] FISHER, Matthew, Daniel RITCHIE, Manolis SAVVA, Thomas FUNKHOUSER a Pat HANRAHAN, 2012. Example-based synthesis of 3D object arrangements. *ACM Transactions on Graphics* [online]. **31**(6), 1. ISSN 07300301. Dostupné z: doi:10.1145/2366145.2366154

[6] GAGNIUS, Paul, 2017. *Markov Chains: From Theory to Implementation and Experimentation*. 1. vyd. B.m.: Wiley. ISBN 1-119-38755-8.

[7] GHAHRAMANI, Zoubin, 2004. Unsupervised Learning. *GATSBY COMPUTATIONAL NEUROSCIENCE UNIT* [online]. [vid. 2019-04-02]. Dostupné z: http://mlg.eng.cam.ac.uk/zoubin/papers/ul.pdf

[8] GIACOMELLO, Edoardo, Pier Luca LANZI a Daniele LOIACONO, 2018. DOOM Level Generation using Generative Adversarial Networks. In: *IEEE Games, Entertainment, Media Conference (GEM) 2018* [online]. [vid. 2018-06-14]. ISBN 978-1-5386-6304-2. Dostupné z: http://arxiv.org/abs/1804.09154

[9] GLOROT, Xavier, Antoine BORDES a Yoshua BENGIO, 2011. Deep Sparse Rectifier Neural Networks. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* [online]. s. 9. Dostupné z: http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

[10] GOODFELLOW, Ian, 2016. NIPS 2016 Tutorial: Generative Adversarial Networks. *arXiv:1701.00160 [cs]* [online]. [vid. 2018-06-14]. Dostupné z: http://arxiv.org/abs/1701.00160

[11] GOODFELLOW, Ian, Yoshua BENIGO a Aaron COURVILLE, 2016. *Deep Learning* [online]. B.m.: MIT Press. Dostupné z: http://www.deeplearningbook.org

[12] GOODFELLOW, Ian, Jean POUGET-ABADIE, Mehdi MIRZA, Bing XU, David WARDE-FARLEY, Sherjil OZAIR, Aaron COURVILLE a Yoshua BENGIO, 2014. Generative Adversarial Nets. In: Z. GHAHRAMANI, M. WELLING, C. CORTES, N. D. LAWRENCE a K. Q. WEINBERGER, ed. *Advances in Neural Information Processing Systems 27* [online]. B.m.: Curran Associates, Inc., s. 2672–2680 [vid. 2018-06-04]. Dostupné z: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[13] HENDRIKX, Mark, Sebastiaan MEIJER, Joeri VAN DER VELDEN a Alexandru IOSUP, 2013. Procedural Content Generation for Games: A Survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*. **9**(1), 24. ISSN 1551-6857.

[14] HUNTER, John, Dale DARREN a Eric FIRING, 2019. matplotlib. *matplotlib* [online]. Dostupné z: https://matplotlib.org/

[15] ISOLA, Phillip, Jun-Yan ZHU, Tinghui ZHOU a Alexei EFROS, 2016. Image-to-Image Translation with Conditional Adversarial Networks. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)* [online]. [vid. 2019-03-15]. ISBN 978-1-5386-0457-1. Dostupné z: http://arxiv.org/abs/1611.07004

[16] JAIN, Rishabh, Aaron ISAKSEN, Christoffer HOLMGA a Julian TOGELIUS, 2016. Autoencoders for Level Generation, Repair, and Recognition. *Proceedings of the ICCC Workshop on Computational Creativity and Games*. 9.

[17] JEBARA, Tony, 1996. Discriminative, Generative and Imitative Learning. 212.

[18] JEBARA, Tony, 2004. *Machine Learning Discriminative and Generative* [online]. 1. vyd. United States: Springer US. ISBN 978-1-4419-9011-2. Dostupné z: https://www.springer.com/gp/book/9781402076473

[19] JORDAN, M. I. a T. M. MITCHELL, 2015. Machine learning: Trends, perspectives, and prospects. *Science* [online]. **349**(6245), 255–260. ISSN 0036-8075, 1095-9203. Dostupné z: doi:10.1126/science.aaa8415

[20] KAPUR, Arnav, Shreyas KAPUR a Pattie MAES, 2018. AlterEgo: A Personalized Wearable Silent Speech Interface. In: *23rd International Conference on Intelligent User Interfaces* [online]. New York, NY, USA: ACM, s. 43–53 [vid. 2019-03-08]. IUI '18. ISBN 978-1-4503-4945-1. Dostupné z: doi:10.1145/3172944.3172977

[21] KARRAS, Tero, Samuli LAINE a Timo AILA, 2018. A Style-Based Generator Architecture for Generative Adversarial Networks. *arXiv:1812.04948 [cs, stat]* [online]. [vid. 2019-03-21]. Dostupné z: http://arxiv.org/abs/1812.04948

[22] KENNEY, 2019. Kenney. *Kenney.nl* [online]. Dostupné z: https://www.kenney.nl/assets

[23] KESKAR, Nitish Shirish, Dheevatsa MUDIGERE, Jorge NOCEDAL, Mikhail SMELYANSKIY a Ping Tak Peter TANG, 2016. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. *arXiv:1609.04836 [cs, math]* [online]. [vid. 2019-04-08]. Dostupné z: http://arxiv.org/abs/1609.04836

[24] KINGMA, Diederik a Max WELLING, 2013. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]* [online]. [vid. 2019-03-19]. Dostupné z: http://arxiv.org/abs/1312.6114

[25] KRIZHEVSKY, Alex, Ilya SUTSKEVER a Geoffrey HINTON, 2012. ImageNet Classification with Deep Convolutional Neural Networks. In: F. PEREIRA, C. J. C. BURGES, L. BOTTOU a K. Q. WEINBERGER, ed. *Advances in Neural Information Processing Systems 25* [online]. B.m.: Curran Associates, Inc., s. 1097–1105 [vid. 2019-03-09]. Dostupné z: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[26] LECUN, Yann, Léon BOTTOU, Yoshua BENGIO a Patrick HAFFNER, 1998. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE* [online]. **86**(11) [vid. 2019-03-08]. ISSN 0018-9219. Dostupné z: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

[27] MEDLER, David a Michael DAWSON, 1994. Using Redundancy to Improve the Performance of Artificial Neural Networks. *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*. 8.

[28] MICHELON DE CARLI, Daniel, Fernando BEVILACQUA, Cesar Tadeu POZZER a Marcos Corderio DORNELLAS, 2011. A Survey of Procedural Content Generation Techniques Suitable to Game Development. In: *2011 Brazilian Symposium on Games and Digital Entertainment*: *2011 Brazilian Symposium on Games and Digital Entertainment* [online]. Salvador: IEEE, s. 26–35. ISBN 978-0-7695-4648-3. Dostupné z: doi:10.1109/SBGAMES.2011.15

[29] MITCHELL, Tom M., 1997. *Machine Learning*. New York: McGraw-Hill. McGraw-Hill series in computer science. ISBN 978-0-07-042807-2.

[30] NAIL, Alex, 2019. NN-SVG: Publication-Ready Neural Network Architecture Schematics. *NN-SVG* [online]. Dostupné z: http://alexlenail.me/NN-SVG/index.html

[31] OLEJ, Vladimír a Petr HÁJEK, 2010. *Úvod do umělé inteligence*. B.m.: Univerzita Pardubice Fakulta ekonomicko-správní. ISBN 978-80-7395-307-2.

[32] OLIPHANT, Travis, 2007. Python for Scientific Computing. *Computing in Science & Engineering* [online]. **9**(3), 10–20. ISSN 1521-9615. Dostupné z: doi:10.1109/MCSE.2007.58

[33] OUHRABKA, Adam, 2019. *Veřejný repozitář k této bakalářské práci* [online]. Dostupné z: https://github.com/Edmaniak/Thesis2019Final

[34] PEREIRA, Francisco Camara, 2008. *Creativity and AI: A Conceptual Blending approach*. Portugal: University of Coimbra. ISBN 978-3-11-019856-0.

[35] REVOW, Michael, Christopher K.I WILLIAMS a Geoffrey E. HINTON, 1996. Using Generative Models for Handwritten Digit Recognition. *IEEE Transactions on pattern analysis and machine intelligence*. **18**(6), 15.

[36] SMITH, Gillian, 2015. An Analog History of Procedural Content Generation. In: *Proceedings of the 2015 Conference on the Foundations of Digital Games (FDG 2015)* [online]. s. 6. Dostupné z: http://sokath.com/main/files/1/smith-fdg15.pdf

[37] SUAREZ, Patricia L., Angel D. SAPPA a Boris X. VINTIMILLA, 2017. Infrared Image Colorization Based on a Triplet DCGAN Architecture. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* [online]. Honolulu, HI, USA: IEEE, s. 212–217 [vid. 2019-03-15]. ISBN 978-1-5386-0733-6. Dostupné z: doi:10.1109/CVPRW.2017.32

[38] SUMMERVILLE, Adam a Michael MATEAS, 2016. Super Mario as a String: Platformer Level Generation Via LSTMs. In: [online]. [vid. 2019-03-10]. Dostupné z: http://arxiv.org/abs/1603.00930

[39] SUMMERVILLE, Adam, Sam SNODGRASS, Matthew GUZDIAL, Christoffer HOLMGÅRD, Amy K. HOOVER, Aaron ISAKSEN, Andy NEALEN a Julian TOGELIUS, 2017. Procedural Content Generation via Machine Learning (PCGML). *arXiv:1702.00539 [cs]* [online]. [vid. 2018-05-31]. Dostupné z: http://arxiv.org/abs/1702.00539

[40] TOGELIUS, Julian, Alex CHAMPANDARD, Pier Luca LANZI, Michael MATEAS, Ana PAIVA, Mike PREUSS a Kenneth STANLEY, 2013. Procedural Content Generation: Goals, Challenges and Actionable Steps. In: Simon LUCAS, Michael MATEAS, Mike PREUSS, Pieter SPRONCK a Julian TOGELIUS, ed. *Artificial and Computational Intelligence in Games* [online]. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl Follow-Ups, s. 61–75 [vid. 2018-05-31]. ISBN 978-3-939897-62-0. Dostupné z: doi:10.4230/DFU.Vol6.12191.61

[41] TOGELIUS, Julian, Noor SHAKER a Mark J. NELSON, 2016. Introduction. In: *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. B.m.: Springer. ISBN 978-3-319-42714-0.

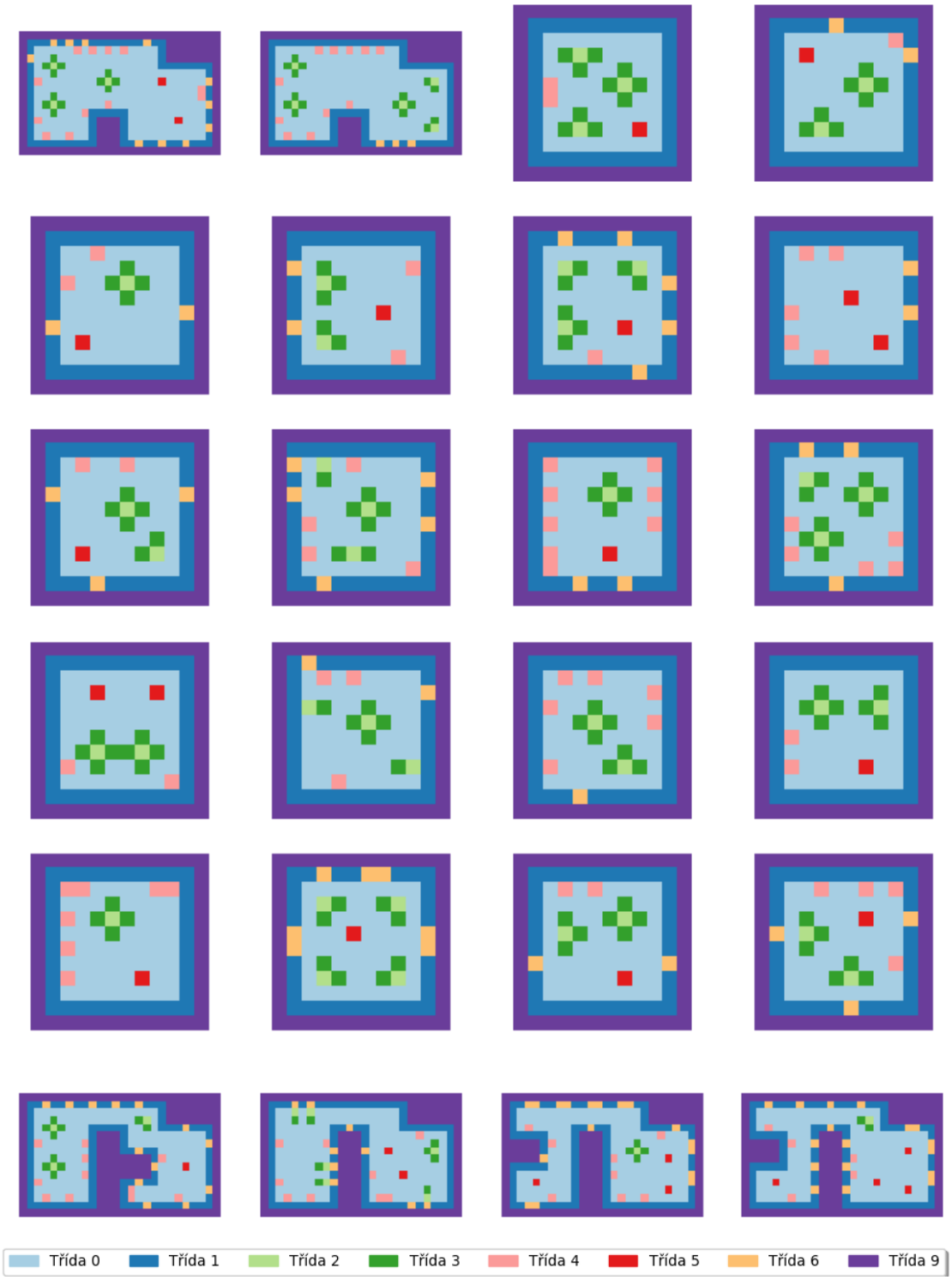
[42] TOGELIUS, Julian, Georgios N. YANNAKAKIS, Kenneth O. STANLEY a Cameron BROWNE, 2010. Search-Based Procedural Content Generation. In: Cecilia DI CHIO, Stefano CAGNONI, Carlos COTTA, Marc EBNER, Anikó EKÁRT, Anna I. ESPARCIA-ALCAZAR, Chi-Keong GOH, Juan J. MERELO, Ferrante NERI, Mike PREUß, Julian TOGELIUS a Georgios N. YANNAKAKIS, ed. *Applications of Evolutionary Computation* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, s. 141–150 [vid. 2018-05-31]. ISBN 978-3-642-12238-5. Dostupné z: doi:10.1007/978-3-642-12239-2_15

[43] VAN DER WALT, Stefan, CHRIS COLBERT a GAËL VAROQUAUX, 2011. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering* [online]. **13**(2), 22–30. ISSN 1521-9615. Dostupné z: doi:10.1109/MCSE.2011.37

[44] YANNAKAKIS, G. N. a J. TOGELIUS, 2011. Experience-Driven Procedural Content Generation. *IEEE Transactions on Affective Computing* [online]. **2**(3), 147–161. ISSN 1949-3045. Dostupné z: doi:10.1109/T-AFFC.2011.6

Přílohy

Vizualizace vstupní datové sady



Popis souborů na přiloženém CD

K bakalářské práci je přiloženo CD s implementací algoritmu. Kromě balíčku implementation.final, ve kterém je finální algoritmus, obsahuje CD všechny ostatní experimenty, které byly implementovány při hledání vhodného řešení.

Stěžejní je však již zmíněný balíček implementation.final. Zde lze najít Application skript, prostřednictvím kterého lze vektorizovat a natrénovat matice scén umístěných v adresáři data_source. Natrénované neuronové sítě jsou umístěny v adresáři /networks a vektorizované csv data jsou ve složce /data. Třídy Statistics a Visualiser slouží k podpůrným vizualizačním a statistickým účelům. Vektorizaci na datech provádí třída DataPreparator a generování řídí třída Generator.

Soubor	**Popis**
Application.py	Výchozí třída pro vektorizaci, natrénování a následné generování.
DataPrep.py	Funkcionalita vektorizace vstupních scén.
Generator.py	Funkcionalita volání správných neuronových sítí a následné generování a umístění objektu do prostoru.
Statistics.py	Funkcionalita pro záznam statistik při trénování neuronových sítí.
Visualiser.py	Nástroj pro vizualizaci matic s pomocí matplotlib.

Důležité jsou rovněž přítomné testy. To, jaké kapitoly odpovídají jednotlivým testům ukazuje následující tabulka.

Soubor	**Kapitola**	**Účel testu**
test1.py	7.1.1	Test obsazeného souseda.
test2.py	7.1.2	Test volného souseda.
test3.py	7.1.3	Opakovaný test sousednosti.
test4.py	7.1.4	Testování blízkého souseda na objektu skříně.
test5.py	7.1.5	Testování prostorovosti.
test6.py	7.1.6	Testování zasazení objektu mezi jiné.
test7.py	7.2	Integrační testování.

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Ouhrabka Adam	K Jeslím 443, Semily - Podmoklice	I1600586

TÉMA ČESKY:

Strojové učení v počítačové grafice

TÉMA ANGLICKY:

Machine learning in computer graphics

VEDOUCÍ PRÁCE:

Ing. Bruno Ježek, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cílem práce je prostudovat problematiku strojového učení a jeho aplikace v počítačové grafice se zaměřením na oblast modelování scény. Na vhodném příkladu odzkoušet vybrané principy a metody učení.

Osnova práce:
1. Provést literární rešerši a seznámit se s aktuálním poznáním v oblasti strojového učení.
2. Vytvořit přehled postupů a metod používaných v počítačové grafice pro návrh modelu scény podporovaný metodami strojového učení.
3. Prozkoumat vybrané dostupné technologie pro implementaci metod strojového učení.
4. Zvolit vhodný příklad využití a navrhnout jeho softwarovou implementaci.
5. Pro implementované řešení provést testování a zhodnotit dosažené výsledky.

SEZNAM DOPORUČENÉ LITERATURY:

SUMMERVILLE, Adam, Sam SNODGRASS, Matthew GUZDIAL, Christoffer HOLMG?RD, Amy K. HOOVER, Aaron ISAKSEN, Andy NEALEN a Julian TOGELIUS, 2017. Procedural Content Generation via Machine Learning (PCGML). arXiv:1702.00539 [cs] [online]. [vid. 2018-05-31]. Dostupné z: http://arxiv.org/abs/1702.00539

TOGELIUS, Julian, Alex J. CHAMPANDARD, Pier Luca LANZI, Michael MATEAS, Ana PAIVA, Mike PREUSS a Kenneth O. STANLEY, 2013. Procedural Content Generation: Goals, Challenges and Actionable Steps. In: Simon M. LUCAS, Michael MATEAS, Mike PREUSS, Pieter SPRONCK a Julian TOGELIUS, ed. Artificial and Computational Intelligence in Games [online]. Dagstuhl, Germany: Schloss DagstuhlLeibniz-Zentrum fuer Informatik, Dagstuhl Follow-Ups, s. 6175 [vid. 2018-05-31]. ISBN 978-3-939897-62-0. Dostupné z: doi:10.4230/DFU.Vol6.12191.61

Mitchell, T. M. (2017). Machine learning. New York: McGraw Hill.

MICHALSKI, R. S., J. G. CARBONELL a T. M. MITCHELL, 2013. Machine Learning: An Artificial Intelligence Approach. B.m.: Springer Science & Business Media. ISBN 978-3-662-12405-5.

Podpis studenta: ..... **Datum:** .....

Podpis vedoucího práce: ..... **Datum:** .....