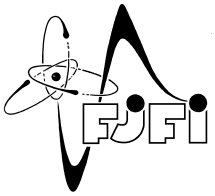
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta jaderná a fyzikálně inženýrská



Využití automatického zpracování leteckých snímků pro procedurální generování realistických 3D modelů měst

Utilizing automatic aerial image processing for procedural generation of realistic 3D city models

Diplomová práce

Autor: **Bc. Světlana Smrčková**

Vedoucí práce: **Ing. Pavel Strachota, Ph.D.**

Akademický rok: 2019/2020

České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská

Katedra: matematiky Akademický rok: 2018/2019

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	Bc. Světlana Smrčková
Studijní program:	Aplikace přírodních věd
Obor:	Aplikované matematicko-stochastické metody
Název práce (česky):	Využití automatického zpracování leteckých snímků pro procedurální generování realistických 3D modelů měst
Název práce (anglicky):	Utilizing automatic aerial image processing for procedural generation of realistic 3D city models

Pokyny pro vypracování:

1. Prostudujte moderní počítačové algoritmy pro zpracování leteckých snímků za účelem klasifikace objektů, např. budov, silnic, atd.
2. Použijte vhodné softwarové nástroje (např. TensorFlow) pro klasifikaci vybraných objektů v leteckých snímcích a případně i dalších vhodných volně dostupných materiálech (např. mapové podklady, katastrální mapy atd.)
3. Navrhněte metody využití výsledných dat pro získání vhodného souboru příznaků, popisujícího charakter města (např. centrum města s výškovými budovami vs. rezidenční čtvrti na okraji města).
4. Pokuste se modifikovat algoritmus pro pocedurální generování města vyvíjený na KM FJFI, aby využíval získané příznaky ke zvýšení realističnosti vygenerovaných modelů.

Doporučená literatura:

1. V. Mnih, Machine learning for aerial image labeling. PhD thesis, Department of Computer Science, University of Toronto, 2013.
2. H. Miura, S. Midorikawa and K. Fujimoto, Autmated building detection from high-resolution satellite image for updating GIS building inventory data. In '13th World Conference on Earthquake Engineering', Vancouver, B.C., Canada, 2004, paper no. 678.
3. A. Raikar, G. Hanji, Automatic building detection from satellite images using internal gray variance and digital surface model. International Journal of Computer Applications 145 (3), 2016, 25–33.
4. M. Schwarz, P. Müller, Advanced Procedural Modeling of Architecture. ACM Transactions on Graphics 34 (4), 2015, 107:1–107:12.

Jméno a pracoviště vedoucí diplomové práce:

Ing. Pavel Strachota, Ph.D.
KM FJFI ČVUT v Praze, Trojanova 13, 120 00 Praha 2

Jméno a pracoviště konzultanta:

Datum zadání diplomové práce: 31.10.2018

Datum odevzdání diplomové práce: 6.5.2019

Doba platnosti zadání je dva roky od data zadání.

V Praze dne 24. října 2018

. . . . . .
garant oboru

. . . . . .
vedoucí katedry

. . . . . .
děkan

Poděkování:
Chtěl bych zde poděkovat především svému školiteli Pavlu Strachotovi za pečlivost, ochotu, vstřícnost a odborné i lidské zázemí při vedení mé diplomové práce.

Čestné prohlášení:
Prohlašuji, že jsem tuto práci vypracoval samostatně a uvedl jsem všechnu použitou literaturu.

V Praze dne 7. července 2020 Bc. Světlana Smrčková

Název práce:

Využití automatického zpracování leteckých snímků pro procedurální generování realistických 3D modelů měst

Autor: Bc. Světlana Smrčková

Obor: Aplikace přírodních věd

Zaměření: Aplikované matematicko-stochastické metody

Druh práce: Diplomová práce

Vedoucí práce: Ing. Pavel Strachota, Ph.D, ČVUT, FJFI, katedra matematiky

Abstrakt: Tato práce se zabývá detekci budov v satelitních a leteckých snících dvou měst (Toronto a Praha) pomocí konvolučních neuronových sítí. Takto detekované objekty jsou dále zpracovány vzhledem ke zjištění charakteristik městské zástavby jako je vzdálenost budov, obvod bloku, protažení budov atd. Tyto charakteristiky budou následně implementovány do programu pro procedurální generování města k dosažení realističtějších výsledků.

Klíčová slova: detekce objektů, konvoluční neuronové sítě, procedurální modelování, rozpoznávání obrazu, TensorFlow.

Title:

Utilizing automatic aerial image processing for procedural generation of realistic 3D city models

Author: Bc. Světlana Smrčková

Abstract: This work deals with building recognition in aerial images of two cities (Toronto and Prague) by means of convolutional neural networks. These detected buildings are utilized to extract characteristics of urban development i.g. distances of buildings, blocks circumference, eccentricity of blocks etc. These characteristics are implemented into the existing program for procedural city generation to achieve realistic results.

Key words: convolution neural networks, object detection, object recognition, procedural modeling, TensorFlow.

Obsah

Úvod

Počítačové vidění je odvětví výpočetní techniky a vývoje softwaru zabývající se vytvářením zařízení schopných získávat informaci ze zachyceného obrazu. Může se jednat mimo jiné i o zpracování videa, obrazu z více zdrojů nebo např. dat z lékařských vyšetření [5].
Pro počítačové vidění je typická snaha porozumět obecné trojrozměrné scéně a interpretace obrazových dat a používá se například v oblastech [35]

- ovládání procesů (autonomní vozidla nebo průmysloví roboti),
- detekce jevů (sledování změn bezpečnostního kamerového záznamu),
- modelování objektů nebo prostředí (analýza obrazů z medicínských zobrazovacích technik).

Postupy v počítačovém vidění jsou různé, patří mezi ně např. umělá inteligence a detekce pomocí neuronových sítí. Počítačové vidění lze rozdělit na dvě základní oblasti:

- klasifikace obrazu - cílem je identifikovat, co data představují,
- detekce objektů - v tomto případě je snaha detekovat a lokalizovat hledané objekty v datech.

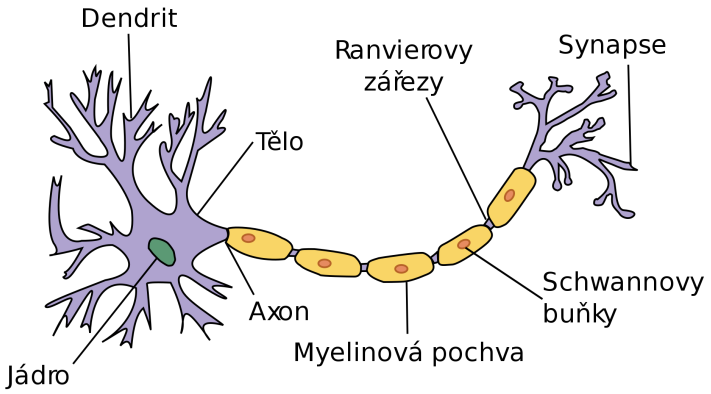
V naší práci budeme kombinovat obě oblasti, jelikož cílem je detekovat objekty v satelitních snímcích a ty správně klasifikovat.
Na detekovaných budovách chceme zjistit jejich typické charakteristiky a tyto znalosti následně využít k modifikaci program pro procedurální generování města [29], aby jeho výsledky více odpovídaly realitě.
Prvním krokem bude postavit a natrénovat neuronovou síť, která bude schopna detekovat budovy na satelitních snímcích, v druhém kroku budeme zjišťovat zajímavé parametry a statistické charakteristiky ohledně budov v reálných městech. V posledním kroku se pokusíme modifikovat program pro generování města, abychom dostali reálnější simulaci.
Následující práce je rozdělana do pěti kapitol. V první kapitole se podrobně zabýváme teorií neuronových sítí. Ve druhé kapitole jsou popsány námi zvolené neuronové sítě a jejich implementace. Třetí kapitola se zabývá jejich porovnáním. Ve čtvrté kapitole zjišťujeme zajímavé charakteristiky rozložení budov ve městě. Závěrečná kapitole je věnována implementaci zjištěných charakteristik do programu procedurálního generování města a jeho modifikaci.

Kapitola 1

Strojové učení a rozpoznávání obrazu

1.1 Neuronové sítě (NS)

Umělé neuronové sítě jsou inspirovány přírodou a principy fungování lidského mozku [20]. Biologický mozek se skládá z neuronů a ty jsou tvořeny dendrity, tělem neuronové buňky, jádrem neuronové buňky, axonem, myelinovou pochvou, Schwannovými buňkami, Ranvierovými zářezy a synapsemi, viz obr. 1.1. Samozřejmě neurony využívané v umělé inteligenci se od stavby těch lidských liší, ale jsou inspirovány myšlenkou, na které pracuje právě i biologický mozek. Rozhodnutí je založené na množství buněk, neuronů, které zpracovávají vstupní informaci. Informace na začátku nevstupuje do všech neuronů naráz, ale pouze do vstupní vrstvy, a potom je teprve propuštěna do dalších, tzv. skrytých vrstev neuronů. Tyto neurony informaci zpracovávají a propouštějí do dalších vrstev na základě její relevantnosti. Lze říci, že na způsobu, jakým informace projde neuronovou sítí, resp. které neurony aktivuje a které ji následně propustí do dalších vrstev, závisí její interpretace. Neurony jsou vzájemně propojeny synapsemi. Každá synapse (spoj) je asociována právě s jednou váhou. To, jak je informace důležitá pro daný neuron a jak ji neuron následně interpretuje, je mimo jiné reprezentováno váhou. Více si o váhách a stavbě neuronu uvedeme v následující kapitole 1.1.1.



Obrázek 1.1: Stavba neuronu v lidském mozku. Ilustrace je převzata z [2].

1.1.1 Stavba neuronové sítě a neuronu

1.1.1.1 Neuron

Neuron je základní stavební jednotkou pro NS. Má n vstupů a jeden výstup. Z výstupu neuronu je šířen signál dále do dalších neuronů a vrstev NS nebo je vysílán do okolí. Neuron se skládá z těchto částí:

- Synapse, neboli spojnice mezi neurony. Počet spojení určuje mohutnost sítě a její schopnost uchovat informaci.

- Váha, kterou lze označit za uchovatele paměti NS. Váha je přiřazena ke každé synapsi a definuje hodnotu přenášené informace pro daný neuron.
- Práh a aktivační funkce jsou další důležitou částí neuronu. Prahem označujeme hodnotu, která aktivuje výstup neuronu. Aktivační funkce pak reprezentuje výstupní informaci neuronu. Můžeme říci, že aktivační funkce zpracuje a interpretuje vstupní data.

Tyto termíny blíže popíšeme a upřesníme v dalším textu.

1.1.1.2 Neuronová síť

Neuronová síť se skládá ze základních stavebních jednotek a těmi jsou neurony. Tyto neurony jsou uspořádány do vrstev. Mezi neurony v každých dvou sousedních vrstvách existují spojení - synapse. Vrstvy můžeme rozdělit do tří základních skupin:

- Vstupní vrstva - Takto nazýváme první vrstvu, která je na vstupu do NS. Tato vrstva je odlišná od ostatních tím, že na vstupu do každého neuronu je pouze jedna synapse, která je asociovaná právě s jednou váhou. Blíže se o váhách zmíníme níže v této sekci.
- Skryté vrstvy - Vrstvy, které jsou za vstupní vrstvou, mimo té poslední.
- Výstupní vrstva - jedná se o vrstvu, která může obsahovat jeden nebo více neuronů, na jejichž výstupu zakládáme predikci pro vstupní data. Tato vrstva mívá zpravidla jinou aktivační funkci než zbylé vrstvy, ale toto téma probereme níže v textu 1.1.2.

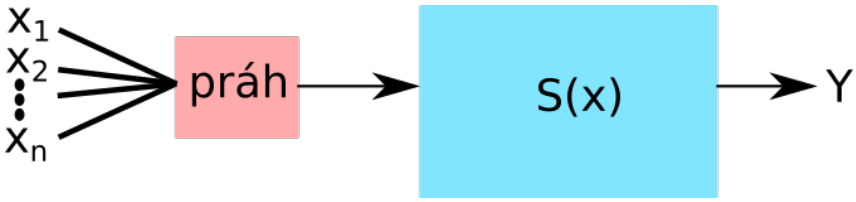
Neuronové sítě lze dále rozdělit podle typu a způsobu propojení jednotlivých neuronů a vrstev nebo na základě uspořádání vrstev NS, viz 1.1.3.

1.1.1.3 Zpracování informace neuronem

Popišme si základní schéma průchodu informace přes neuron, reakci neuronu a následně si toto schéma definujme matematicky. Na vstupu do neuronu máme libovolná vstupní data. Tyto data se dostanou do neuronu po spojnicích, které jsou asociovány s příslušnou váhou.
Než neuron může zpracovat vstupní data, musí být splněna "vstupní podmínka". Tato podmínka by se dala interpretovat, jako míra relevantnosti informace (dat) pro daný neuron. Tato "relevantnost"je spočtena jako součet všech příspěvků spojnic, které vedou do neuronu. Příspěvek spojnice je roven součinu hodnoty vstupních dat, které jsou spojnicí přenášeny, a příslušnou váhou dané spojnice. Pokud tato suma dosáhne určité hodnoty, která je definována jako práh, neuron je aktivován a bude reagovat a zpracovávat daná data dále. Pokud práh je nepřekročen, neuron zůstává v neaktivním stavu a informaci nijak nezpracovává a neprovádí žádnou reakci na daná data. To v praxi znamená, že neuron nevyšle žádný signál ostatním neuronů nebo do okolí a data tak neprostoupí do dalších neuronů.
Předpokládejme, že součet příspěvků spojnic překročil daný práh a data tak vstoupí do aktivního neuronu a ten na ně bude reagovat. Neuron nyní spočte tzv. aktivační funkci a její hodnotu vyšle spojnicemi do ostatních neuronů, které jsou s ním spojeny. Tato výstupní hodnota se stane vstupní hodnotou do dalších neuronů.
Grafické znázornění základních částí neuronu lze vidět na obr.1.2.
Definujme matematický model neuronu [28]:

$$Y = S\left(\sum_{i=1}^{N}(w_i x_i) - h\right) \tag{1.1}$$

kde x_i je i-tý vstup a w_i je váha i-té spojnice (synapse) neuronu. h je práh, $S(x)$ je aktivační funkce neuronu a Y je výstup z neuronu. Váha w_i vyjadřuje a ukládá zkušenost neuronu z trénování. Čím vyšší má hodnotu, tím je příslušný vstup do neuronu důležitější. Váha může nabývat hodnoty mezi 0 a 1. Je-li $\sum_{i=1}^{N}(w_i x_i) < h$, pak neuron je v pasivním stavu, neaktivuje se a tedy nevysílá informaci do dalších neuronů (tzn. $Y = 0$). Blíže si aktivační funkce popíšeme v sekci 1.1.2.



Obrázek 1.2: Schéma zpracování dat neuronem.
Pro lepší přehled zobrazme výsledky a parametry nejlepších modelů v tabulce.

1.1.2 Aktivační funkce

Neurony dělíme na binární a spojité podle toho, jaký je zvolen typ aktivační funkce (AF). Uveďme si jejich základní výčet s jejich krátkým popisem. Detailněji jsou AF popsané v [3] nebo [28]. Z těchto zdrojů jsme také čerpali. Grafické znázornění některých AF lze vidět na obr.1.3. Volba aktivační funkce je mnohdy ovlivněna zkušenostmi s řešením podobných úloh a často je nutné experimentovat.

1.1.2.1 Aktivační funkce jako binární práh

Zaměřme se na tzv. aktivní dynamiku neuronu. Po předložení vstupního vektoru x je vnitřní potenciál neuronu definován

$$z = \sum_{i=1}^{n} w_i x_i - h \tag{1.2}$$

x_i jsou jednotlivé složky vstupního vektoru, w_i jsou jejich příslušné váhy a h je konstantní hodnota prahu. Hodnota vnitřního potenciálu z je vstupním argumentem pro aktivační funkci. Odezvu, neboli výstup, neuronu pak můžeme modelovat jako

$$y = S(z) \tag{1.3}$$

V souvislosti s rovnicí 1.1 a 1.2, zde je práh z praktických důvodů modelován jako nultá váha. Obecně neuron s binárním prahem má práh nastaven na nulu, díky nastavení biasu. Neuron je aktivní, když $w^T x + b > h \Rightarrow x + (b - h) > 0$.

1.1.2.2 Sigmoidální aktivační funkce

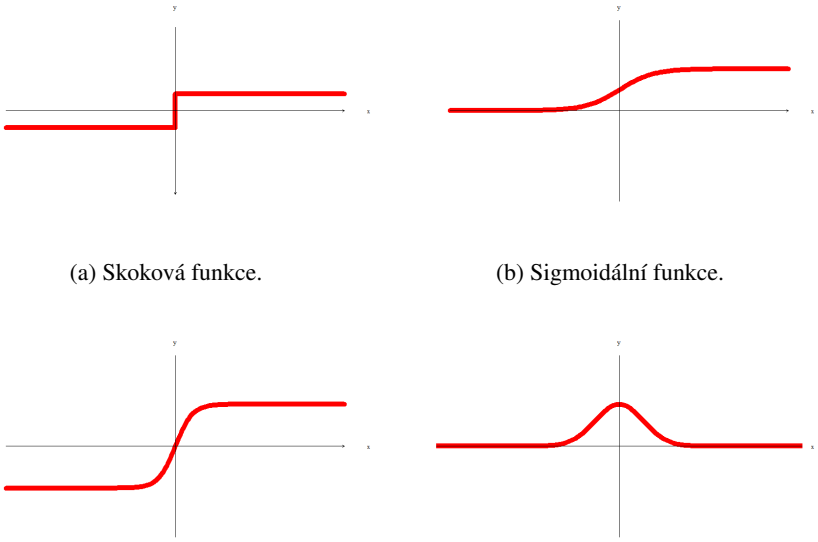
Tato funkce je definována následujícím vztahem

$$y = S(z) = \frac{1}{1 + e^{-z}} \tag{1.4}$$

Kde $z = w^t x + b$ je vstup ze sítě do sigmoidální aktivační funkce. Rozeberme možné výstupy z této aktivační funkce

- Pokud je vstup $z > 0$ velké pozitivní číslo, pak y se blíží rychle k 1, viz obr.1.3b.
- Je-li $z < 0$ velké záporné číslo, pak y rychle klesá k 0.
- Když $z = 0$, potom $y = \frac{1}{2}$.

Výstup ze sigmoidálního neuronu je hladký a má všude spojité první derivace, které jsou důležité pro trénování NS. Jelikož obor hodnot je mezi 0 a 1, tato aktivační funkce se hodí pro binární klasifikaci, více o této základní úloze pro NS popíšeme v 1.1.5. Protože tato funkce je nelineární, dává nám možnost klasifikovat komplexnější úlohy. Vizualizaci této funkce vidíme na obr. 1.3b.



(a) Skoková funkce.

(b) Sigmoidální funkce.

(c) Hyperbolická funkce.

(d) Aktivační funkce radiálního typu.

Obrázek 1.3: Grafická ukázka některých aktivačních funkcí.

1.1.2.3 Softmax

Jedná se o zobecnění sigmoidální aktivační funkce a je velmi vhodná pro klasifikaci do více tříd. Mějme k výstupních tříd a váhový vektor pro i-tou třídu $w^{(i)}$, potom predikovaná pravděpodobnost pro i-tou třídu je výstupem pro vstupní vektor $x \in R^{n\times 1}$ daná vztahem:

$$P(y_i = 1 \mid x) = \frac{e^{w^{(i)T}x+b^{(i)}}}{\sum_{j=1}^{k} e^{w^{(j)T}x+b^{(j)}}} \tag{1.5}$$

kde $b^{(i)}$ je bias pro každou výstupní třídu. Podívejme se na spojitost mezi sigmoidální a softmax aktivační funkcí pro binární klasifikaci. Označme dvě výstupní třídy y_1 a y_2 a korespondující vahové vektory $w^{(1)}$ a $w^{(2)}$. Nechť také k nim příslušející biasy $b^{(1)}$ a $b^{(2)}$. Řekněme že třída y_1 je pozitivní třída a proto $y_1 = 1$.

$$P(y_1 = 1 \mid x) = \frac{e^{w^{(1)T}x+b^{(1)}}}{e^{w^{(1)T}x+b^{(1)}} + e^{w^{(2)T}x+b^{(2)}}} = \frac{1}{1 + e^{-(w^{(1)}-w^{(2)})^T x-(b^{(1)}-b^{(2)})}} \tag{1.6}$$

Vidíme, že pravděpodobnost predikce pozitivní třídy je stejná pro SoftMax aktivační funkci pro klasifikaci do dvou tříd, jako pro sigmoidální aktivační funkci.

1.1.2.4 Aktivační funkce ReLu

V usměrněném lineárním neuronu je výstup roven vstupu do neuronu, když jsou všecny vstupy větší než 0, jinak neuron vrací 0. Výstup z ReLu, může být reprezentován jako:

$$y = S(z) = max(0; z), \tag{1.7}$$

kde $z = w^T x + b$. Relu je považována za revoluční přístup v hlubokém učení. Její velkou výhodou je snadná spočitatelnost. Další výhodou je, že ReLu vhodně spojuje výpočet aktivační funkce s hodnotou jejího gradientu. Gradient aktivační funkce je konstantní, když je vstup do sítě pozitivní, jinak je gradient nulový.
Existuje několik variant aktivačních funkcí, které jsou založeny na ReLu, např. PReLu. Pro standardní

Relu může gradient nabývat nuly i pro záporné hodnoty vstupních hodnot a tím zastavit učení. Pro model, který má nenulový gradient i pro záporné vstupy, může bý vhodné použít PReLu, která je definována:

$$y = \max(0; z) + \beta \min(0; z), \tag{1.8}$$

kde $z = w^T x + b$ je vstup do PReLu a β je parametr učení. Když β je nastavena na -1, pak $y = |z|$ a aktivační funkce se nazývá absolutní hodnota ReLu. Když je β nastavena jako malá pozitivní hodnota, např 0.01, pak se aktivační funkce nazývá slabá ReLu.

1.1.2.5 Aktivační funkce tangens hyperbolický

Tato aktivační funkce je definována jako

$$y = S(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{1.9}$$

$z = w^T x + b$ je vstup do aktivační funkce. Tangens hyperbolický nabývá hodnot mezi -1 a 1. sigmoidální aktivační funkce je saturovaná kolem nuly, což může být problém ve vrstvách, kde je výstupní hodnota blízká nule. Zde se může objevit problém mizícího gradientu a tedy zastavení trénování. tangent hyperbolický je saturován kolem -1 a 1. Další jeho výhodou je dobře definovaný gradient kolem nuly. Vizualizaci této funkce vidíme na obr. 1.3c.

1.1.3 Rozdělení NS

NS můžeme rozlišovat z mnoha hledisek, např. dle stavby NS (její architektury), topologie a způsobu učení atd. Uveďme si proto několik příkladů pro zmíněné oba způsoby dělení NS.
Nejprve zmiňme základní architektury NS:

- Perceptron - Označní matematického modelu biologického neuronu [9]. Konkrétně se jedná o nejjednodušší model dopředné NS, který je tvořen pouze jedním neuronem. Využití takovéto sítě je značně omezené, jelikož ji lze použít pouze pro lineárně separovatelné množiny. Pojmem perceptron se také dříve označoval základní stavební prvek neuronové sítě z toho vznikl např. označení vícevrstvá perceptronová síť. V dnešní době se však pro základní stavební jednotku používá spíše označení "neuron".

- Vrstevnatá NS - Jedná se o rozšíření základního modelu perceptronu o další vrstvy. Každá vrstva se skládá z několika jednotek, které jsou vzájemně propojeny spoji. Díky takovému to složitějšímu modelu můžeme řešit širší množinu úloh [9].

- Rekurentní NS - Zatím se uvažovaly všechny modely, kde spojnice asociované s váham vedli informace a data stále jedním směrem a to od vstupu do neuronu k jeho výstupu. Takovým to sítím říkáme dopředné. Celá NS tak tvoří acyklický graf. Pokud do sítě přidáme i spoje, které vedou zpět do předchozích vrstev, síť se bude nazývat rekurentní. Tyto sítě se využívjí např. pro předpovídání časových řad, strojový překlad, nebo generování textu [31].

- Kohenenovy mapy - [1], zvané také jako SOM (Self-Organizing Maps neboli samoorganizující se mapy). Patří do skupiny samoučících se NS, tzn. sítí, které se učí bez učitele. Pro tento typ trénování nejsou potřeba data, které ke vstupním datům mají korespondující výstupní data. Ty v případě tzv. učení s učitelem udávají konečný cílový stav, do kterého se má síť učením dostat. Právě jejich získání bývá často velkým problémem (časová i finanční náročnost). Naopak u SOM (Kohonenovy mapy) nám například stačí jen skupina nahraných řečových signálů a během učení si síť již sama nalezne společné znaky a odlišnosti.

- Modulární NS - Biologické studie ukazují, že lidský mozek nepracuje jako jediná masivní síť, ale jako soubor malých sítí. Tento výzkum dal zrod konceptu modulárních neuronových sítí, ve kterých několik malých sítí spolupracuje nebo soutěží, aby vyřešily daný problém [3].

Dle topologie NS rozlišujeme dva elementární typy:

- Feed forward (dopředná) - V tomto typu NS vedou váhy vždy jen jedním směrem (ze vstupní vrstvy k výstupní) a celá neuronová síť tedy tvoří acyklický graf. Takové neuronové sítě se chovají zcela reaktivně a jejich předchozí vstupy nijak neovlivňují vstupy následující. To se hodí pro mnoho běžných problémů, jako jsou např. klasifikace obrázků apod. [31]
- Back forward (rekurentní) - Pokud mají být vstupem neuronové sítě posloupnosti různých délek, kde se jednotlivé položky ovlivňují, je vhodnější neuronové síti přidat nějaký vnitřní stav, reprezentovaný typicky pomocí spoje, který vede zpět. Sítě s takovými spoji se nazývají rekurentní a používají se např. pro předpovídání časových řad, strojový překlad, nebo generování textu [31].

NS, která se používá především pro prací s obrazovým vstupem má tyto základní části:

1. První vrstva je vstupní, zde vstupují data do NS.
2. Následující vrstvy jsou konvoluční, zde se aplikuje n filtrů a výstupem je n příznakových map. Více si o těchto vrstvách řekneme v 1.1.13.
3. Další vrstva se nazývá škálovací. Tady se aplikuje na všechny příznakové mapy maxpooling. Máme-li vstupní oblast $k \times k$, výstupem pak bude pouze nejvyšší hodnota z oblasti. Tímto dosáhneme redukci dimenze, s tím je však spojená ztráta informace.
4. Poslední vrstvou jsou výstupní neurony na jejichž výstupu se zakládá predikce o vstupních datech. V případě rozpoznávání obrazu se konkrétně jedná o klasifikaci do tříd.

Blíže si tyto vrstvy popíšeme v následující sekci, viz 1.1.13.

1.1.4 Učení NS

Učení NS je v principu snaha nastavit váhy neuronů tak, aby co nejlépe popisovaly data a dosáhli jsme nejlepší možné predikce. Učení rozdělme na dva základní typy, dle procesu učení na:

- Učení s učitelem - Využívá se zpětná vazba. Po předložení vzoru zjistíme na aktuálním nastavení sítě, predikci. Tuto predikci porovnáme s požadovaným výsledkem (vzorem), na základě toho se zjistí chyba predikce. Spočítáme korekci sítě a upravíme váhy a prahy NS. Tento postup se opakuje, dokud se nedosáhne optimálního výstupu.
- Učení bez učitele - Při trénování se nevyhodnocuje výstup NS. Predikci není s čím porovnat, jelikož správný výsledek není znám. Na vstupu do NS je sada vzorů, které si síť sama třídí. Třídění do skupin probíhá dle typických zástupců nebo NS si přizpůsobí topologii vlastnostem vstupu.

Dále se budeme zabývat jen učením s učitelem, které je vhodné pro naši úlohu.

1.1.5 Perceptron a učení perceptronu

Perceptrony jsou lineární binární klasifikátory, které využívají nadroviny k separaci dat do dvou tříd, [28]. Obecně tyto třídy nazýváme jako kladná (ozn. 1) a záporná (ozn. 0). Nadrovina pro binární separaci je reprezentována rovnicí

$$w^{*T} x^* + b = 0, \tag{1.10}$$

kde jednotkový váhový vektor je kolmý k nadrovině a bias b vyjadřuje vzdálenost nadroviny od počátku. Vektor $w^* \in R^{n \times 1}$ je vybrán vzhledem ke směru orientace pozitivní třídy. Algoritmus pro učení perceptronu je založen na správném nastavení vah a biasu tak, aby docházelo ke správné klasifikaci vstupu. Klasifikace vstupních dat $x^* \in R^{n \times 1}$ je založena na výsledku rovnice nadroviny:

$$w^{*T} x^* + b = c \tag{1.11}$$

Pokud v rovnici (1.11) je

- $c = 0$, leží v této nadrovině a budou přiřazeny do negativní třídy.
- $c > 0$, budou klasifikovány do pozitivní třídy.
- $c \leq 0$, budou klasifikovány do negativní třídy.

Perceptron obvykle neuchovává váhový vektor w^* jako jednotkový vektor, ale spíše jako jakýkoliv obecný vektor . V takovém případě bias b neodpovídá vzdálenosti nadroviny od počátku, ale násobku této vzdálenosti, kde škálovací faktor je norma vektoru w. V souhrnu, když w^* je obecný vektor kolmý k nadrovině a orientovaný ve směru pozitivní třídy, potom $w^{*T}x^* + b = 0$ stále reprezentuje nadrovinu, kde b představuje vzdálenost nadroviny od počátku krát norma vektoru w^*. Cílem strojového učení je najít váhy w^* a b, tj. parametry nadroviny.
Mějme parametrický vektor po přidání biasu, jej ozn. $w \in R^{(n+1)\times 1}$ a vstupní příznakový vektor po přidaní konstanty 1, ozn. $x \in R^{(n+1)\times 1}$. Dostáváme:

$$x^* = [x_1, x_2, x_3 \dots x_n]^T, \tag{1.12}$$

$$x = [1, x_1, x_2, x_3 \dots x_n]^T, \tag{1.13}$$

$$w^* = [w_1, w_2, w_3 \dots w_n]^T, \tag{1.14}$$

$$w = [b, w_1, w_2, w_3 \dots w_n]^T. \tag{1.15}$$

Nyní máme nadrovinu v R^{n+1}. Nadrovina je dána jejím parametrickým váhovým vektorem $w \in R^{(n+1)\times 1}$. Klasifikační pravidla jsou pak následující:

- $w^T x = 0$ je rovnice odpovídající nadrovině.
- Všechny body, které splňují nerovnici $w^T x > 0$, spadají do pozitivní třídy. Z toho vyplývá, že klasifikace je nyní pouze determinována úhlem mezi vektorem w a x. Pokud vstupní vektor x svírá s w úhel mezi $-90°$ a $90°$, pak vektor x bude klasifikovaný do pozitivní třídy.
- Pokud x splňuje nerovnici $w^T x \leq 0$, pak tyto body budou klasifikovány do negativní třídy.

Nyní máme vše připraveno pro učení perceptronu. Mějme m vstupních vektorů a k nim korespondující vektor správných klasifikací, ozn. i-tý vstupní vektor $x^{(i)} \in R^{(n+1)\times 1}$, pro $i \in \{1, 2, \dots m\}$. Dále ozn. i-tý výstupní vektor $y^i \in \{0, 1\}$, pro $\forall i \in \{1, 2, \dots m\}$. Hodnoty výstupního vektoru y^i odpovídají správné klasifikaci pro vstupní vektor x^i.
Učení perceptronu lze rozdělit na tyto kroky:

1. Nejprve začneme s náhodnou množinou vah $w \in R^{(n+1)\times 1}$.
2. Vypočítáme predikovanou třídu pro vstupní vektor. Když platí $w^T x^{(i)} > 0$ pro vstup $x^{(i)}$, pak predikovaná třída je pozitivní, tzn. $y_p^{(i)} = 1$, jinak $y_p^{(i)} = 0$.
3. Nově nastavíme váhový vektor w:
   - Když $y_p^{(i)} = 0$ a aktuální třída je $y^{(i)} = 1$, váha se upraví jako $w = w + x^{(i)}$.
   - Když $y_p^{(i)} = 1$ a aktuální třída je $y^{(i)} = 0$, váha se upraví jako $w = w - x^{(i)}$.
   - Když $y_p^{(i)} = y^{(i)}$, pak váha se neupravuje a zůstává, tak jak je.
4. Vracíme se k bodu 2 a procházíme algoritmus pro další vstupní vektor.
5. Zastavíme učení, pokud všechny vstupy jsou klasifikovány správně.

Perceptron je schopný klasifikovat pouze do dvou tříd. Klasifikace bude správná za předpokladu, že existuje vhodný váhový vektor w, který umí lineárně separovat vstupní data do dvou tříd. V takovém případě, konvergenční teorém perceptronu garantuje konvergenci vektoru w při výše popsaném algoritmu. Detailní popis teorému je např. v [21].

1.1.6 Ztrátové funkce

Ztrátová funkce (někdy ozn. jako chybová funkce), se využívá k modifikaci vah. Modifikováním vah dosáhneme lepšího modelu, který bude lépe predikovat. Ztrátová funkce v sobě nese informaci o tom, jak moc je náš model dobrý a jak se predikce modelu liší od požadovaného výsledku.
Cílem trénování, učení NS, je tedy minimalizovat ztrátovou funkci, viz [11], [12], [13]. Místo ztrátové funkce se někdy používá cenová funkce, která má opačnou hodnotu než ztrátová funkce. Zatímco ztrátu se snažíme minimalizovat, cenovou funkci se naopak snažíme maximalizovat.
Výběr ztrátové funkce je důležitý vzhledem k definování citlivosti výstupu. Mezi nejpoužívanější ztrátové funkce patří:

- Ztrátové funkce používané pro regresní úlohy:

  1. Mean Squared Error Loss (MSE)
     Vyjadřuje přesnost odhadů pomocí střední hodnoty druhých mocnin rozdílu mezi měřením a skutečností [19]. Obecně platí, že čím je odhad přesnější, tím je střední kvadratická chyba menší.
     Nechť $\hat{\eta}$ je odhad nějakého parametru η. Pro jednoduchost předpokládejme, že η je jednorozměrná veličina. Definujme střední hodnotu čtverce chyby $\mathrm{MSE}(\hat{\eta})$:

     $$\mathrm{MSE}(\hat{\eta}) = E[(\hat{\eta} - \eta)^2] \tag{1.16}$$

     Vzhledem k definici je zřejmé, že MSE je vždy nezáporná hodnota, nulová bude tehdy pokud odhad je naprosto přesný. Důležitou vlastností takto definované MSE je, že ji lze teoreticky rozložit na součet čtverce vychýlení (ozn. jako bias nebo jako systematická chyba) a rozptyl odhadu (ozn. náhodná chyba).

     $$\mathrm{MSE}(\hat{\eta}) = \mathrm{Bias}(\hat{\eta})^2 + \mathrm{Var}(\hat{\eta}) \tag{1.17}$$

  2. Mean Squared Logarithmic Error Loss (MSLE)
     Tuto metodu lze interpretovat jako měření poměru mezi skutečnou a odhadovanou hodnotou [6]. MSLE se zajímá pouze o relativní rozdíl mezi odhadem a skutečnou hodnotou pro N realizací náhodné veličiny $\eta = (\eta_1 \dots \eta_N)$ a N predikcí $\hat{\eta} = (\hat{\eta}_1 \dots \hat{\eta}_N)$. Penalizuje více podhodnocený odhad než nadhodnocený odhad. MSLE definujme:

     $$\mathrm{C}(\eta, \hat{\eta}) = \frac{1}{N} \sum_{j=1}^{N} (\log(\eta_j + 1) - \log(\hat{\eta}_j + 1))^2 = \sum_{j=1}^{N} \log^2 \left(\frac{\eta_j + 1}{\hat{\eta}_j + 1} \right). \tag{1.18}$$

  3. Mean Absolute Error Loss (MAE)
     Tato ztrátová funkce se spočte jako suma absolutních hodnot rozdílu mezi odhadem a pravou hodnotou náhodné veličiny [19]

     $$\mathrm{MAE} = \frac{\sum_{i=1}^{N} |\eta_j - \hat{\eta}_j|}{N} \tag{1.19}$$

     Tato metoda je mnohem robustnější vzhledem k odlehlým pozorováním oproti MSE. Proto se tato ztrátová funkce využívá zejména předpokládáme-li, že v trénovacích datech je hodně odlehlých pozorování.

- Ztrátová funkce pro binární klasifikaci:

  1. Binary Cross-Entropy
     Tuto ztrátovou funkci používáme, pokud výstupem z neuronu je rozhodnutí ano nebo ne [16].

Cílem takovéhoto neuronu je klasifikovat, zda vstupní data patří do dané třídy nebo nikoliv. Tuto ztrátovou funkci definujme:

$$C(\eta, \hat{\eta}) = -\frac{1}{N}\sum_{j=1}^{N}(\eta_j \log(\hat{\eta}_j) + (1-\eta_j)\log(1-\hat{\eta}_j)) \tag{1.20}$$

η resp. $\hat{\eta}$ je náhodná proměnná, resp. její odhad. Binary cross-entropy měří, jaké chyby jsme se dopustili pro každý odhad, který se snaží vstupní data klasifikovat do správné třídy (tzn. výstup je 0 nebo 1). Tyto chyby jsou následně zprůměrovány přes všechny odhady a tak obdržíme výslednou chybu.

2. Hinge Loss
   Jedná se o alternativní ztrátovou funkcí k funkci Cross-Entropy při klasifikaci pro binární úlohy [14]. Definujeme

$$C(\eta, \hat{\eta}) = \sum_{j=1}^{N}(\max(0, 1-\eta_j\hat{\eta}_j)), \tag{1.21}$$

   kde $\hat{\eta}$ je odhad veličiny η, která nabývá hodnoty hodnoty -1 nebo 1.

3. Squared Hinge Loss
   Tuto ztrátovou funkci používáme v úlohách řešících binární problém [14]. Nejčastěji se kombinuje s aktivační funkcí tangent hyperbolický v poslední vrstvě. Definujeme

$$C(\eta, \hat{\eta}) = \sum_{j=1}^{N}(\max(0, 1-\eta_j\hat{\eta}_j)^2) \tag{1.22}$$

   kde $\hat{\eta}$ je odhad veličiny η, která nabývá hodnoty hodnoty -1 nebo 1

- Ztrátové funkce pro klasifikační úlohy s více třídami:

1. Multi-Class Cross-Entropy Loss
   Tuto ztrátovou funkci využíváme především, pokud chceme vstupní data klasifikovat do M tříd [17]. Tuto funkci definujme:

$$C(\eta, \hat{\eta}) = -\sum_{j=1}^{N}\sum_{i=1}^{M}(\eta_{ij}\log(\hat{\eta}_{ij}), \tag{1.23}$$

   kde $\hat{\eta}_{ij}$ je odhad náhodné proměnné η_{ij}, že j-tý vzorek patří do i-té třídy. V tomto případě ztrátová funkce srovná rozdělení predikce a skutečné náhodné veličiny.

2. Sparse Multiclass Cross-Entropy Loss
   Pro klasifikaci do více tříd můžeme použít také tuto ztrátovou funkci [16]. Představme si, že máme klasifikační problém o M třídách, tzn. v poslední vrstvě máme M neuronů s aktivační funkcí softmax. Když bychom použili jako ztrátovou funkci Multi-Class Cross-Entropy Loss, dostaneme M-dimenzionální vektor. Tento vektor má všechny prvky nulové kromě těch, který odpovídají třídě, do kterých jsme vstupní data klasifikovali. Problém je v tom, že při této ztrátové funkci se nám může stát, že na výstupu můžeme data klasifikovat do více tříd zároveň. Toto může být výhodné, např. když budu klasifikovat náladu podle obrázku. Člověk může být zároveň veselý, nadšený a šťastný. Ovšem problém nastane, když bychom měli úlohu, kde jsou třídy disjunktní, např. klasifikujeme do kategorií rostlina, přístroj a zvíře. Pokud použijeme Sparse Multiclass Cross-Entropy Loss, obdržíme jedinou výstupní hodnotu (nikoliv vektor). Tato hodnota pak reprezentuje třídu, do které vstupní data klasifikujeme. Tuto ztrátovou funkci definujme jako

$$C(\eta, p) = -\sum_{i=1}^{M}\eta_{o,i}\log(p_{o,i}). \tag{1.24}$$

tato definice předpokládá, že počet tříd M je větší než 2, η je binární veličina, která nabývá hodnot 0 nebo 1, p je predikovaná pravděpodobnost, že pozorování o patří do třídy i, tzn. $\eta_{o,i}$.

3. Kullback-Leibler Divergence Loss
   V podstatě se jedná o vzdálenost mezi dvěma distribucemi pravděpodobnosti [15]. definujme ji jako

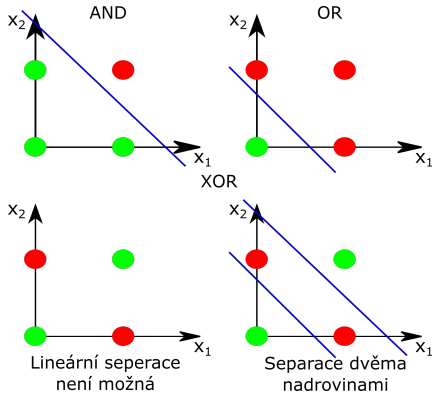
$$\mathrm{KL}(P\|Q) = \sum P(X) \log\left(\frac{P(X)}{Q(X)}\right). \tag{1.25}$$

   Jednoduše řečeno, rovnice (1.25) nám říká, kolik můžeme ztratit nebo získat entropie, když změníme distribuční rozdělení P na Q. Proto se této funkci říká také relativní entropie.
   Tato ztrátová funkce lze využít na podobné úlohy jako např. Multiclass Cross-Entropy Loss.

Ztrátová funkce má velmi důležitou a obtížnou úlohu, jelikož musí jedním číslem výjářit všechny aspekty a vlastnosti modelu tak, aby zlepšením tohoto čísla jsme vylepšili i model [18], [33].

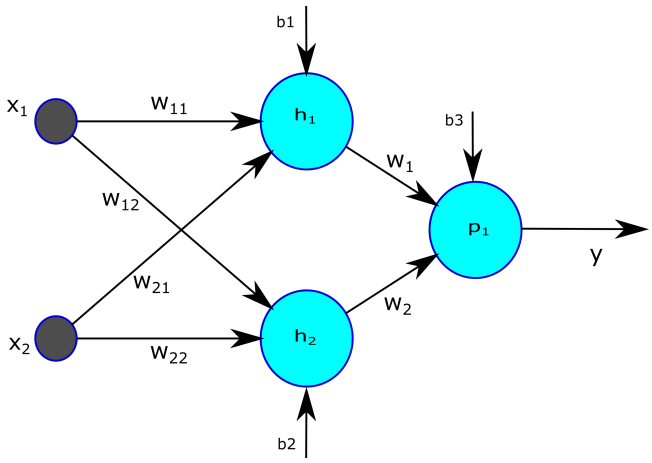
1.1.7 Pravidla učení pro vícevrstvé sítě

Vícevrstvé sítě obsahují skryté vrstvy, díky tomu jsou schopny se naučit dělat nelineární rozhodnutí. Proto je potřeba, aby výstup z aktivační funkce byl nelineární, jako např. sigmoidální funkce, tanh, ReLu atd. Výstupní neuron by pak měl mít pro binární klasifikaci sigmoidální aktivační funkci. Ta je potřeba, aby byly splněny podmínky pro Binary Cross-Entropy ztrátovou funkci. Na této funkci závisí pravděpodobnost klasifikace do konkrétní třídy.



Obrázek 1.4: Vizualizace separace nadrovinou logické operace AND (vlevo nahoře), OR (vpravo nahoře) a XOR (dole).

Toho lze využít při modelování základních logických funkcí. V minulosti byla jedním z hlavních argumentů proti NS neschopnost perceptronu vyřešit XOR úlohu. XOR je logická funkce, která není lineárně separabilní (na rozdíl od logických funkcí AND, OR a NOT). Jeden perceptron proto nemůže sám o sobě řešit úlohu, založenou na XOR logické funkci. Uvažujeme-li tuto úlohu ve dvoudimenzionálním prostoru, potřebujeme ještě jednu nadrovinu, která rozdělí prostor, tzn. přidáme ještě jeden neuron na výstup prvního. Obecně lze říci, že přidáváním neuronů lze řešit komplexnější a složitější úlohy, což je motivace k tvorbě vícevrstvých a hlubokých NS. Jednoduše řečeno XOR úloha spočívá v tom, že se snažíme dvoudimenzionální prostor separovat pomocí dvou přímek.



Obrázek 1.5: Ukázka architektury NS, která je schopná řešit XOR problém.

Vzhledem k předchozím úvahám zkusme řešit XOR úlohu [28]. Nechť máme vstupní vektor do NS $\boldsymbol{x} \in (x_1, x_2)$, kde $x_1 \in (0, 1)$ a $x_2 \in (0, 1)$. Existují celkem čtyři kombinace vstupů a k nim příslušející výstupy (klasifikace do třídy):

- $x_1 = 0$, $x_2 = 1$ a $y = 1$
- $x_1 = 1$, $x_2 = 0$ a $y = 1$
- $x_1 = 1$, $x_2 = 1$ a $y = 1$
- $x_1 = 0$, $x_2 = 0$ a $y = 1$

Ztrátovou funkci zvolme log-loss funkci, kterou budeme minimalizovat vzhledem k parametrům modelu, tzn. vzhledem k váhovému vektoru a biasu. Předpokládejme, že všechny neurony v síti mají sigmoidální aktivační funkci. Nechť vstup a výstup neuronu h_1 ve skryté vrstvě je i_1 a z_1. Obdobně mějme neuron h_2 ve skryté vrstvě a jemu odpovídající vstup i_2 a výstup z_2. Nakonec označme vstup a výstup z výstupní vrstvy p_1 jako i_3 a z_3. Dostáváme vztahy

$$i_1 = w_{11}x_1 + w_{21}x_2 + b_1, \tag{1.26}$$

$$i_2 = w_{12}x_1 + w_{22}x_2 + b_2, \tag{1.27}$$

$$z_1 = \frac{1}{1 + e^{-i_1}}, \tag{1.28}$$

$$z_2 = \frac{1}{1 + e^{-i_2}}, \tag{1.29}$$

$$i_3 = w_1 z_1 + w_2 z_2 + b_3, \tag{1.30}$$

$$z_3 = \frac{1}{1 + e^{-i_3}}. \tag{1.31}$$

Ozn. θ parametrický vektor, který hromadně označuje všechny váhy a biasy $(w_1, w_2, w_{11}, w_{12}, w_{21}, w_{22}, b_1, b_2, b_3)$. Uvažujme ztrátovou funkci Binary Cross-Entropy. Celkovou ztrátovou funkci pro XOR problém můžeme vyjádřit jako

$$C(\theta) = \sum_{j=1}^{4} -y^{(j)} \log z_3^{(j)}(x^{(j)}, \theta) - (1 - y^{(j)}) \log(1 - z_3^{(j)}(x^{(j)}\theta)) \tag{1.32}$$

Potom může probíhat učení modelu skrze minimalizaci funkce C:

$$\theta^* = \arg\min_{\theta} C(\theta) \tag{1.33}$$

Gradient funkce C v tomto minimu, vzhledem k parametru θ, by měl být roven 0. Toto minimum proto můžeme vypočítat např. pomocí algoritmu gradient descent, viz sekce 1.1.8.1.

1.1.8 Optimalizační algoritmy implementované v TensorFlow

Optimlizační algoritmy se používají ke korekci vah a hledání minimální chyby, způsobené rozdílem mezi predikcí a očekávaným výsledkem. Software TensorFlow, který používáme, nabízí několik optimalizačních algoritmů, které jsou detailně popsány v [28]. Některé si uveďme spolu s jejich charakteristikou.

1.1.8.1 Gradient descent

Gradient descent je iterativní optimalizační algoritmus, který hledá lokální minimum funkce. Tento algoritmus se k lokálnímu minimu přibližuje jednotlivými kroky, které jsou úměrné k záporné hodnotě gradientu dané funkce v konkrétním bodě [10].

Modifikační pravidlo pro gradient descent je v tomto případě

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla C(\theta^{(t)}), \tag{1.34}$$

kde η je koeficient rychlosti učení a $\theta^{(t+1)}$ resp. $\theta^{(t)}$ jsou parametrické vektory iterace $t+1$ resp. t. Uvažujme konkrétní váhu $w_k^{(t)}$ z parametrického vektoru θ. Modifikační pravidlo pro změnu této váhy $w_k^{(t)}$ na novou váhu $w_k^{(t+1)}$ pomocí algoritmu gradient descent bude vypadat:

$$w_k^{(t+1)} = w_k^{(t)} - \eta \frac{\partial C(w_k^t)}{\partial w_k}, \tag{1.35}$$

pro $w_k \in \theta$. Jelikož výpočet gradientu vektoru není tak snadný jako v lineární nebo logistické regresi, tak se v NS váhy řídí logistickým uspořádáním. Pravidlo řetězení derivací však nám umožní zjednodušení výpočtu parciálních derivací s ohledem na váhy (včetně zkreslení). Toto pravidlo se využívá v metodě zpětné propagace a umožňuje snadný výpočet gradientu, viz sekce 1.1.8.2.
Vraťme se ještě k aktivačním funkcím, viz 1.1.2, a porovnejme Relu se sigmoidální funkcí vzhledem k využití gradientu. Sigmoidální funkce má gradient nulový (nebo skoro nulový) pro velkou část vstupních hodnot (pozitivních i negativních), proto NS může trpět problémem tzv. mizení gradientu. Konstantní gradient využívaný v ReLu tomuto problému předchází. Důsledkem tak je, že pokud se NS trénuje pomocí algoritmu gradient descent, síť v případě konstantního gradientu ukončí trénování.

Gradient může být spočten buď pro celý dataset, pak mluvíme o *batch gradient descent*, nebo nebo pro náhodné vzorky dat, pak mluvíme o stochastic gradient descent. Jelikož batch gradient descent je pomalý a *stochastic gradient descent* zase pro jednoduchá data může být značně zašuměn, tak se používá varianta *mini-batch gradient descent* [22]. Tato metoda dosahuje mnohem lepších výsledků než předchozí dvě. Jelikož tento základní algoritmus je známý a dobře prozkoumaný, je zde neustálá snaha o jeho vylepšení pmocí heuristik a numerických metod. Bližší popis nejrůznějších metod nalezneme např. v [23, 27, 39, 40].

1.1.8.2 Aplikace metody zpětné propagace na výpočet gradientu

Tato metoda je užitečná pro propagaci chyby z výstupní vrstvy zpět do sítě a na jejím základě modifikovat váhy. pomocí řetězového pravidla můžeme snadno spočítat gradienty v předchozích vrstvách. Pravidlo si ukažme na modelovém příkladu. Uvažujme XOR architekturu sítě, viz [28] vizualizovanou na obr. 1.5. Máme vstupní vektor $\boldsymbol{x} = [x^1, x^2]^T$, který je klasifikován do třídy y, takže ztrátová funkce pro jednu iteraci výpočtu vypadá

$$C(\theta) = -y \log z_3 - (1-y)\log(1-z_3). \tag{1.36}$$

$$\frac{\partial C}{\partial w_1} = \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial w_1}, \tag{1.37}$$

$$\frac{dC}{dz_3} = \frac{z_3 - y}{z_3(1-y)}. \tag{1.38}$$

Nyní

$$z_3 = \frac{1}{1+e^{-z^3}} \tag{1.39}$$

$$\frac{dz_3}{di_3} = z_3(1-z_3) \tag{1.40}$$

$$\frac{dC}{di_3} = \frac{dC}{dz_3}\frac{dz_3}{di_3} = \frac{z_3 - y}{z_3(1-z_3)} z_3(1-z_3) = z_3 - y \tag{1.41}$$

Můžeme pozorovat, že derivace ztrátové funkce podle vstupu do výstupní vrstvy není nic jiného než chyba při odhadu výstupu $z_3 - y$:

$$\frac{\partial i_3}{\partial w_1} = z_1 \tag{1.42}$$

$$\frac{\partial C}{\partial w_1} = \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial w_1} = (z_3 - y)z_1 \tag{1.43}$$

Podobně

$$\frac{\partial C}{\partial w_2} = \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial w_2} = (z_3 - y)z_2 \tag{1.44}$$

$$\frac{\partial C}{\partial b_3} = \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial b_3} = (z_3 - y) \tag{1.45}$$

Nyní spočteme parciální derivace vzhledem k váhám v první vrstvě

$$\frac{\partial C}{\partial z_1} = \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial z_1} = (z_3 - y)w_1 \tag{1.46}$$

$\frac{\partial C}{\partial z_1}$ lze považovat za chybu výstupu neuronu ve skryté vrstvě h_1. Tato chyba se proporcionálně propaguje dále do vah ve skrytých vrstvách, které jsou spojeny s výstupem tohoto neuronu. Pokud by zde bylo několik jednotek, pak $\frac{\partial C}{\partial z_1}$ by byl tvořen příspěvkem chyby od každé výstupní jednotky. Obdobně

$$\frac{\partial C}{\partial i_3} = \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial z_1}\frac{dz_1}{di_1} = (z_3 - y)w_1 z_1(1 - z_1) \tag{1.47}$$

člen $\frac{\partial C}{\partial i_1}$, můžeme považovat za chybu na vstupu do neuronu ve skryté vrstvě h_1. Tato chyba se snadno spočte když $w_1 z_1(1 - z_1)$ vynásobíme $\frac{\partial C}{\partial z_1}$.

$$\frac{\partial C}{\partial w_{11}} = \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial z_1}\frac{dz_1}{di_1}\frac{\partial i_1}{\partial w_{11}} = (z_3 - y)w_1 z_1(1 - z_1)x_1 \tag{1.48}$$

$$\frac{\partial C}{\partial w_{21}} = \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial z_1}\frac{dz_1}{di_1}\frac{\partial i_1}{\partial w_{21}} = (z_3 - y)w_1 z_1(1 - z_1)x_2 \tag{1.49}$$

$$\frac{\partial C}{\partial b_1} = \frac{dC}{dz_3}\frac{dz_3}{di_3}\frac{\partial i_3}{\partial z_1}\frac{dz_1}{di_1}\frac{\partial i_1}{\partial w_{21}} = (z_3 - y)w_1 z_1(1 - z_1) \tag{1.50}$$

Jakmile máme parciální derivace ztrátové funkce pro každý vstup do všech neuronů, můžeme spočítat parciální derivaci ztrátové funkci s ohledem na váhu , která přispívá ke vstupním datům do neuronu, stačí nám vstup vynásobit pouze touto váhou.

1.1.8.3 Adagrad

Jedná se o optimalizátor založený na first-order (derivace prvního řádu), podobně jako gradient descent, ale s některými modifikacemi. Namísto jednoho globálního koeficientu učení (viz η v 1.1.8.1), je tento koeficient normalizován pro každou dimenzi, na které je aktivační funkce (funkce ocenění) závislá. Mějme ztrátovou funkci $C(\theta)$, kde $\theta = [\theta_1, \ldots, \theta_n]^T \in R^{n\times 1}$, pak modifikační pravidlo pro θ_j je:

$$\theta_j^{t+1} = \theta_j^t - \frac{\eta}{\sqrt{\sum_\tau^t \theta_j^{(\tau)2} + \epsilon}}\frac{\partial C^{(t)}}{\partial \theta_j}, \tag{1.51}$$

kde η je koeficient učení a θ_j^t a θ_j^{t+1} jsou hodnoty i-tého parametru v t-té iteraci resp. v $t + 1$ iteraci. Maticový zápis pro modifikaci parametrů vektoru θ lze vyjádřit jako:

$$\theta^{t+1} = \theta^t - \eta G_t^{-1}\nabla C(\theta^t), \tag{1.52}$$

kde G_t je diagonální matice, která obsahuje l^2 normu předchozích gradientů až do iterace t pro každou dimenzi a má tvar

$$G_t = \begin{pmatrix} \sqrt{\sum_{\tau=1}^{t} \theta_1^{(\tau)2} + \epsilon} & \dots & 0 \\ \vdots & \sqrt{\sum_{\tau=1}^{t} \theta_j^{(\tau)2} + \epsilon} & \vdots \\ 0 & \dots & \sqrt{\sum_{\tau=1}^{t} \theta_n^{(\tau)2} + \epsilon} \end{pmatrix}. \tag{1.53}$$

Někdy řídké příznaky, tzn. příznaky nevyskytující se často v datech, mohou být velmi užitečné pro optimalizační úlohu. Pro základní nebo stochastický gradient descent, učící koeficient klade stejnou důležitost na všechny příznaky v každé iteraci. Jelikož koeficient učení je stejný, pak celkový přínos neřídkých příznaků bude větší než řídkých příznaků. Adagard algoritmus pomáhá zabraňovat ztrátě těchto kritických informací z řídkých příznaků, díky modifikaci pomocí jiných koeficientů učení pro různé příznaky. Řídké příznaky kvantity $\sqrt{\sum_{\tau=1}^{t} \theta_1^{(\tau)2} + \epsilon}$ by měly být malé, a proto celkový koeficient učení bude velký. Tento optimalizátor je vhodný pro použití v NLP (natural language processing) nebo pro zpracování obrazu s řídkými daty.

1.1.8.4 RMSprop

Tato metoda je vhodná pro malé soubory dat. RMSprop pochází z odolných zpětno-propagačních (Rprop) optimalizačních technik. Tento optimalizátor řeší problém, kdy gradient neukazuje na minimum v případech, kde kontury aktivační funkce jsou eliptické. Výjimečnost Rprop je v tom, že nevyužívá hodnotu gradientu vah, ale pouze znamének determinujících, jakým směrem modifikovat každou váhu. Popišme logiku s jakou Rprop pracuje:

- Pro všechny váhy začínáme se stejným znaménkem modifikace váhy: $\Delta_{ij}^{(t=0)} = \Delta_{ij}^{(0)} = \Delta$
  Dále nastavíme přípustné minimum a maximum váhy, ozn. $\Delta_{\min}$, resp. $\Delta_{\max}$. Indexy i, resp. j odpovídají řádkovému resp. sloupcovému indexu vah zapsaných do matice.

- V každé iteraci zkontrolujeme znaménko předchozích i aktuálních složek gradientu. Uvažujme, že symboly předchozích i aktuálních složek gradientu pro váhy synapsí jsou stejné např.

  $$\operatorname{sgn}\left(\frac{\partial C^t}{\partial w_{ij}} \frac{\partial C^{t-1}}{\partial w_{ij}}\right) = +1. \tag{1.54}$$

  Dále ozn. koeficient učení η_+ a nastavme ho např. na hodnotu $\eta_+ = 1.2$, viz[28]. Poté dojde k modifikaci podle pravidla:

  $$\Delta_{ij}^{(t+1)} = \min\left(\eta_+ \Delta_{ij}^{(t)}, \Delta_{\max}\right), \tag{1.55}$$

  $$w_{ij}^{t+1} = w_{ij}^{t} - \operatorname{sgn}\left(\frac{\partial C^t}{\partial w_{ij}}\right) \Delta_{ij}^{(t+1)}. \tag{1.56}$$

  Pokud je rozdíl mezi znaménky pro předchozí a aktuální složky gradientu, např. $\operatorname{sgn}\left(\frac{\partial C^t}{\partial w_{ij}} \frac{\partial C^{t-1}}{\partial w_{ij}}\right) = -1$, pak koeficient učení poklesne a ozn. η_- a nastavme ho např. na hodnotu $\eta_- = 0.5$, viz[28]. Potom modifikační pravidla jsou:

  $$\Delta_{ij}^{(t+1)} = \max\left(\eta_- \Delta_{ij}^{(t)}, \Delta_{\min}\right), \tag{1.57}$$

  $$w_{ij}^{t+1} = w_{ij}^{t} - \operatorname{sgn}\left(\frac{\partial C^t}{\partial w_{ij}}\right) \Delta_{ij}^{(t+1)}. \tag{1.58}$$

- Pokud $\frac{\partial C^t}{\partial w_{ij}} \frac{\partial C^{t-1}}{\partial w_{ij}} = 0$, je modifikační pravidlo:

  $$\Delta_{ij}^{(t+1)} = \Delta_{ij}^{(t)}, \tag{1.59}$$

  $$w_{ij}^{t+1} = w_{ij}^{t} - \operatorname{sgn}\left(\frac{\partial C^t}{\partial w_{ij}}\right) \Delta_{ij}^{(t+1)}. \tag{1.60}$$

1.1.8.5 Adadelta

Tento optimalizátor je variantou Adagrad, která je méně agresivní ve smyslu zmenšování koeficient učení. Adadelta optimalizátor bude konstantně škálovat koeficient učení pro každou synapsy i v jednotlivých iteracích pomocí rozdělení na kvadratické středy. Proto pro každou váhu je koeficient učení ve své podstatě monotónní klesající funkce s počtem iterací, takže se stává nekonečně malým. Efektivní koeficient učení v Adadelta optimalizátoru bere více v úvahu lokální odhady aktuálního gradientu a nezmenšuje se tak rychle jako v Adagrad metodě. Tím dosáhneme plynulejšího učení. Pravidlo pro modifikaci vah v Adadelta metodě lze zapsat

$$g_{ij}^t = \gamma g_{ij}^t + (1-\gamma)\left(\frac{\partial C^{(t)}}{\partial w_{ij}}\right)^2, \tag{1.61}$$

$$w_{ij}^{t+1} = w_{ij}^t - \frac{\eta}{\sqrt{g_{ij}^t + \epsilon}} \frac{\partial C^{(t)}}{\partial w_{ij}}, \tag{1.62}$$

kde γ je exponenciální rozpadová konstanta, η je konstantní koeficient učení a g_{ij}^t představuje efektivní gradient průměrných čtverců v iteraci t. Ozn. v rovnici (1.62) $\mathrm{RMS}(g_{ij}^t) = \sqrt{g_{ij}^t + \epsilon}$ takže

$$w_{ij}^{t+1} = w_{ij}^t - \frac{\eta}{\mathrm{RMS}(g_{ij}^t)} \frac{\partial C^{(t)}}{\partial w_{ij}} \tag{1.63}$$

Nechť $h_{ij}^{(t)}$ je průměr čtverce váhy, která byla modifikována v iterci $t, \beta \in (0,1)$ je rozkladová konstanta a $\Delta w_{ij}^{(t)}$ je váha upravená v iteraci t. Pak pravidlo pro modifikci $h_{ij}^{(t)}$ a finální pravidlo pro modifikaci vah v Adadelta optimalizátoru vyjádříme

$$h_{ij}^{(t)} = \beta h_{ij}^{(t-1)} + (1-\beta)\left(\Delta w_{ij}^t\right)^2, \tag{1.64}$$

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \frac{\sqrt{h_{ij}^{(t)} + \epsilon}}{\mathrm{RMS}(g_{ij}^{(t)})} \frac{\partial C^{(t)}}{\partial w_{ij}} \tag{1.65}$$

a s využitím rovnice (1.65) dostáváme

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \frac{\mathrm{RMS}(h_{ij}^{(t)})}{\mathrm{RMS}(g_{ij}^{(t)})} \frac{\partial C^{(t)}}{\partial w_{ij}}. \tag{1.66}$$

Hlavní výhodou tohoto optimalizátoru je naprostá eliminace konstantního koeficientu učení. Když nebudeme uvažovat eliminaci konstantního koeficientu učení, pak jsou metody Adadelta a RMSprop stejné. Obě metody byly vyvíjeny nezávisle ve stejném období pro řešení problému rychlého rozkladu (rychlé degradace) koeficientu učení v Adagrad.

1.1.8.6 Adam

Adam neboli adaptivní odhad momentu je jinou optimalizační metodou, která stejně jako Adadelta a RMSprop má adaptivní učící koeficient pro každou váhu a parametr. Adam neuchovává pouze aktuální průměr druhé mocniny gradientu, ale i minulého průměru druhé mocniny gradientu. Mějme koeficient degradace průměru gradientů m_{ij}^t a průměr druhé mocniny gradientů v_{ij}^t, pro každou váhu w_{ij} je označme po řadě β_1 resp. β_2. Nechť η je konstantní faktor koeficientu učení. Potom modifikační pravidla vah pro Adam optimalizátor jsou následující:

$$m_{ij}^{(t)} = \beta_1 m_{ij}^{(t-1)} + (1-\beta_1)\frac{\partial C^{(t)}}{\partial w_{ij}^{(t)}} \tag{1.67}$$

$$v_{ij}^{(t)} = \beta_2 v_{ij}^{(t-1)} + (1-\beta_2)\left(\frac{\partial C^{(t)}}{\partial w_{ij}^{(t)}}\right)^2 \tag{1.68}$$

Normalizovaný průměr gradientů $\hat{m}_{ij}^{(t)}$ a normalizovaný průměr druhé mocniny gradientů $\hat{v}_{ij}^{(t)}$ se vypočítají jako:

$$\hat{m}_{ij}^{(t)} = \frac{m_{ij}^{(t)}}{1+\beta_1^{(t)}} \tag{1.69}$$

$$\hat{v}_{ij}^{(t)} = \frac{v_{ij}^{(t)}}{1+\beta_2^{(t)}} \tag{1.70}$$

Finální pravidlo pro modifikaci každé váhy je:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \frac{\eta}{\sqrt{\hat{v}_{ij}^{(t)}+\epsilon}}\hat{m}_{ij}^{(t)} \tag{1.71}$$

1.1.9 Softwarové nástroje pro deep learning

Mezi známé frameworky a balíčky, které se používají pro hluboké učení, jsou následující [20]:

- Torch - Mezi podporované jazyky patří např. C/Cuda. Tato knihovna obsahuje mimo jiné spoustu podpůrných balíčků pro zpracování obrazu.
- Cafee - V tomto balíčku je kladen důraz především na rychlost a modularitu. Mimo jiné je tato knihovna hojně používána pro zpracování obrazu. Jazyky, které podporuje, jsou např. C/C++, Python a MATLAB.
- TensorFlow - Jedná se o soubor knihoven podporovaných mimo jiné na Linuxu. Mezi jazyky, které jsou pro tento balíček podporovány patří např. C++ a Python. Do TensorFlow je také plně integrovaný Keras, který byl dříve samostatně.
- Apache MXNet - Je vzdělávací framework se snadno použitelným API pro strojové učení. MXNet obsahuje rozhraní Gluon, které umožňuje provádět hluboké učení v cloudu, mimo jiné i v mobilních aplikacích [4].
- Theano - Knihovna v Pythonu, která vám umožňuje efektivně definovat, optimalizovat a vyhodnocovat matematické výrazy zahrnující vícerozměrná pole, stejně jako provádět hluboké učení NS [38].
- CuDNN - knihovna od NVIDIA s algoritmy implementujícími trénování hlubokých NS na CUDA.

1.1.10 Základní pojmy pro implementaci NS

Neuronové sítě, které jsme si popisovali výše viz. 1.1, jsou trénovány přes mini-batch stochastického gradientu. Seznamme se s několika základními pojmy, které budeme potřebovat pro implementaci učení v TensorFlow, ale i v jiných sotwarech:

- Velikost dávky dat, v anglické literatuře ozn. jako batch size, udává počet trénovacích dat v každé mini-dávce. Velikost jedné mini-dávky by měla být vybrána, tak aby se dosáhlo dostatečně dobrého odhadu gradientu pro celý trénovací dataset a zároveň dostatečného šumu, aby algoritmus mohl uniknout z falešných lokálních minim, které nedávají dobré zobecnění. Po klasifikaci vstupních dat, jejichž počet odpovídá velikosti mini-dávky, dojde k modifikaci (korekci) vah v síti. Tomuto procesu říkáme trénování neboli učení NS.
- Počet mini-dávek, je celkový počet mini-dávek v celém trénovacím datasetu.

- Epochy - Jedna epocha se skládá z jednoho úplného tréninku přes celý dataset.

Když to tedy shrneme, celé učení NS je rozděleno do několika epoch. Tyto epochy jsou v podstatě jakési cykly opakování a prohloubení již naučených vzorů, přičemž každá epocha se dělí na několik mini-dávek. V dávce dochází právě k jednomu kolu učení a zpětné reflexi. Tedy ověření přesnosti naučení a korekci chyb.
Tento proces lze přirovnat k učení člověka, např. Představme si, že za týden píšeme test a chceme se na něj naučit. Každý den si budeme opakovat naučenou látkou a prohlubovat naše znalosti, tedy jeden den odpovídá jedné epoše. Každý den si nejprve vyberu daný počet příkladů (jedna mini-dávka), spočtu je, ověřím jejich výsledky a na základě toho opravím postup v příkladech. Tento postup odpovídá tréninku. Ten samý den náhodně vyberu opět stejný počet příkladů a proces, trénink, několikrát zopakuji. Tedy den (epocha) se skládá z několika mini-dávek, tréninků.

1.1.11 Přeučení (overfitting)

K přeučení dochází, když si náš model zapamatuje trénovací data, tzn. model má skvělé výsledky na trénovacích datech, ale špatné na testovací resp. validační sadě. K přeučení modelu dochází, když se model snaží predikovat trend na základě dat, která jsou příliš zašuměná. K tomu může dojít, když je model příliš složitý a má zbytečně moc parametrů. Model je nepřesný, jelikož trend neodráží skutečná data. Zda dochází k přeučení můžeme posoudit, porovnáme-li výsledky z trénovací množiny (tzn. model znal správné výsledky) a testovací množiny (tzn. model nezná správné výsledky). Cílem učení je zobecnit trend dat v trénovací množině na jakákoliv data, která se týkají dané úlohy. Tzn. v budoucnu chceme, aby model správně predikoval i data, které před tím nikdy neviděl, ale která spadají do domény problému. Základní techniky, jak předcházet přeučení, jsou:

- Zjednodušení modelu - Snížíme počet neuronů nebo synapsí.
- Zkrácení učení - Snížíme počet epoch.
- Regularizace - V datech mohou vznikat náhodné vzory a my se tyto vzory snažíme narušit.
- Prořezávání - Snížíme počet synapsí mezi neurony neboli je budeme náhodně vypínat v průběhu trénování.
- Rozšíření dat - Přidáme nová trénovací data do datasetu.

Detailní popis jednotlivých metod nalezneme např. v [34]

1.1.12 Nedoučení (underfitting)

Zcela opačný problém než přeučení, sekce 1.1.11, představuje nedoučení. Model, který je nedoučený, nemůže správně predikovat data z trénovací množiny, a už vůbec ne data z testovací množiny. Jedním z důvodů může být složitost modelu a přílišný počet parametrů modelu. Oproti přeučení však data nepokryjí všechny případy a nebo nejsou zastoupeny v dostatečné míře, a proto síť si nemůže správně matematicky namapovat vstupní data na výstupní.

1.1.13 Konvoluční NS (CNN)

Myšlenkou konvolučních neuronových sítí je lokální porozumění obrazu. Tato idea je nastíněna v [36]. Jednou z praktických výhod je menší počet potřebných parametrů, což má výrazný vliv na čas potřebný k trénování NS. Stejně tak dalším výhodným důsledkem je potřeba menší množiny trénovacích dat. Místo toho, abychom každému pixelu v obrazu přiřazovali váhu, a každý pixel tak byl brán jako jednotlivý vstup do NS, do konvolučních NS vstupuje malá část z obrazu, kterému je přiřazena váha atd. To znamená, že obraz je zkoumán na úrovni jednotlivých malých bloků, ale ne na úrovni pixelů. Procházení obrázku po jednotlivých oknech zajišťuje konvoluční vrstva, viz sekce 1.1.15. Typická CNN

má několik konvolučních vrstev. Další výhodou CNN je, že počet parametrů je nezávislý na velikosti originálního obrázku.
CNN má následující základní komponenty, [28]:

- Vstupní vrstva
- Konvoluční vrstva
- Aktivační funkce
- Pooling vrstva
- Plně spojené vrstvy

1.1.14 Vstupní vrstva

Vstupními daty do téhle vrstvy jsou obrázky. Obecně obrázky vstupují do trénování, jako čtyřrozměrné tenzory, kde první dimenze je specificky pro index obrázku, druhá a třetí dimenze jsou určené pro výšku a šířku obrázků a čtvrtá dimenze odpovídá různým kanálům obrazu. Pro barevné obrázky obecně máme tři kanály, RGB, zatímco pro černobílé obrázky máme pouze jeden kanál.

1.1.15 Konvoluční vrstva

Konvoluce je operace mezi dvěma funkcemi f_1 a f_2 téhož argumentu definovaných v diskrétním případě jako:

$$f(n) = f_1(n) * f_2(n) = \sum_{m=-0}^{n} f_1(m) * f_2((n-m)) \tag{1.72}$$

a ve spojitém případě:

$$f(t) = f_1(t) * f_2(t) = \int_{-\infty}^{\infty} f_1.(\tau) * f_2((1-\tau))\mathrm{d}\tau \tag{1.73}$$

pokud má f_2 konečný support, funkce $f_2(t)$ se nazývá konvoluční jádro. Takto definovaná konvoluce jak pro spojitý případ, tak pro diskrétní je definovaná v jednodimenzionálním prostoru. Tyto definice lze rozšířit i do dvoudimenzionálního prostoru, to nás zajímá především proto, že budeme aplikovat konvoluci na obrazová data. Pro diskrétní variantu:

$$f(m,n) = f_1(m,n) * f_2(m,n) = \sum_{l=0}^{N-1} \sum_{k=0}^{M-1} f_1(m-k, n-l) * f_2(k,l) \tag{1.74}$$

TensorFlow podporuje 2D i 3D konvoluce. Výstupem konvoluční vrstvy jsou příznakové mapy, které vznikly aplikací 2D konvoluce pomocí 2D filtrů o specifické velikosti na vstupní data do této vrstvy. Konvoluce probíhá podél dimenzí, zatímco konvoluce do hloubky kanálů neprobíhá. Pro každou hloubku kanálu se vytvoří příznakové mapy o stejné velikosti, ty se nakonec sečtou podél dimenze hloubky, než projdou dále do aktivační funkce ReLu. Tyto filtry pomáhají detekovat příznaky v obrázku. Hlubší konvoluční vrstvy v NS se učí komplikovanější příznaky. Např. inicializační konvoluční vrstva se učí detekovat hrany v obraze, zatímco další vrstva se učí detekovat spojení těchto hran do geometrických útvaru, jako je čtverec nebo kruh. V CNN se specifikují pouze velikosti filtrů. Váhy jsou inicializovány náhodnou hodnotou před začátkem trénování. Váhy filtrů se učí během trénování CNN, proto nemohou být reprezentovány tradičními filtry pro zpracování obrazu jakými jsou filtry Sobel, Gaussian, Mean, Median atd. Uveďme několik běžně používaných pojmů v souvislosti s konvolučními vrstvami:

- Velikost filtru - definuje výšku a šířku jádra filtru. Např. jádro filtru o velikosti 3×3 se bude skládat z 9 vah. Obecně se filtry nepřekrývají a to jak při jejich inicializaci, tak při jejich posouvaní přes celý vstupní obrázek. Technicky, když je konvoluce prováděna bez překrývání jader filtrů, jedná se o tzv. křížovou korelaci a nikoli o konvoluci. Nicméně, jelikož lze také provádět učení i s překrýváním filtrů, v obecnosti mluvíme o konvoluci.

- Krok (v angl. literatuře stride) - definuje počet pixelů podél každé osy o které posuneme okno v průběhu konvoluce. Běžně se konvoluce signálů vypočítává pro každý pixel v každé pozici okna. V takovém případě se jedná o krok jedna v každém směru podél os. Nicméně, během konvoluce můžeme přeskočit každý druhý pixel a tak zvolíme krok 2. Pokud zvolíme krok dva podél všech os, potom výsledný obrázek po konvoluci bude mít velikost přibližně $\frac{1}{4}$ vstupního obrázku.

- Vycpávka (v angl. padding) - po provedení konvoluce o specifické velikosti filtru, je výsledný obrázek menší než byl na vstupu. Padding je postup, kdy přidáme nulový rám kolem obrázku tak, abychom mohli lépe kontrolovat výstup z konvolučních vrstev. Výsledkem je obrázek rozměru $L^* \times L^*$ po konvoluci o délce L^* dostaneme jako:

$$L^* = \frac{L - K + 2P}{S} + 1. \tag{1.75}$$

  Pro jednoduchost obrázek $L \times L$ a jádro $K \times K$, P počet přidaných nul podél jedné dimenze a S je krok konvoluce. Obecně pro krok o velikosti 1 se dimenze redukuje o $\frac{K-1}{2}$ na každém konci, kde K je délka jádra. Pro zachování velikosti vstupního obrázku je vyžadována "vycpávka" délky $\frac{K-1}{2}$.

1.1.15.1 Shromažď'ovací vrstva

Tato vrstva se v anglické literatuře označuje jako pooling vrstva. Shromažď'ovací operace obecně provádí shrnutí lokalit v obraze. Lokality jsou dány velikostí filtru, proto se také nazývají pole vnímání. Sumarizace zpravidla nastává ve formě maxim (maxpooling) nebo průměrů (average pooling). V maximovém shrnutí se zvolí pixel s maximální intenzitou daného pole vnímání jako reprezentant této lokality. V průměrovém shrnutí, je jako reprezentant každé lokality zvolen průměr intenzity pixelů v dané lokalitě. Shrnutí tak redukují dimenzi obrazu. Velikost jádra, která definuje velikost lokalit, se obvykle volí 2×2, zatímco krok se volí jako 2. S takto zvolenými parametry dosáhneme redukce dimenze přibližně na $\frac{1}{4}$ původního obrazu.

1.1.15.2 Prořezávací vrstva a regularizace

V anglické literatuře je obvykle označována jako dropout vrstva. Prořezávání slouží k regularizaci vah v plně spojené CNN, abychom se vyhnuli přeučení sítě. Prořezávání NS znamená, že v průběhu tréninku je určitý poměr neuronů, jak skrytých tak i viditelných, náhodně vynechán. Tyto neurony tedy zůstanou neaktivní. Toto nastane pro každý trénovací vzorek v mini batchi. Zbylé neurony se tak mohou učit všechny důležité příznaky samostatně a nebude tak vznikat závislost a spolupráce mezi neurony. Když jsou určité neurony vynechány, tak jsou samozřejmě vynechány i všechny synapse a váhy s nimi spojené, ať už vstupní nebo výstupní. Když neurony mezi sebou příliš spolupracují, vzniká tak mezi nimi závislost a to vede k neúspěšnému učení různorodých příznaků. Vysoká míra kooperace vede k přeučení. Pokud jsou neurony vypínány náhodně, dosáhneme tak pokaždé jiného nastavení sítě. Předpokládejme, že máme NS s N neurony. Počet možných kombinací, jak nastavit prořezávání NS, je 2^N. Pro každý trénovací vzorek v mini-batchi jsou náhodně vybrány neurony, které se vypnou. Toto vypínaní neuronů má pravděpodobnostní rozdělení, dle kterého se řídí náhodný výběr. Tedy trénování NS s prořezáváním neuronů je vlastně trénování na množině různých konfigurací NS. Průměrováním predikce dosáhneme redukce odchylky v celkovém modelu a minimalizujeme tak riziko přeučení. Tím pádem dostaneme přesnější a stabilnější predikce.
Pro představu, uvažujme problém klasifikace do dvou tříd. Trénujme dva různé modely M_1 a M_2. Pravděpodobnost přiřazení do třídy 1 v modelu M_1 ozn. p_{11} a pravděpodobnost přiřazení do druhé třídy v témže modelu ozn. p_{12}. Obdobně pro model M_2 definujme pravděpodobnost přiřazení do první třídy jako p_{21} a do druhé jako p_{22}. Dostáváme průměrnou pravděpodobnost pro celkový model, který vznikl z průměrováním modelu M_1 a M_2, pro klasifikaci do první třídy jako $\frac{p_{11}+p_{21}}{2}$ a pro druhou třídu $\frac{p_{12}+p_{22}}{2}$.

Kromě aritmetického průměru pro pravděpodobnost přiřazení do celkového modelu můžeme použít i geometrický průměr. V tomto případě, ale budeme potřebovat provézt normalizaci, neboť součet středních geometrických hodnot pro nové pravděpodobnosti musí být roven 1. Pravděpodobnosti pro celkový model by pak vypadaly $\frac{\sqrt{p_{11} \times p_{21}}}{\sqrt{p_{11} \times p_{21}} + \sqrt{p_{12} \times p_{22}}}$ resp. $\frac{\sqrt{p_{12} \times p_{22}}}{\sqrt{p_{11} \times p_{21}} + \sqrt{p_{12} \times p_{22}}}$.
Při testování modelu není možné spočítat predikce přes všechny možné konfigurace a zjistit tak jejich průměr. Místo toho použijeme NS s upravenými váhami. Když neuron v NS během učení není vypnutý s pravděpodobností p, potom výsledná váha, která spojuje neuron s následující vrstvou, je zmenšená o násobek pravděpodobnosti p. Obecně tato aproximace vzhledem k celému datasetu funguje velice dobře.

1.2 Architektury pro CNN

V této části si popíšeme architekturu několika běžně používaných sítí. Tyto sítě lze požít mimo klasifikace i na segmentaci, lokalizaci i detekci objektů, samozřejmě s jejich malou modifikací. Jelikož se jedná o hojně používané modely, pro většinu z nich jsou již předtrénované sítě, které lze použít k rychlejšímu a efektivnějšímu učení vlastní NS. Tato metoda se nazývá v angl. literatuře jako transfer learning nebo fine-tuning modelů. Zpracování obrazu v CNN probíhá na principu, že první konvoluční vrstva se učí detekovat všeobecné příznaky, jako jsou hrany a oblouky. Jak síť narůstá, tak další konvoluční vrstvy se učí detekovat složitější a komplexnější příznaky, které jsou relevantní pro daný dataset.

1.2.1 LeNet

Tato architektura je považována za první úspěšnou CNN, vyvinutou pro klasifikaci ručně psaných číslic [28]. LeNet5 je její pozdější vylepšená verze. Tato CNN jako vstup přijímá obrázek o velikosti 32×32 pixelů. Konvoluční vrstvy pak vyprodukují 6 příznakových map o velikosti 28×28 pixelů. 6 příznakových map je převzorkováno tak, aby produkovaly šest výstupních obrázků o velikosti 14×14 pixelů. Toto převzorkování je v podstatě pooling operace. Druhá konvoluční vrstva produkuje 16 příznakových map o velikosti 10×10. Druhé převzorkování redukuje příznakovou mapu na velikost 5×5. Dále následují dvě plně spojené vrstvy o 120 neuronů v první a 84 neuronů v druhé vrstvě. Nakonec následuje výstupní vrstva. Klíčovými vlastnostmi LeNet5 jsou

- Pooling vrstva bere jako vzorky sousední bloky o velikosti 2×2 pixelů a sumarizuje je do hodnot intenzity o velikosti 4 pixely. Suma je škálována trénovacími váhami a biasem. Následně tyto hodnoty vstupují do sigmoidální aktivační funkce. Zde je nepatrný rozdíl oproti maximálnímu poolingu nebo průměrnému poolingu.

- Jádro filtru používaného ke konvoluci má velikost 5×5. Výstupními neurony z této konvoluční vrstvy mají jako aktivační funkce RBF (tj. radiální funkce). Tyto RBF se skládají ze SoftMax funkcí. Plně propojená konvoluční vrstva se skládá z 84 neuronů, má 84 propojení ke každé klasifikační třídě s 84 korespondujícími váhami.
  V RBF neuronech pozorujeme euklidovskou vzdálenost mezi vstupy a výstupy tříd. Tyto vzdálenosti jsou reprezentovány váhovými vektory. Čím větší je euclidovská vzdálenost (váhový vektor) mezi vstupními daty a výstupní třídou, tím menší je šance, že data budou klasifikovány do této třídy.
  Stejným způsobem můžeme převézt toto tvrzení na pravděpodobnost, tak že vyloučíme negativní vzdálenosti a normalizujeme je přes různé třídy. Euklidovská vzdálenost přes všechny třídy může fungovat jako ztrátová funkce pro každý vstup. Mějme $x = [x_1 \dots x_{84}]^T \in R^{84 \times 1}$ je výstupní vektor z plně propojené vrstvy. Pro každou třídu, mějme 84 vah příslušných ke spojnicím. Když váhový vektor reprezentující i-tou třídu je $w_i \in R^{84 \times 1}$, potom výstup z neuronu i-té třídy může být dán jako:

$$\| x - w_i \|^2 = \sum_{j=1}^{84} (x_j - w_{ij})^2 \tag{1.76}$$

- Reprezentativní váhy pro každou třídu jsou předem fixovány a neprobíhá jejich učení.

1.2.2 AlexNet

Tato síť se skládá z pěti konvolučních vrstev, maxpooling vrstvy, dropout vrstvy a tří plně spojených vrstev připojených k vstupní a výstupní vrstvě o tisíci neuronů, viz [28].
Vstupní data do sítě mají rozměry $224 \times 224 \times 3$. První konvoluční vrstva produkuje 96 příznakových map, které korespondují s 96 filtry o velikosti $11 \times 11 \times 3$ s velikostí kroku 4 pixely. Druhá konvoluční vrstva produkuje 256 příznakových map, které odpovídají konvolučnímu filtru o velikosti jádra $5 \times 5 \times 48$. První dvě konvoluční vrstvy jsou následovány maxpooling vrstvami, zatímco další tři konvoluční vrstvy jsou uspořádány za sebou bez vložení maxpooling vrstev. Pátá konvoluční vrstva je následována maxpooling vrstvou, dále dvě plně propojené konvoluční vrstvy o 4096 neuronech a nakonec SoftMax výstupní vrstva a klasifikací do tisíce tříd, neboli s tisíci výstupními neurony. Třetí konvoluční vrstva má 384 filtrů o velikosti $3 \times 3 \times 256$, zatímco čtvrtá a pátá konvoluční vrstva má 384 a 256 filtrů o velikosti $3 \times 3 \times 192$. Síť se v posledních dvou plně spojených vrstvách bude prořezávat s koeficientem 0.5, tzn. pří trénování se vypne 50% neuronů. Poznamenejme, že rozměr filtrů pro konvoluci je poloviční oproti počtu příznakových map v předchozí vrstvě, což platí pro všechny konvoluční vrstvy kromě třetí. Mezi klíčové vlastnosti AlexNet patří:

- Používá se ReLu aktivační funkce, kvůli její nelinearitě, snadné spočitatelnosti a konstantnímu nesaturovanému gradientu v opozici k sigmoidální a hyperbolické tangent aktivační funkci, kde se gradient blíží k nule pro velmi vysoké nebo naopak velmi nízké vstupní hodnoty.

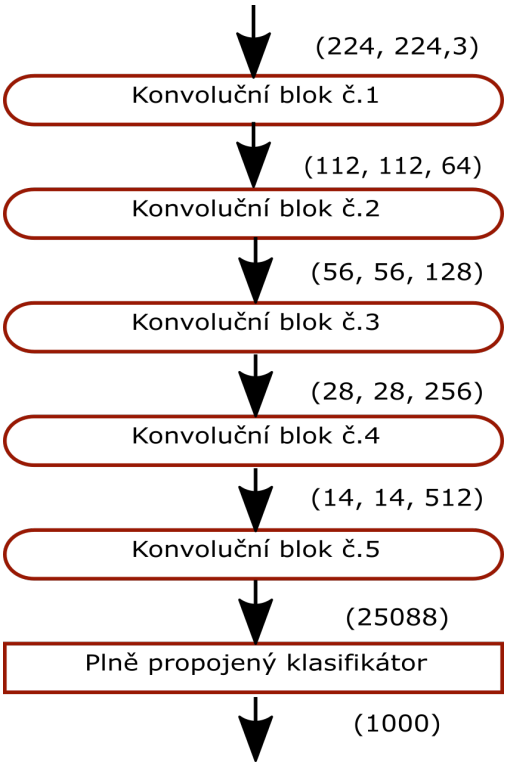
- Je využito prořezávání k redukci přeučení modelu.

1.2.3 VGG16

Tato architektura, viz obr. 1.2.3 nepoužívá velké konvoluční filtry, místo toho používá filtry o velikosti 3×3, viz [28]. Dále využívá ReLu aktivační funkci a maxpooling s polem vnímání o velikosti 2×2. Tato síť je založena na úvaze, že použití dvou konvolučních vrstev o velikosti 3×3 je ekvivalentní k použití jedné konvoluční vrstvy o velikosti 5×5, přičemž využíváme všechny výhody filtrů s malými konvolučními jádry. Výhodou je např. redukce počtu parametrů a realizace větší nelinearity díky Relu konvoluci v páru oproti pouze jedné konvoluci. Speciální vlastností této sítě je, že prostorové dimenze míry vstupních dat je redukována konvolucí a maxpoolingem. Počet příznakových map roste úměrně se zvyšujícím se počtem filtrů, vzhledem k tomu, jak jdeme do hloubky NS. Vstupními daty do sítě jsou obrázky o velikosti $224 \times 224 \times 3$. První dvě vrstvy produkují 64 příznakových map. Každá z vrstev je následována maxpooling vrstvou. Konvoluční filtry mají rozměry 3×3 s krokem o velikosti 1 a vycpávkou o velikosti 1. Maxpooling je velikosti 3×2 s krokem 2 v celé síti. Třetí a čtvrtá konvoluční vrstva produkuje 128 příznakových map, opět každá vrstva je následována maxpoolingem. Zbytek sítě je sestaven v podobném smyslu. Síť je zakončena třemi plně spojenými vrstvami o 4096 neuronech. Každá tato vrstva je následována výstupní SoftMax vrstvou o tisíci třídách. Pro plně propojené vrstvy je prořezávání nastaveno na 50%. Všechny neurony v síti mají ReLu aktivační funkci.

1.2.4 ResNet

ResNet je konvoluční síť o hloubce 152 vrstev, viz [28]. Tato síť implementuje unikátní myšlenku residuálních bloků. V tradičních metodách během provádění konvoluce nebo ostatních transformací zkoušíme fitovat základní mapování na originální data, abychom vyřešili klasifikační úlohu. Avšak koncept ResNet residuálních bloků se snaží naučit síť k fitování residuálního mapování namísto přímého mapování ze vstupních dat na výstupní. V každém malém bloku operací vložíme vstupní data do bloku na výstup. Tento koncept je založen na hypotéze, že je lehčí trénovat mapování reziduí než trénovat mapování na originální vstup.



Obrázek 1.6: Ukázka populární architektury VGG16. Vidíme zde sestavení vrstev a tvar výstupních dat z nich (hodnoty v závorce - velikost příznakové mapy a jejich počet).

1.2.5 Přenesené učení

Tento způsob učení zahrnuje v širším slova smyslu ukládání znalostí získaných při řešení jiných problémů a jejich uplatnění při řešení jiných úloh v podobné doméně. Tento způsob učení je úspěšný v nejrůznějších oblastech hlubokého učení. Modely hlubokého učení mají obecně velký počet parametrů kvůli povaze skrytých vrstev a schématu spojení jednotlivých neuronů. Trénovat takto obrovský model vyžaduje velké množství dat, jinak bude model trpět problémem přeučení. Dostatečné množství zpravidla není k dispozici, ale z povahy problémů je vhodné právě hluboké učení, abychom obdrželi přiměřeně dobré výsledky. V případech detekce objektů ve zpracovaní obrazu, lze přenesené učení použít pro generování generických příznaků z předtrénovaných hlubokých modelů a následně tyto příznaky využít k budování jednoduchých modelů. Předtrénované modely jsou na rozsáhlých zdrojových datech, a proto tyto modely dosahují vysoké spolehlivosti.

Kapitola 2

Implementace rozpoznávání budov

V této práci jsme se pokusili implementovat dva modely NS. V obou případech se jedná o algoritmy založené na CNN a jsou typu VGG16 při různém nastavení parametrů.
Pro rozpoznávání objektů lze použít mnoho různých algoritmů, my jsme si vybrali metodu, kdy jednotlivé obrázky rozdělíme do několika bloků a tyto bloky následně předkládáme NS ke klasifikaci. Jedná se primárně o binární klasifikační úlohu.
Detailním popisem vstupních dat se budeme zabývat v sekci 3.1, ale pro porozumění dalšímu textu zmiňme několik základních faktů. Data jsou rozdělena do tří základních množin: trénovací, validační a testovací. Jelikož bude NS trénovat metodou učení s učitelem, ke všem obrázkům máme mapové podklady s vyznačenými půdorysy budov. Tyto mapové podklady budeme nazývat vzory a budou fungovat jako kontrolní materiály. Satelitní snímky, na kterých se NS bude snažit detekovat budovy, budeme nazývat vstupní obrazy nebo data.
Popišme si postup přípravy a klasifikaci dat pro obě NS, viz obr. 2.1

1. Načtení dat:
   Načítáme vždy tři sady dat, trénovací, validační a testovací. Obrázky jsou uloženy do pole. K těmto obrázkům jsou načteny i jejich vzory, tzn. mapové podklady s vyznačenými půdorysy budov, opět jsou uloženy do pole.

2. Vytvoření bloků:
   Každý vstupní obrázek i vzor je rozdělen do menších bloků o velikosti $k \times k$. Pokud obrázek nelze rozdělit beze zbytku do bloku, budou okrajové části doplněny černou barvou, tak aby délka resp. šířka obrázku byla dělitelná beze zbytku velikostí bloku. Samozřejmě se vždy zachovává asociace mezi vstupními obrázky a jejich bloky se vzory a jejich bloky.

3. Vytvoření vzorů - interpretace bloků:
   Pole $k \times k \times d$, kde k je velikost bloku a d je počet načtených vzorů, které obsahuje bloky vzorů interpretujeme do pole 1 a 0 o délce d, tzn. každý blok bude kódován jedním číslem podle toho, jestli bude označen jako budova (1) nebo jako okolí (0). Podívejme se blíže, jakým způsobem bude rozhodnuto o tom, zda daný blok je budova nebo okolí. Nyní mluvíme o blocích, které vznikly ze vzorů, tzn. víme, kde se nachází budova. Pokud plocha bloku bude obsahovat ze stejné nebo větší části budovu než je práh (numerická hodnota daná uživatelem) bude blok označen 1 (tzn. budova), jinak 0 (okolí). Tento vektor budeme dále v tomto textu nazývat jako vzor nebo interpretace.

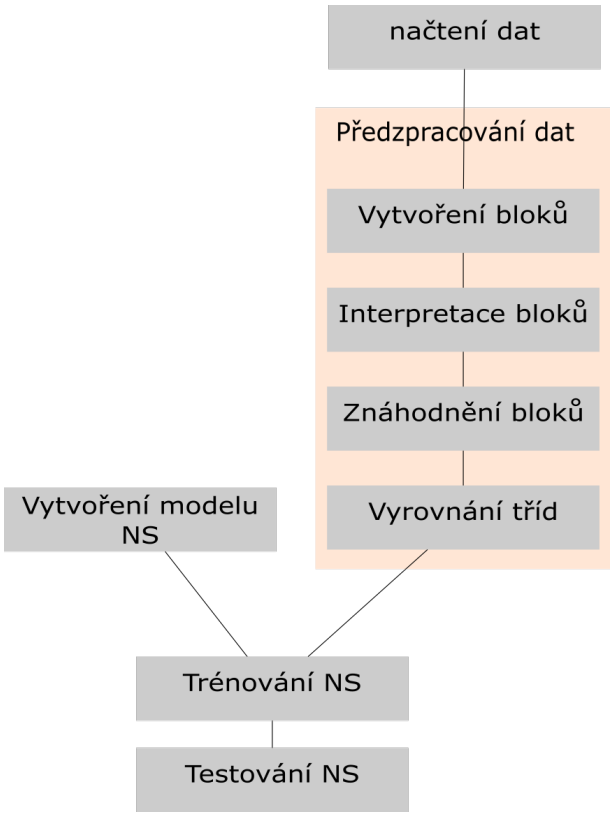
4. Znáhodnění datasetu a jeho vyváženost. Bloky je potřeba v jednotlivých množinách promíchat, abychom zabránili vytváření náhodných vzorů, které by se síť mohla naučit a následně by špatně predikovala na testovacích datech. Dalším důležitým krokem je vyrovnání počtu tříd. V tomto případě je to velice důležité, jelikož z povahy dat, je třída 0 obsažena nepoměrně více než třída 1.

5. Sestavení NS a nastavení jejich parametrů:
   Bližší popis nastavení NS a jejích parametrů bude v sekci 2.2.

6. Trénování NS

7. Testování NS na datech z testovací množiny:
   Tato data NS během trénování nikdy neviděla. Poslední fází je sestavení čtyř experimentů na nichž budeme testovat naše natrénované sítě a budeme je porovnávat vzhledem k volbě prahu velikosti bloku a měřítka satelitních snímků viz kapitola 3. Na závěr porovnáme modely mezi sebou při nejlepším nastavení parametrů pro každý z nich.

Detailně si tyto kroky a jejich propojení mezi nimi popišme v následujícím textu viz sekce 2.2.



Obrázek 2.1: Ukázka implementace NS a předzpracování dat.

2.1 Závislosti a knihovny

Nejprve popíšeme, jaké knihovny a závislosti naše NS využívají (import knihoven):

- `Tensoflow` - Jedná se o soubor knihoven, jejichž pomocí můžeme sestavovat, trénovat a jinak manipulovat NS. Mezi jazyky, které jsou pro tento balíček podporovány patří např. C++ a Python.
- `keras` - Knihovna pro sestavení architektury neuronové sítě.
- `cv2` - Tato knihovna je také známá jako OpenCV a slouží ke zpracování obrázků a videa v Pythonu, ale i dalších jazycích. Používá se k různým způsobům analýzy audiovizuálních dat, např. rozpoznávání a detekce obličejů, čtení licencí, editování fotografií, podporu strojového vidění a mnoho dalšího. My tuto knihovnu využíváme pouze k načtení a změně velikosti obrázků.
- `numpy` - Populární matematická knihovna, snadno a rychle zpracovává velké multi-dimenzionální vektory a matice.
- `pandas` - Softwarová knihovna napsaná pro Python pro manipulaci s daty a jejich analýzu.

- `matplotlib.pyplot` - Knihovna pro Python, které umožňuje snadné vykreslení grafů a obrázků.
- `os` - Vestavěný balíček v Pythonu pro přístup k počítači a systémovým souborům.
- `random` - Balíček, který pomáhá generovat náhodné čísla.
- `gc` - Zkratka pro garbage collector. Důležitý nástroj pro manuální čištění a mazání nepotřebných proměnných.

2.2 Použité architektury NS

Sítě, které jsme vytvořili budeme nazývat model 1 a model 2. Jsou tvořeny sekvenční sítí. Tento typ sítě je vhodný pro jednoduché propojení vrstev, kde každá vrstva má přesně jeden vstupní tenzor a jeden výstupní tenzor. Naopak tato síť není vhodná, pokud by měla více vstupů, více výstupů nebo bychom chtěli vytvořit nelineární propojení vrstev.
Architekturu našich modelů 1 a 2 je typu VGGnet, viz obr. 1.6. Ukázku a popis architektury, kterou jsme v těchto případech použili jsme získali z [25]. Používáme pouze malé sítě a můžeme se povšimnout, že velikost filtru se zvětšuje, když jdeme do hlubších vrstev NS:
$32 \rightarrow 64 \rightarrow 128 \rightarrow 512 \rightarrow 1$
Nyní si popišme postup sestavení modelu 1 a 2 a jejich vrstvy(typ těchto vrstev je pro oba modely stejný, liší se pouze v jejich počtu a uspořádání, viz tabulky 2.1 a 2.2):

1. Nejprve vytvoříme sekvenční model.
2. První vrstva je vytvořena funkcí `.add()`. Typ této vrstvy je 2D konvoluční, funkce `Conv2D`. Tato první vrstva se nazývá vstupní (input) a má několik důležitých parametrů, které je nutno nastavit:
   - Velikost filtru (Filter size) - Zvolili jsme 32. Toto je hodnota výstupní dimenze, např. počet výstupních filtrů z konvoluce.
   - Velikost jádra (Kernel size) - Zvolili jsme $[3, 3]$. Tato funkce specifikuje výšku a šířku konvolučního 2D okna.
   - Aktivační funkce (activation) - Aktivační funkci jsme nastavili `"relu"`. ReLu (Rectified Linear Unit) je běžná aktivační funkce, viz 1.1.2. Dalšími variantami jsou např. leaky ReLU a eLU.
   - Vstupní velikost - Vstupní velikost je rovna velikosti bloku,
     tzn. [velikost bloku, velikost bloku, 3]. Z toho lze vidět, že bloky do sítě vstupují, jako barevný (RGB) obrázek.
3. Druhou vrstvou je `MaxPool2D` - Tato funkce redukuje prostorovou velikost budoucích příznaků, navíc pomáhá redukovat počet parametrů a výpočtů v síti, také pomáhá snížit riziko přeučení.
4. Další vrstvou je vyrovnávací (Flatten) - Konvoluční 2D vrstvy vytěžují a učí se prostorové příznaky, které potom vstupují dál do plně propojené vrstvy a proto musí být tato data zploštěny (vyhlazeny resp. redukce dimenze). Toto právě zajišťuje vyhlazovací vrstva.
5. Následuje vrstva vypínající náhodně ostatní vrstvy (Dropout) - Nastavili jsme hodnotu na 0.5. V síti jsou náhodně vypuštěny některé vrstvy a pak probíhá učení na redukované síti. Tímto způsobem se síť stává nezávislá a nespoléhá na jedinou vrstvu. Toto opět pomáhá předcházet přeučení. Volba parametru 0.5 znamená, že se náhodně vypne polovina vrstev.
6. Poslední vrstva se nazývá výstupní - Tato vrstva má výstupní dimenzi 1 a jinou aktivační funkci, v našem případě sigmoidální. Výstupem je pravděpodobnost s jakou se na bloku nachází budova. Sigmoidální funkce vrací číslo mezi 0 a 1, proto je ideální pro náš případ, viz sekce 1.1.2.

Nyní model zkompilujeme, k tomu je potřeba nastavit tři parametry.

- Ztrátovou funkci (Loss) - Zvolíme funkci binary-crossentropy, viz sekce 1.1.6, tuto funkci bude optimalizér minimalizovat. Ztrátovou funkci jsme vybrali podle toho, že máme klasifikační problém o dvou třídách.
- Optimalizační funkce - Zvolili jsme funkci `rmsprop`.
- Metrika - Tato metrika bude měřit naši výkonnost během učení sítě. Jelikož máme klasifikační problém o dvou třídách vybereme funkci `acc`.

Historie učení se ukládá do proměnné, proto si můžeme následně zobrazit grafy pro ztrátovou funkci a přesnost.
Dalším krokem je načtení testovacího datasetu a naučenou síť použijeme k predikci pro tento dataset. Přesnost modelu pak můžeme prověřit spočtením confusion matrix a porovnáním počtů falešně pozitivních resp. negativních, chybně a správně detekovaných budov.
Jak jsme zmínili výše, rozdíl mezi modelem 1 a 2 je v počtu a uspořádání jednotlivých vrstev:

- model 1 - Počet konvolučních bloků je 4 a jejich uspořádání lze vidět v tab. 2.1.
- model 2 - Počet konvolučních bloků je 3 a jejich uspořádání lze vidět v tab. 2.2.

Můžeme tedy říci, že porovnáním modelu 1 a 2 zkoumáme vliv počtu konvolučních bloků na přesnost detekce v našich datech.

Následující tabulka přehledně ukazuje sestavení modelu 1 a počty trénovaných parametrů:

Vrstva	Velikost výstupu	Parametry
Konvoluce2D	(48, 48, 32)	896
MaxPooling2D	(24, 24, 32)	0
Konvoluce2D	(22, 22, 64)	18496
MaxPooling2D	(11, 11, 64)	0
Konvoluce2D	(9, 9, 128)	73856
MaxPooling2D	(4, 4, 128)	0
Konvoluce2D	(2, 2, 128)	147584
MaxPooling2D	(1, 1, 128)	0
Flatten	(128)	0
Dropout	(128)	0
Dense	(512)	66048
Dense	(1)	513
Celkem parametrů:		307393

Tabulka 2.1: Architektura NS pro model 1. Velikost výstupu udává velikost výstupu příznakové mapy (první dvě čísla) a počet příznakových map.

Následující tabulka přehledně ukazuje sestavení modelu 2 a počty trénovaných parametrů:

Vrstva	velikost výstupu	Parametry
Konvoluce2D	(10, 10, 32)	896
MaxPooling2D	(5, 5, 32)	0
Dropout	(5, 5, 32)	0
Konvoluce2D	(5, 5, 64)	18496
MaxPooling2D	(2, 2, 64)	0
Dropout	(2, 2, 64)	0
Konvoluce2D	(2, 2, 64)	36928
MaxPooling2D	(1, 1, 64)	0
Dropout	(1, 1, 64)	0
Flatten	(64)	0
Dense	(512)	33280
Dropout	(512)	0
Dense	(1)	513
Celkem parametrů:		90113

Tabulka 2.2: Architektura NS pro model 2. Velikost výstupu udává velikost výstupu příznakové mapy (první dvě čísla) a počet příznakových map.

2.3 Načtení a předzpracování obrázků

Po načtení trénovacího a validačního datasatu obrázky rozdělíme do bloků. Velikost bloků závisí na nastavení vstupních parametrů (testujeme především velikosti hrany bloku o 2px, 5px, 10px, 15px, 20px, 25px, 35px, 50px). Tyto bloky jsou náhodně promíchány a jejich počet redukován tak, aby splňovaly následující podmínky:

- Poměr bloků, které jsou označeny jako budova a pozadí je 1:1. Toto platí pro trénovací i validační dataset.

- Poměr velikostí trénovacího a validačního datasetu je 80:20.
- Množství dat v trénovací množině nesmí přesáhnout množství dat, které jsme schopni předzpracovat (cca 300 000 bloků).

Poté jsou v případě potřeby (v závislosti na modelu a typu NS) bloky přeškálovány na požadovanou velikost.
Interpretaci našich bloků jsme získali tak, že vzory příslušející k originálním obrázků jsme také rozdělili na bloky. Poté jsme pro každý blok spočetli plochu, která je pokryta budovou. Pokud tato plocha je větší než polovina bloku, blok originálu příslušejí k bloku vzoru je označen jako budova tzn. 1, jinak jako pozadí tj. 0. V tuto chvíli jsou data připravena pro trénování.
Dalšími důležitými proměnnými pro trénování je velikost mini-dávky, neboli velikost trénovacího datasetu v jedné epoše.

Kapitola 3

Srovnání modelů a jejich výsledky

3.1 Datasety

V práci jsou využity dvě různé sady dat. Jednou sadou dat jsou satelitní snímky pokrývají velkou část Toronta. Tento dataset jsme získali z [24]. Druhou sadou dat jsou satelitní snímky pro Prahu, které jsme získali z [26]. Snímky zobrazují především hustě zastavěné oblasti a centrum města. Tyto datasety jsme využili pro všechny naše experimenty popsané v sekci 3.2.
Datasety se skládají ze satelitních snímků a jim příslušejících mapových podkladů s vyznačenými půdorysy budov. Data pro obě města jsou také rozdělena do trénovací, validační a testovací množiny, viz 3.1. Na trénovací a validační množině probíhá učení NS a testovací množina slouží ke změření výkonosti NS na datech, které nikdy neviděla, abychom docílili objektivity měření.
Mapové podklady nazývejme vzory, a satelitní snímky nazýváme originály. Vzory obsahují chyby, jelikož některé budovy nejsou vyznačeny. K tomuto dojde např. pokud mapové podklady vznikly dříve a nebyly aktualizovány.
Datasety jsou rozděleny do množin následovně:

Data	Trénovací mn.	Validační mn.	Testovací mn.
Praha	12	5	10
Toronto	102	4	10

Tabulka 3.1: Rozdělení datasetů na trénovací, validační a testovací množinu.

Satelitní snímky Prahy mají měřítko 1px = 1dm a velikost foceného území byla 500×500m pro každý snímek. Snímky Toronta mají měřítko 1px = 1, 2m a jeden snímek zobrazuje území o rozloze 1200×1200m.

3.2 Porovnání modelů

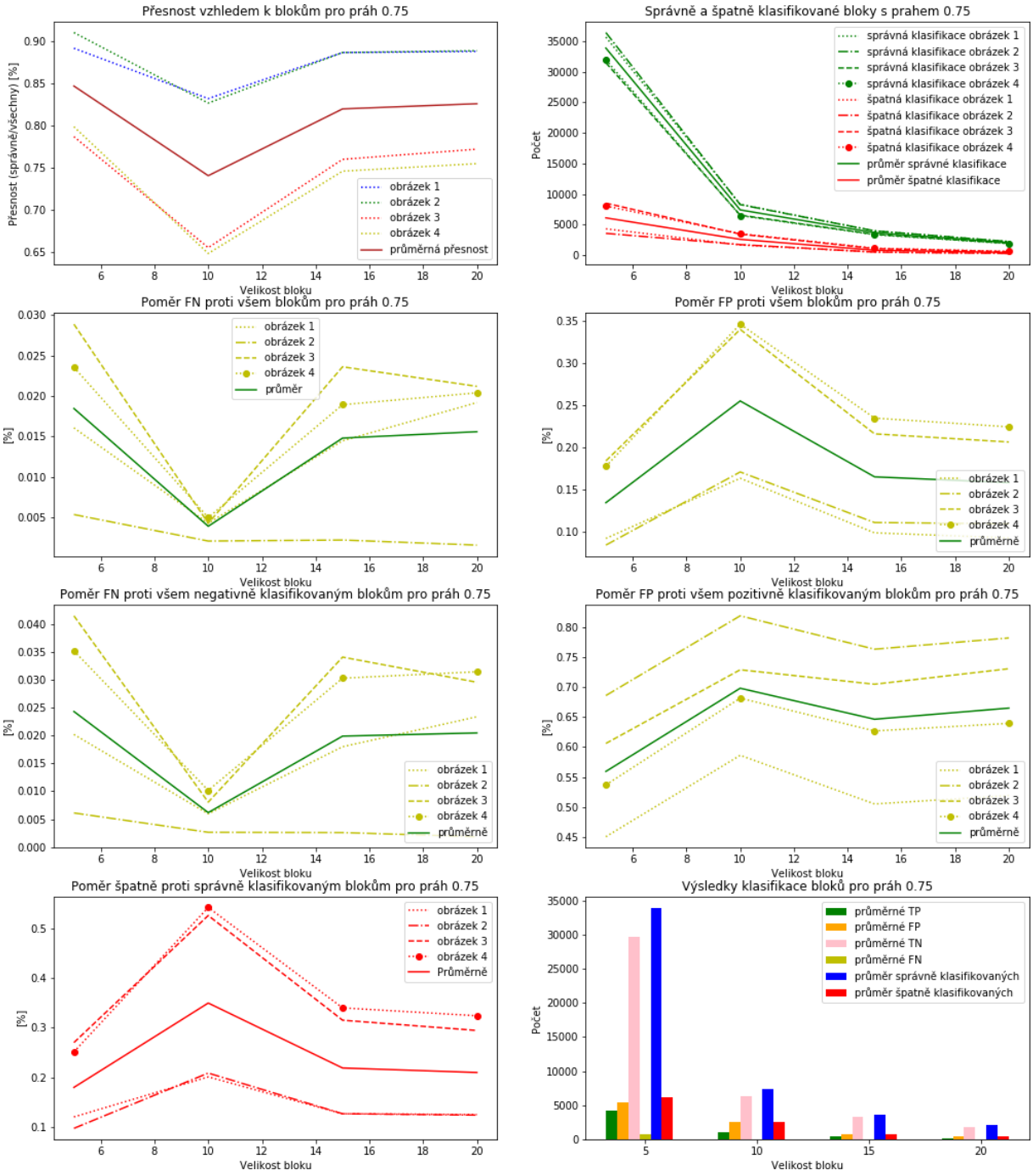
Detailní popis modelů jsme uvedli v sekci 1.2.3, nyní se zabývejme jejich porovnáním. Jedno z hledisek, na které jsme se zaměřili při experimentování, bylo kromě typu použité sítě i měřítko satelitních snímků. Testovali jsme snímky z oblasti Toronta a Prahy s různými měřítky. Sítě jsme také porovnávali vzhledem k nastavení prahu. Práh ovlivňuje rozhodnutí, zda daný blok bude označen jako budova nebo pozadí. Hodnoty prahu byly 0.75 a 0.5.
Kvalitu modelu jsme měřili pomocí parametrů:

- Přesnosti - poměr správně detekovaných bloků, vůči všem testovaným.
- Falešná pozitivita (FP) - počet bloků, které byly označeny jako budova, ale ve skutečnosti se nejednalo o budovu.

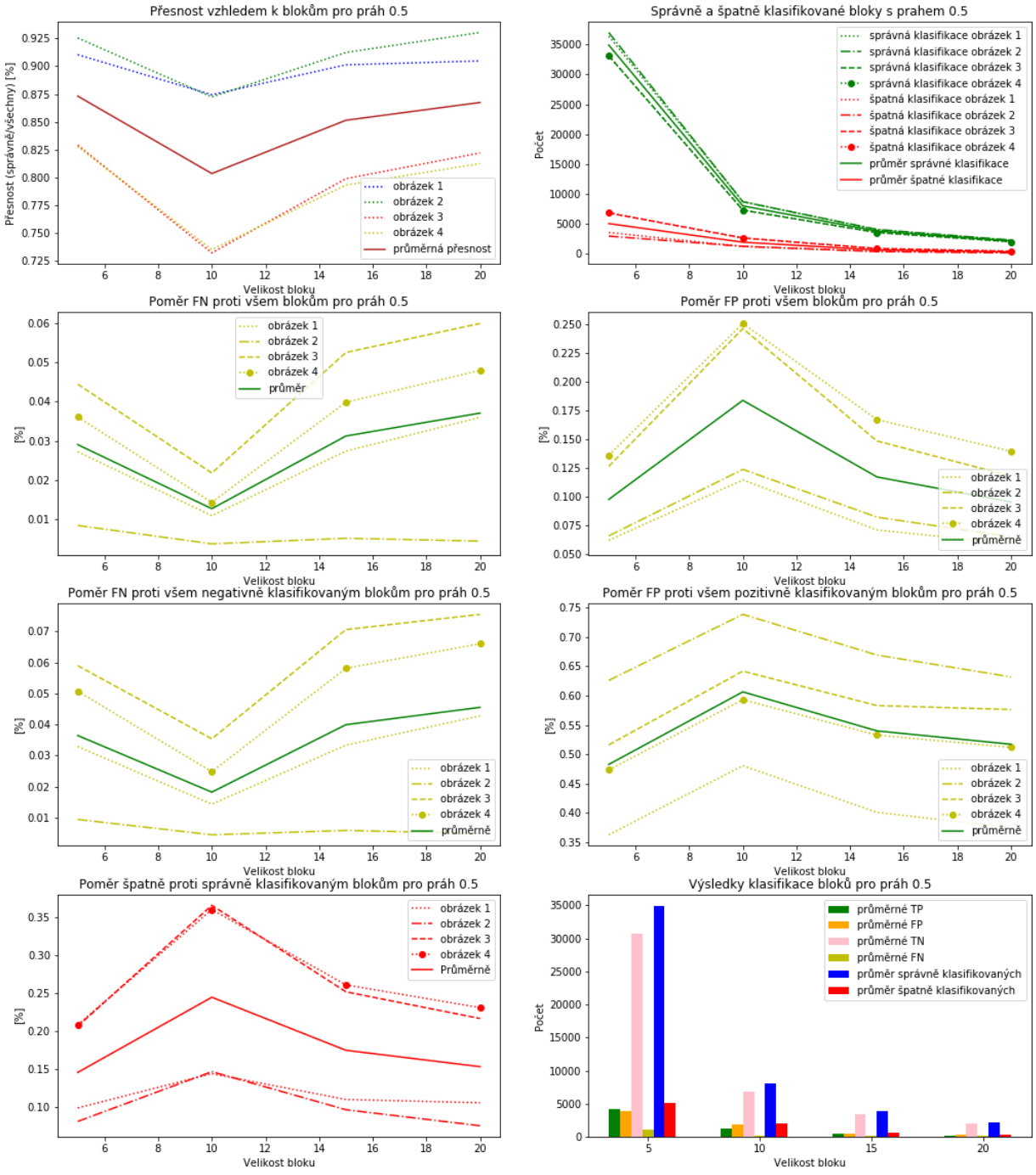
- Falešná negativita (FN) - počet bloků, které nebyly označeny jako budova, ale ve skutečnosti se jednalo o budovu.
- Pravdivě negativní (TN) - bloky, které byly označeny jako pozadí a ve skutečnosti se jednalo o pozadí.
- Pravdivě pozitivní (TP) - bloky, které byly označeny jako budova a ve skutečnosti se jednalo o budovu.
- Správně detekované (right) - součet TP a TN.
- Špatně detekované bloky (wrong) - součet všech FP a FN.

3.2.0.1 Experiment 1

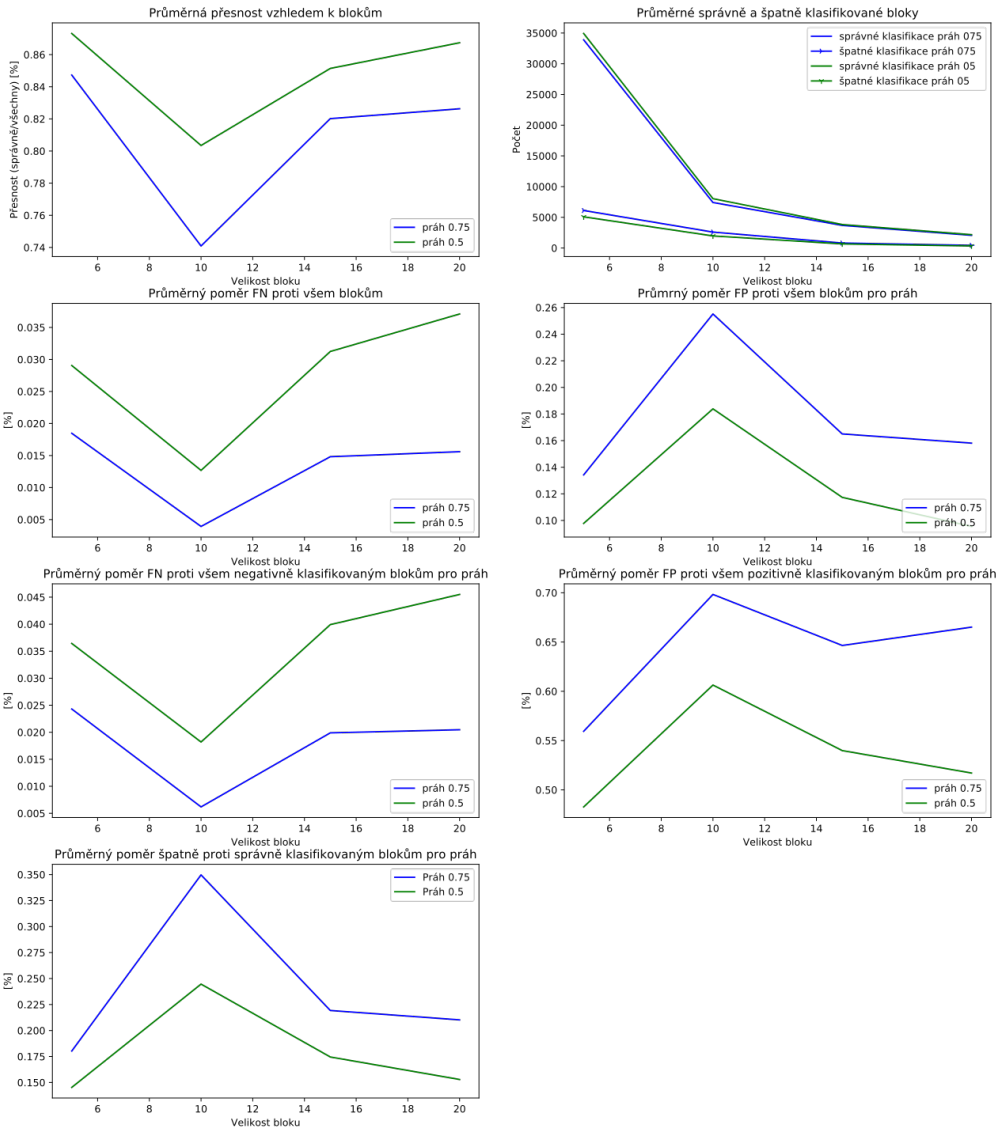
V tomto experimentu je použit model 1. V tomto případě jsme použili snímky Prahy a porovnání NS vzhledem k velikosti prahu můžeme vidět na obr.3.1-.3.3.
Pro práh 0.75 se jeví jako nejlepší model s velikostí bloku 5, viz obr. 3.1. K tomuto názoru nás vede několik důvodů. Průměrná přesnost je 85%, dosahuje nejlepšího poměru mezi špatně a správně klasifikovanými bloky. Stejný závěr učiníme, i pokud jsme zvolili práh 0.5, jehož detailní statistiky vidíme na obr. 3.2. Dále dosahuje lepších hodnot z pohledu FN.
Pokud porovnáme modely vzhledem ke zvolenému prahu, zjistíme, že lepší přesnosti dosáhneme modelem s prahem 0.5, a to 87.5% a i v ostatních grafech má tento práh lepší parametry. Tento závěr je celkem překvapivý, jelikož při vizuálním porovnaní predikovaných map pro dané prahy můžeme udělat opačný závěr, viz obr. 3.4-3.5.



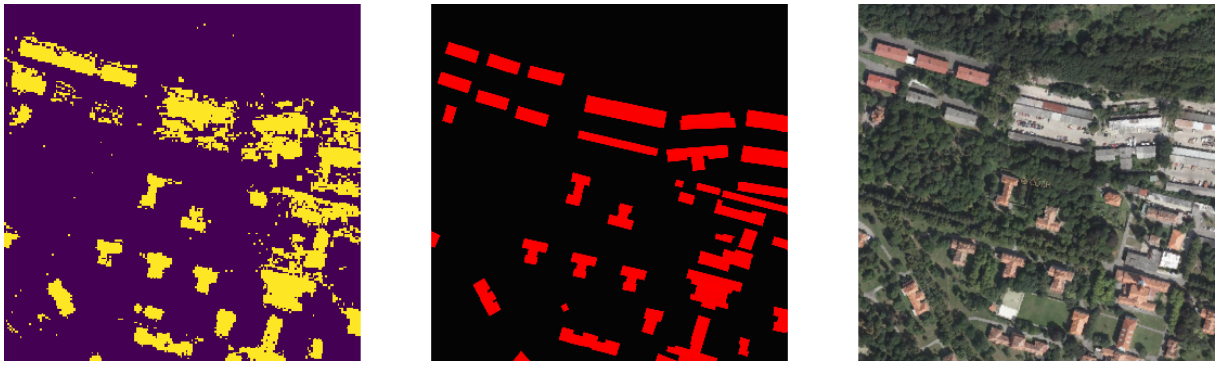
Obrázek 3.1: Vyhodnocení kvality klasifikace pro model 1. Data použitá pro trénování jsou z ČR práh byl 0,75.



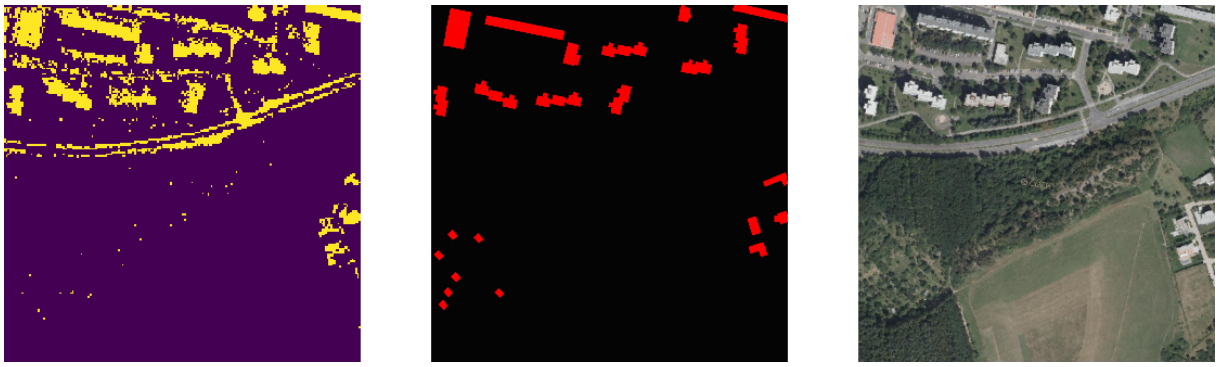
Obrázek 3.2: Vyhodnocení kvality klasifikace pro model 1. Data použitá pro trénování jsou z ČR práh byl 0,5.



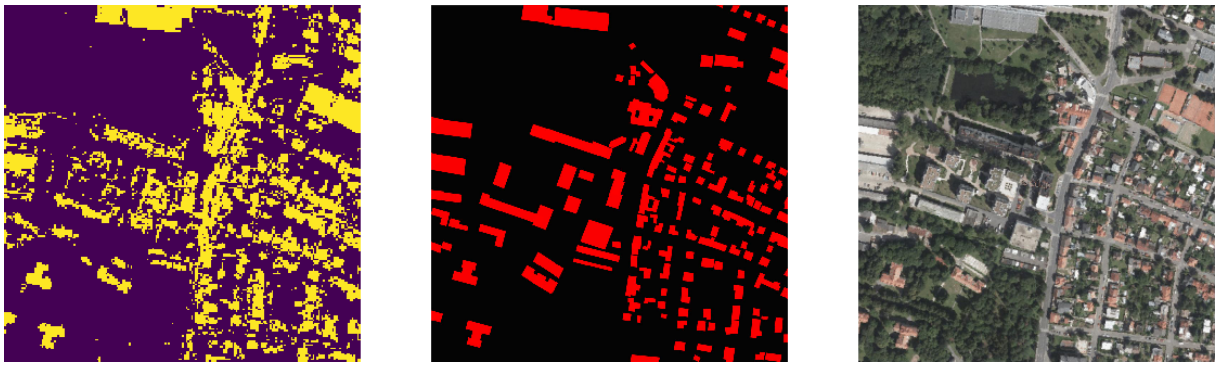
Obrázek 3.3: Vyhodnocení kvality klasifikace pro model 1. Data použitá pro trénování jsou z ČR, vzájemné porovnání průměrných hodnot pro hodnoty prahu $0,5$ a $0,75$.



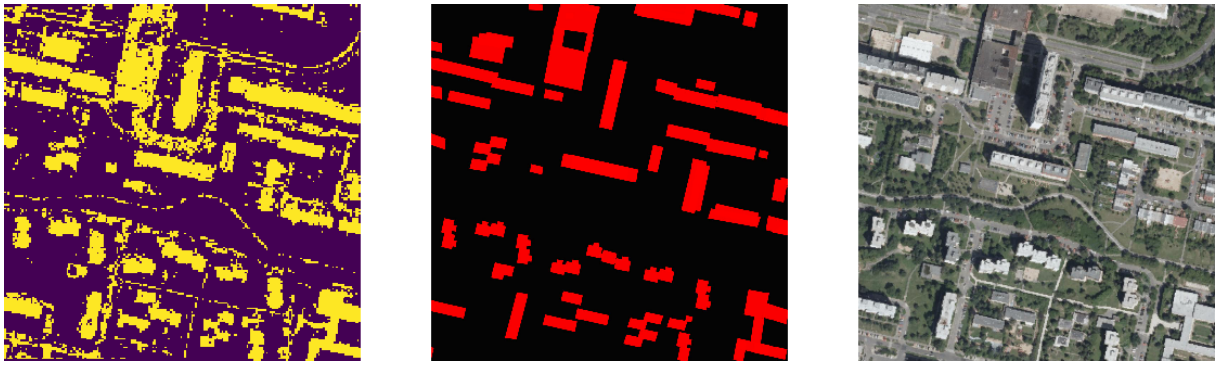
(a)



(b)

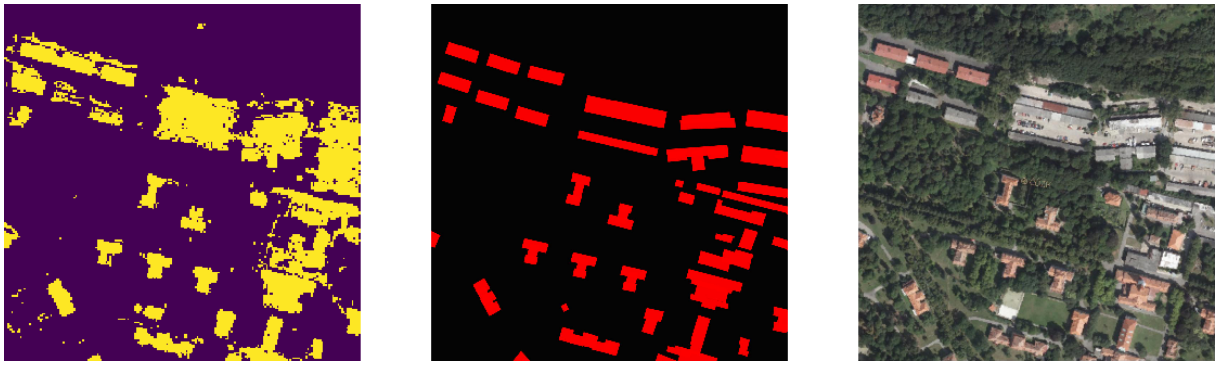


(c)

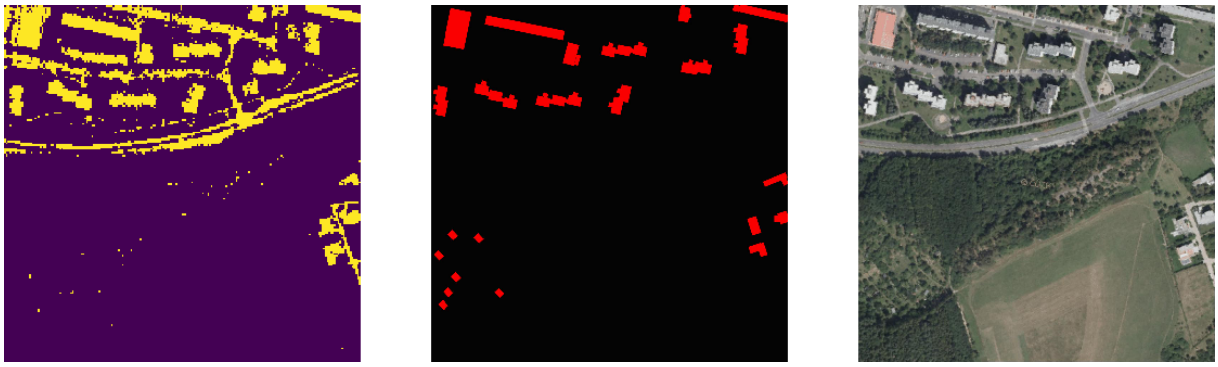


(d)

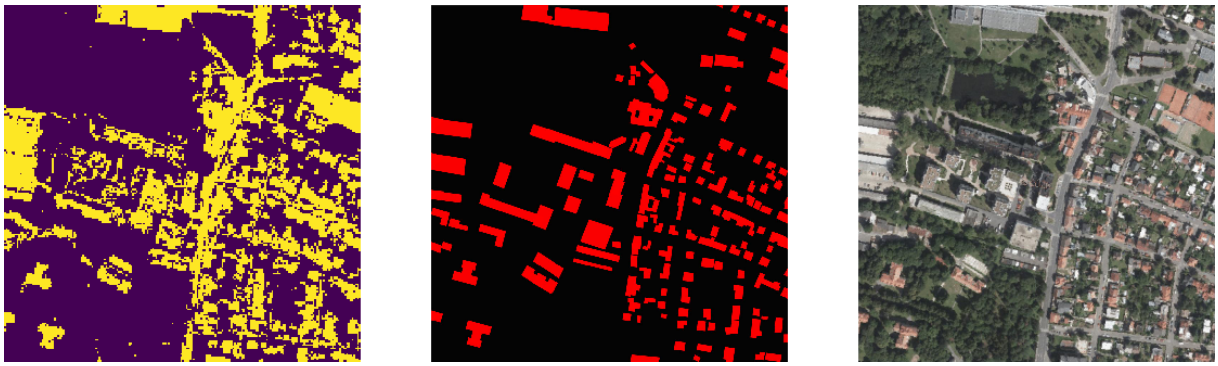
Obrázek 3.4: Zobrazení detekce budov pro nejlepší nastavení modelu 1 s použitým prahem 0.5. Vlevo výsledek detekce NS, uprostřed vzor, vpravo originál.



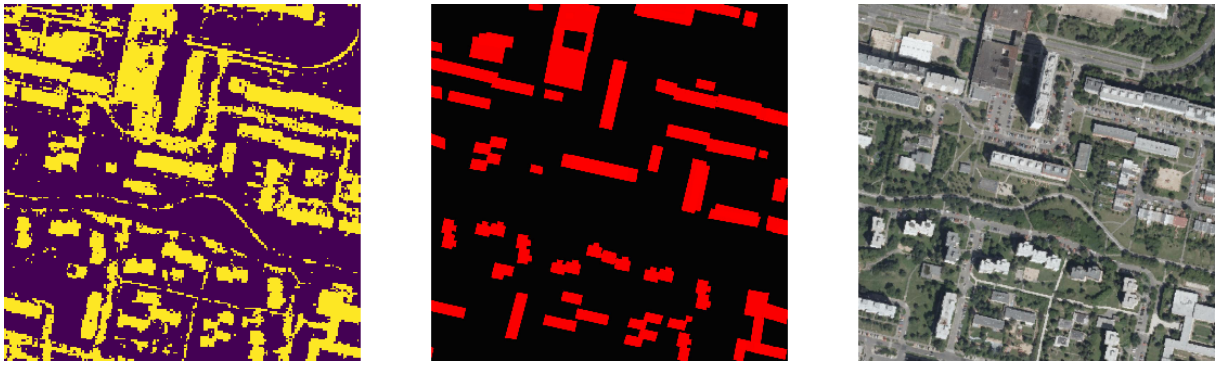
(a)



(b)



(c)

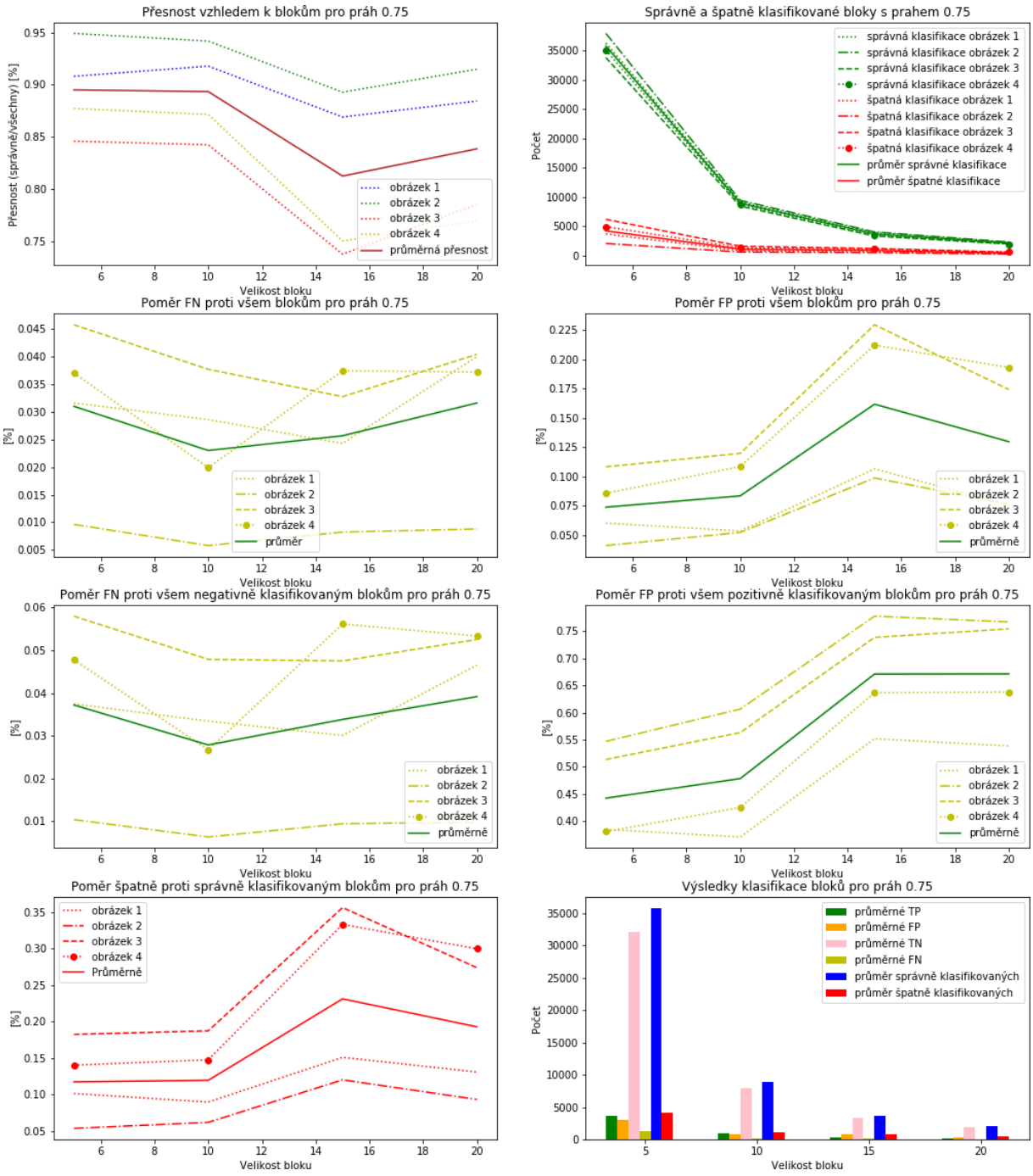


(d)

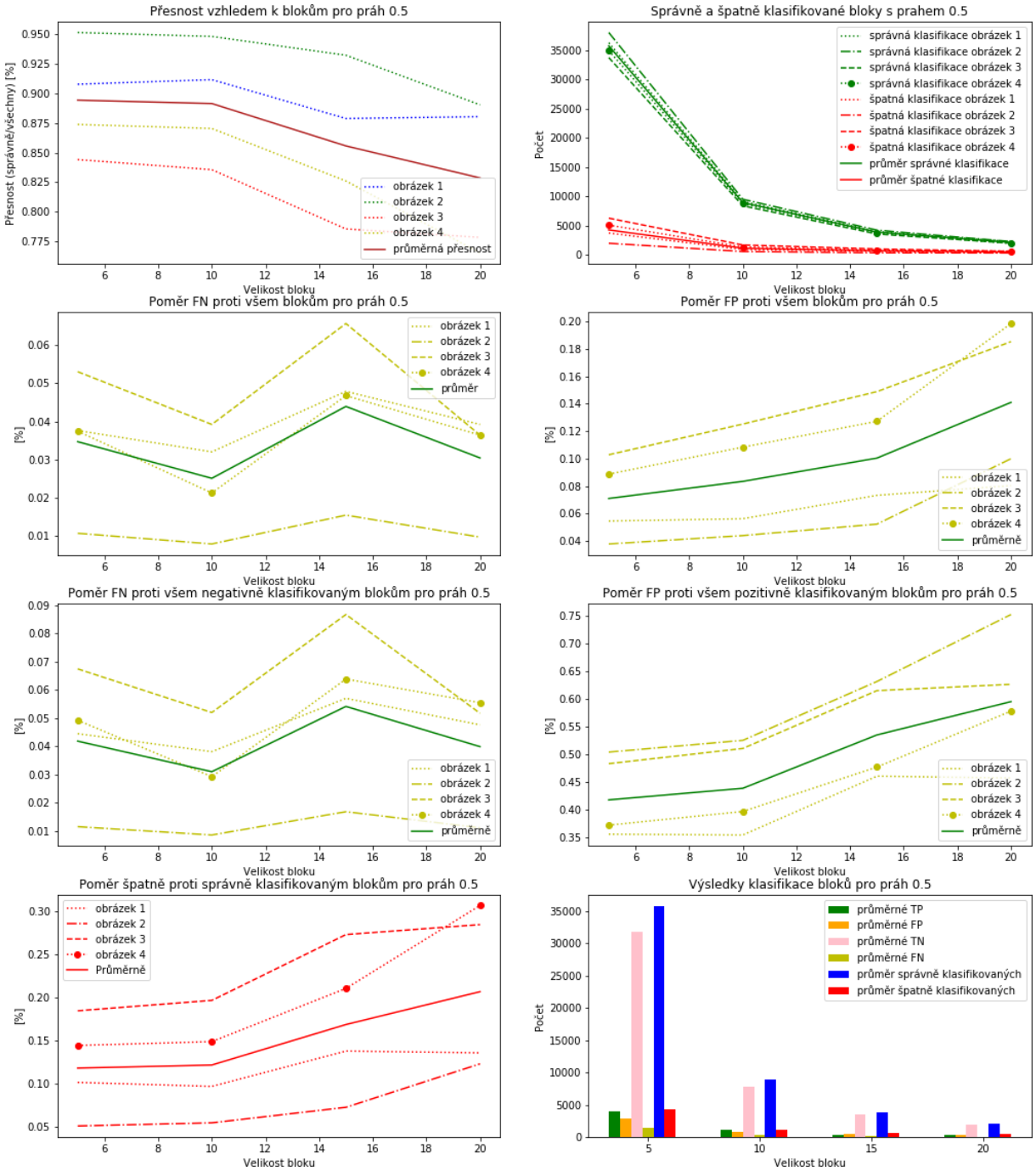
Obrázek 3.5: Zobrazení detekce budov pro nejlepší nastavení modelu 1 s použitým prahem 0.75. Vlevo výsledek detekce, uprostřed vzor a vpravo originál.

3.2.0.2 Experiment 2

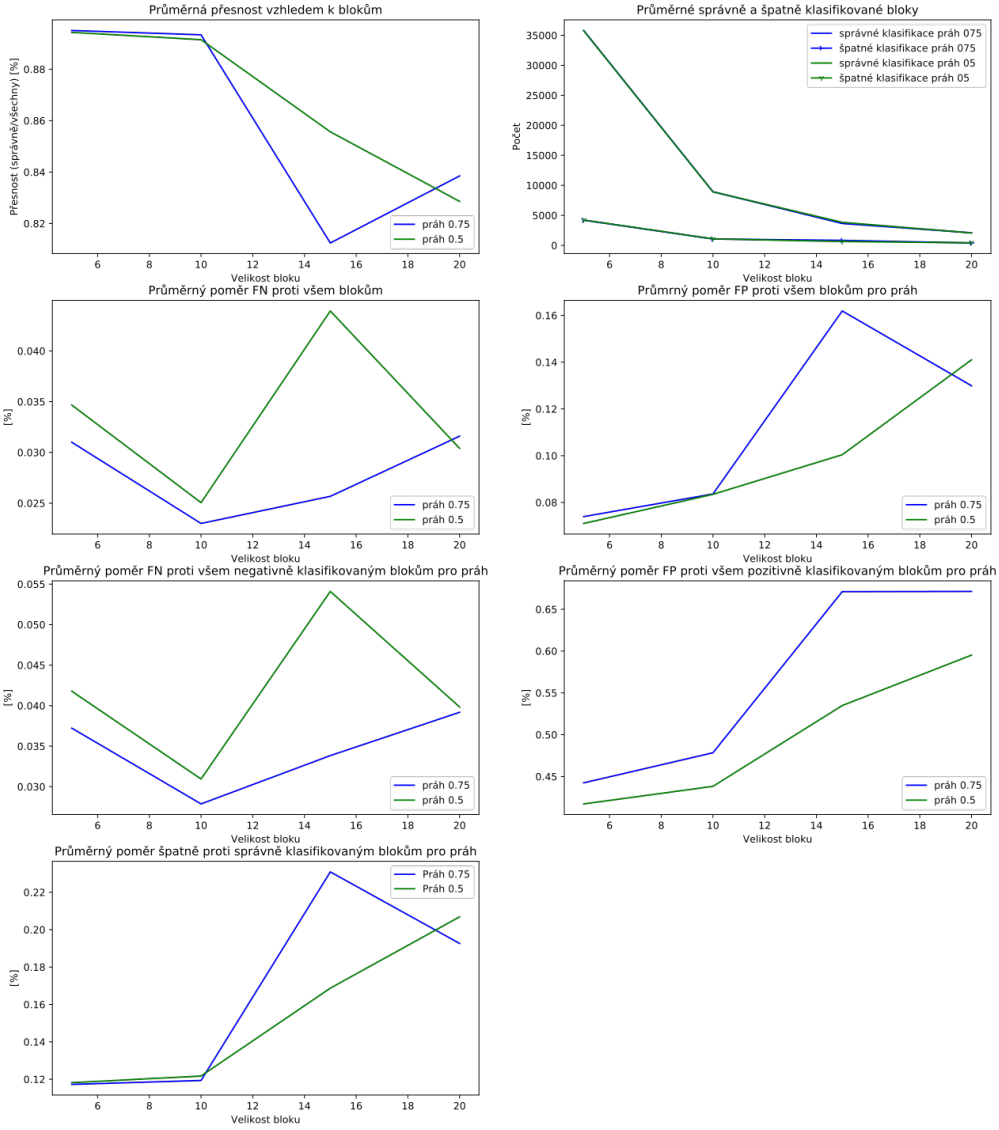
V tomto experimentu používáme model 2 a dataset pro Prahu. Zkoumejme nejprve model s prahem 0.75. Zde nejlepší přesnosti, téměř 90%, dosahují modely o velikosti bloku 5 a 10, viz obr. 3.6. V grafech, které zkoumají FN, vychází nejlépe model s velikostí bloku 10, zatímco u FP vychází lépe bloky o velikosti 5. Celkově se o něco lépe jeví model pro velikost bloku 5.
Zkoumáme-li grafy pro práh 0.5, nejlépe vychází model pro bloky velikosti 5, viz obr.3.7, a to jak v parametru přesnosti, tak i v dalších parametrech.
Porovnáme-li modely pro různé prahy (0.75 a 0.5), viz obr.3.8, dosahují přibližně stejné přesnosti, ale FN má lepší model s prahem 0.75, naopak model s prahem 0.5 je lepší u FP. Učiňme tedy závěr, že NS model 2 dosahuje srovnatelných výsledků pro oba prahy při stejné velikosti bloku 5, ale nepatrně lepší je model s prahem 0,5.



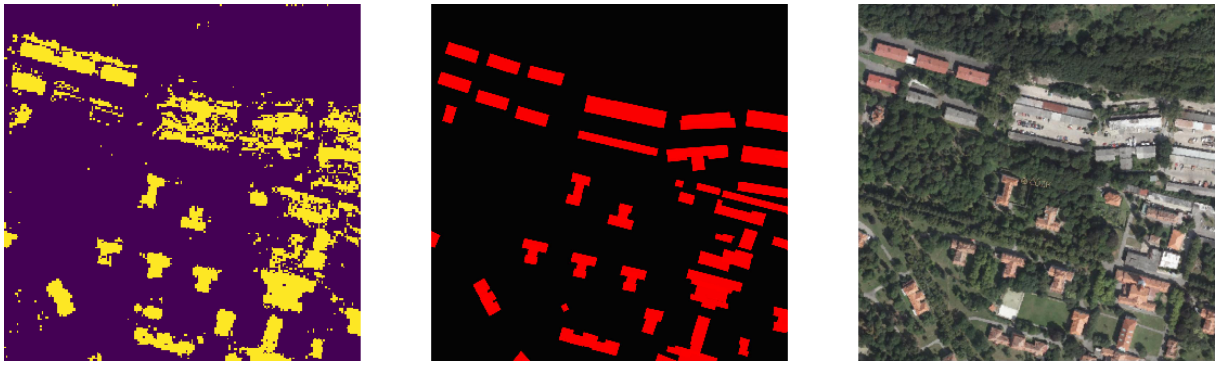
Obrázek 3.6: Vyhodnocení kvality klasifikace pro model 2, trénovaného na datech z ČR, práh 0,75.



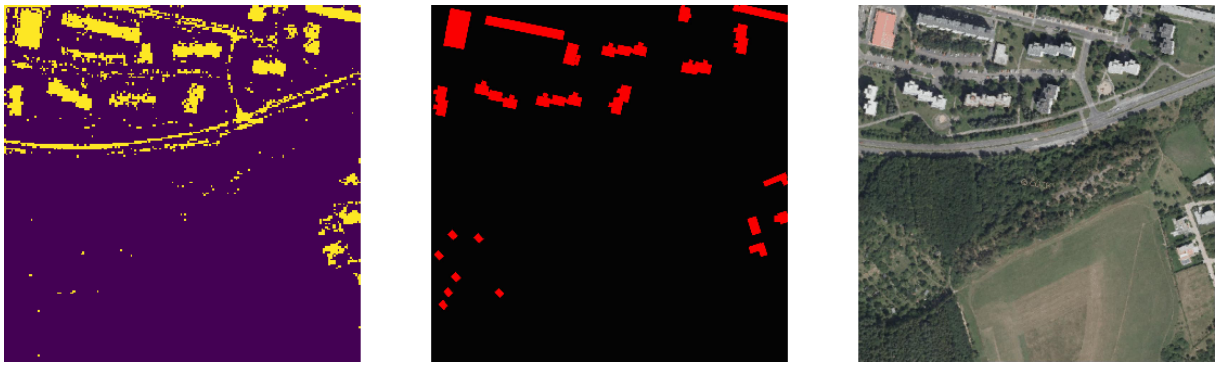
Obrázek 3.7: Vyhodnocení kvality klasifikace pro model 2, trénovaného na datech z ČR, práh 0,5.



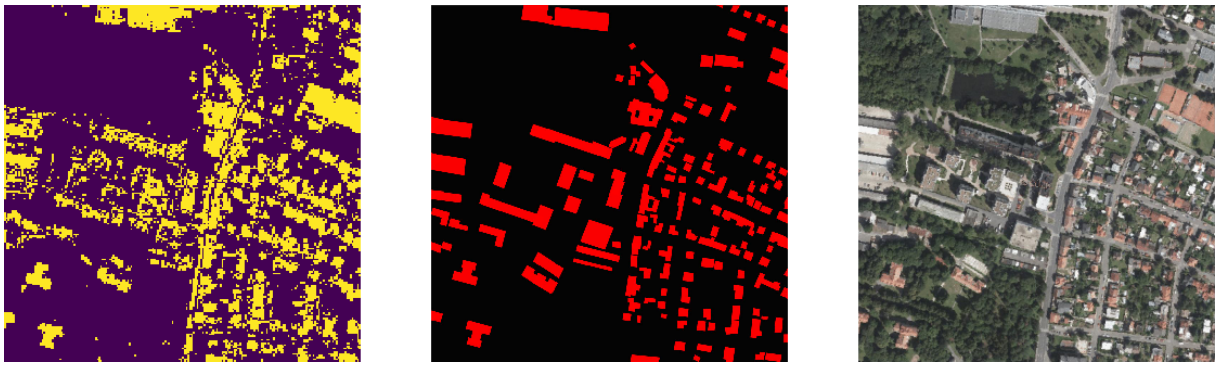
Obrázek 3.8: Vyhodnocení kvality klasifikace pro model 2, trénovaného na datech z ČR, pro hodnoty prahu $0,5$ a $0,75$.



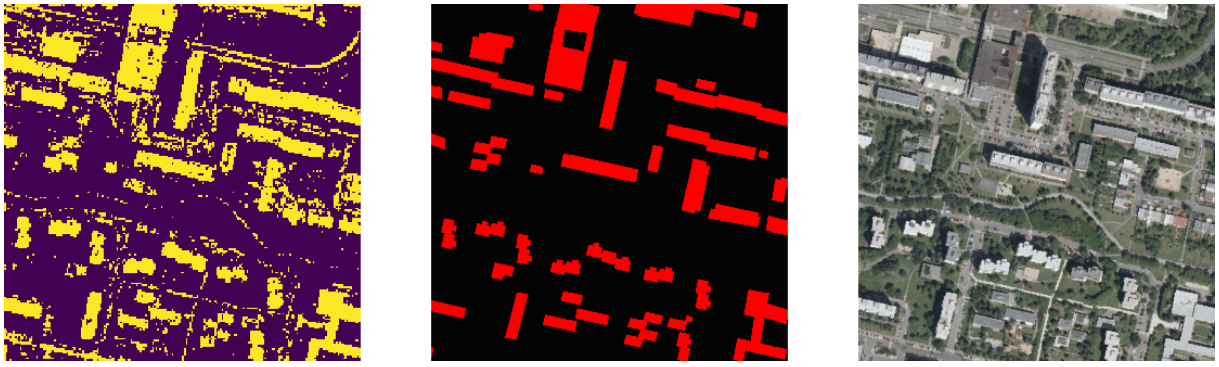
(a)



(b)



(c)



(d)

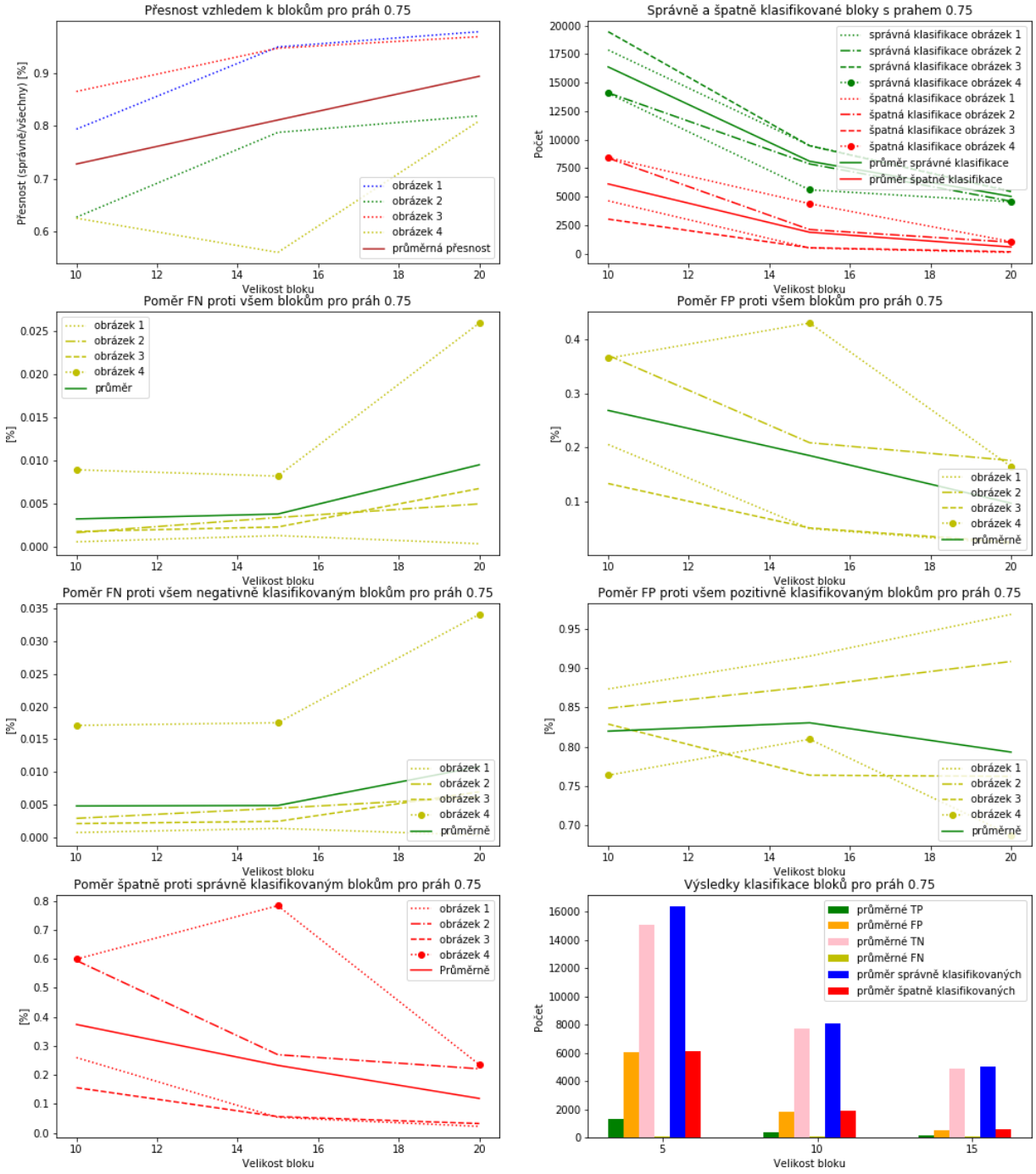
Obrázek 3.9: Zobrazení detekce budov pro nejlepší nastavení modelu 2 s použitým prahem 0.75. Vlevo výsledek detekce NS, uprostřed vzor a vpravo originál.

3.2.0.3 Experiment 3

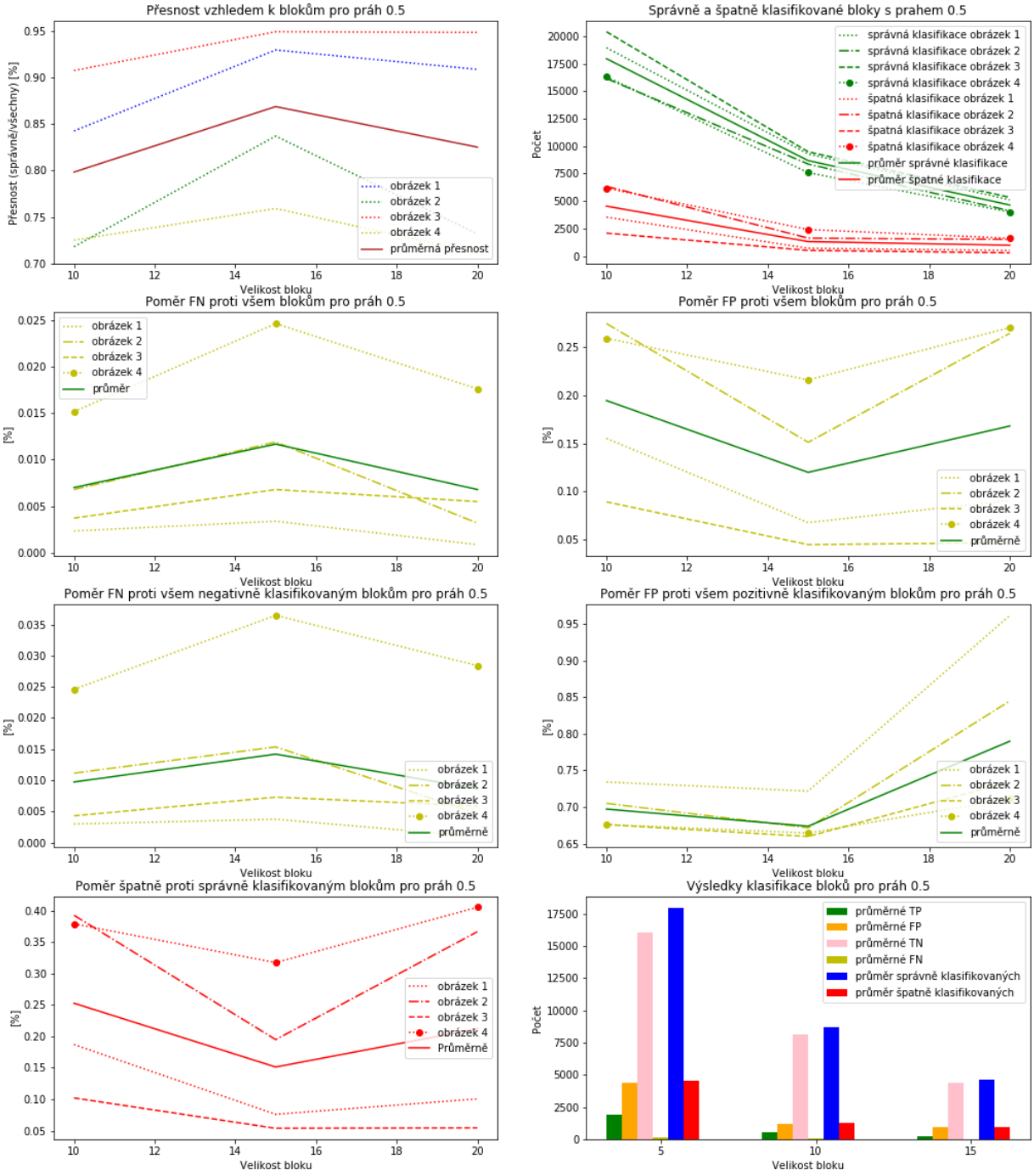
V tomto experimentu používáme model 1. Data jsou satelitní snímky Toronta.
Když budeme zkoumat model s prahem 0.75, viz obr. 3.10, jako nejlepší z hlediska přesnosti vychází modely s větší velikostí bloku. Nejlepší z nich je model o velikosti bloku 20 (přesnost téměř 90%). Z pohledu poměru FN vychází lépe model s velikostí bloků 10. Podíváme-li se naopak na FP vůči všem bloků lepší model je pro velikost bloků 20. Celkově jako lepší model se jeví model s velikostí bloku 20.
Nejlepší se jeví model 1 pro práh 0.5. Dosahuje nejlepší přesnosti pro velikosti bloku 15 a pro tuto velikost bloku dosahuje nejlepších výsledků i z hlediska FN a poměru špatně a dobře detekovaných. Model s prahem 0.75 sice má větší přesnost, ale zároveň má mnohem větší hodnoty pro FN a FP.

3.2.0.4 Experiment 4

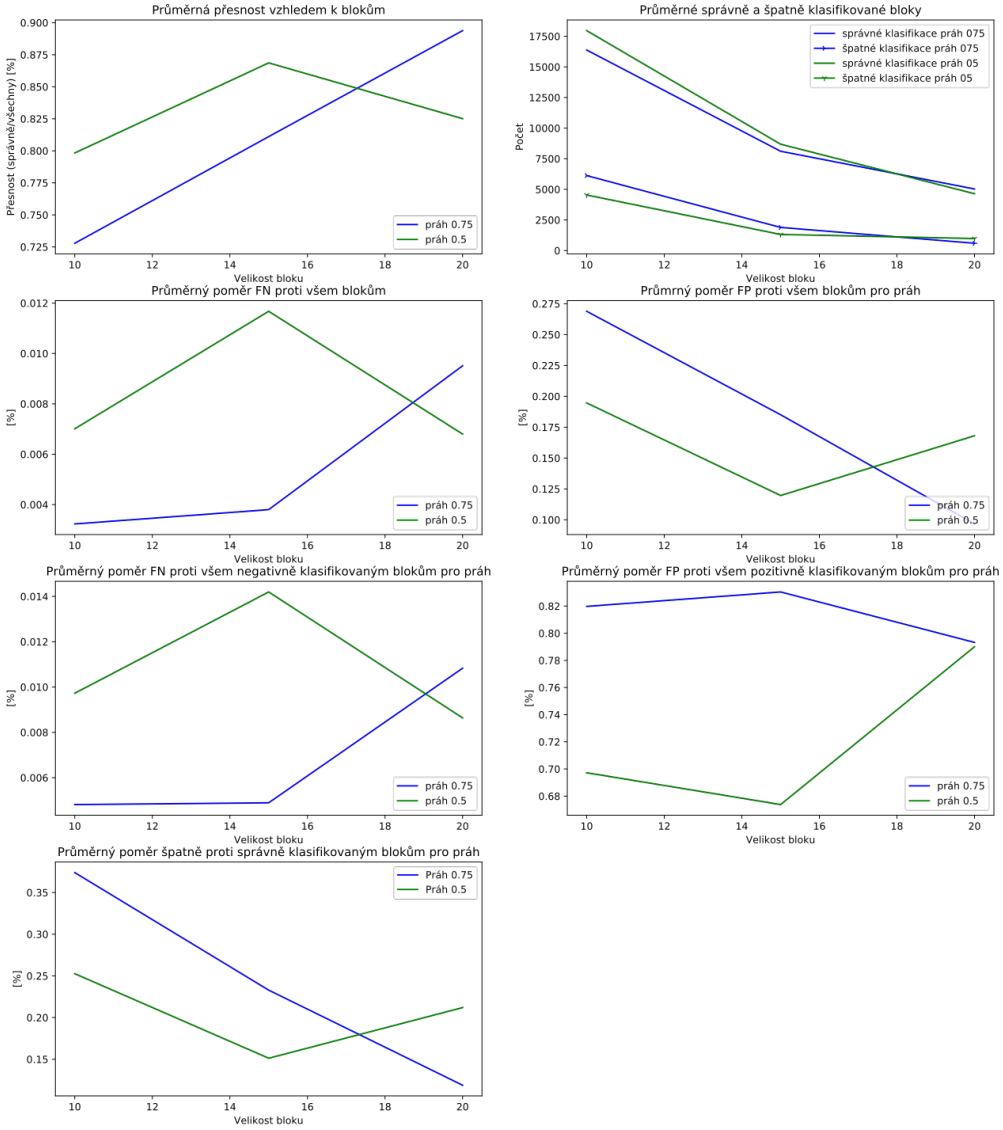
V tomto experimentu jsme použili model 2 a data jsou satelitní snímky Toronta.
Když budeme analyzovat modely s prahem 0.75, viz obr. 3.15, tak nejvyšší přesnosti dosahujeme u modelu s velikostí bloku 10, ale toto je pouze nejvyšší průměrná přesnost. Lze pozorovat, že přesnost pro větší bloky je velice rozdílná pro různé typy obrázků a v extrémních případech dosahuje více než 95% (blok o velikosti 16, resp. 18, resp. 20). Když budeme zkoumat poměr falešně pozitivních (FN) klasifikací v poměru ke všem, resp. k negativně klasifikovaným (blok byl označen jako pozadí), velice slušných výsledků dosahujeme od velikosti bloku 13 a větší, přibližně méně než 1%. Oproti tomu u falešně pozitivních (FP) vůči všem, resp. všem pozitivním klasifikacím (blok označen jako budova), dosahujeme lepších výsledků pro menší bloky. Celkově se jeví jako nejlepší model o velikosti bloku 10. Avšak pozorujeme, že výsledky jsou silně ovlivněny typem obrázku.
Když podobnou analýzu provedeme pro modely s prahem 0.5, vidíme, že s rostoucí velikostí bloku obecně dosahujeme lepšího modelu, přičemž nejlepší modely jsou s velikostí bloku 13 a 15.
Když bychom chtěli modely porovnat vzhledem k volbě prahu, lepší se jeví model pro práh 0.75, jelikož dosahuje vyšší přesnosti a nižších poměru pro FN.



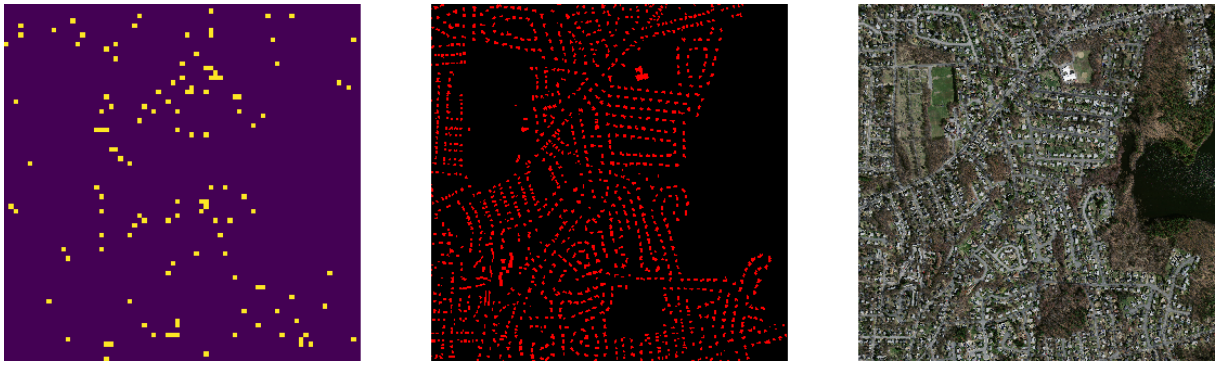
Obrázek 3.10: Vyhodnocení kvality klasifikace modelu 2 pro práh 0,75. Data použitá pro trénování jsou snímky Toronta.



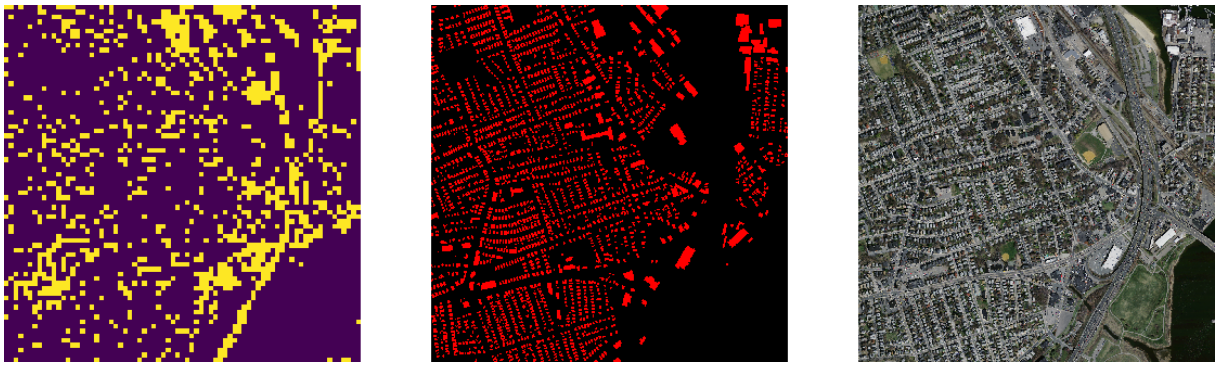
Obrázek 3.11: Vyhodnocení kvality klasifikace modelu 2 pro práh 0,75. Data použitá pro trénování jsou snímky Toronta.



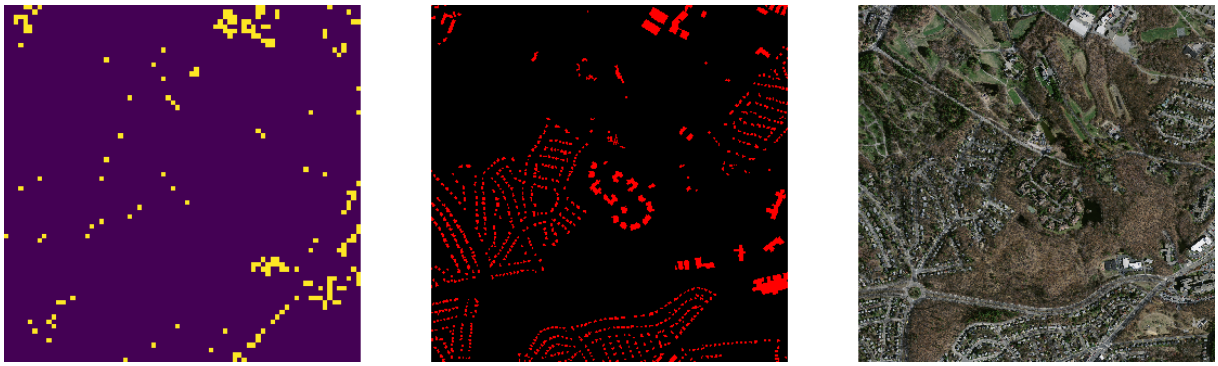
Obrázek 3.12: Vyhodnocení kvality klasifikace modelu 2 pro prahy 0,5 a 0,75. Data použitá pro trénování jsou snímky Toronta.



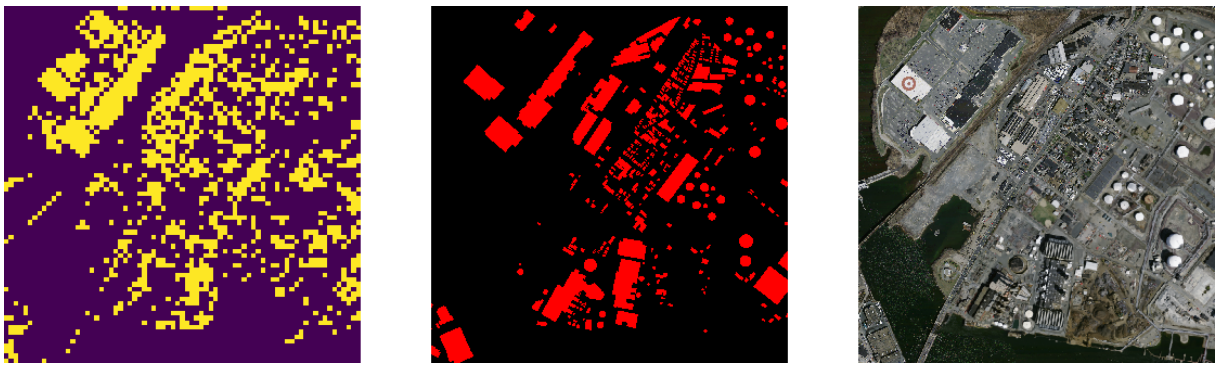
(a)



(b)

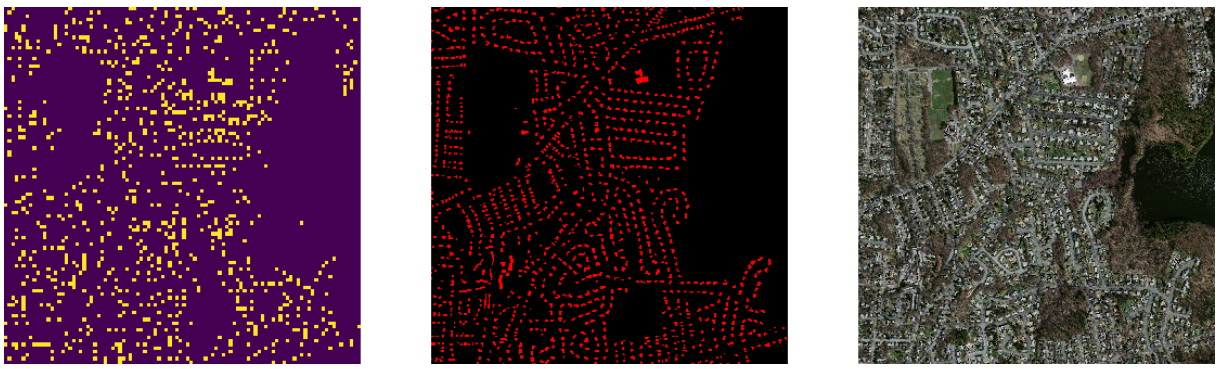


(c)

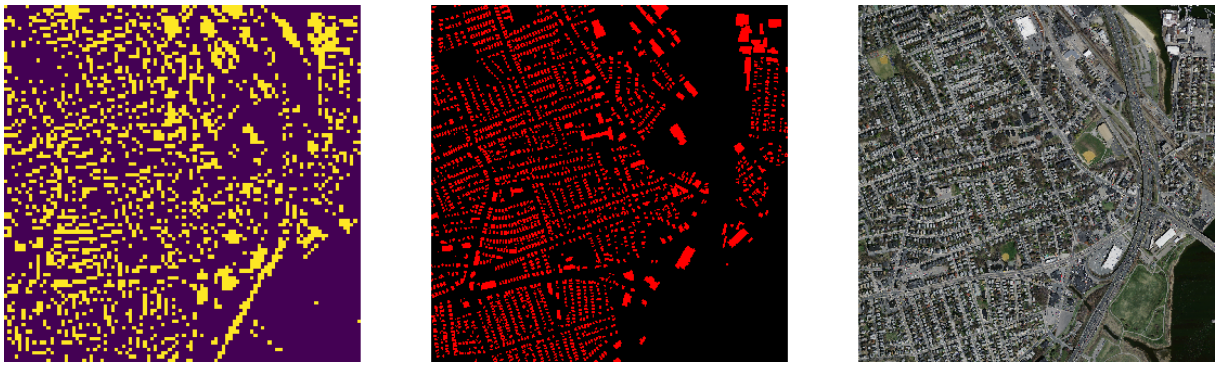


(d)

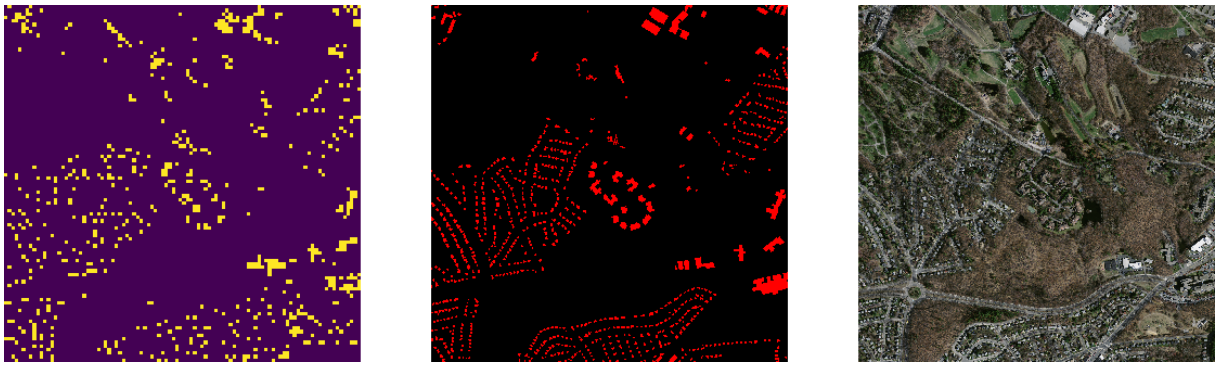
Obrázek 3.13: Zobrazení detekce budov pro nejlepší nastavení modelu s použitým prahem 0.75. Vlevo výsledek detekce NS, uprostřed vzor a vpravo originál.



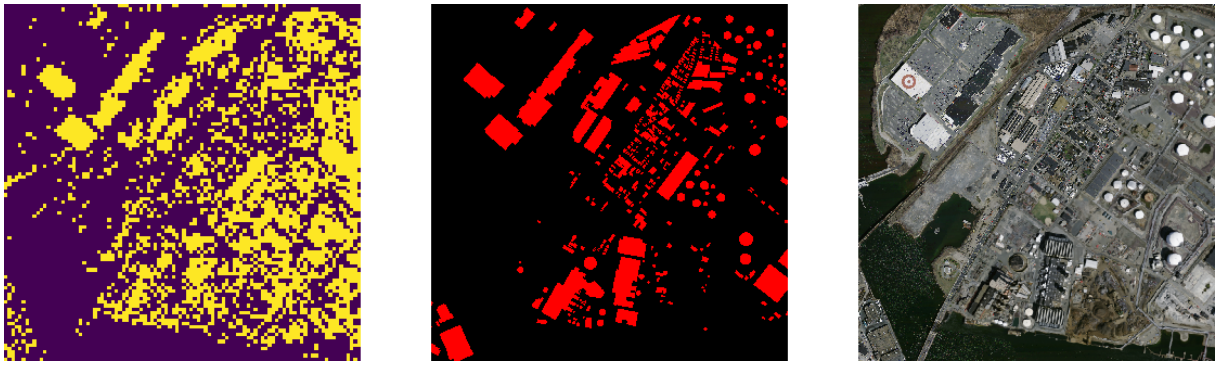
(a)



(b)

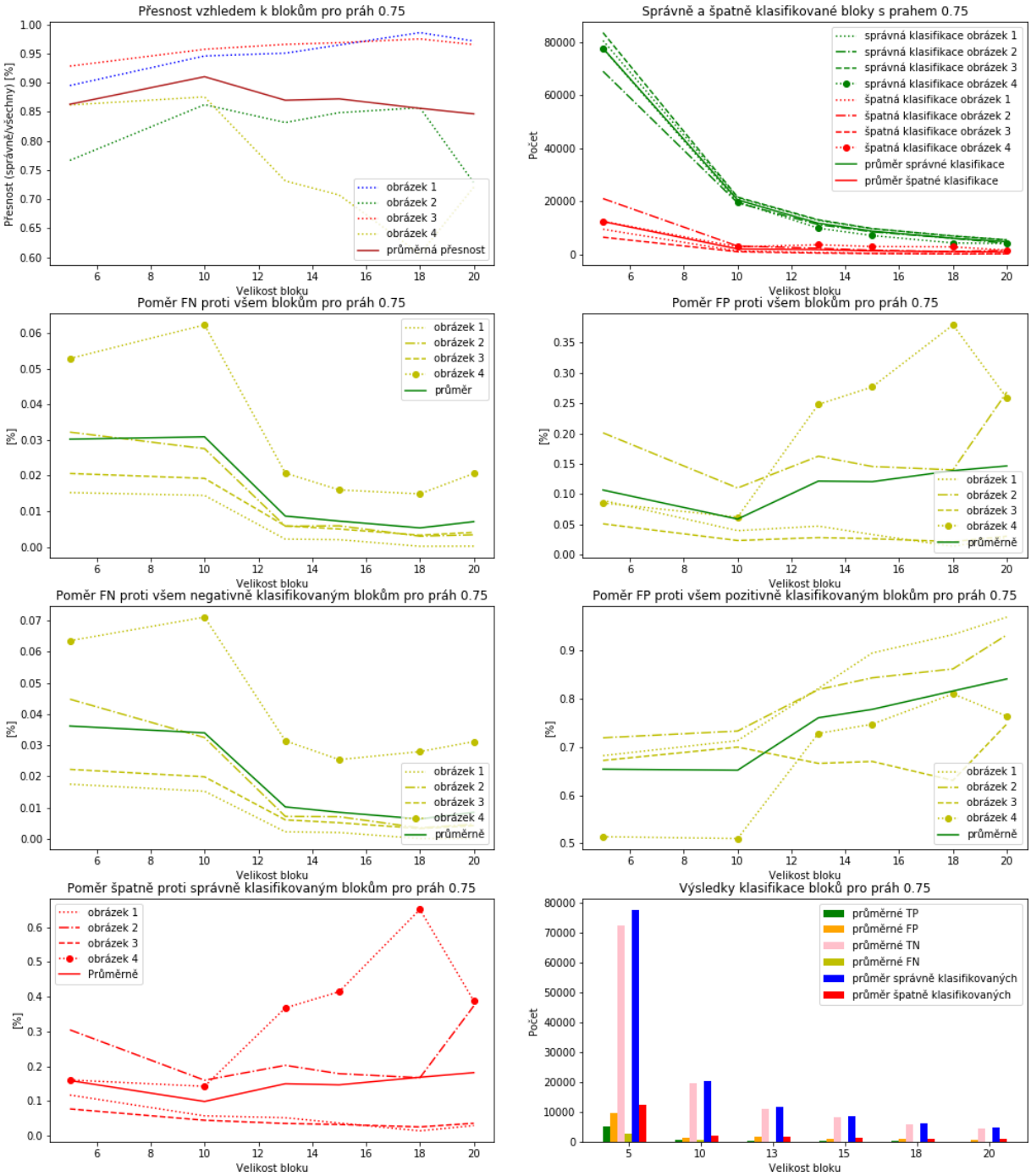


(c)

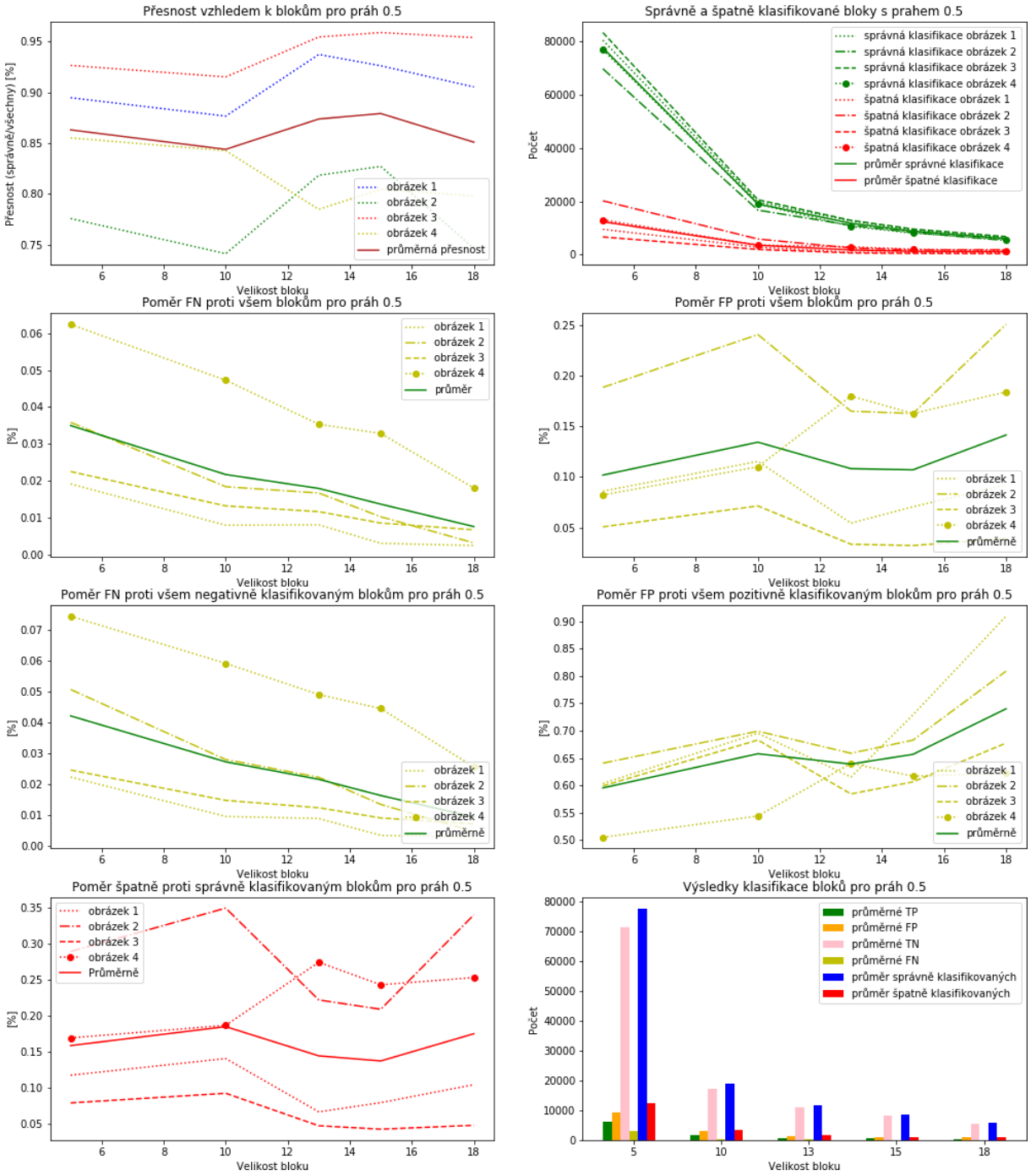


(d)

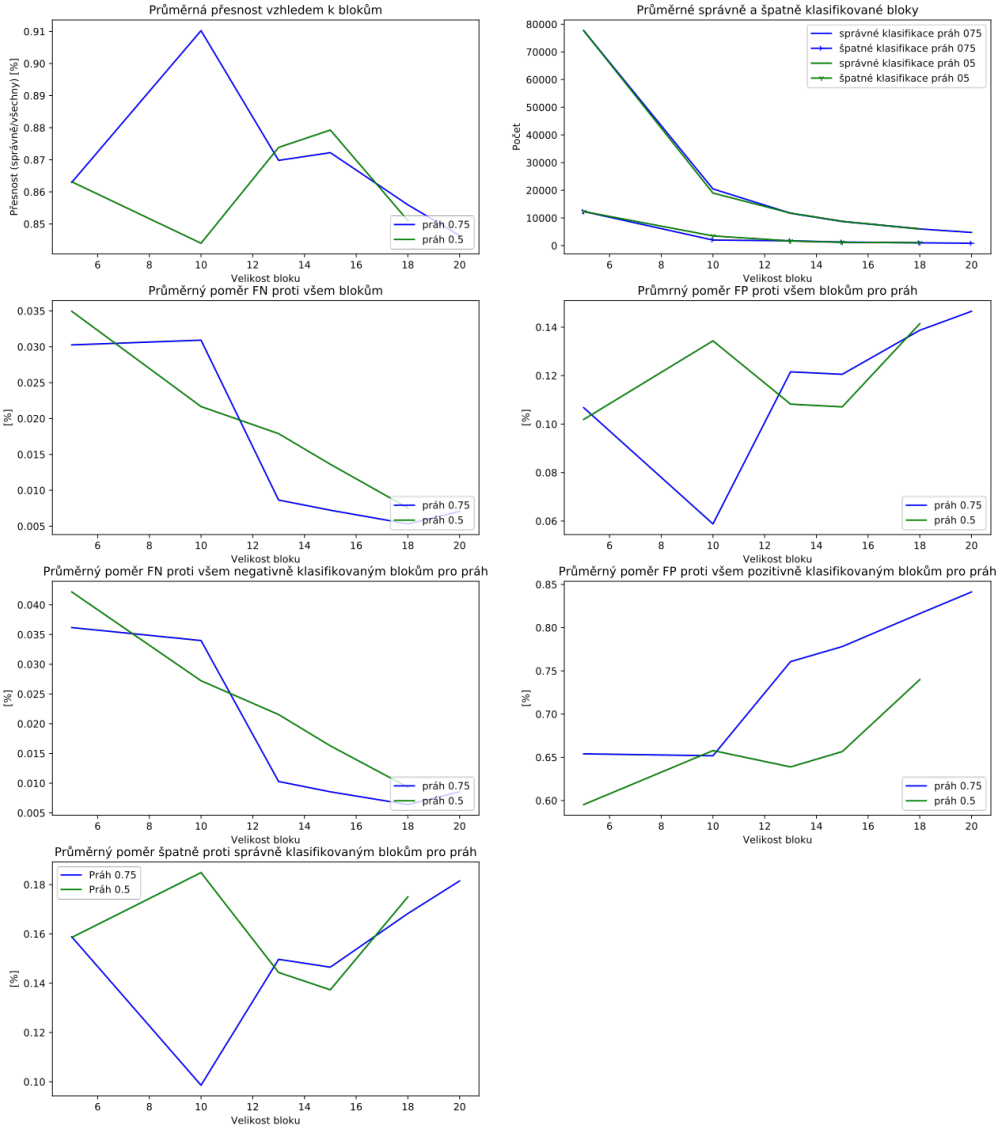
Obrázek 3.14: Zobrazení detekce budov pro nejlepší nastavení modelu s použitým prahem 0.5. Vlevo výsledek detekce NS, uprostřed vzor a vpravo originál.



Obrázek 3.15: Vyhodnocení kvality klasifikace modelu 2 trénovaného na satelitních snímcích Toronta a práh 0,75.



Obrázek 3.16: Vyhodnocení kvality klasifikace modelu 2 trénovaného na satelitních snímcích Toronta a práh 0,5.



Obrázek 3.17: Vyhodnocení kvality klasifikace modelu 2 trénovaného na satelitních snímcích Toronta, Porovnání prahů 0,5 a 0,75.

3.2.1 Shrnutí výsledků NS

Pro lepší přehled zobrazme výsledky a parametry nejlepších modelů v tabulce.

Model	Velikost bloku	Práh	Průměrná přesnost
model 1	5	0.5	0.875
model 2	5	0.75	0.9

Tabulka 3.2: Přehled nejlepších výsledků pro modely 1 a 2 pro detekci budov na satelitních snímcích Prahy.

Model	Velikost bloku	Práh	Průměrná přesnost
model 1	15	0.5	$0,87$
model 2	10	0.75	$0,91$

Tabulka 3.3: Přehled nejlepších výsledků pro modely 1 a 2 pro detekci budov na satelitních snímcích Toronta.

Kapitola 4

Statistické znaky rozložení budov ve městě

Abychom mohli modifikovat algoritmus pro procedurální modelování města k získání realističtějších výsledku, musíme zjistit některé parametry ze satelitních snímků.
Parametry, které budeme sledovat v obrázcích jsou

- Obvod bloku budov.
- Protaženost bloku budov.
- Vzájemná vzdálenost budov.
- Plocha bloku budov.

Znaky budeme zkoumat na deseti vybraných obrázcích Prahy, které nebyly v trénovací, validační ani testovací sadě pro NS, viz obr. 4.1-4.3. Pro testování jsme použili nejlepší nastavení pro model 2. Porovnávali jsme rozložení sledovaných znaků pro predikované obrázky z NS vůči příslušejícím vzorům.

(a) obr. 81 (b) obr. 82

Obrázek 4.1: Satelitní snímky, na kterých jsme počítali statistické znaky

(a) obr. 85 (b) obr. 91

(c) obr. 92 (d) obr. 94

Obrázek 4.2: Satelitní snímky, na kterých jsme počítali statistické znaky

(a) obr. 95

(b) obr. 97

(c) obr. 99

(d) obr. 100

Obrázek 4.3: Satelitní snímky, na kterých jsme počítali statistické znaky

4.1 Úprava predikovaných map z NS

Pro získání potřebných parametrů, musíme v predikovaných obrázcích i ve vzorech provést úpravy kvůli šumu a nepřesně detekci. Základní modifikaci provedl algoritmus založený na Perona-Malikově anizotropní difuzi, viz [30]. Poté jsme ještě provedli operaci pomocí knihovny `opening`.
Na obr. 4.4, můžeme porovnat rozdíl mezi predikovaným obrázkem z NS bez úpravy a jeho úpravou.
Při kontrole predikovaných obrázků, lze vidět, že NS detekuje některé objekty špatně. Silnice je často interpretována jako budovu. Na druhou stranu u mapových podkladů, na jejichž základě vznikly vzory pro NS, je problém s jejich aktuálností. Z tohoto důvodu si můžeme v některých případech povšimnout, že NS detekuje budovy, které lze vidět na satelitních snímcích, ale v mapových podkladech a tedy i ve vzorech nejsou zaznamenány. Bohužel i v tomto případě tento objekt bude do statistiky zahrnut jako FP, a tím vznikají chyby v měření přesnosti.

Obrázek 4.4: Vizualizace úpravy predikovaného obrázku z NS (vlevo) pomocí anizotropní difuzi a openingu (vpravo).

4.2 Segmentace objektů

Nejprve jsme provedli detekci hranice objektů, ať už se jedná o vzory nebo o predikované obrázky. K tomuto jsme používali knihovnu v Pythonu, `scikit-image`. Abychom mohli bloky správně identifikovat, musíme obrázky rozšířit o nulový rám. Tím dosáhneme uzavření všech hranic objektů. Poté využijeme funkci `find_contours` z knihovny `scikit-image`. Do této knihovny je potřeba znát parametr práh, pro rozpoznání objektů. Tento práh byl nalezen pomocí Otsu algoritmu [8].
Bohužel tato knihovna neumí zjistit hierarchii hranic, tzn. neumí rozpoznat vnitřní a vnější hranici objektů. V dalším kroku proto musím najít a odstranit všechny vnitřní hranice, abychom nezkreslili statistiku. Vnitřní hranici budeme hledat tak, že budeme procházet všechny hranice a zaznamenáme nejvyšší, nejnižší body, stejně jako bod nejvíce vlevo a vpravo. Tyto body pak porovnáme vůči ostatním hranicím a pokud budou splněny podmínky, hranice bude vymazána. Popišme si tento algoritmus a jeho podmínky. Mějme hranici A a hranici B, o hranici A řeknu, že je vnitřní hranicí hranice B, pokud platí následující podmínky:

1. Nejvyšší bod hranice A bude níže než nejvyšší bod hranice B, ale zároveň výše než nejnižší bod hranice B.

2. Nejnižší bod hranice A bude výše než nejnižší bod hranice B a níže než nejvyšší bod hranice B.

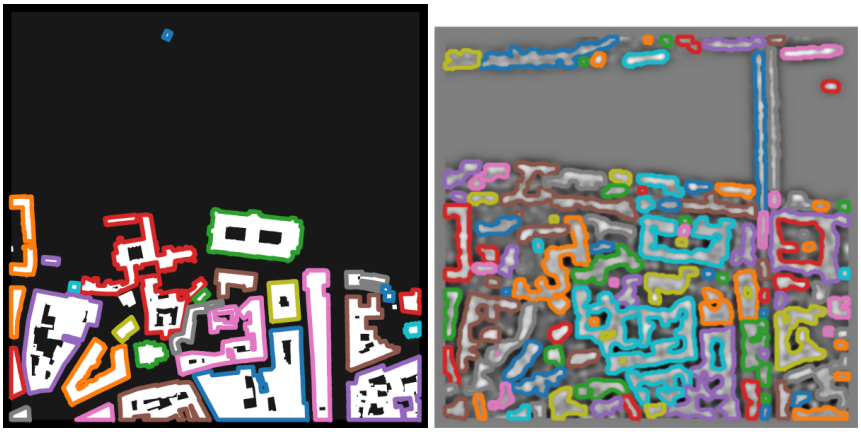
3. Bod nejvíce vlevo hranice A bude ležet vpravo od budu, který je nejvíce vlevo hranice B a současně se nachází vlevo od bodu hranice B, který je nejvíce vpravo.

4. Obdobně pro bod nejvíce vpravo hranice A, který leží vlevo od bodu hranice B, který je nejvíce vpravo, ale zároveň se nachází vpravo od bodu, který leží nejvíce vlevo hranice B.

Formulujme tento postup matematicky. Bod nejvíce vpravo hranice A resp. B ozn. $[x_{PA}, y_{PA}]$ resp. $[x_{PB}, y_{PB}]$, bod nejvíce vlevo hranice A resp. B ozn. $[x_{LA}, y_{LA}]$ resp. $[x_{LB}, y_{LB}]$, nejvyšší bod hranice A resp. B ozn. $[x_{VA}, y_{VA}]$ resp. $[x_{VB}, y_{VB}]$ a nakonec nejnižší bod hranice A resp. B ozn. $[x_{NA}, y_{NA}]$ resp. $[x_{NB}, y_{NB}]$. Podmínky zapišme matematicky.

1. $[y_{VA}] < [y_{VB}]$ a $[y_{VA}] > [y_{NB}]$
2. $[y_{NA}] > [y_{NB}]$ a $[y_{NA}] < [y_{VB}]$
3. $[x_{LA}] > [x_{LB}]$ a $[x_{LA}] < [x_{PB}]$
4. $[x_{PA}] < [x_{PB}]$ a $[x_{PA}] > [x_{LB}]$

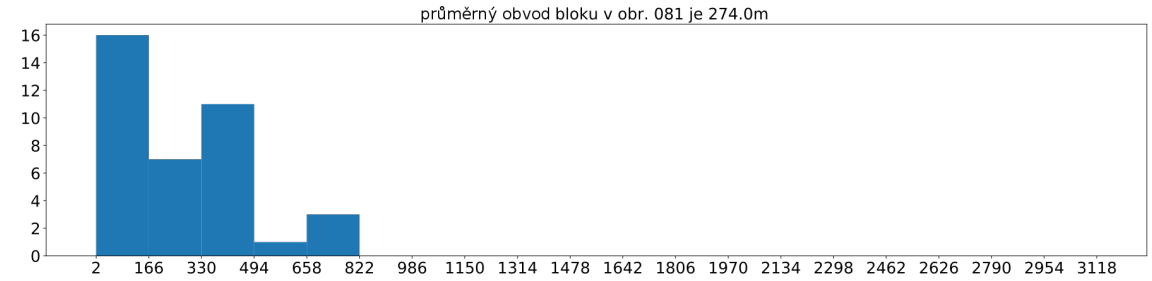
Tento algoritmus sice není robustní, ale pro naše účely je postačující. Tento poslední krok provádíme pouze ve vzorech, jelikož v predikovaných obrázcích dochází často ke slévání objektů. Z tohoto důvodu se tento krok neosvědčil a lepších výsledků dosahujeme bez odstranění vnitřních hranic. Můžeme se podívat na ukázku tohoto postupu na obr. 4.5. Hranice budov jsou uloženy do pole.



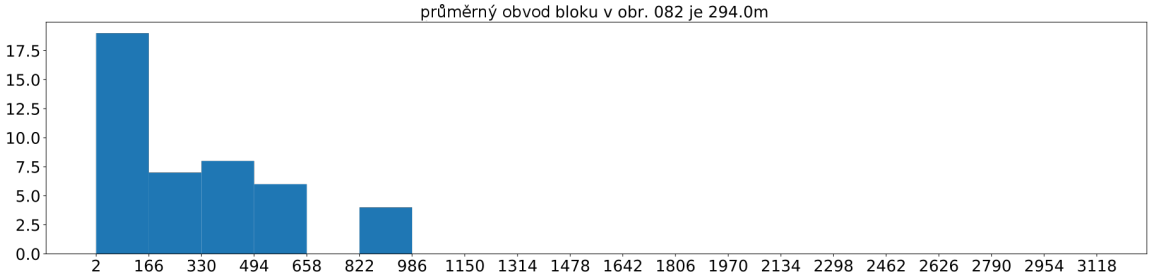
Obrázek 4.5: Vizualizace hledání hranice bloku. Vlevo je vzor a vpravo je obrázek z NS.

4.3 Obvod bloku budov

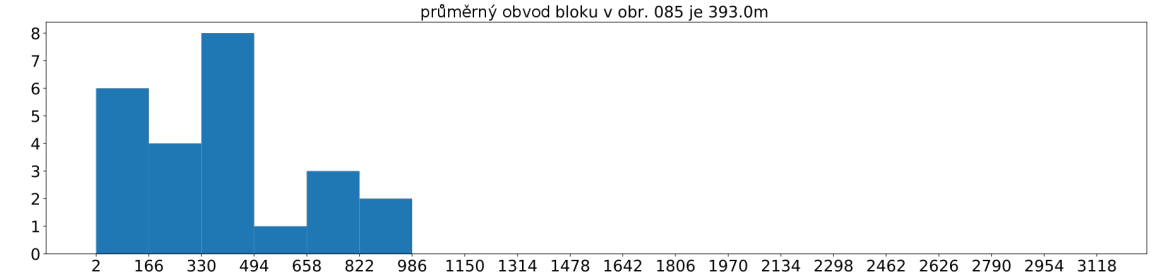
Nyní se podívejme na rozdělení délky bloků v jednotlivých vzorových obrázcích i na celkové rozdělení přes všechny vzory, viz obr. 4.6 - 4.8.



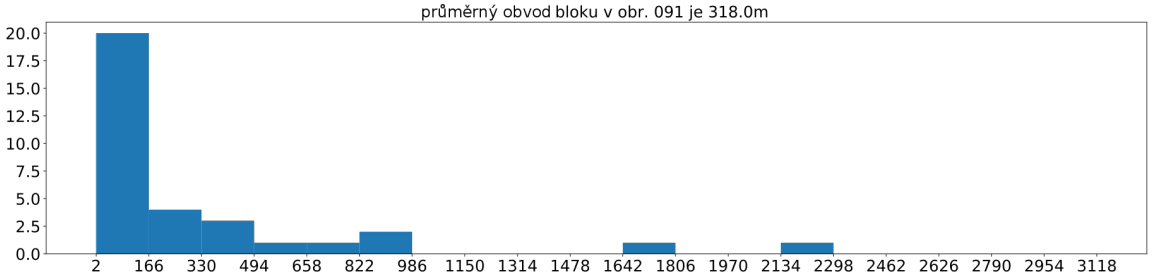
(a)



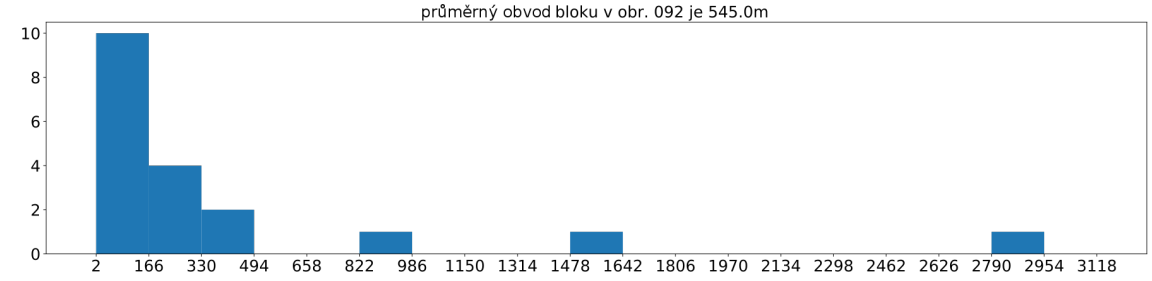
(b)



(c)

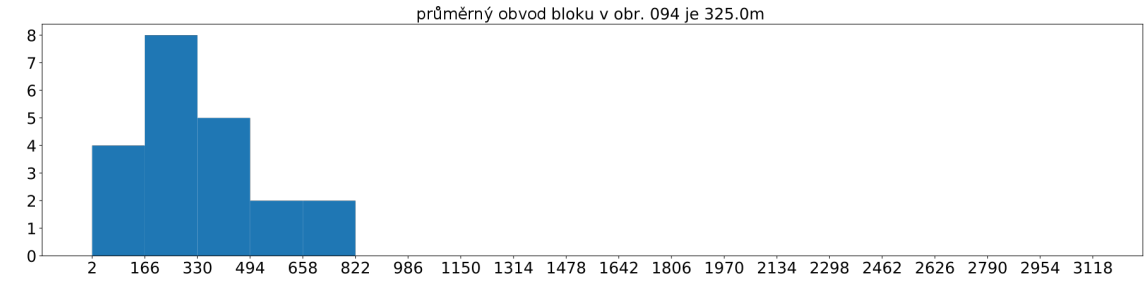


(d)

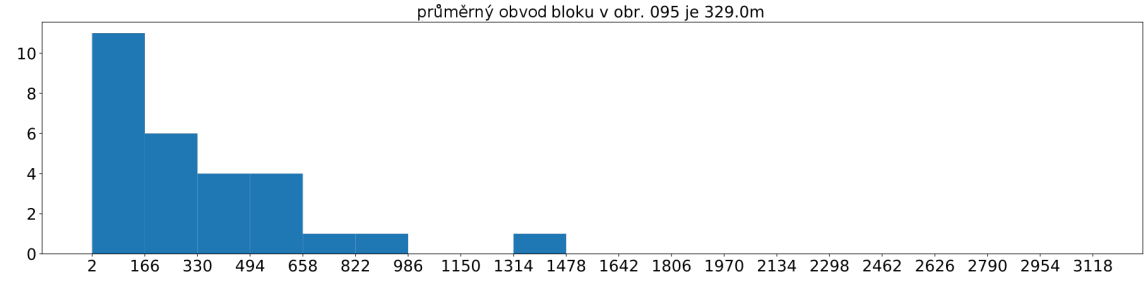


(e)

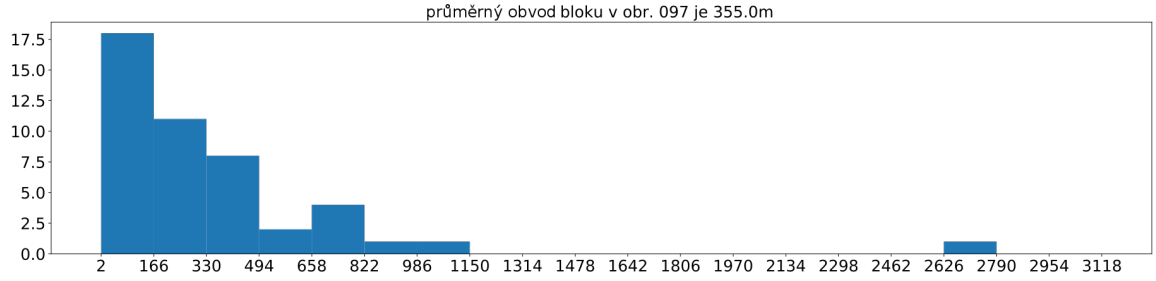
Obrázek 4.6: Histogram obvodu bloků pro vzory obr. 81-92. Jednotky na ose x jsou v metrech.



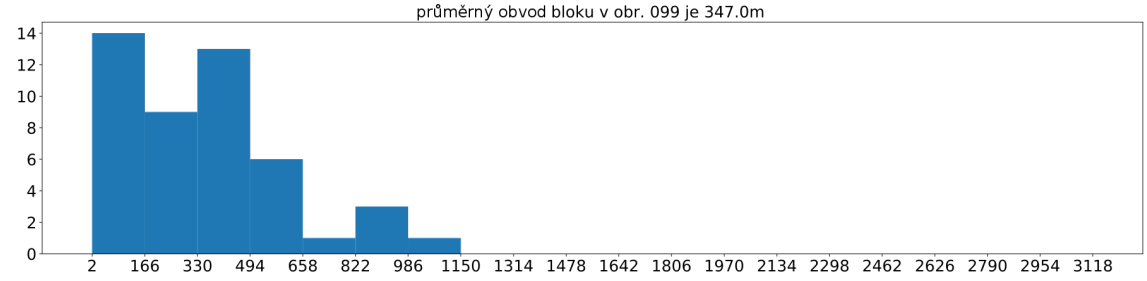
(a)



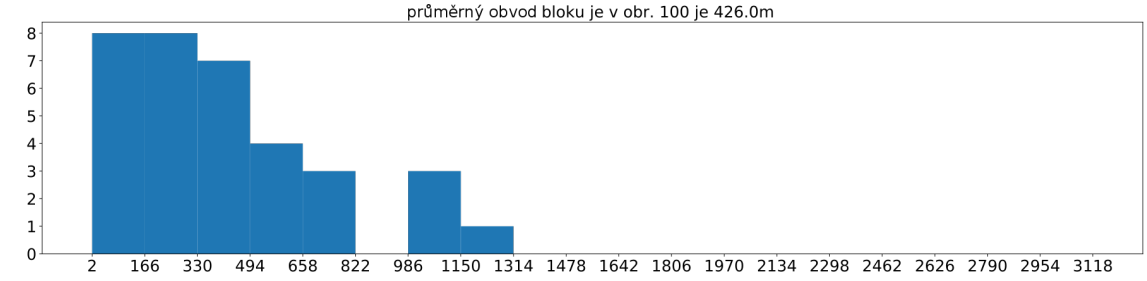
(b)



(c)

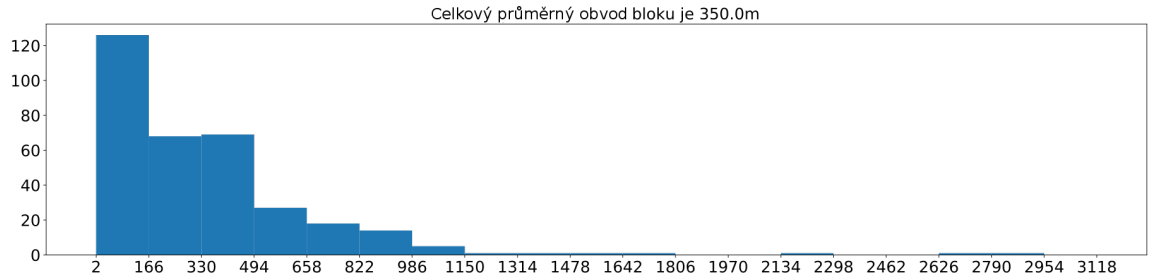


(d)



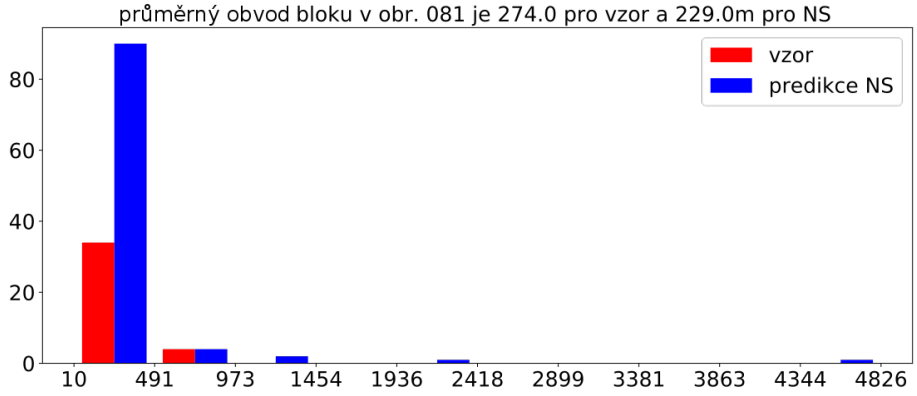
(e)

Obrázek 4.7: Histogram obvodu bloků pro vzory obr. 94-100. Jednotky na ose x jsou v metrech.

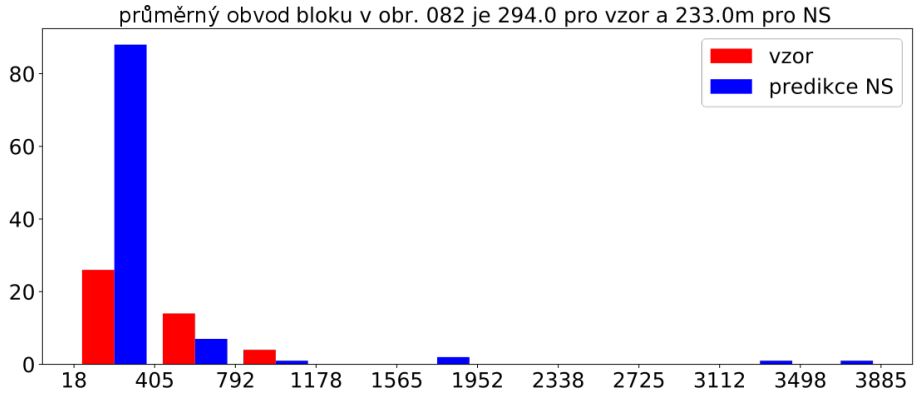


Obrázek 4.8: Histogram obvodu bloků pro vzory přes všechny obrázky. Jednotky na ose x jsou v metrech.

Když se podíváme na histogramy, pozorujeme ve všech grafech klesající trend podobný log-normálnímu rozdělení.
Nyní porovnejme obvod bloku ve vzoru s obrázky získanými predikcí NS, viz obr.4.9-4.11. Průměrný obvod bloku ve vzoru je 350 m a v predikovaném obr. je 183 m. Tyto parametry použijeme k modifikaci programu pro procedurální generování města. Ze samotných hodnot můžeme usuzovat, že při predikci docházelo velmi často k rozmělnění velkých bloků do menších.

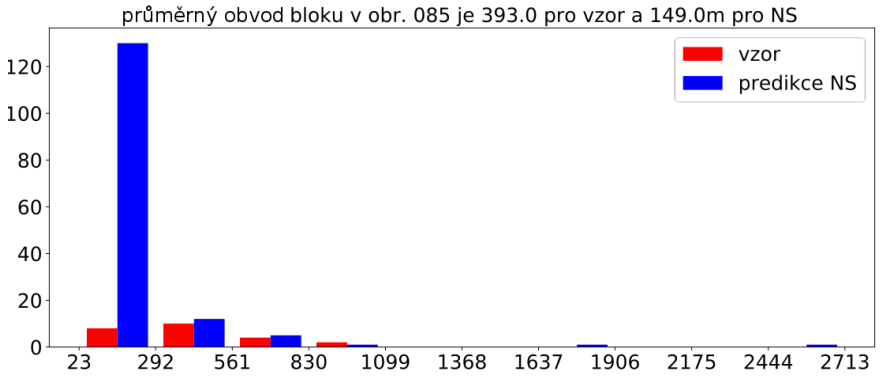


(a)

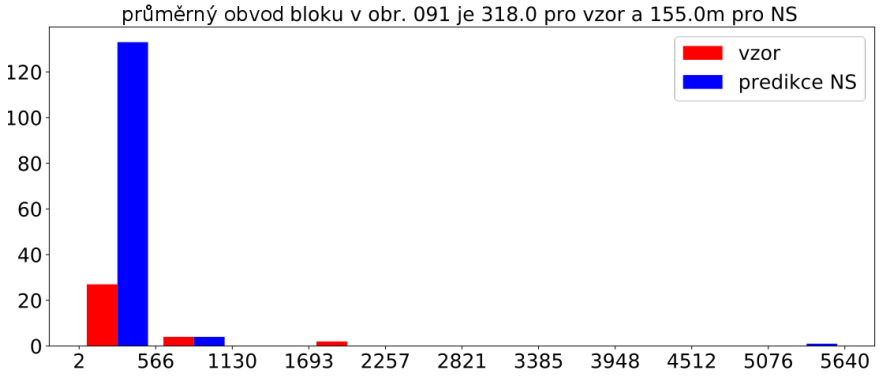


(b)

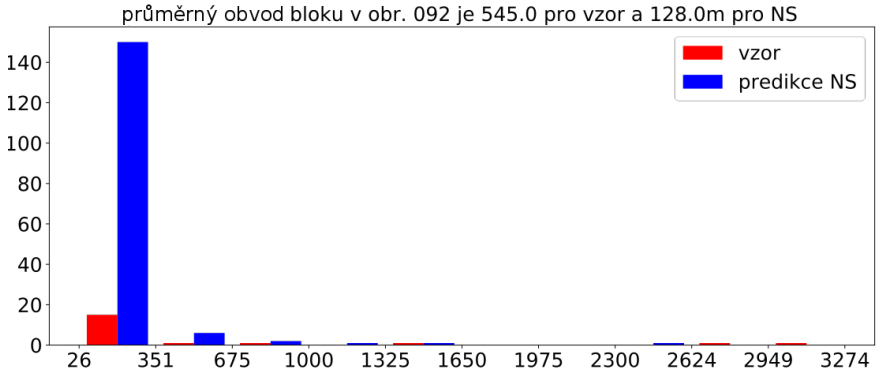
Obrázek 4.9: Histogram obvodu bloků pro predikované obr. 81-82. Jednotky na ose x jsou v metrech.



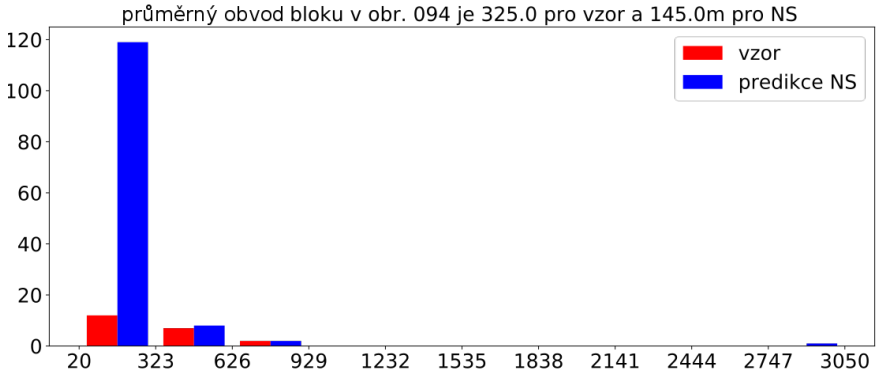
(a)



(b)

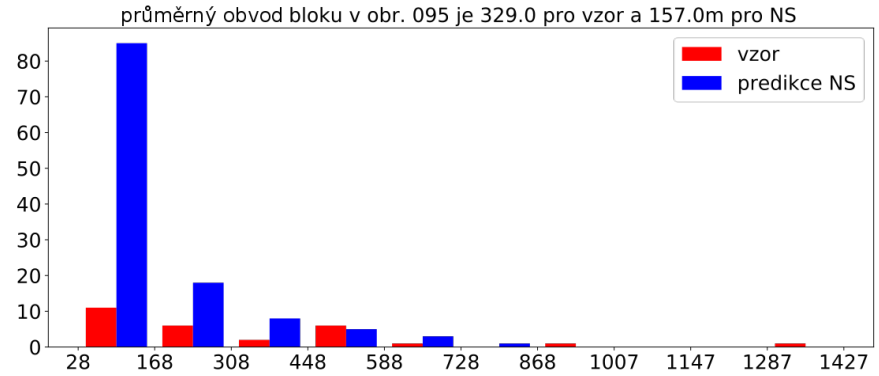


(c)

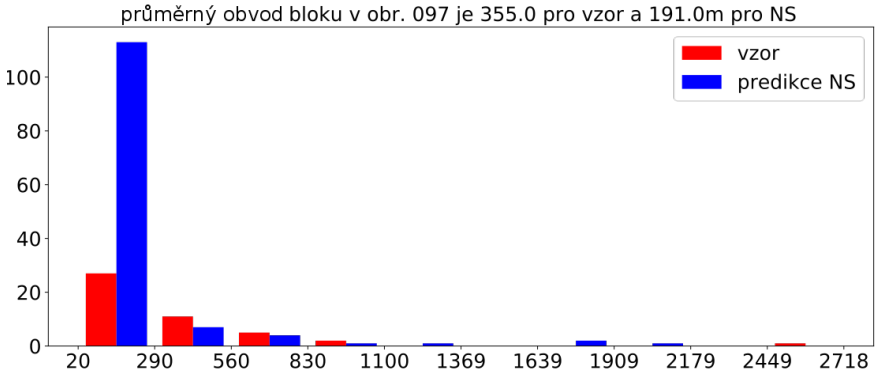


(d)

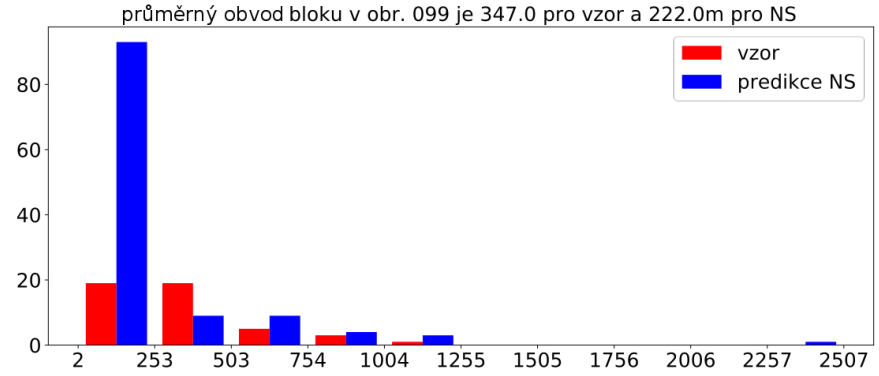
Obrázek 4.10: Histogram obvodu bloků pro predikované obr. 85-94. Jednotky na ose x jsou v metrech.



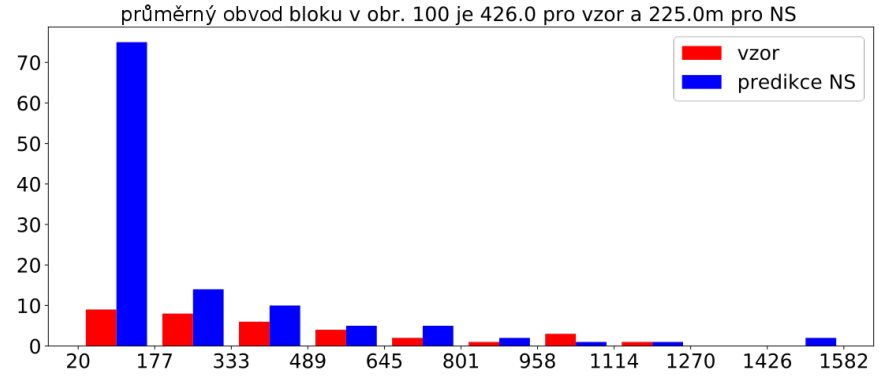
(a)



(b)



(c)



(d)

Obrázek 4.11: Histogram obvodu bloků pro predikované obr. 95-100. Jednotky na ose x jsou v metrech.

obr.	081	082	085	091	092	094	095	097	099	100
korelační koef.	0.796	0.958	-0.441	1.	-1.	0.988	0.238	0.998	0.501	-0.130

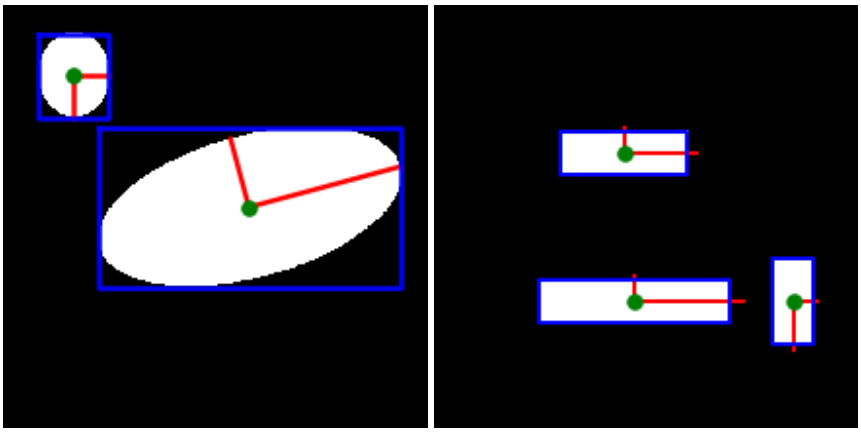
Tabulka 4.1: Korelační koeficienty pro délku obvodu bloků budov.

V následující tabulce 4.1 jsou uvedeny korelační koeficienty vypočtené pro jednotlivé obrázky srovnáním histogramů obvodu bloků získaných ze vzorů a z predikcí NS.

U šesti obr. můžeme pozorovat silnou korelaci vzorů a predikovaných NS. Když se podíváme na jejich vizualizaci, tak to jsou ty, které nemají mnoho chyb při predikci, tzn. není záměna v detekci silnic a budov, navíc budovy př predikci NS nejsou příliš rozdrobeny na menší objekty. Budovy ve vzorech jsou pravidelné a mají mezi sebou široké rozestupy, zároveň zde nedochází příliš k míchání městského prostředí a přírody.
Vycházejme z předpokladu, že NS bude v detekci dělat chyby, tyto chyby se budou nejčastěji v detekovaných obrázcích projevovat tak, že větší budova bude rozdělena na několik menších, a proto předpokládáme vyšší výskyt kratších kontur. Další hypotéza je, že především u obrázků, které mají užší silnice, bude docházet naopak ke slévání budov. Toto by se pak na histogramu projevilo menším počtem středně velkých budov a nepatrným zvýšením (v řádu jednotek) budov s dlouhými konturami. Oba tyto jevy můžeme pozorovat jednak v histogramech, ale stejně tak i v predikovaných obrázcích, viz obr. 4.9-4.11.

4.4 Protažení budov

Protažení budov budeme definovat jako poměr délek vedlejší a hlavní poloosy elipsy, která je opsaná budově. Na vizualizaci opsané elipsy a poloos se můžeme podívat na obr. 4.12 a 4.13. Opět pro získání délky poloos využijeme knihovnu `scikit-image` a v ni funkci `measure region properties`.

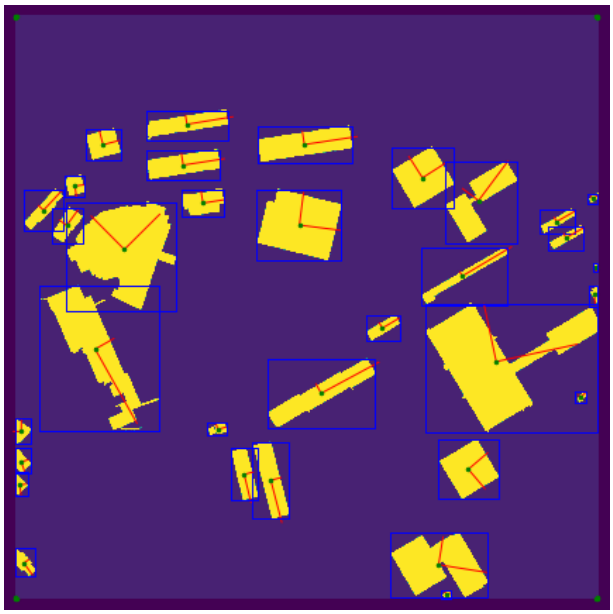


(a) Vizualizace poloos v detekovaných elipsách.

(b) Získání znaků polos na ukázkovém příkladu.

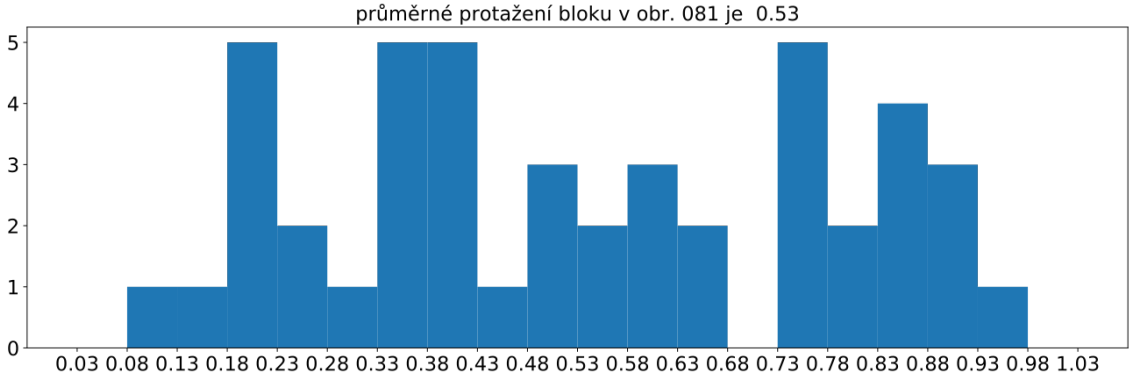
Obrázek 4.12: Vizualizace poloos (obdélník opsaný elipse nesymbolizuje budovu).

Podívejme se na rozložení protažení ve vzorech, viz obr. 4.14-4.16. Na celkové rozložení protažení budov můžeme pozorovat, že rozdělení přibližně kopíruje Gaussovu křivku, kde průměrné protažení ve vzorech je 0.55, tzn. že vedlejší poloosa je poloviční vzhledem k hlavní poloose (poměr 1:2).

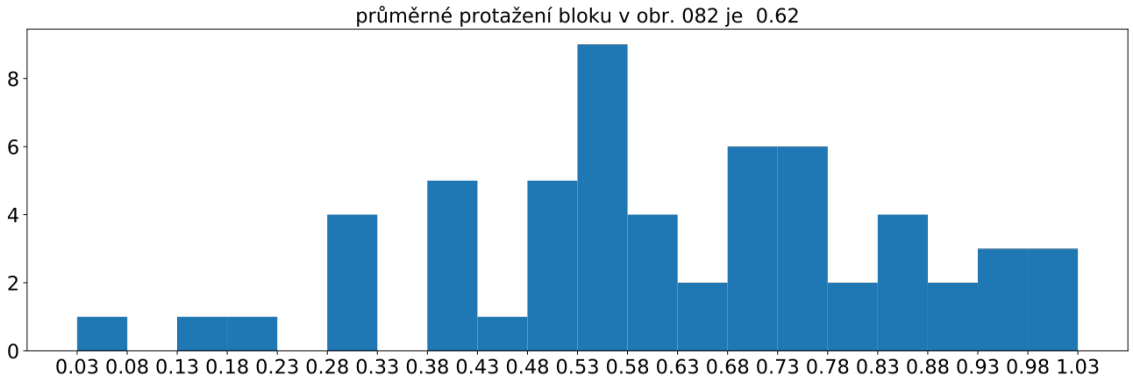


(a) Ukázka vizualizace poloos v detekovaných budovách.

Obrázek 4.13: Vizualizace poloos.

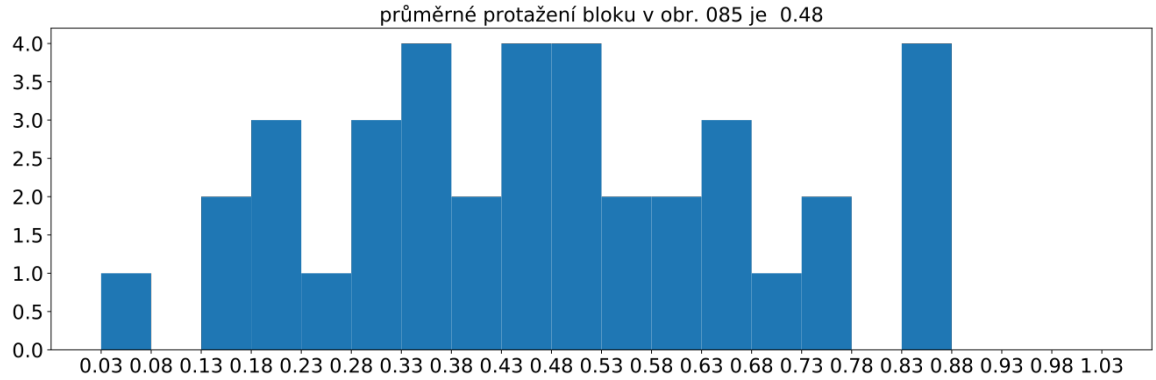


(a)

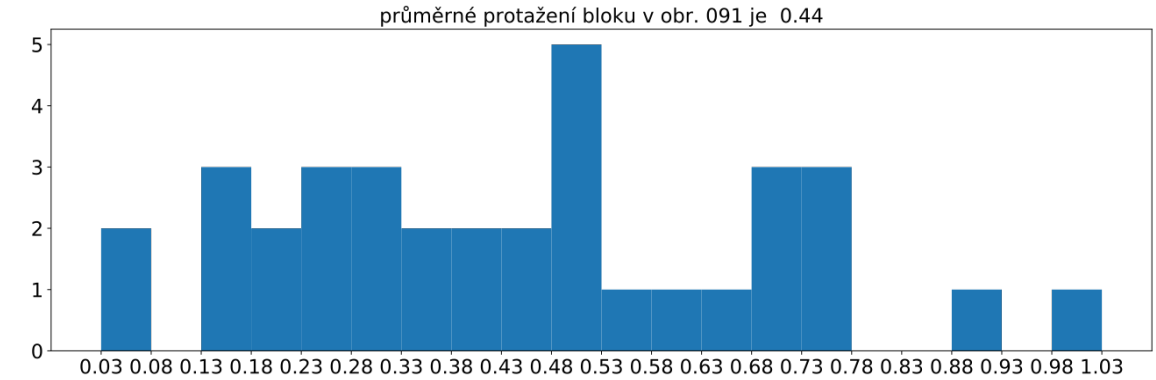


(b)

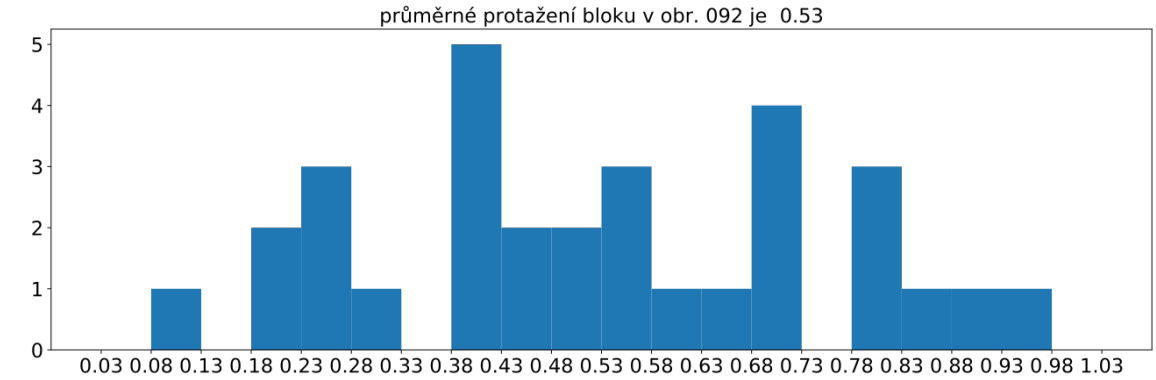
Obrázek 4.14: Rozložení poloos pro vzory v obr. 81-82.



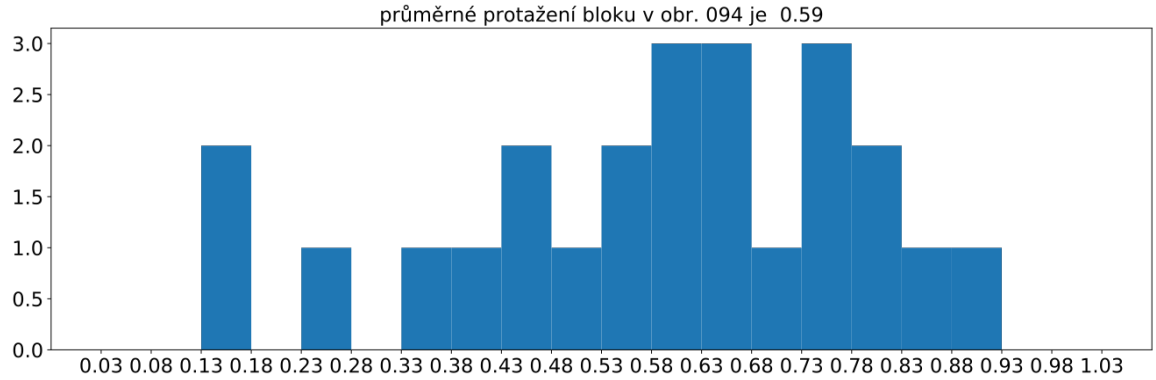
(a)



(b)

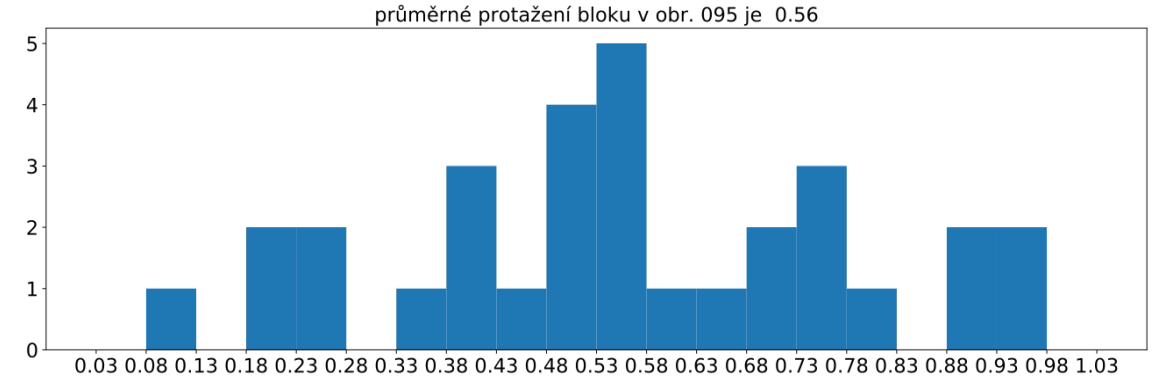


(c)

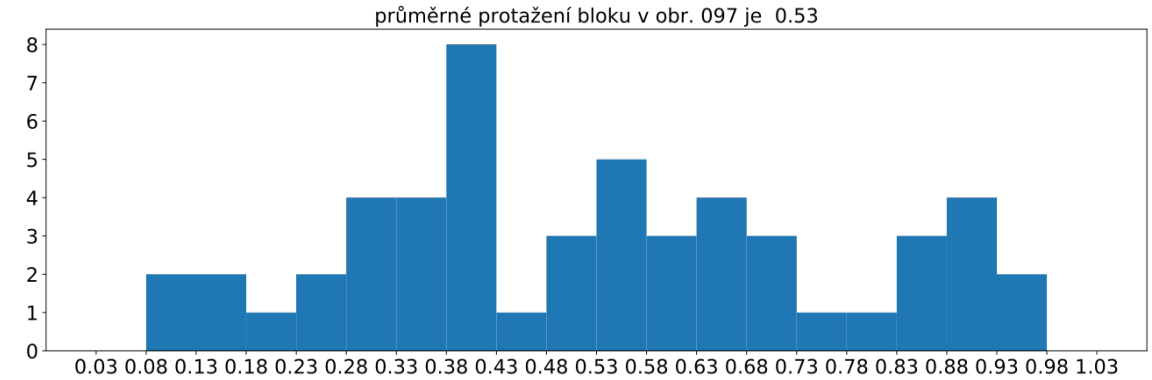


(d)

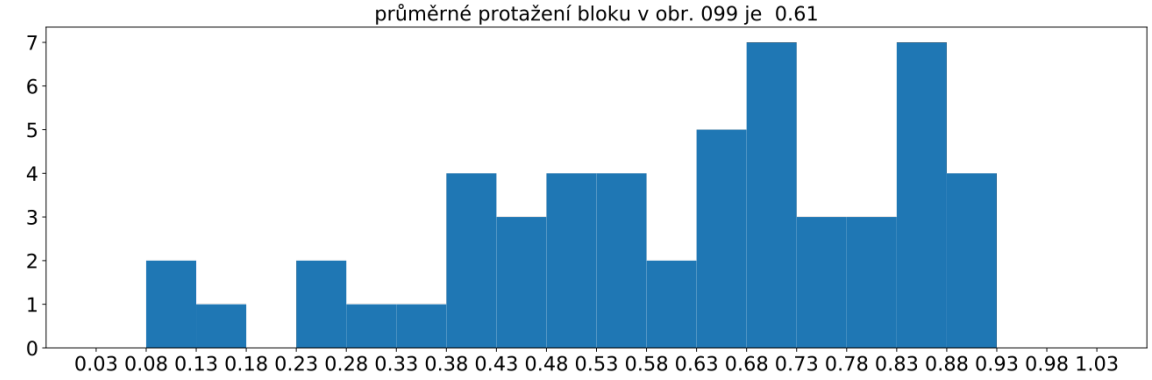
Obrázek 4.15: Rozložení poloos pro vzory v obr. 55-94.



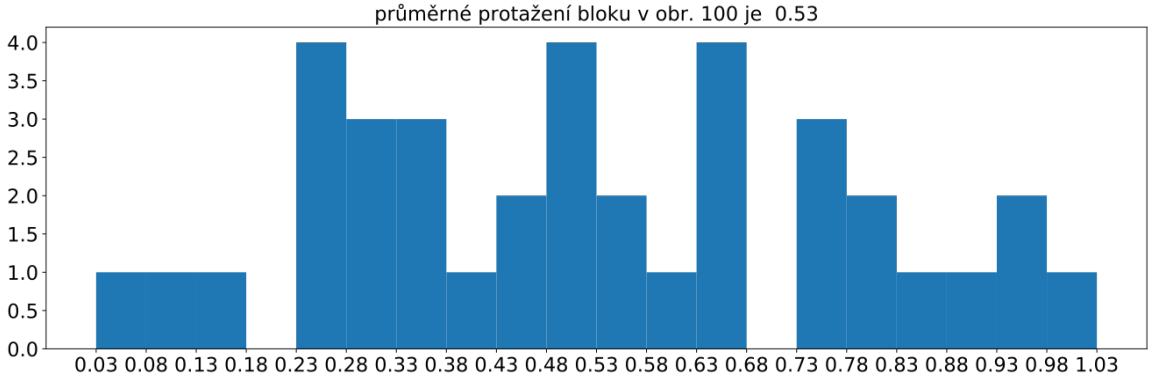
(a)



(b)



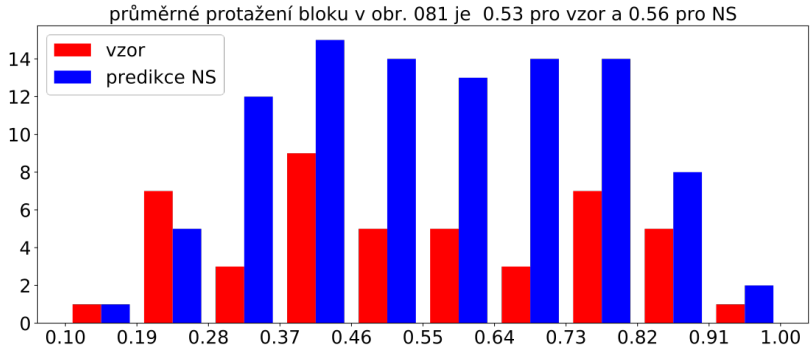
(c)



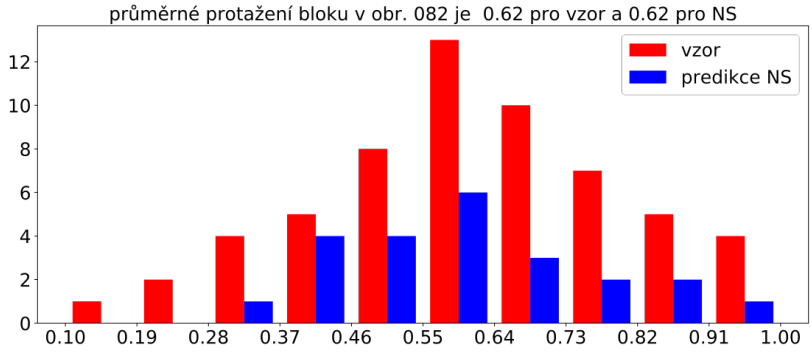
(d)

Obrázek 4.16: Rozložení poloos pro vzory v obr. 95-100.

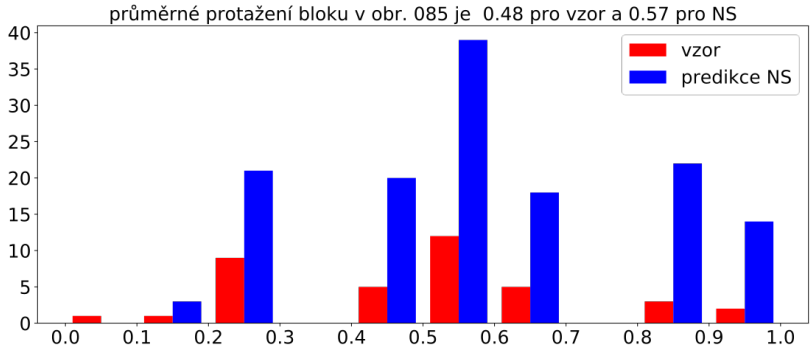
Nyní porovnejme protažení budov ve vzorových a v predikovaných obrázcích, viz obr. 4.17-4.19. Vidíme, že protažení budov má v predikovaných a vzorových obrázcích podobné rozložení. V predikovaných obrázcích je vidět vyšší četnost, ale to je způsobeno tím, že došlo k rozpadu větších bloků na menší, jak jsme již zmínili výše, viz sekce 4.3. Na druhou stranu, se nám potvrzuje hypotéza, že ke zkreslení, parametru délky bloku pro predikované obrázky, došlo právě rozmělněním bloků a nikoliv primárně špatnou detekcí budov, kdy např. silnice byla detekována jako budova. Toto tvrzení opíráme o předpoklad, že kdyby zkreslení bylo způsobeno hlavně špatnou predikcí, protažení budov na predikovaných obrázcích, by neodpovídala protažení vzorech.
Výrazné narušení podoby rozdělení můžeme zaznamenat v obr. 4.19a,4.19b a 4.19c, proto můžeme usoudit, že zde došlo k výrazným chybám při detekci a při pohledu na predikované obrázky se nám to potvrzuje, viz obr. 4.1-4.3.
Průměrné protažení v predikovaných obrázcích je $0,58$, tedy skoro stejná jako pro vzory.



(a)

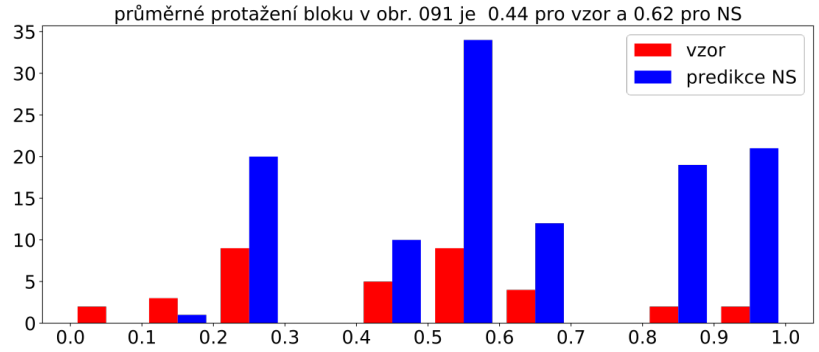


(b)

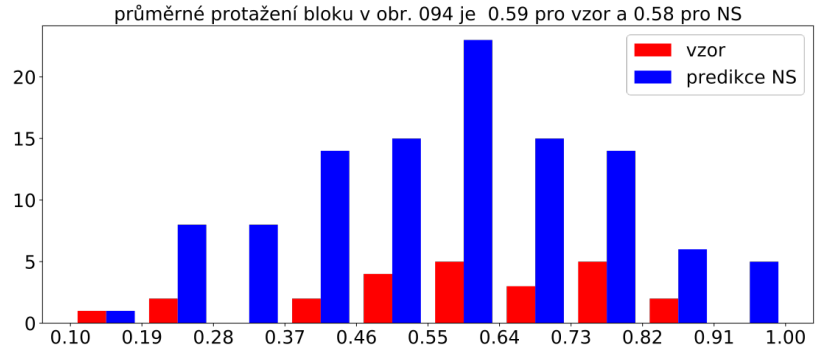


(c)

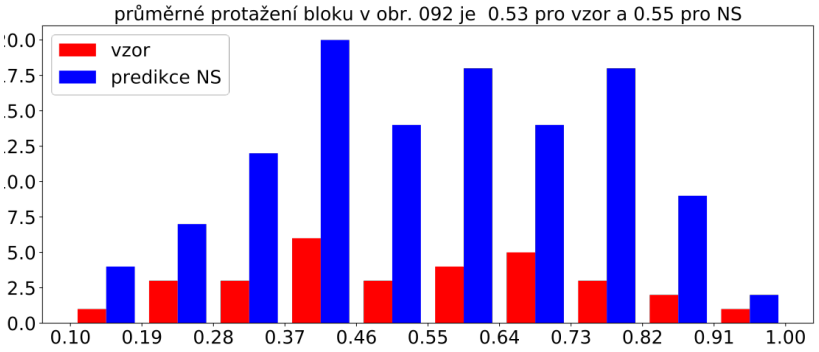
Obrázek 4.17: Rozložení poloos pro predikované obr. 81-85.



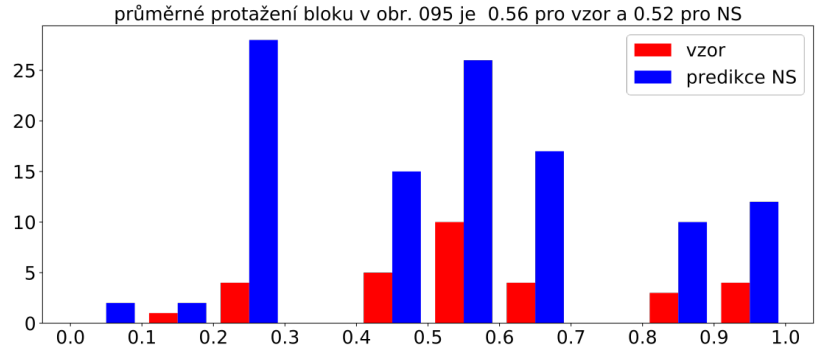
(a)



(b)

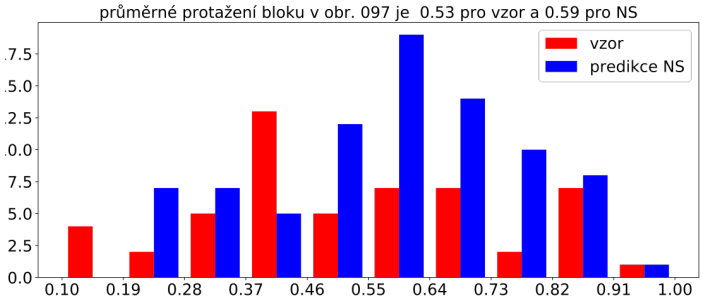


(c)

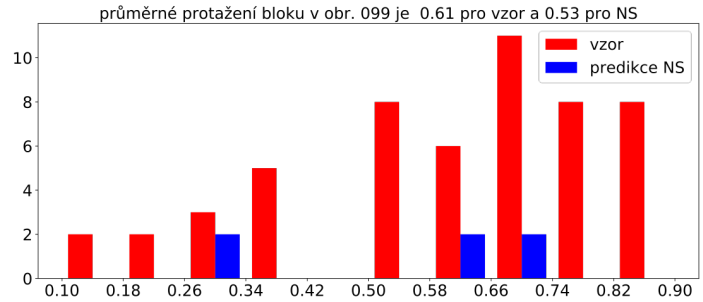


(d)

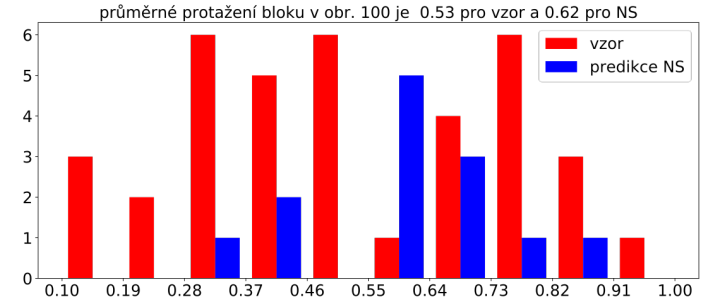
Obrázek 4.18: Rozložení poloos pro predikované obr. 91-95.



(a)



(b)



(c)

Obrázek 4.19: Rozložení poloos pro predikované obr. 97-100.

Podívejme se ještě na korelaci predikce se vzory v jednotlivých obrázcích vzhledem k protažení viz tabulka 4.2. Silnou závislost pozorujeme pouze u třech obrázků, zatímco u zbytku je patrná nezávislost, popřípadě slabá negativní závislost.

obr.	**081**	**082**	**085**	**091**	**092**	**094**	**095**	**097**	**099**	**100**
korelační koef.	0.306	0.846	0.615	0.145	0.738	0.113	0.101	-0.007	-0.166	-0.345

Tabulka 4.2: Korelace mezi vzory a predikovanými obrázky NS pro parametr protažení budov.

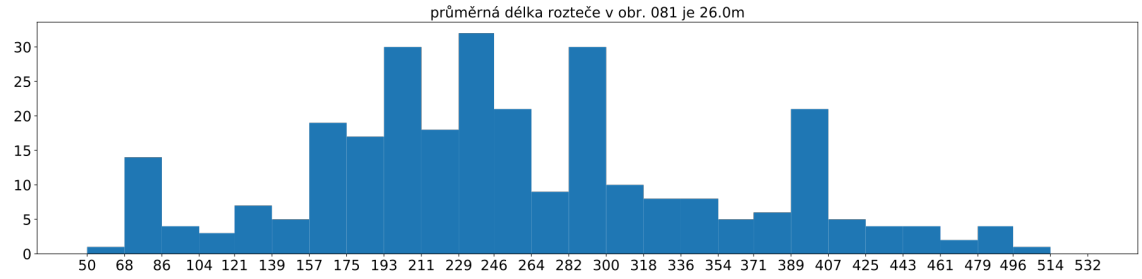
4.5 Vzájemná vzdálenost budov

Vzájemnou vzdálenost budov budeme v dalším textu zkráceně označovat jako rozteč. Rozteč jsme počítali zvlášť pro každou budovu. Vybrali jsme si 20 bodů na každé kontuře (konturou ozn. detekovanou hranici budovy) a z každého bodu jsme vedli čtyři paprsky, dva horizontálně a dva vertikálně. Tímto způsobem jsme zjišťovali, v jaké vzdálenosti paprsek narazí do další budovy. Poté jsme vybraly paprsek s nejkratší nenulovou délkou a ten jsme zvolili za rozteč v tomto bodě. Zároveň jsme zavedli některé omezující podmínky, např. dolní a horní hranici pro rozteč. Pro dolní hranici jsme vyšli ze zákonných norem [7], kdy minimální šířka silnice je $7,5$ m a chodníku 2m. Samozřejmě jsme si vědomi, že některé obrázky jsou z historického centra města, a proto nemusí reálná šířka ulic odpovídat normám, proto jsme zvolili minimální délku rozteče 5m. Horní hranici nastavujeme zvláště proto, jelikož v testovacích obrázcích je např. i řeka nebo lesík a nechceme, aby naše statistika roztečí byla zatížená i těmito vzdálenostmi, jelikož se snažíme změřit šířku ulice. Z tohoto důvodu nechceme zaznamenávat např. vzdálenost budov přes řeku. Samozřejmě na obrázcích jsou i prvky jako je náměstí nebo kruhová křižovatka, ale v tomto případě rozteč chceme zaznamenat. Proto jsme horní hranici nastavili na desetinu rozměru obrázku, tzn. 50m (přibližný poloměr pro větší kruhový objezd).
Tento algoritmus sice není robustní, ale pro naši úlohu je dostatečný. Dalšími možnostmi, jak měřit rozteč mezi budovami je vyslání paprsků ve více směrech, nebo směr paprsku určit jako normálu k hranici budovy v daném místě. Vizualizaci výsledků algoritmu pro zjištění roztečí vidíme na obr. 4.20.
Předpokládáme, že rozdělení roztečí bude četnější u nižších vzdálenosti než u těch větších.

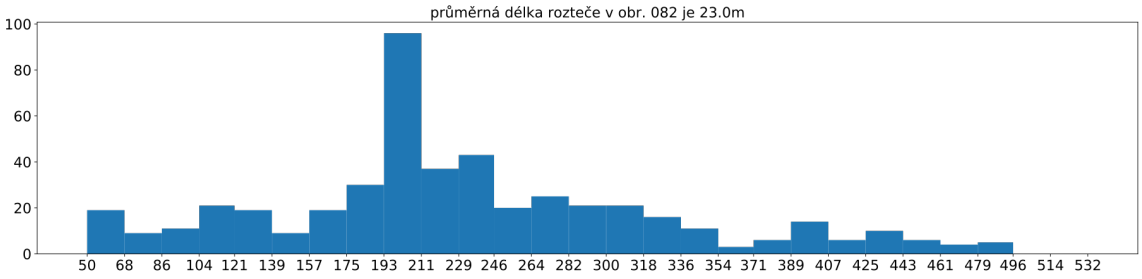
Obrázek 4.20: Vizualizace paprsků v algoritmu pro výpočet roztečí.

Distribuci velikosti roztečí ve vzorech můžeme pozorovat v obr. 4.21-4.22.

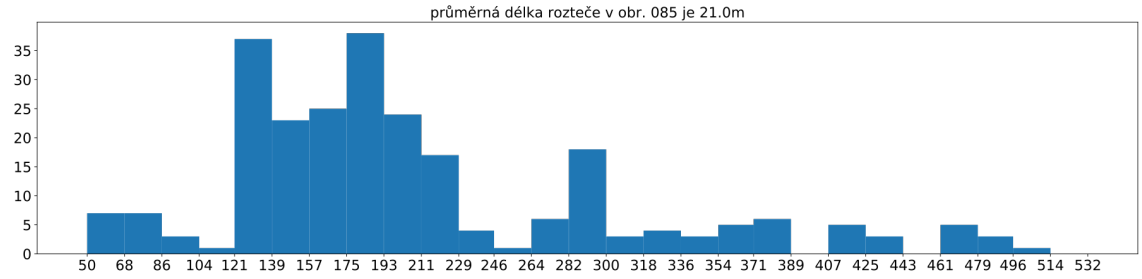
Rozdělení roztečí ve vzorech má pro různé obrázky odlišný charakter. V některých případech je podobné log-normálnímu rozdělení, jindy má tvar Gaussovy křivky a nebo má dva vrcholy, které sami o sobě mají Gaussovo rozdělení.
Průměrná rozteč ve vzorech je 21 m, toto odpovídá dobře realitě, protože normová vzdálenost pro silnici s chodníky po obou stranách je $11,5$m, jelikož typ našich obrázků odpovídá městské zástavbě, kde právě takovýto typ ulice lze předpokládat, zároveň je zde i kruhový objezd nebo náměstí, kde lze předpokládat větší rozteč. Prozkoumejme blíže obrázky, viz obr. 4.1-4.3, které mají v histogramech vidět dva vrcholy. Hypotéza je, že se bude jednat právě o satelitní snímky s objezdy nebo náměstím apod.
Na obr. 92, lze pozorovat větší zelené plochy (zahrady atd.) a vidíme zde i nádvoří. Na obr. 91 je situace stejná. V obr. 95, 97, 99 a 100, kde pozorujeme sestupný trend, je vidět hustá městská zástavba většinou jen s jedním středně velkým prvkem typu náměstí, velká křižovatka, resp. malý park, což dobře odpovídá předpokladu.
Na obr. 4.23-4.25, můžeme pozorovat porovnání rozdělení roztečí mezi vzory a výsledky predikce NS. U predikovaných obrázků vidíme log-normální trend, který se výrazně liší od vzorů. Průměrná délka rozteče u těchto obrázků je 14m.



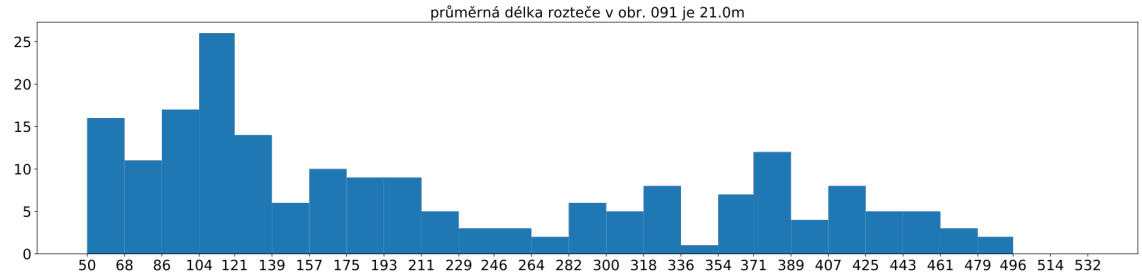
(a)



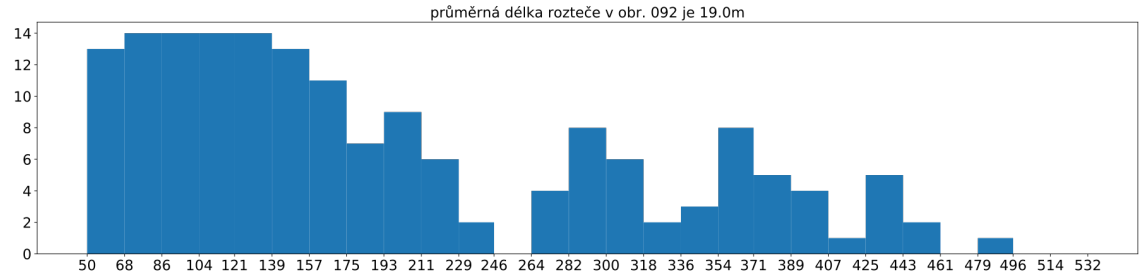
(b)



(c)

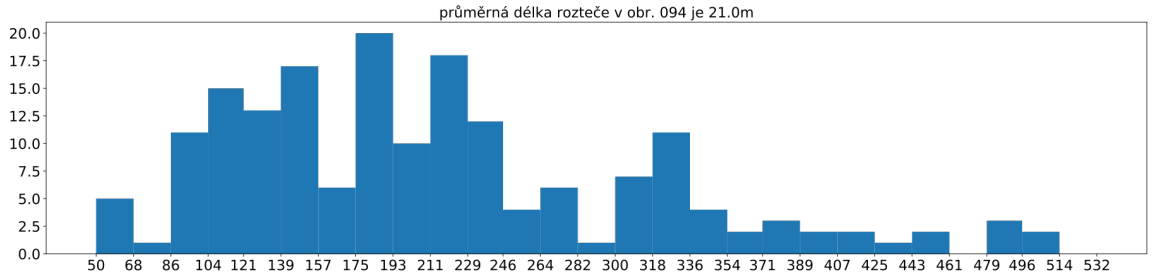


(d)

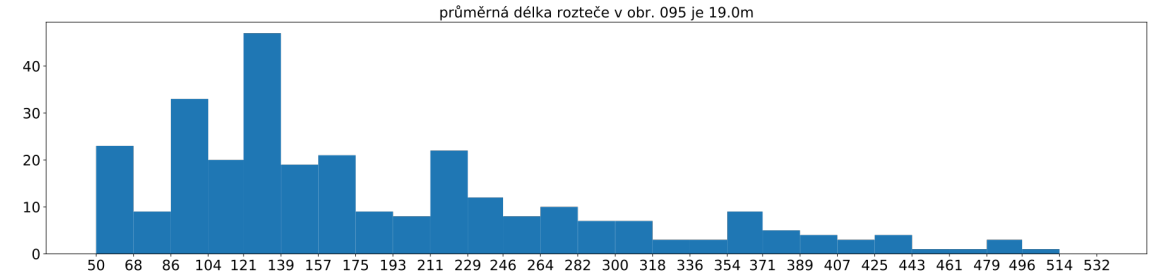


(e)

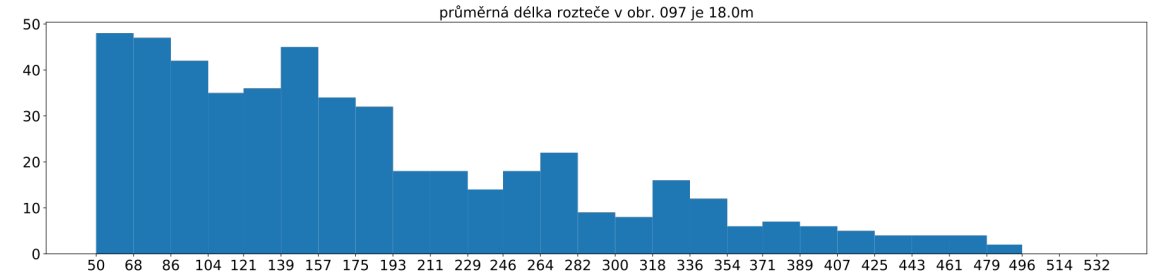
Obrázek 4.21: Histogram roztečí mezi budovami pro vzory obr. 81-92. Na ose x je délka rozteče v decimetrech a na ose y je četnost.



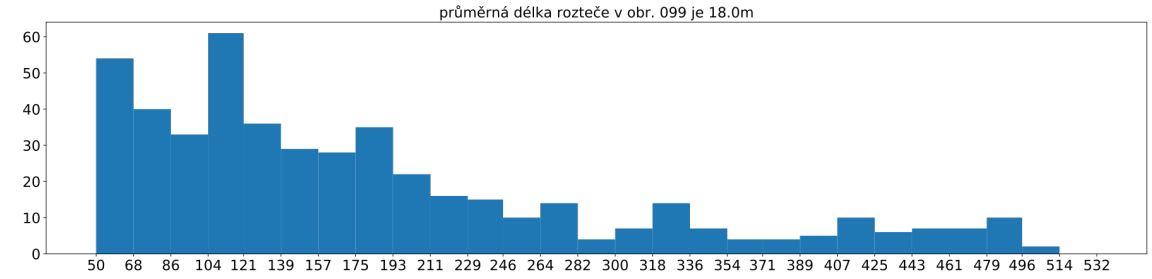
(a)



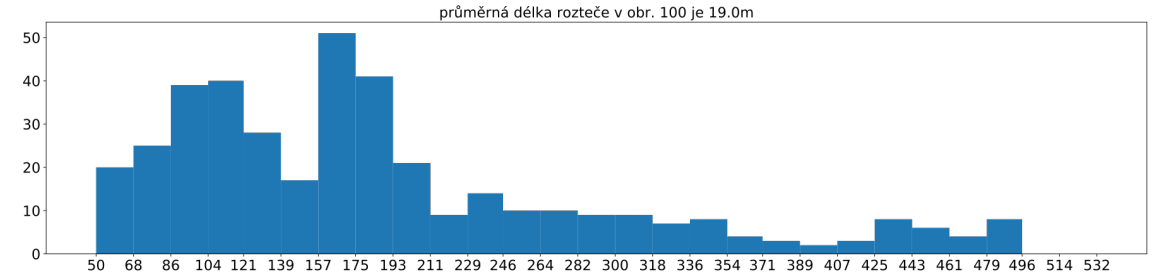
(b)



(c)

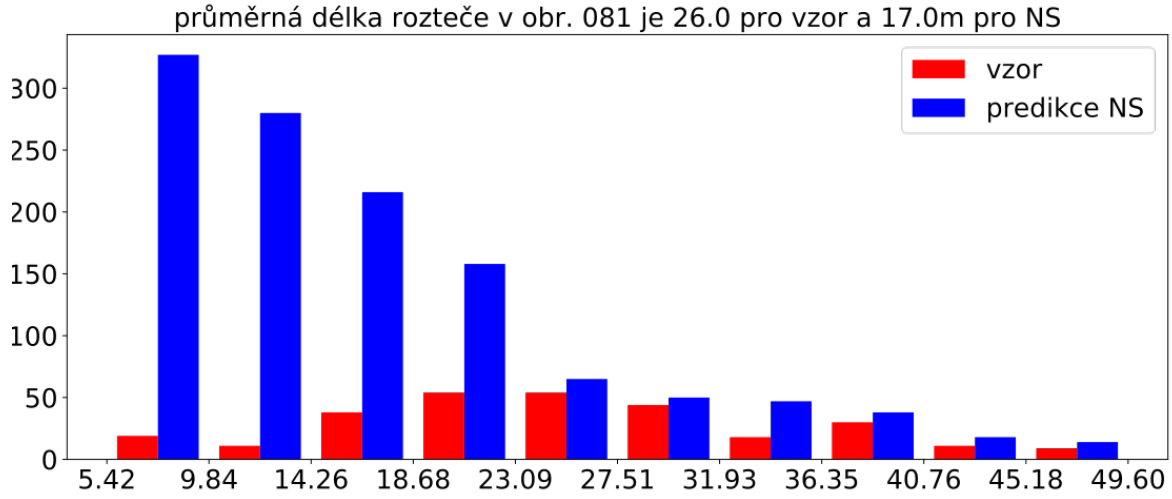


(d)

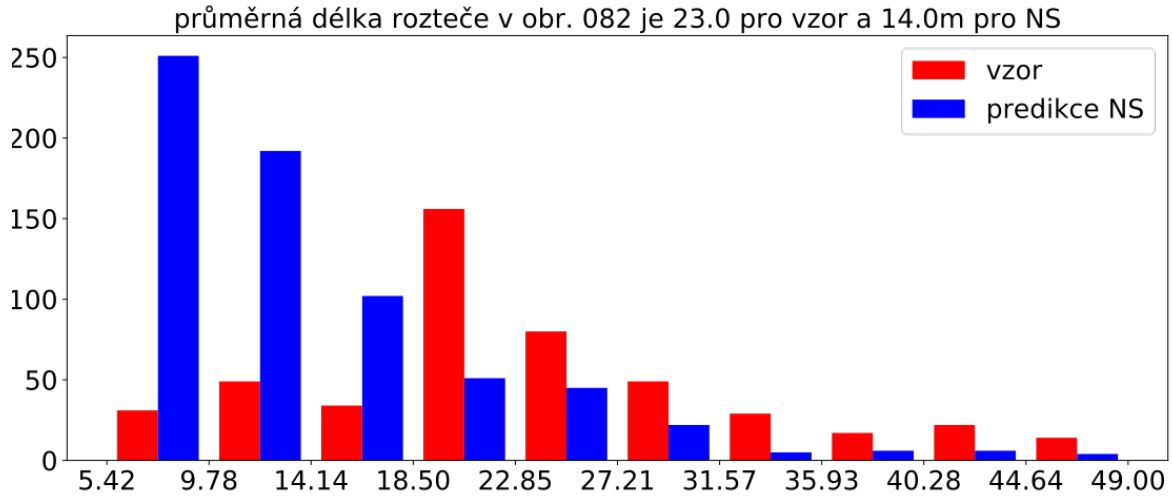


(e)

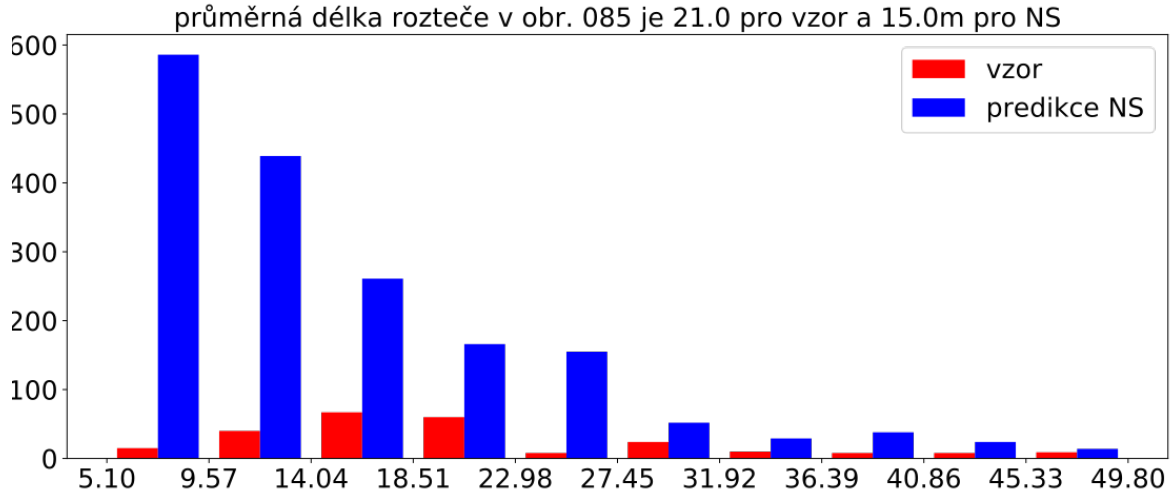
Obrázek 4.22: Histogram roztečí mezi budovami pro vzory obr. 94-100. Na ose x je délka rozteče v decimetrech a na ose y je četnost.



(a)

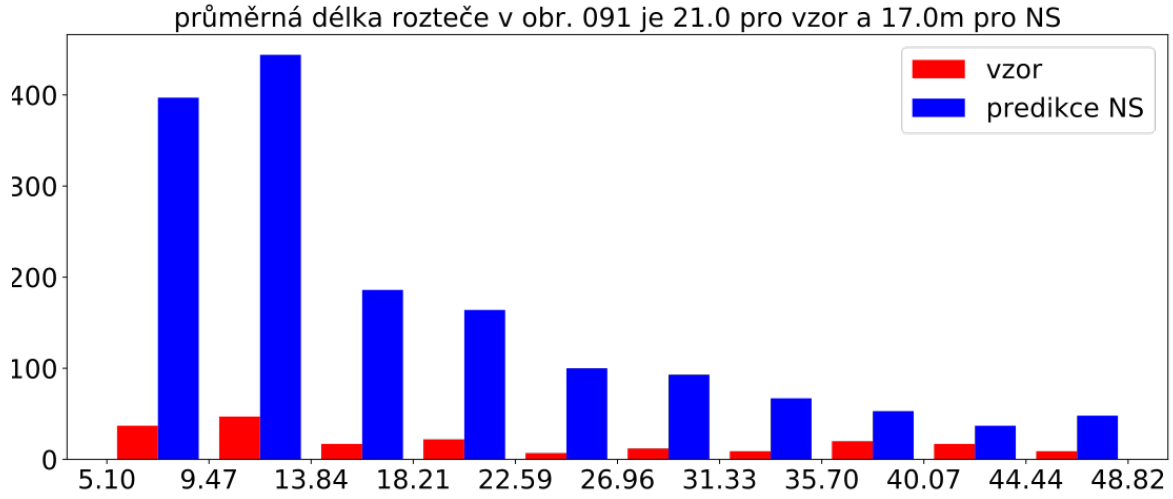


(b)

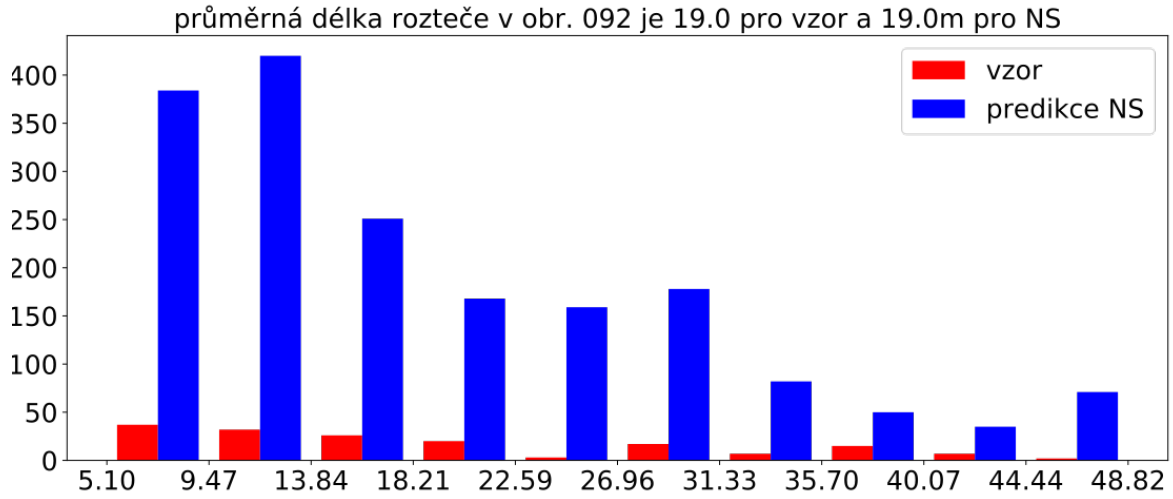


(c)

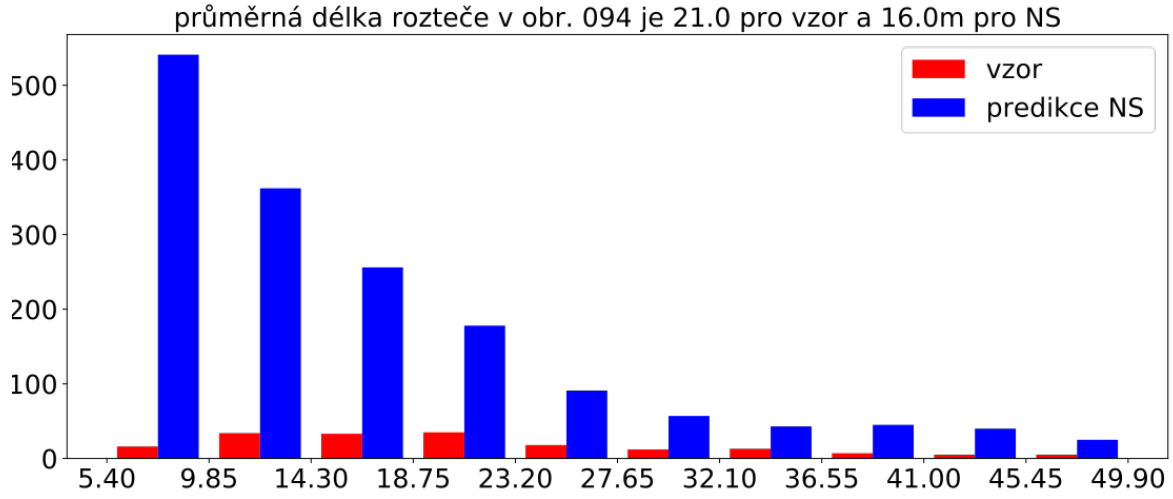
Obrázek 4.23: Histogram roztečí mezi budovami pro predikované obr. 81-85. Na ose x je délka rozteče v metrech a na ose y je četnost.



(a)

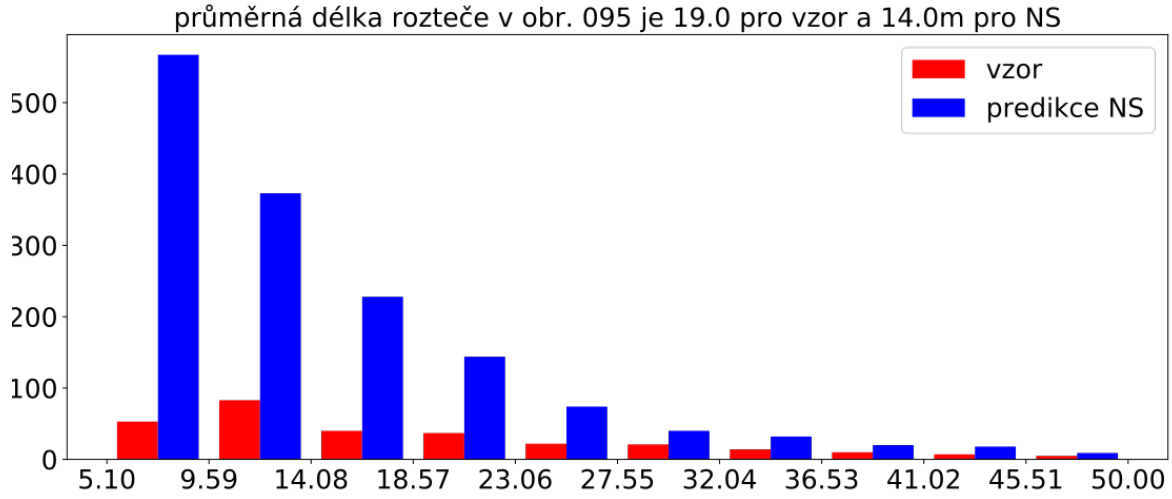


(b)

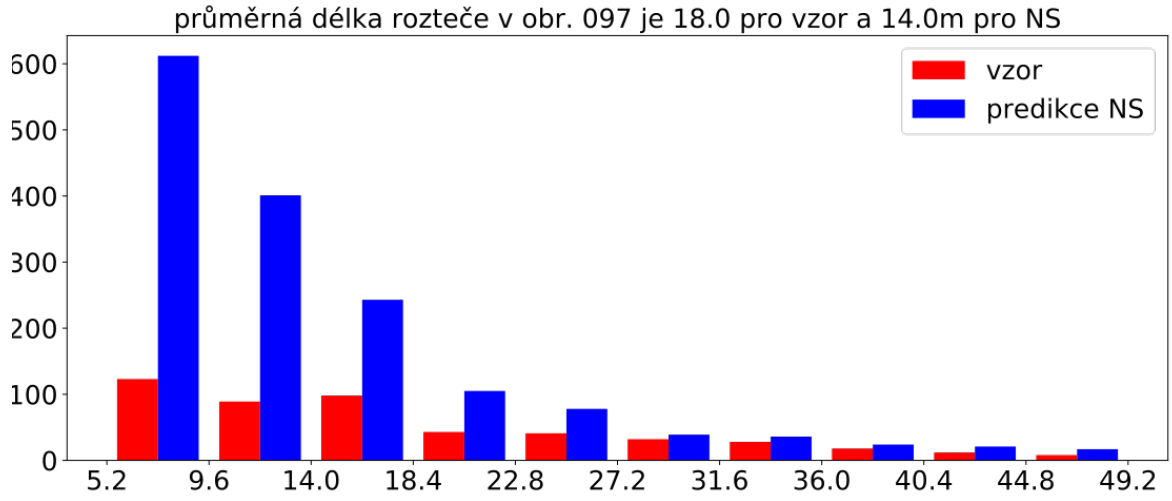


(c)

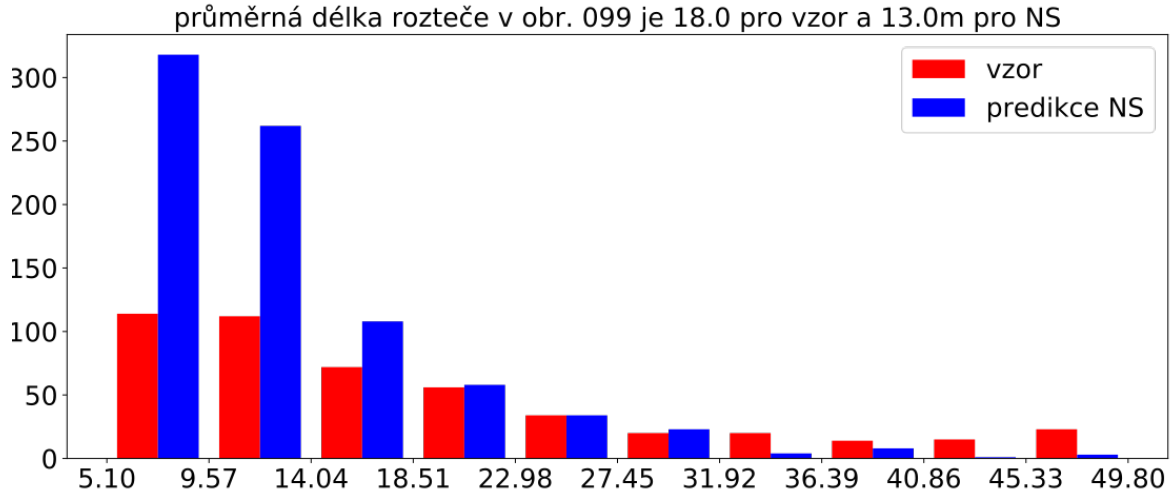
Obrázek 4.24: Histogram roztečí mezi budovami pro predikované obr. 91-94. Na ose x je délka rozteče v metrech a na ose y je četnost.



(a)

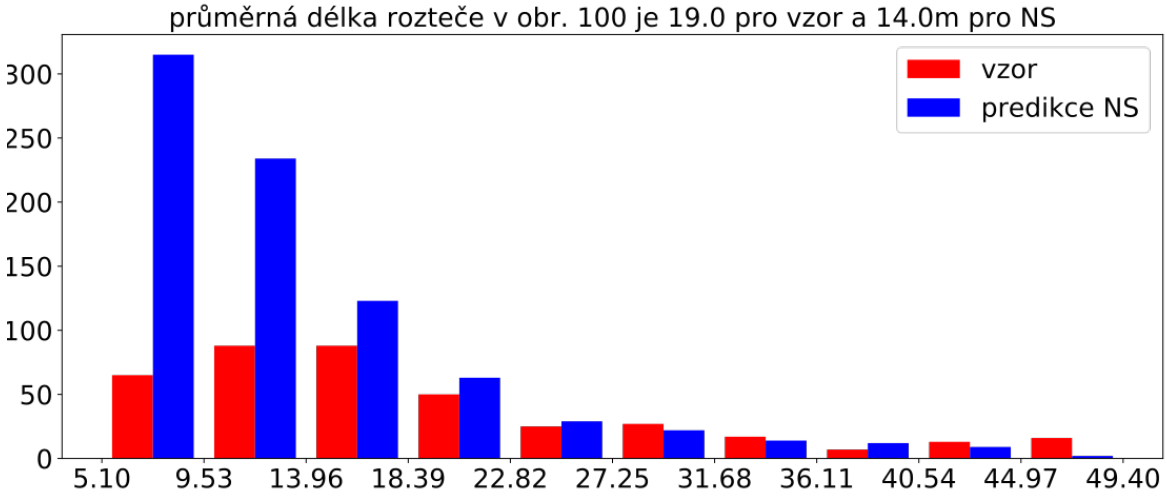


(b)



(c)

Obrázek 4.25: Histogram roztečí mezi budovami pro predikované obr. 95-99. Na ose x je délka rozteče v metrech a na ose y je četnost.



(a)

Obrázek 4.26: Rozložení roztečí mezi budovami pro predikované obr.100. Na ose x je délka rozteče v metrech a na ose y je četnost.

Podívejme se ještě na korelaci predikce se vzory v jednotlivých obr. vzhledem k roztečím v tabulce 4.3.

obr.	**081**	**082**	**085**	**091**	**092**	**094**	**095**	**097**	**099**	**100**
korelační koef.	-0.048	0.025	0.328	0.910	0.870	0.535	0.840	0.938	0.968	0.796

Tabulka 4.3: Korelace vzorů a predikovaných obrázků vzhledem k naměřeným roztečím.

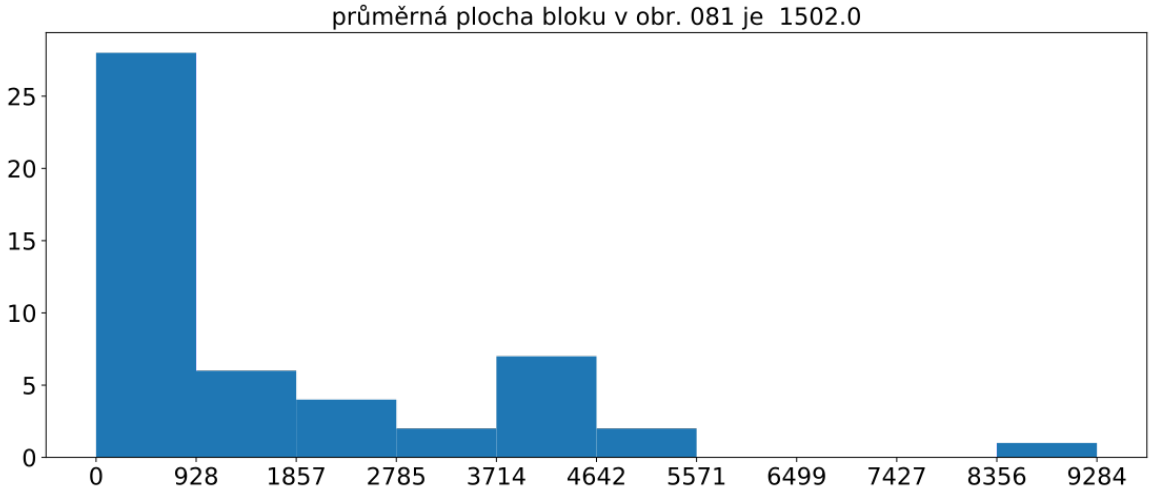
Vidíme, že silná korelace je u šesti obrázků, což jsou většinou ty s hustou zástavbou.

4.6 Plocha bloku

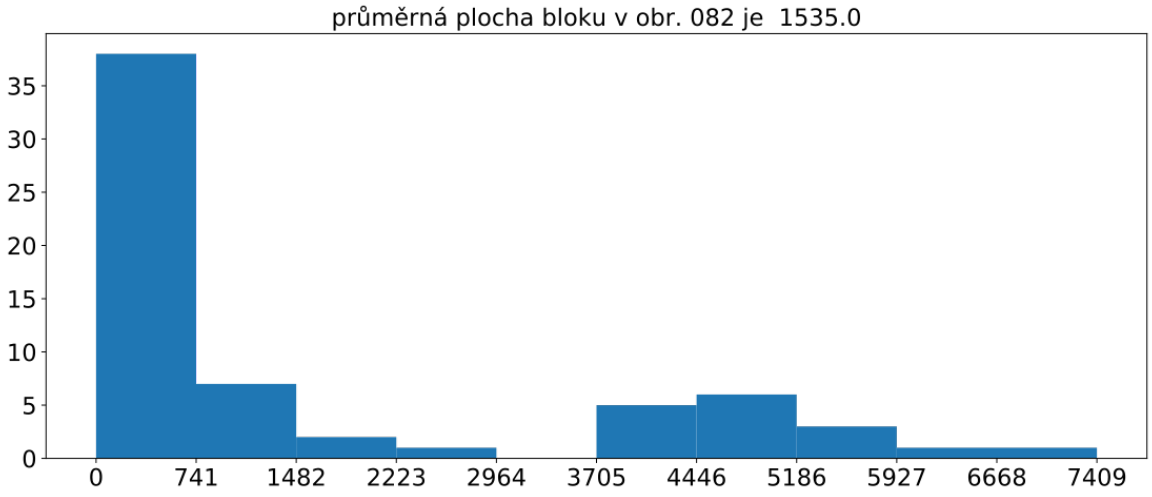
K zjištění, jakou plochu blok zabírá, jsme opět použili knihovnu `scikit-image`. Tento parametr je důležitý, jelikož na základě jeho hodnoty, kterou implementujeme do programu pro procedurální generování města, lze ovlivňovat typ pozemku, tj. zda bude stavební nebo bude ponechán jako zelená plocha. Podívejme se na rozdělení plochy bloků na obr. 4.27-4.30.

Průměrná plocha bloku budov ve vzorech je 2101m^2. Z grafů lze vidět, že tento parametr bude mít log-normální rozdělení.

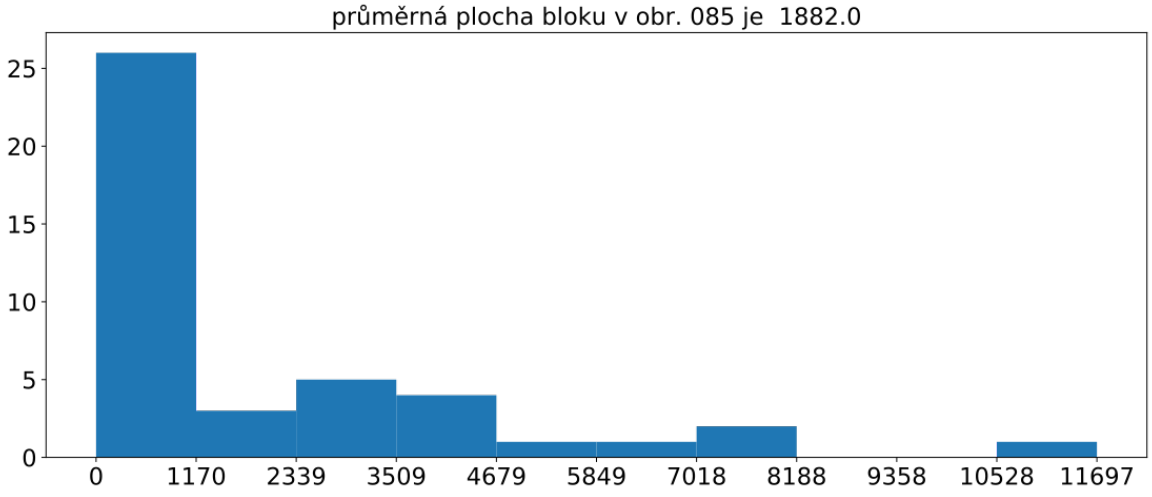
Na obr. 4.31-4.33, můžeme pozorovat porovnání rozdělení plochy mezi vzory a predikovanými. Vidíme, že se rozdělení velice liší, což nejspíš způsobeno hlavně tím, že se velké bloky při detekci rozpadly na menší, jak jsme již zmínili v sekci 4.3.



(a)

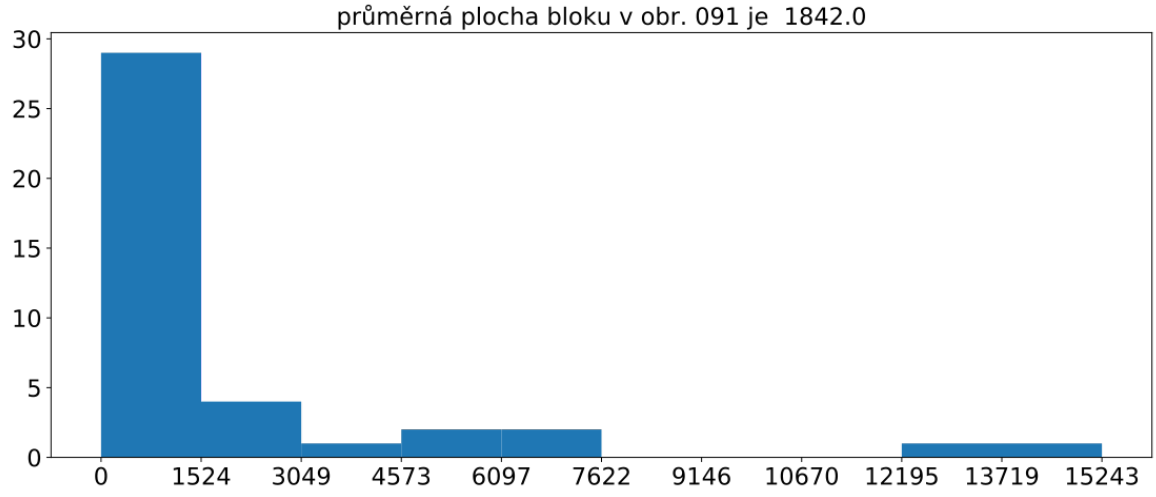


(b)

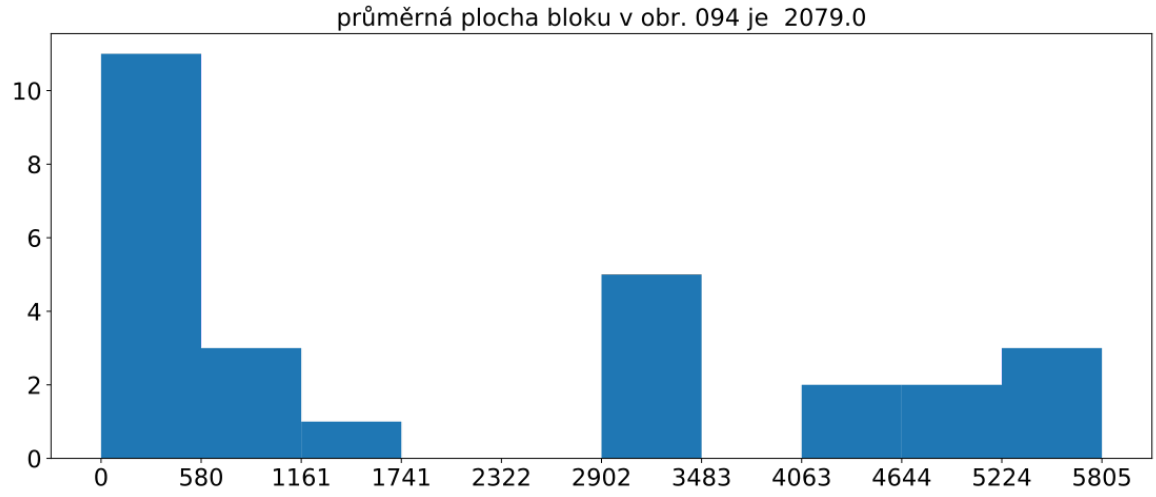


(c)

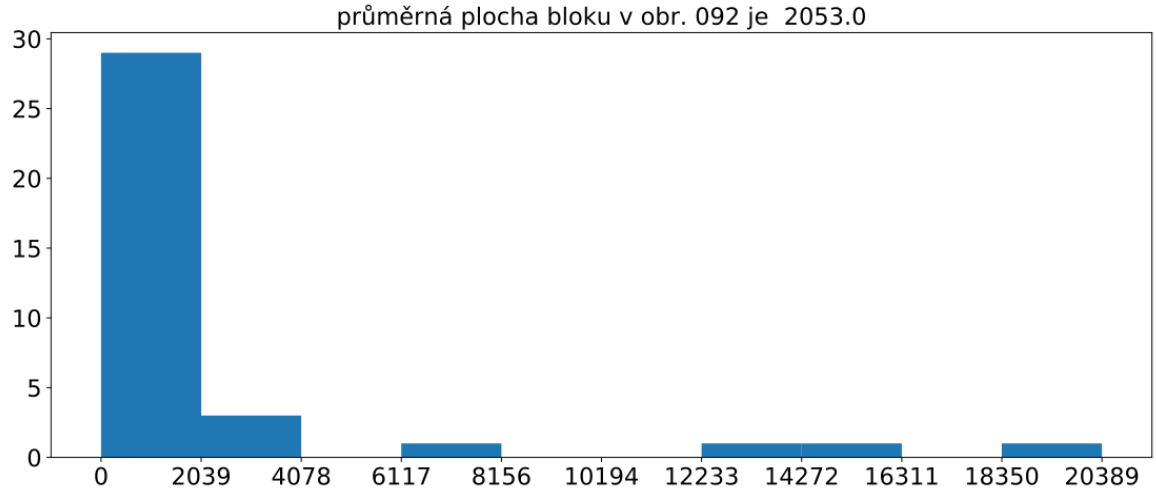
Obrázek 4.27: Histogram plochy bloku budov pro vzory obr. 81-85. Na ose x jsou m^2.



(a)

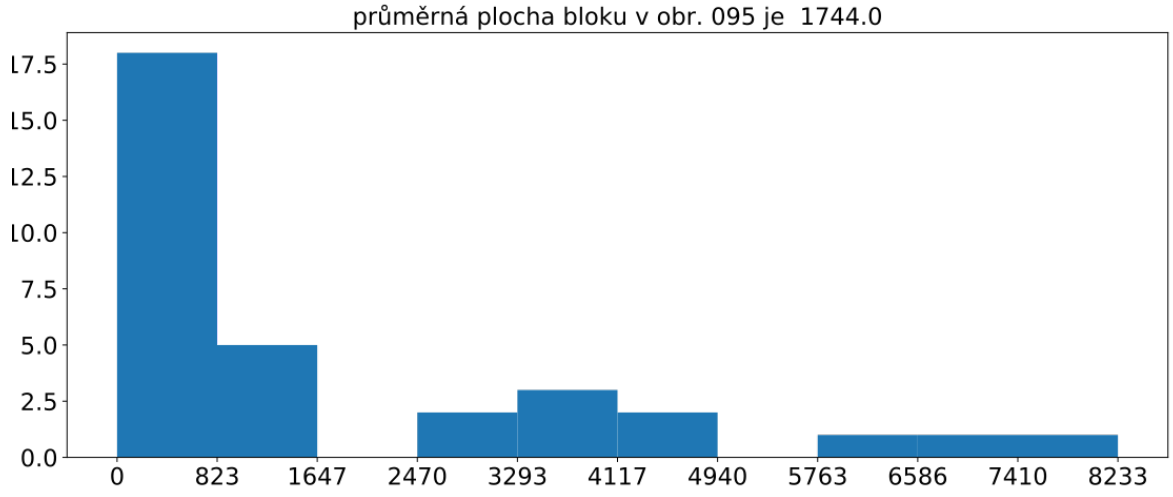


(b)

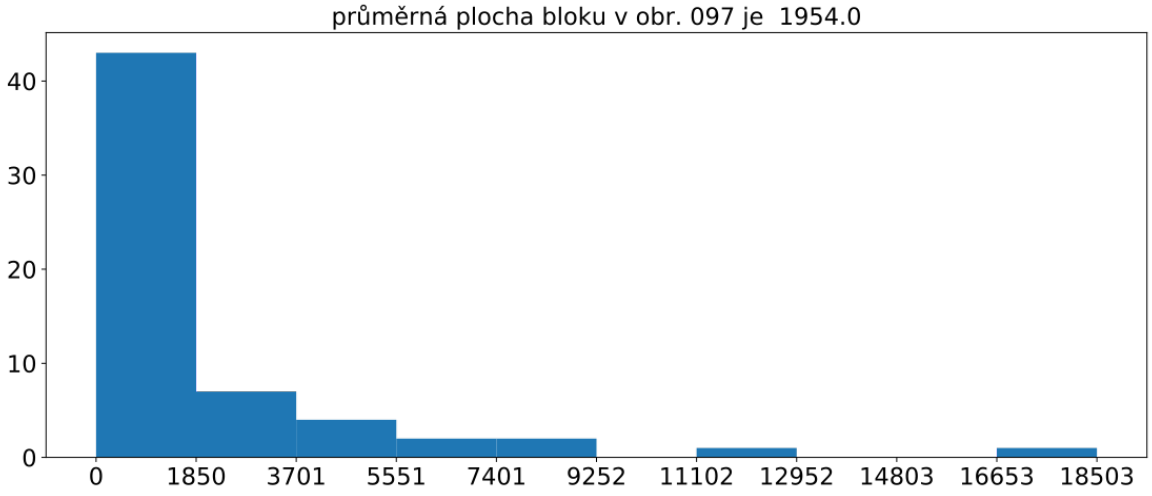


(c)

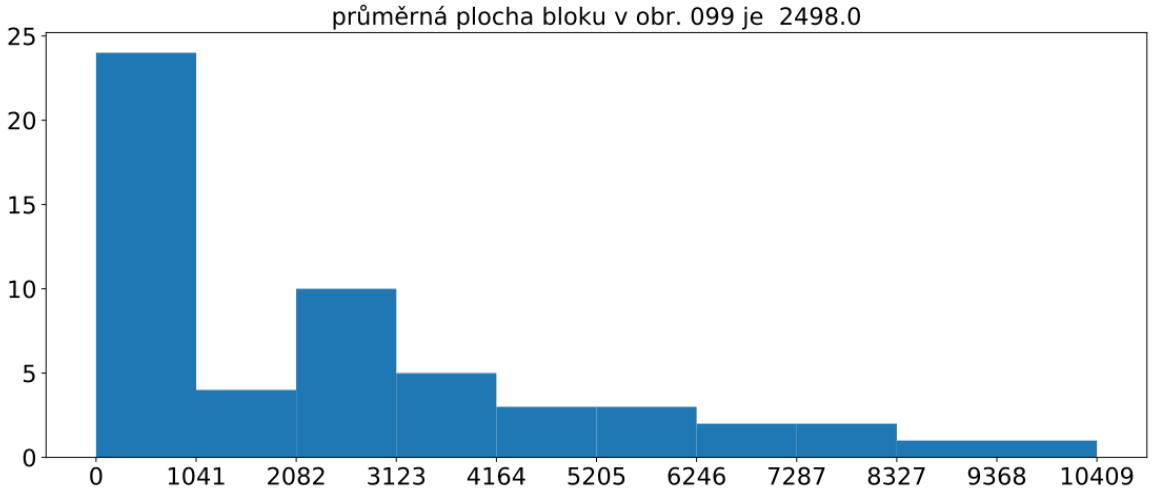
Obrázek 4.28: Histogram plochy bloku budov pro vzory obr. 91-94. Na ose x jsou m^2.



(a)

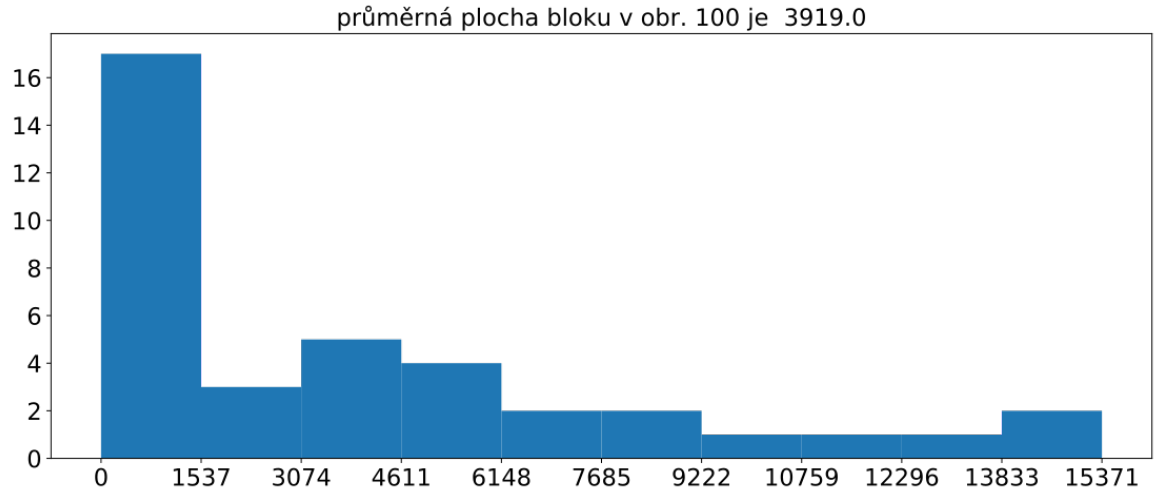


(b)



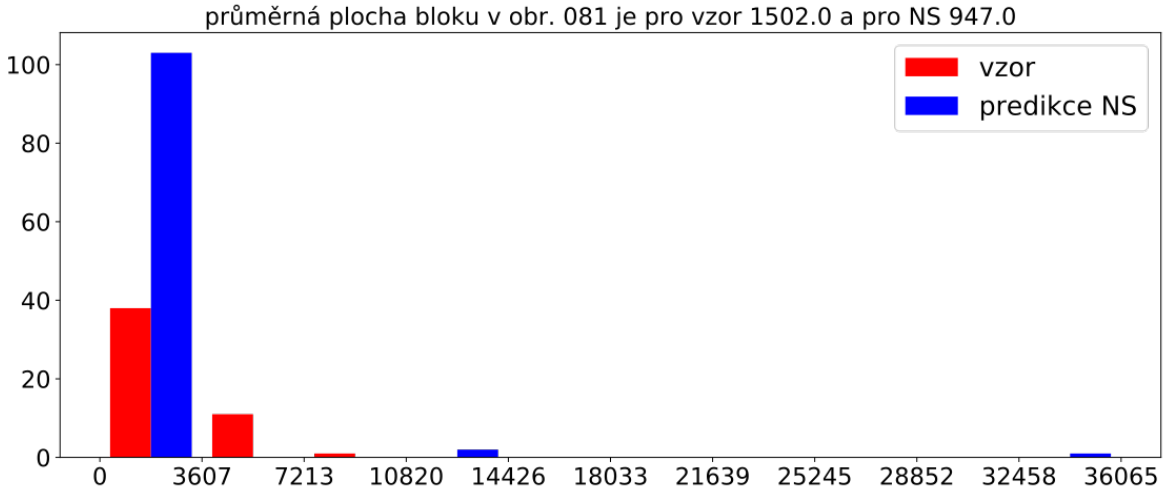
(c)

Obrázek 4.29: Histogram plochy bloku budov pro vzory obr. 95-99. Na ose x jsou m^2.

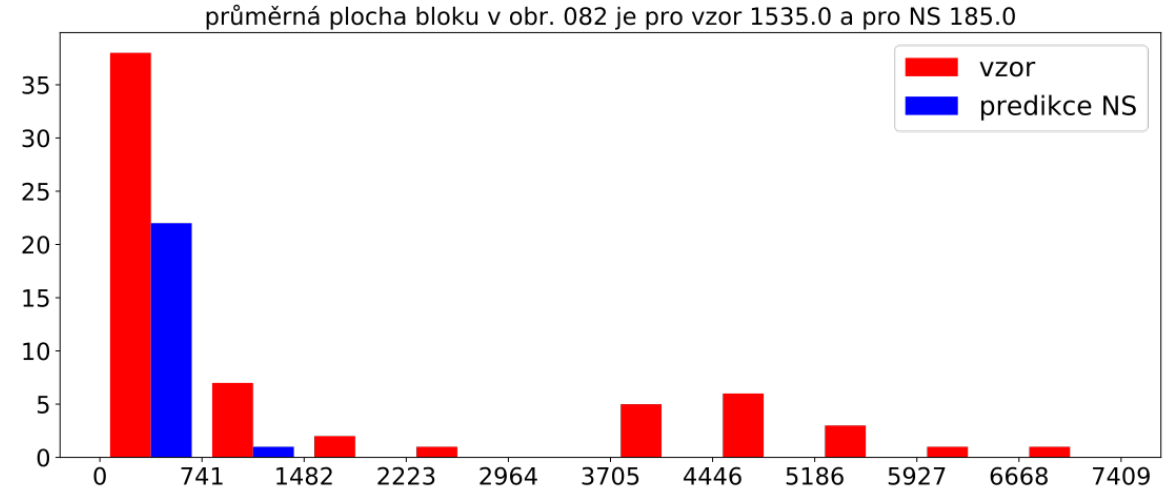


(a)

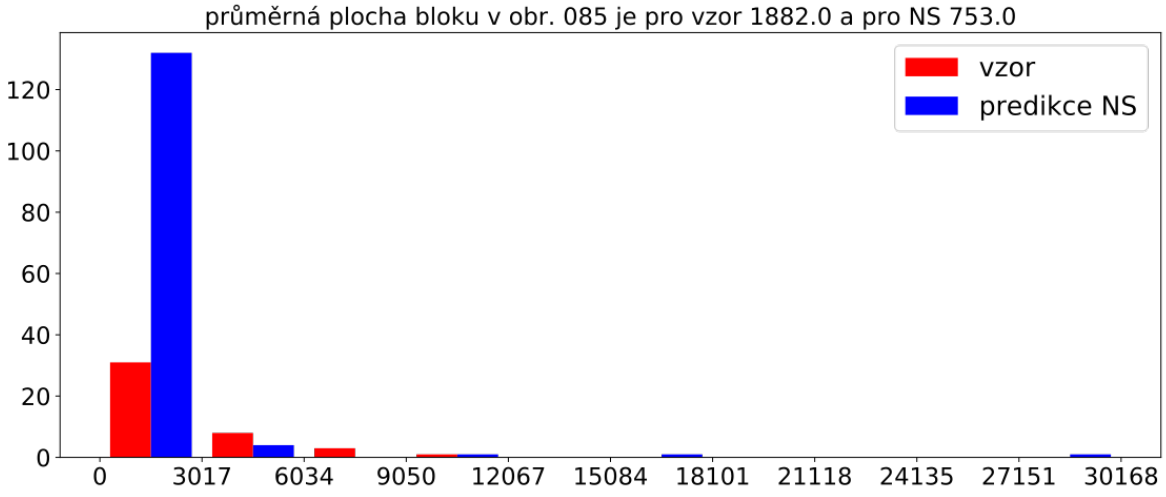
Obrázek 4.30: Histogram plochy bloku budov pro vzory obr. 95-99. Na ose x jsou m^2.



(a)

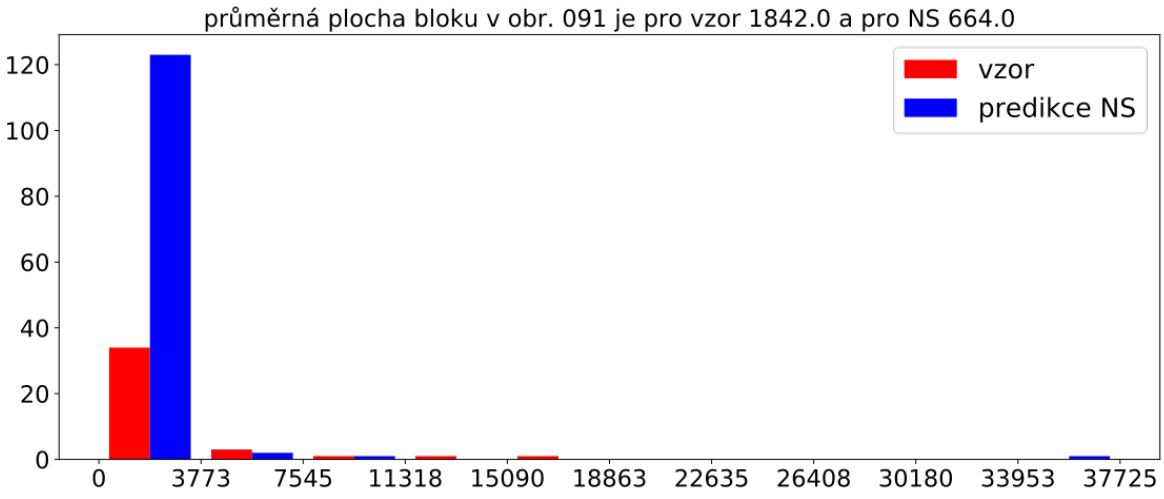


(b)

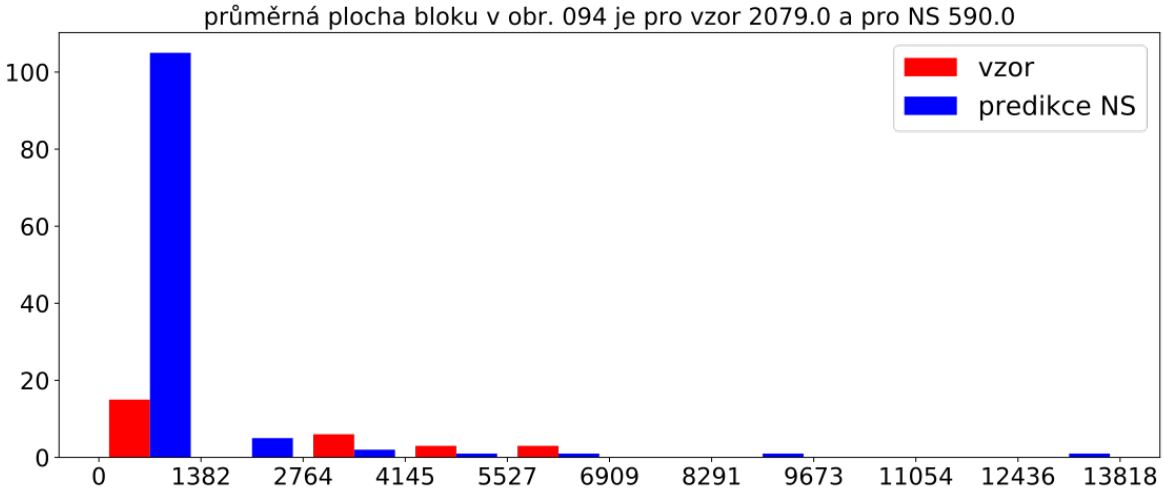


(c)

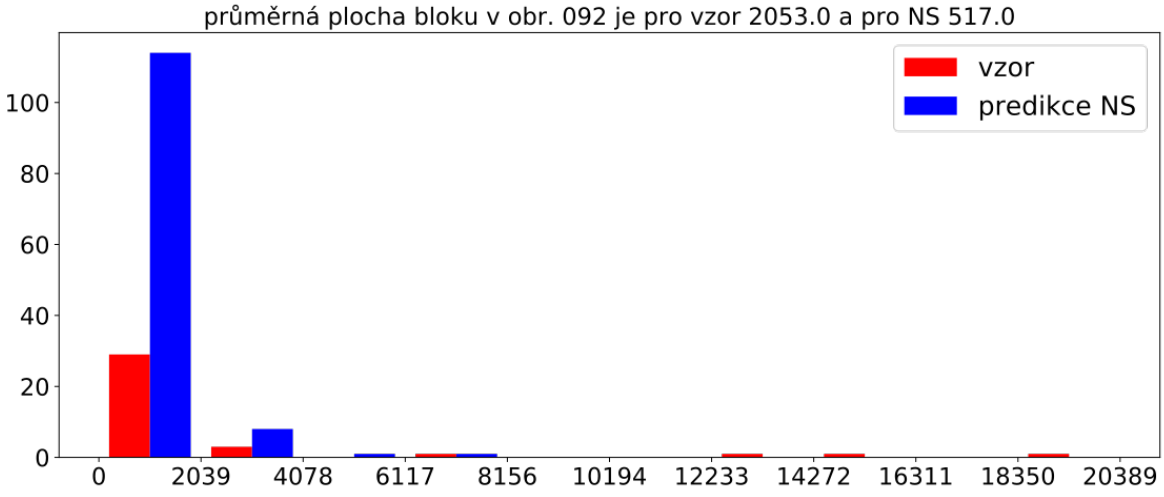
Obrázek 4.31: histogram plochy bloku budov pro predikované obr. 81-85. Na ose x jsou m^2.



(a)

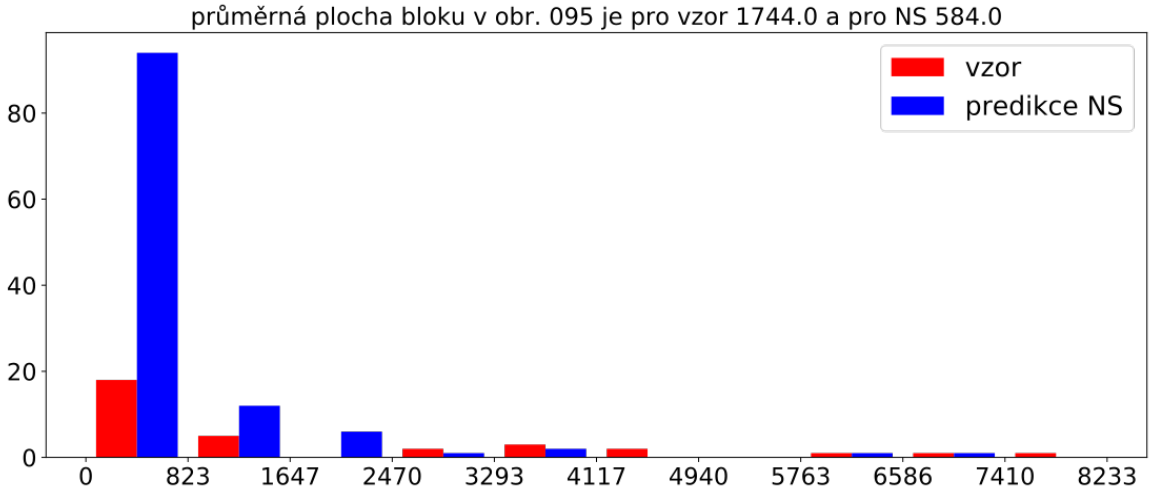


(b)

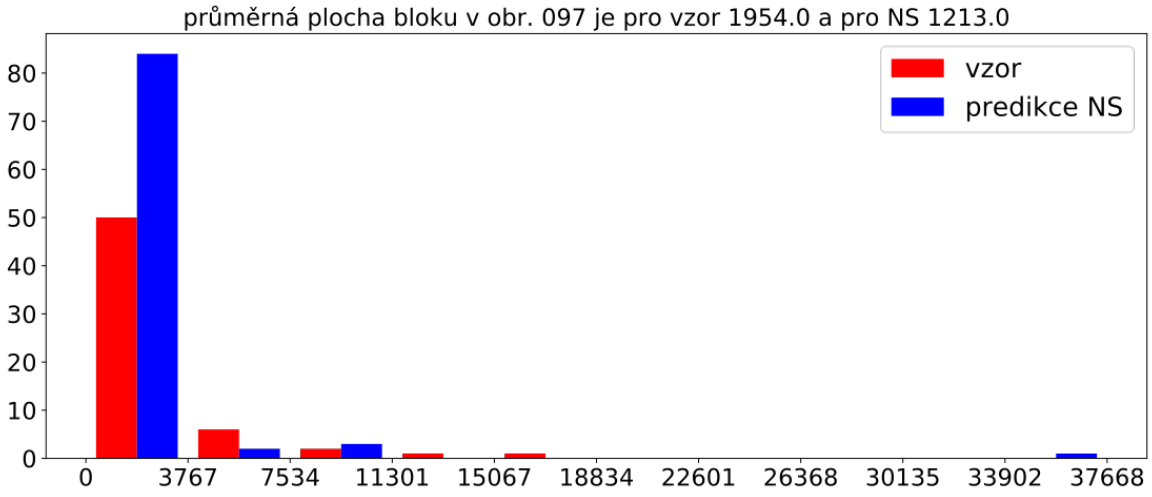


(c)

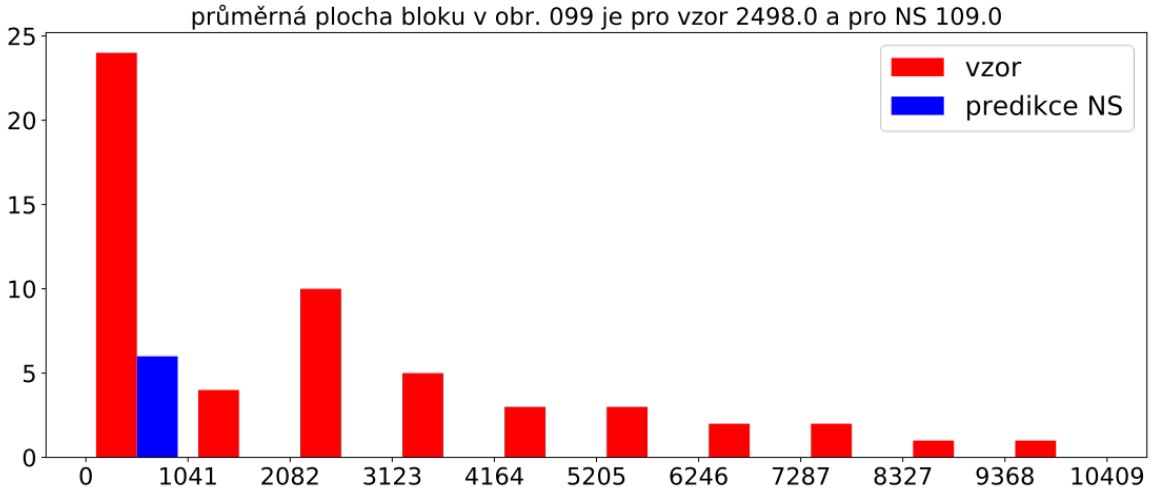
Obrázek 4.32: Histogram plochy bloku budov pro predikované obr. 91-94. Na ose x jsou m^2.



(a)

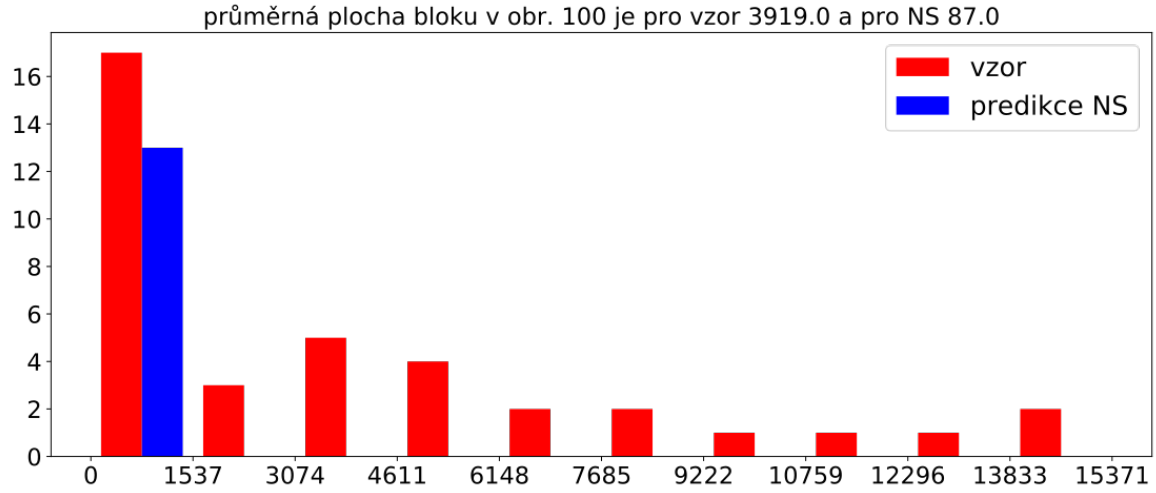


(b)



(c)

Obrázek 4.33: Histogram plochy bloku budov pro predikované obr. 95-99. Na ose x jsou m^2.



Obrázek 4.34: Histogram plochy bloku budov pro predikované obr. 100. Na ose x jsou m^2.

Podívejme se ještě na korelaci mezi vzory a predikovanými obrázky v tabulce 4.4.

kor. koef. - obr.	**081**	**082**	**085**	**091**	**092**	**094**	**095**	**097**	**099**	**100**
plocha bloku	0.0958	0.983	0.971	0.997	0,905	0.988	0.975	0.995	0.927	0.961

Tabulka 4.4: Korelace vzorů a predikovaných obrázků vzhledem k naměřeným plochám.

4.7 Shrnutí parametrů

Podívejme se na přehled průměrných hodnot parametrů, viz tabulka 4.6 a jejich korelace mezi vzory a predikovanými obrázky, viz tabulka 4.5

obr.	**081**	**082**	**085**	**091**	**092**	**094**	**095**	**097**	**099**	**100**
obvod	0.796	0.958	-0.441	1.	-1.	0.988	0.238	0.998	0.501	-0.130
protažení	0.306	0.846	0.615	0.145	0.738	0.113	0.101	-0.007	-0.166	-0.345
rozteč	-0.048	0.025	0.328	0.910	0.870	0.535	0.840	0.938	0.968	0.796
plocha	0.0958	0.983	0.971	0.997	0,905	0.988	0.975	0.995	0.927	0.961

Tabulka 4.5: Shrnutí korelace parametrů v příslušných histogramech.

parametr	**vzor**	**predikovaný obr.**
délka bloku	350 m	183 m
protaženost	0.55	0.58
rozteče	21 m	14 m
plocha bloku	2 101 m^2	565m^2

Tabulka 4.6: Přehled průměru měřených parametrů, jejichž hodnoty budou použity pro modifikaci programu pro procedurální modelování města, viz sekce 5

Z tabulky 4.6 lze vidět, že nejvíce se liší průměrné parametry délky bloku a jeho plochy. U predikovaných obrázků NS, došlo v případě délky bloku ke zmenšení skoro na polovinu, což koresponduje s tím, že plocha bloku budov se zmenšila téměř na čtvrtinu. Jelikož tyto dva parametry spolu úzce souvisí a jak jsme zmínili výše v sekci 4.3 docházelo častěji k rozdrobení velkých bloků budov, než k jejich sjednocení.
Oproti tomu vidíme, že průměrný parametr pro potažení bloku budovy u predikovaných obrázků NS přibližně odpovídá protažení ve vzorech. Pokud jde o průměrnou rozteč u predikovaných obrázků jsme naměřili rozteč zhruba o 33% menší.
Pokud jde o porovnání histogramů měřených parametrů viz 4.5, tak jako nejlepší parametr se jeví plocha bloku budov a rozteč mezi nimi.

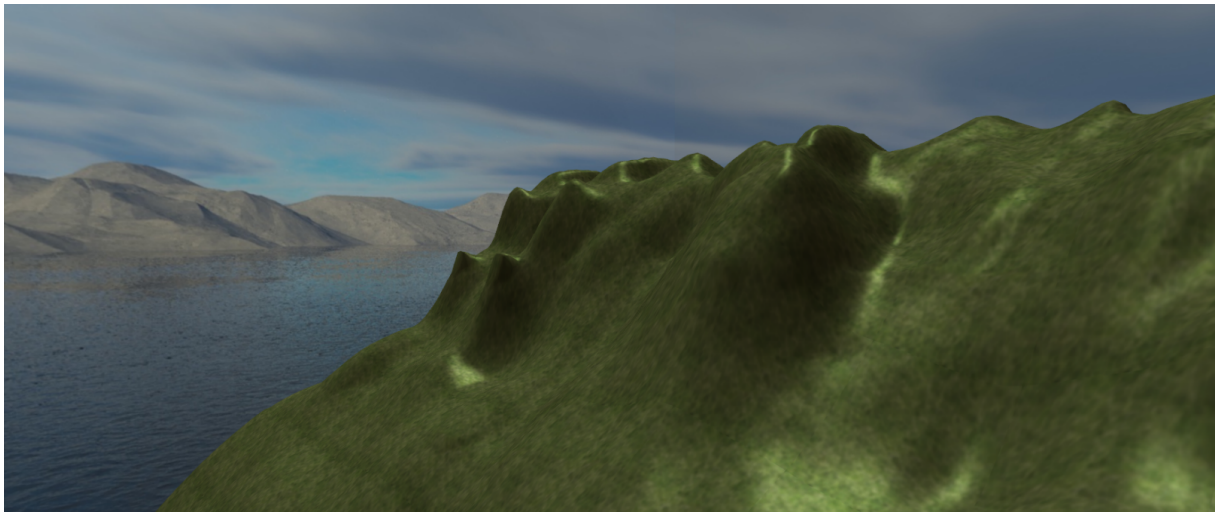
Kapitola 5

Implementace do programu procedurálního generování města

Cílem programu pro procedurální generování města je náhodné generování realisticky vypadajícího města. Takový program může najít využití při urbanistickém plánování, v počítačových hrách atd.
Program je vyvíjen na katedře matematiky a na jeho vývoji se podílelo několik studentů. Hlavní částí je procedurální generování terénu a města. Program vygeneruje terén a na něm síť ulic a stavebních pozemků, na kterých se posléze umístí budovy [29]. Další částí byla implementace procedurálně generovaných budov. Na této části jsme pracovali v rámci mého výzkumného úkolu [37]. Výstupem této práce byla databáze budov a program pro jejich další generování dle požadovaných vlastností. Nyní bychom chtěli program pro procedurální generování města modifikovat tak, aby jeho výsledky více odpovídaly realitě, tzn. zahrnout do něj znaky, které jsme zjistili ze satelitních snímků.
Program můžeme modifikovat pomocí několika parametrů, na jejichž základě se generuje jednak vzhled krajiny, ale i soustava ulic a stavebních pozemků. Mezi tyto parametry patří:

- Výška terénu.
- Šířka terénu.
- Délka silnice mezi křižovatkami.
- Šířka silnice.
- Označení bloku jako stavebního pozemku.

Pro nastavení těchto parametrů jsme použili průměrné hodnoty měřených znaků, viz kapitola 4.
Na obr. 5.1 vidíme vzhled krajiny, způsob generování ulic a pozemků před změnou parametrů. Na první pohled si povšimneme, že krajina je příliš členitá s vysokým převýšením a také rozdělení pozemků neodpovídá realitě. Přesně tyto nedostatky se pokusíme změnou parametrů vylepšit, viz obr. 5.2.
Porovnejme vizualizaci krajiny před modifikací programu na obr. 5.1 s krajinou v již modifikovaném programu, viz obr. 5.2. Z těchto vizualizací je vidět, že se nám podařilo vhodným nastavením parametrů zredukovat členitost a výšku terénu. Zároveň se nám podařilo nastavit členění pozemků, kdy pozemky jsou menší a jejich větší množství.
Bohužel, program pracuje především s nastavením parametrů pro jednotlivé budovy, zatímco my jsme primárně měřili vlastnosti celých bloků budov. Dalším problémem je, že program nám neumožní změnit všechny parametry v potřebných poměrech, aniž bychom provedli výrazné a zásadní změny v jeho kódu. Jelikož přepisovat potřebné části v kódu by bylo příliš časově naročné, tak jsme od tohoto kroku upustili a měnili jsme pouze konstantní parametry.

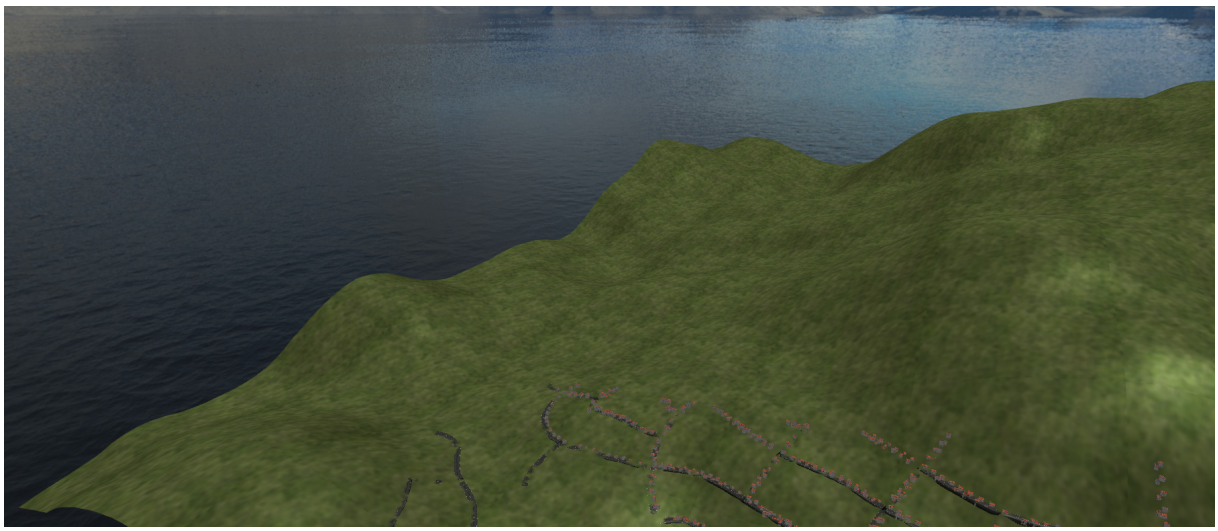


(a) Celkový vzhled krajiny.

(b) Vizualizace města.

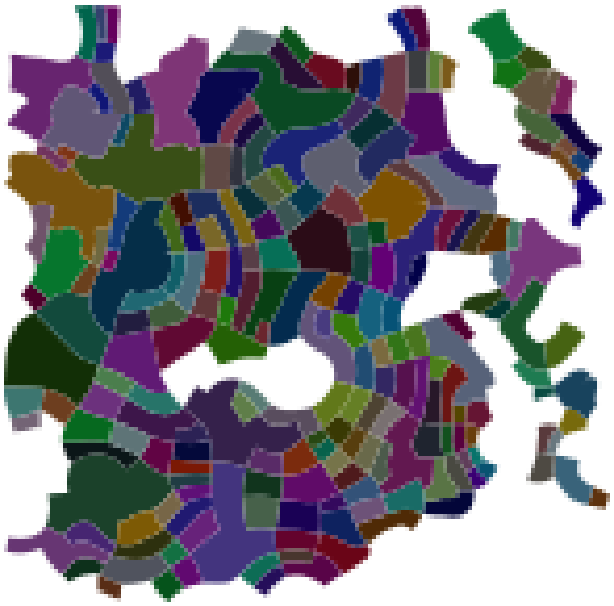
(c) Vizualizace rozčlenění města na pozemky.

Obrázek 5.1: Vizualizace výstupu z programu pro procedurální generování města před úpravou parametrů.



(a) Celkový vzhled krajiny.

(b) Vizualizace města.



(c) Vizualizace rozčlenění města na pozemky.

Obrázek 5.2: Vizualizace výstupu z programu pro procedurální generování města po úpravou parametrů.

Závěr

Na závěr bychom mohli práci shrnout z hlediska

1. kvality detekce budov neuronovou sítí,
2. porovnání získaných znaků z predikovaných snímků a ze vzorů,
3. přiblížení výsledků programu pro procedurální modelování města realitě.

Ohledně prvního bodu vidíme, že i nejlepší NS má poměrně velkou chybovost. Sice najde některé budovy, které nebyly původně v mapových podkladech, ale často detekuje chybně silnici jako budovu. Také nedetekuje přesně okraje budov, ale často větší budovy rozseká na menší části. Naopak, je-li zástavba příliš hustá a ulice úzké, dochází ke slití budov dohromady. Na základě toho by stálo za úvahu vyzkoušet detekci budov pomocí jiných a případně složitější algoritmů pro detekci objektů v obraze (např. mask-RCNN a YOLO viz [32]).
Jelikož máme velikou chybovost již v predikci budov, není překvapivé, že statistické porovnání znaků mezi predikovanými obrázky a vzory se značně liší. Ze statistických znaků, které jsme se snažili zjistit, odpovídá nejlépe znak protažení. Toto zjištění je překvapivé. Přestože dojde k rozmělnění nebo slití objektů, zdá se, že víceméně zůstává zachovaný poměr stran.
Nyní se podívejme na poslední část. Jejím cílem bylo přiblížit náhodně generované město realitě. Bohužel, program není vhodně konstruován vzhledem k příznakům, které jsme zjišťovali. Z povahy dat jsme měřili příznaky pro celé bloky budov, ale program pracuje především s jednotlivými budovami.
V programu jsme měnili pouze konstantní parametry. Modifikaci programu by bylo možné vylepšit, pokud by parametry nebyly pouze konstantní, ale náhodně by se generovaly z rozdělení, které jsme zjistili ze satelitních snímků. Dalšího vylepšení by bylo možné docílit, pokud by jsme budovy náhodně generovali z databáze dle požadovaných vlastností, např. podle protažení, velikosti atd. V současné situaci se do programu vybírají budovy náhodně bez ohledu na jejich vhodnost v daném místě.
Vhodnou manipulací parametrů se nám podařilo docílit realističtější krajiny a rozumnějšího generování pozemků. Nicméně tento program má několik nedostatků vzhledem k jeho struktuře a grafickém zobrazení, které nám nedovoluje dosáhnout výraznějších změn bez značného zásahu přímo do struktury kódu. Jelikož větší zásah do kódu by byl náročný, z časových důvodů jsme jej neuskutečnili.

Literatura

[1] Samoučící se neuronová síť - som, Kohonenovy mapy, author = Antonín Vojáček, month = leden, howpublished = https://www.kiv.zcu.cz/studies/predmety/uir/ns/samouc_nn2.pdf, year = 2020, owner = svetla, timestamp = 2020.01.13.

[2] https://cs.wikibooks.org/wiki/Nervove_bunky, 20. Říjen 2019. Kresba stavby neuronove bunky.

[3] Umělá neuronová síť. https://cs.wikipedia.org/wiki/Um%C4%9Bl%C3%A1_neuronov%C3%A1_s%C3%AD%C5%A5#/media/Soubor:HardLimitFunction.png, říjen 2019.

[4] Apache MXNet on AWS. https://aws.amazon.com/mxnet/, červen 2020.

[5] Computer vision. https://cs.wikipedia.org/wiki/Po%C4%8D%C3%ADta%C4%8Dov%C3%A9_vid%C4%9Bn%C3%AD, červen 2020.

[6] Mean squared logarithmic error (MSLE). https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-logarithmic-error-(msle), červen 2020.

[7] Norma ČSN 73 6110 - projektování místních komunikací. https://www.mmr.cz/cs/ministerstvo, červen 2020.

[8] Otsu's method. https://en.wikipedia.org/wiki/Otsu%27s_method, červen 2020.

[9] Perceptron. https://cs.wikipedia.org/wiki/Perceptron, leden 2020.

[10] S. Amari. Backpropagation and stochastic gradient descent method. In *Neurocomputing*, volume 5, page 185–196, 1993.

[11] J. Brownlee. *Better Deep Learning Train Faster, Reduce Overfitting, and Make Better Predictions*. Machine Learning Mastery, 2018.

[12] J. Brownlee. How to choose loss functions when training deep learning neural networks. https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/, leden 2020.

[13] J. Brownlee. Loss and loss functions for training deep learning neural networks. https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/, leden 2020.

[14] K. Chowdhury. Understanding loss functions : Hinge loss. https://medium.com/analytics-vidhya/understanding-loss-functions-hinge-loss-a0ff112b40a1, červen 2020.

[15] M. Cokluk. Kullback-Leibler divergence loss vs. (weighted) cross entropy loss. https://medium.com/@mertcokluktttt/kullback-leibler-divergence-loss-vs-weighted-cross-entropy-loss-79126dccc8a2, červen 2020.

[16] D. Godoy. Understanding binary cross-entropy / log loss: a visual explanation. https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a, červen 2020.

[17] R. Goméz. Understanding categorical cross-entropy loss, binary cross-entropy loss, softmax loss, logistic loss, focal loss and all those confusing names. https://gombru.github.io/2018/05/23/cross_entropy_loss/, červen 2020.

[18] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Massachusetts Institute of technology, 2016.

[19] P. Grover. 5 regression loss functions all machine learners should know. https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0, červen 2020.

[20] J. J. Hopfield. *Neural networks and physical systems with emergent collective computational abilities.* Proc. Natl. Acad. Sci. USA, duben 1982.

[21] S. Kalyanakrishnan. The perceptron learning algorithm and its convergence. https://www.cse.iitb.ac.in/~shivaram/teaching/old/cs344+386-s2017/resources/classnote-1.pdf, leden 2017.

[22] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48, 2012.

[23] J. Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, page 735–742, 2010.

[24] V. Mnih. *Machine Learning for Aerial Image Labeling*. PhD thesis, University of Toronto, 2013.

[25] R. Odegua. Image classification from scratch in keras. https://towardsdatascience.com/image-detection-from-scratch-in-keras-f314872006c9, 11. August 2019.

[26] OpenStreetMap contributors. Planet dump. https://planet.osm.org, 2019.

[27] R. Pascanu and Y. Bengio. Revisiting natural gradient for deep networks. *arXiv*, 2013.

[28] S. Pattanayak. *Pro Deep Learning with TensorFlow*. Apress, 2017.

[29] M. Pavlíček. *Komplexní modulární algoritmus pro procedurální generování modelu města v počítačové grafice*. 2018. Diplomová práce.

[30] P. Perona and J. Malik. *Scale-space and edge detection using ansotropic diffusion*. 1990.

[31] M. Pilát. Neuronové sítě - rbf sítě a rekurentní sítě. https://martinpilat.com/cs/prirodou-inspirovane-algoritmy/neuronove-site-rbf-site-rekurentni-site, leden 2020.

[32] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[33] R. Reed and R. J. Marks II. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. A Bradford Book, 1999.

[34] A. Sagar. 5 techniques to prevent overfitting in neural networks. https://towardsdatascience.com/5-techniques-to-prevent-overfitting-in-neural-networks-e05e64f9f07, leden 2020.

[35] S. Sharwood. Artificial intelligence. https://www.theregister.com/2017/04/19/cloud_vision_api_defeated_by_noise/, červen 2020.

[36] N. Shukla. *Machine Learning with TensorFlow*. Manning Publications, 2017.

[37] S. Smrčková. *Algoritmy pro procedurální modelování objektů a scén v počítačové grafice*. ČVUT, FJFI, 2018. Výzkumný úkol.

[38] Theano Development Team. Theano. `http://deeplearning.net/software/theano/`, červen 2020.

[39] O. Vinyals and D. Povey. Krylov subspace descent for deep learning. *arXiv*, (1111.4259), 2011.

[40] S. Wiesler, A. Richard, R. Schluter, and H. Ney. Mean-normalized stochastic gradient for large-scale deep learning. In *Acoustics, Speech and Signal Processing (ICASSP)*, page 180–184. IEEE International Conference, 2014.