Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Segmentace obrazu pomocí neuronových sítí

Plzeň 2023 Marián Kadlečík

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Marián KADLEČÍK**
Osobní číslo:	**A19B0083P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Segmentace obrazu pomocí neuronových sítí**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s dostupnými datovými kolekcemi pro úlohu segmentace.
2. Seznamte se s algoritmy pro segmentaci obrazu pomocí neuronových sítí.
3. Implementujte minimálně dva vybrané algoritmy pro segmentaci.
4. Otestujte implementované algoritmy na reprezentativní sadě netriviálních dat a zhodnoťte výsledky.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce

Vedoucí bakalářské práce: **Ing. Martin Prantl, Ph.D.**
Nové technologie pro informační společnost

Datum zadání bakalářské práce: **3. října 2022**
Termín odevzdání bakalářské práce: **4. května 2023**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 25. října 2022

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 4. května 2023

Marián Kadlečík

Abstract

This work deals with image segmentation via the usage of neural networks. The beginning of the work focuses on the problem of segmentation itself with a description of a couple of existing methods that do not use neural networks. Afterwards the work shifts it's focus to neural networks, their structure, functionality and usage for segmentation with an analysis of multiple existing architectures. Those architectures are U-Net (2015), DoubleU-Net (2020) and ResUNet++ (2019), which were then implemented into a program that can be used to train them on different data and subsequently use them for segmentation purposes. Finally the implemented architectures are trained on two seperate datasets, Carvana and IMCDB dataset, with their individual results compared in the last section of this work. The best results were achieved on the DoubleU-Net architecture with a Jaccard index value of 0.9883 for Carvana dataset and 0.9383 for IMCDB dataset respectively.

Abstrakt

Tato práce se zabývá segmentací obrazu za využití neuronových sítí. Začátek práce se zabývá problematikou segmentování s několika existujícími metodami nevyužívající neuronových sítí. Následně se práce již zaměřuje na neuronové sítě, jejich strukturu, funkcionalitu a využití pro segmentaci s analýzou několika vybraných architektur. Vybranými architekturami jsou U-Net (2015), DoubleU-Net (2020) a ResUNet++ (2019), a byly implementovány do programu, který je umožňuje učit na různých datech a využít za účelem segmentace. V konečné části práce se potom porovnávají výsledky mezi jednotlivými architekturami z učení a segmentace na Carvana a IMCDB datasetu. Nejlepších výsledků dosáhla architektura DoubleU-Net s Jaccard index hodnotou 0.9883 pro Carvana dataset a 0.9383 pro IMCDB dataset.

Obsah

1 Úvod

V posledních letech dochází v oboru umělé inteligence a hlubokého učení k neustálým pokrokům a optimalizacím. Jedním z největších odvětví tohoto oboru jsou neuronové sítě, jejichž rapidní vývoj, univerzalita v možnostech využití a neustále se zlepšující výkon způsobují explozivní nárůst v jejich používání v mnoha různorodých oborech. Jedním z těchto využití je právě segmentování obrazu, které umožňuje nejen detekovat a klasifikovat individuální objekty co se v obrazu vyskytují, ale hlavně i lokalizovat kde specificky v obraze jsou. Tato funkcionalita hraje významnou roli například v mnoha oblastech medicíny (detekce nádorů), automobilového průmyslu (autonomní řízení) nebo zpracovávání satelitních snímků (mapování ulic a budov nebo hledání přírodních zdrojů).

Cílem práce je seznámení se s obecnou problematikou segmentace a neuronových sítí k tomu navržených. Následně je za využití těchto znalostí navržen a implementován program, který umožní uživateli vybrat z několika různých síťových architektur k natrénování a následnému uplatnění k segmentaci.

V teoretické části této práce jde o seznámení s problematikou segmentace obrazu a několika obecnými metodami, které k tomu byli vymyšleny. Následovat bude přiblížení neuronových sítí samotných, jejich funkcionality a využití pro segmentaci. Pak se podíváme na několik již existujících neuronových sítí, které byly k segmentaci navrženy a pro účely této práce vybrány k implementaci.

Praktická část práce se pak zabývá programem v němž jsou sítě naimplementovány, jeho strukturou, funkcionalitou a použitými knihovnami. Nakonec jsou v textu prezentovány výsledky segmentování implementovaných sítí na dvou různých datasetech s tím, že jsou výsledky dále mezi sebou porovnány.

2 Segmentace obrazu

Segmentace obrazu je jednou z prvních a nejdůležitějších součásti oborů obrazové analýzy a počítačového vidění. Jedná se o proces digitálního zpracování obrazu, ve kterém dochází k rozdělení obrazu na části, které mají patřičnou významnou vlastnost nebo přísluší určitému objektu. Segmenty samotné jsou nejčastěji značeny na úrovni jednotlivých pixelů, kde je každý pixel klasifikován třídou, podle toho ke které skupině daný pixel patří. Takto segmentovaný obraz je často označován jako segmentační maska nebo mapa, a je mnohem vhodnější pro další zpracovávání, neboť je v něm známo, co, a kde se v obrazu nachází. V podstatě se dá říci, že segmentování obrazu efektivně kombinuje funkcionalitu obrazové klasifikace a detekce objektů na úrovni pixelů.

2.1 Druhy segmentace obrazu

Přístupy k segmentaci se odlišují podle množství extrahovaných informací a k čemu dále mají informace být určeny. Můžou se lišit množstvím hledaných tříd, přístupu k objektům příslušících do stejné třídy, nebo co se v rámci segmentace bere jako jeden objekt. Podle těchto odlišností je možné rozlišit několik obecných přístupů k segmentování.

- Sémantická segmentace (Semantic segmentation)
- Segmentace jednotlivých objektů (Instance segmentation)
- Panoptická segmentace (Panoptic segmentation)

Na obrázku 2.1 jsou přístupy vyobrazeny.

2.1.1 Sémantická segmentace

Jednotlivé třídy sémanticky segmentovaného obrazu hromadně reprezentují všechny objekty, které do dané třídy patří. To například znamená, že všichni lidé v obrazu budou patřit do jedné třídy. Dále k tomuto přístupu spadá segmentování věci, které nelze kvantifikovat, například silnice nebo nebe.

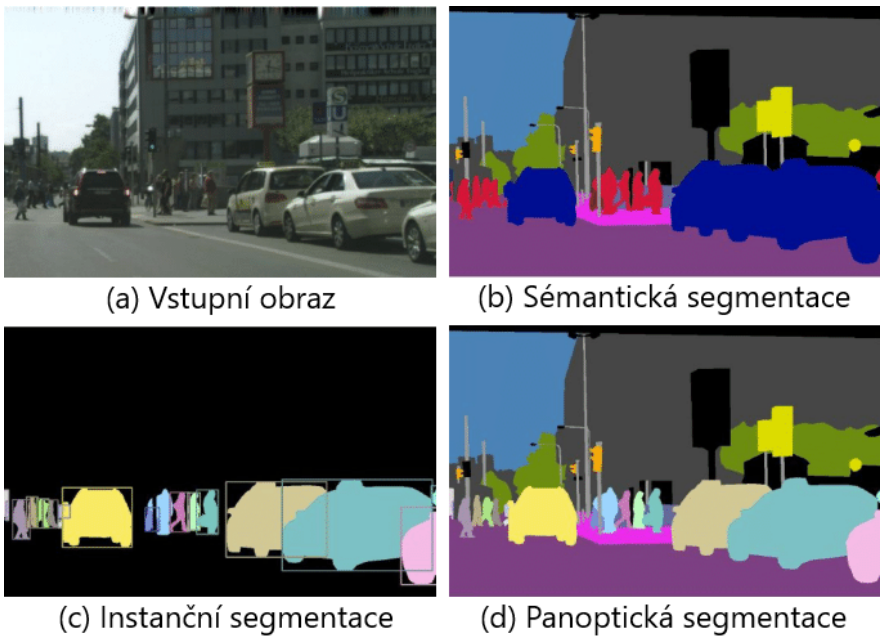
Specifickým případem tohoto druhu segmentace je binární segmentace, ve které se v obraze hledá pouze jedna třída, respektive dvě, a to jestli pixel přísluší hledanému objektu, nebo ne. Tomuto přístupu se budeme v této práci věnovat primárně.

2.1.2 Segmentace jednotlivých objektů

Jak již vychází z názvu, v tomto přístupu se segmentují jednotlivé výskyty objektů. Každý samostatný výskyt objektu patřící do stejné třídy je v obraze označen vlastním, nejčastěji číselným, identifikátorem. Lze potom říci, že se zabývá hlavně kvantifikovatelnými věcmi.

2.1.3 Panoptická segmentace

Panoptická segmentace kombinuje předchozí dva přístupy. Pro každý pixel jsou zde přiřazena dvě označení, a to o kterou věc se jedná a o kolikátý výskyt dané věci se jedná.



Obrázek 2.1: Porovnání druhů segmentace [3].

2.2 Metody segmentace bez neuronových sítí

Segmentování obrazu je úloha, jejíž řešením se výzkumníci z celého světa zabývají již od prvopočátků digitálního obrazu. Protože využití neuronových sítí pro segmentování je metoda, která se skutečně začala prosazovat až v posledních 10 letech, stojí za zmínku i některé metody, které neuronovým sítím předcházely. Mezi tyto metody spadá například využití prahování (tresholding), detekce hran (edge detection), zaplavování (watershed), shluková analýza (clustering) a regionově orientované metody. Většinu těchto

metod lze různými způsoby kombinovat pro dosažení lepších segmentačních výsledků. V následujících dvou sekcích na některé z nich nahlédneme.

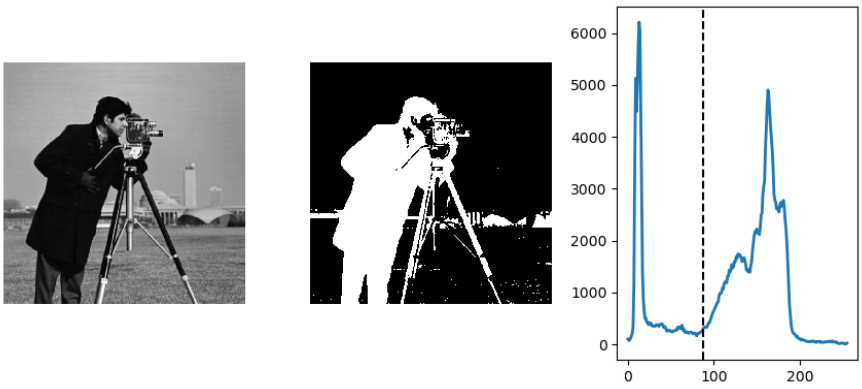
2.2.1 Prahování

Metody využívající prahování pracují se skutečností, že objekty v obraze mají často znatelný rozdíl v intenzitě jasu oproti jeho pozadí. V těchto metodách dochází k porovnávání intenzity jasu jednotlivých pixelů s určitou hodnotou prahu. Pokud je intenzita pixelu vyšší, je mu přiřazena hodnota 1, naopak nižší intenzitě pak hodnota 0. Výstupem je potom binární obraz, kde by ideálně byly hodnotou 1 označeny pouze všechny významné objekty.

V primitivním přístupu je hodnota prahu konstantní pro celý obraz. Stanovená hodnota by měla být ideálně taková, aby došlo pouze k oddělení významných objektů. V realitě to v mnoha případech ovšem není úplně možné, a to kvůli šumu a jiným faktorům (světelné podmínky, stíny atd.). Zavádí se proto využití dynamického prahu, kdy se obraz rozdělí na menší segmenty a práh se potom určí pro každý segment zvlášť.

Určení hodnoty prahu může být metodou pokus omyl, kde zkoušíme různé hodnoty, dokud nedosáhneme požadovaného výsledku. Další možností může být využití Otsu metody [19]. Otsu metoda je založená na stanovení prahu z histogramu obrazu a výpočtu rozptylu jasu pro pixely v popředí a v pozadí. Důležité u Otsu metody je, aby histogram byl tzv. bimodální. To znamená, že obsahuje dva zřetelné vrcholy v hodnotách jasu (pixely v popředí a pozadí respektive).

Na obrázku 2.2 lze vidět příklad prahování s využitím Otsu metody. Vlevo je původní obrázek převeden do černobílého, uprostřed je výsledek prahování a vpravo je bimodální histogram intenzity jasu s vyznačeným prahem.



Obrázek 2.2: Příklad prahování s využitím Otsu metody [26].

2.2.2 Regionově orientované metody

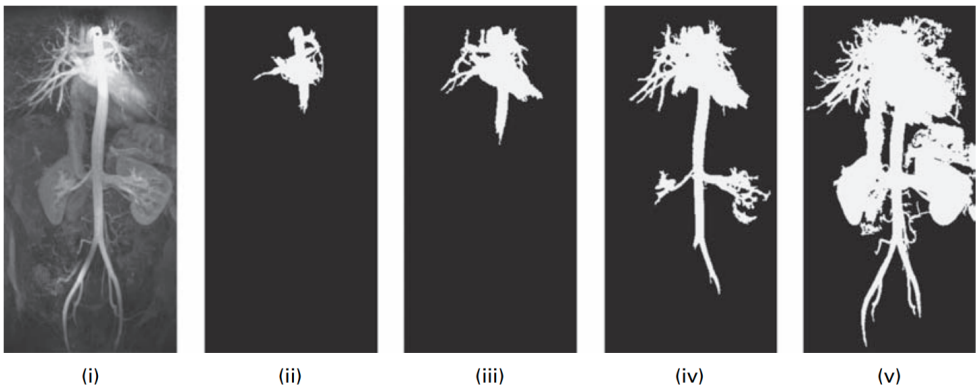
Regionově orientované metody jsou založeny na podobnosti vlastností segmentovaných oblastí. Tyto vlastnosti mohou být například barva, jas, textura atd. Rozlišuje se zde několik přístupů pro vytváření regionů, které budou dále popsány.

Nárůst regionů

Metoda s nárůstem regionů (region growing) pracuje s několika body, které postupně rozšiřuje do celých segmentů. Vlastnosti těchto bodů určují vlastnosti segmentovaných oblastí, které z jednotlivých bodů vznikají.

Na začátku dochází k určení počátečních bodů (seeds). Následuje jejich rozšiřování postupným porovnáváním s nejbližšími sousedními pixely v okolí. Pokud je porovnávaný pixel dostatečně podobný tomu původnímu, pixel je přidán do segmentu. Takto se segmenty rozšiřují do doby, kdy do nich nelze přidat další pixely. Za sousední pixely se zde může považovat všech 8 okolních pixelů nebo pouze 4 pixely bez těch sousedících na diagonálách.

Na obrázku 2.3 lze vidět průběh metody s nárůstem regionů.



Obrázek 2.3: Průběh narůstání regionů. (i) původní obraz s šedým označením počátečního bodu v horní části obrazu, (ii)–(v) postupný nárůst regionu [4].

Slučování regionů

Slučování regionů (region merging) probíhá podobně jako metoda s nárůstem regionů. Na počátku jsou určeny malé regiony, které nejlépe budou tvořeny z pixelů, které si jsou navzájem podobné. Poté začne postupné porovnávání vždy dvou sousedních regionů a jejich sloučení v případě dostatečné podobnosti. To se opakuje, dokud se nesloučí všechny regiony, které si jsou podobné a nezbudou pouze ty, které se již dále sloučit nemohou.

Dělení a slučování regionů

Přístup dělení a slučování regionů (region splitting and merging) se dělí na dvě fáze, které vyplývají z pojmenování. Fáze kdy dochází k dělení regionů a fáze slučování regionů.

V první fázi dochází k rozdělování regionů obrazu, jejichž obsah si není podobný, na menší regiony. Dělení probíhá do té doby, dokud si obsah každého regionu není podobný. Rozdělování probíhá obvykle na kvadranty. Tyto kvadranty lze během toho co se rozdělí, opětovně porovnat navzájem mezi sebou a případně je okamžitě sloučit.

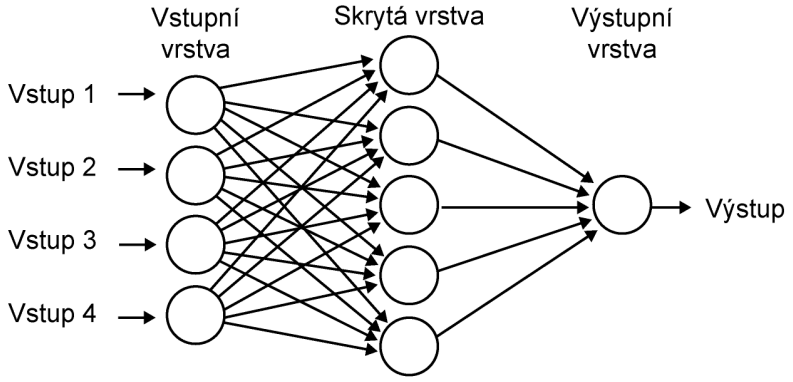
V druhé fázi postupujeme stejně jako v metodě využívající slučování regionů. Sousední regiony se opět navzájem porovnávají mezi sebou a slučují se v případě podobnosti.

3 Neuronové sítě

Vývoj technologií a pokroky v oborech umělé inteligence a hlubokého učení umožnily umělým neuronovým sítím (artificial neural networks), aby se staly jednou z předních metod pro řešení téměř jakýchkoliv úloh, jejichž řešení obnáší predikci. To může být například v oblastech robotiky, financí nebo lékařství. Neuronové sítě jsou výpočetní systémy, založené na principech biologické funkčnosti mozku. Schopnosti mozku se učit, hledat souvislosti a řešit problémy na základě předchozích zkušeností, se po přesunutí do počítačových výpočtů sice stává velmi efektivním, ale výpočetně náročným kvůli obrovskému množství informací, které je nutné zpracovat.

3.1 Struktura neuronových sítí

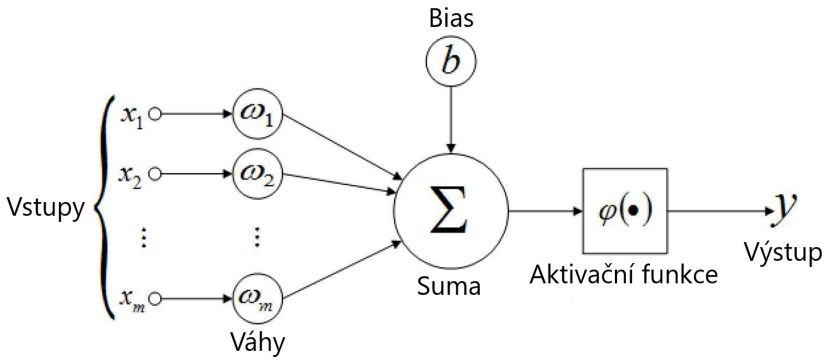
Neuronové sítě jsou nejčastěji více-vrstvou strukturou popsatelnou grafem. Typicky struktura začíná jednou vstupní vrstvou, následovanou žádnou, jednou nebo více skrytými vrstvami a končí vrstvou výstupní. Skryté vrstvy slouží k postupné transformaci vstupních dat do takové podoby, ze kterých síť ve finální výstupní vrstvě dokáže provést predikci výsledku. Celkový počet vrstev je označován jako hloubka neuronové sítě a pro různě komplexní úlohy se navrhují struktury s rozdílnými hloubkami. Každá vrstva je tvořena jedním nebo více umělými neurony. Specifická struktura je potom často označována jako architektura nebo model. Na obrázku 3.1 lze vidět možnou strukturu neuronové sítě.



Obrázek 3.1: Příklad struktury neuronové sítě s jednou skrytou vrstvou.

Jednotlivé neurony jsou potom samotné uzly v síti. Každý neuron má

určené hodnoty vah, které se v průběhu učení můžou měnit. Tyto hodnoty pak neuron využívá k roznásobení a transformaci příchozích dat. V každé vrstvě kromě vstupní se většinou přidává navíc tzv. bias, jehož hodnota je k transformovaným datům přidána a tím je způsoben posun aktivační funkce po ose a usměrnění průchodu dat správným směrem. Zmíněná aktivační funkce svým výsledkem určuje výstup z neuronu a zajišťuje nelinearitu, která je důležitou vlastností neuronových sítí. Na obrázku 3.2 je vyobrazena možná struktura jednoho neuronu v neuronové síti.



Obrázek 3.2: Jednoduchý neuron [1].

3.1.1 Aktivační funkce

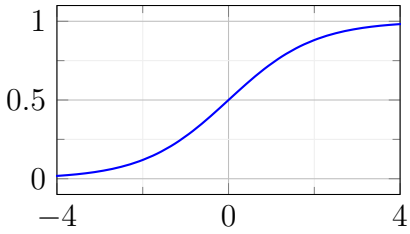
Jak již bylo zmíněno, aktivační funkce určuje výstup z neuronu. Aktivační funkce to je, jelikož rozhoduje, zdali se neuron aktivuje a tím svou výstupní hodnotu bude propagovat dál. Existuje jich velké množství [2], mezi ty nejčastěji používané patří následující:

Sigmoid

Tato funkce vezme jakékoliv reálné číslo a transformuje ho do intervalu $(0; 1)$. Čím vyšší je vstup, tím víc se blíží hodnotě 1, naopak čím menší, tím blíž hodnotě 0. Nulový vstup bude přesně v hodnotě 0,5. Matematický zápis funkce vypadá takto:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

a graf funkce pak takto:

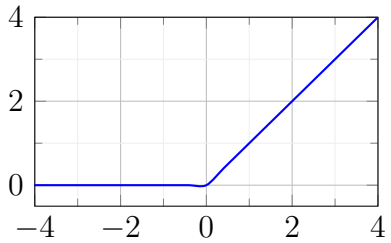


ReLU funkce

ReLU (Rectified Linear Unit) funkce aktivuje neuron pouze pokud hodnota jeho výstupu je vyšší než 0. Matematicky ji lze popsat velmi jednoduše jako:

$$f(x) = max(0, x) \tag{3.2}$$

a graf funkce pak vypadá následovně:



Softmax

Aktivační funkce Softmax je speciální v tom, že se vlastně jedná o kombinaci několika vstupních hodnot, které převede na vektor pravděpodobnostní. Výstup této funkce je pravděpodobnost, že je daná hodnota ta správná v relaci k ostatním. Matematický zápis zde vypadá následovně:

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}. \tag{3.3}$$

Protože je funkce závislá na několika hodnotách, nelze pro ni graf přímo určit.

3.2 Historie neuronových sítí a zpracování obrazu

Přestože hojné využívání neuronových sítí začalo až v posledních 10 letech, jejich prvotní myšlenka přišla na svět již v roce 1943, kdy Warren McCulloch

a Walter Pitts [6] vytvořili jednoduchý matematický model neuronu. Roku 1957 byl Frankem Rosenblattem vymyšlen tzv. perceptron [22], který byl zobecněním předchozího neuronu a vlastně i první teoreticky funkční jednovrstvou neuronovou sítí. O rok později, Rosenblatt, Charles Wightman a další, sestrojili první neuropočítač s názvem *„Mark I Perceptron"*, který byl určen pro rozpoznávání obrazců, přesněji promítaných znaků.

Vývoj pokračoval nadějně až do roku 1969, kdy Marvin Minsky a Seymour Papert [18] publikovali článek, který odhaluje nedostatky perceptronu, jeho jednovrstvé architektury a nemožnost dalších větších pokroků v tomto oboru. Tento článek způsobil převážnou stagnaci nových výzkumů, která trvala až do 80. let. V roce 1986 Rumelhart, Hinton a Williams [23] znovuobjevili a zpopularizovali učící algoritmus zpětného šíření (backpropagation) pro vícevrstvé neuronové sítě, kterým vyřešili Minského a Papertovo problém a otevřeli tak dveře dalším objevům.

Důležitým milníkem byl rok 1998, kdy Yann LeCun s kolektivem [17] představuje LeNet–5, 7–vrstvou konvoluční neuronovou sít pro klasifikaci cifer. V publikaci autoři porovnávají různé metody pro klasifikaci cifer a docházejí k závěru, že konvoluční architektura svými výsledky překonává všechny ostatní modely. Přesto využití konvolučních neuronových sítí nebylo efektivní z důvodu náročnosti na výkon. Až v roce 2012 se Alex Krizhevksy, Ilya Sutskever a Geoffrey E. Hinton [16] podíleli na publikaci, ve které představují architekturu, využívající vysokého výkonu grafických karet při trénování sítě. S variantou této architektury autoři vyhráli *ImageNet Large Scale Visual Recognition Challenge 2012* (ILSVRC) s Top-5 chybovostí[1] 15.3%. Na druhé pozici se umístila architektura, která dosáhla chybovosti 26.2%. Tento masivní rozdíl ukázal světu efektivitu grafických karet při tréningu neuronových sítí a ještě více utvrdil využívání konvolučních sítí pro zpracovávání digitálního obrazu.

3.3 Konvoluční neuronové sítě

Konvoluční neuronové sítě (Convolutional Neural Networks, CNN) jsou sítě specializované ke zpracování dat, které jsou popsatelné mřížkovou topologií. Typickým příkladem takových dat jsou obrazy, jehož pixely lze chápat jako 2–D mřížku [7]. Právě obrazy jsou data, která nás budou převážně zajímat. Je zde vhodné podotknout, že se data obrazu během průchodu sítě často dodatečně rozdělují na několik kanálů. Kanály mohou být např. jednotlivé matice barevných složek obrazu nebo jiné, které postupně vznikají během

[1] Procento klasifikací kde hledaná třída nebyla mezi top 5 predikovanými třídami.

průchodu vrstvami sítě a poskytují síti kontextové informace.

Slovo „konvoluční" v názvu je označení pro matematickou operaci konvoluce, kterou alespoň v jedné vrstvě konvoluční sítě aplikují. Další důležitou vrstvou je tzv. pooling. Obě tyto vrstvy budou dále popsány.

3.3.1 Konvoluční vrstva

Obecná konvoluce je matematická operace, která ze dvou funkcí $f(x)$ a $g(x)$ vytvoří třetí funkci $(f * g)$, která popisuje jak tvar jedné funkce ovlivní funkci druhou. V neuronových sítích se však využívá konvoluce diskrétní, která funguje na podobném principu jako ta obecná. Místo funkcí zde ovšem máme dvě množiny čísel f a g, a jejich konvoluci lze obecně popsat takto:

$$s(t) = (f * g)(t) = \sum_{m=-\infty}^{\infty} f(m)g(t-m), \tag{3.4}$$

kde je, z pohledu konvolučních sítí, f množinou vstupních dat (input), množina g je tzv. konvoluční jádro (kernel) (někdy také konvoluční filtr nebo maska), a výsledkem konvoluce $s(n)$ je tzv. mapa vlastností (feature map) (někdy vektor vlastností nebo mapa příznaků). Pokud jsou naše data dvourozměrná, jako například u obrazu, můžeme tento vzorec dále rozšířit následovně:

$$S(i,j) = (I * K)(i,j) = \sum_{m=1}^{K_w} \sum_{n=1}^{K_h} I(m,n)K(i-m, j-n), \tag{3.5}$$

kde K_w a K_h jsou šířka a výška jádra respektive a množiny hodnot jsou zde reprezentovány maticemi. Výška a šířka jádra se většinou určuje stejná (nejčastěji 3×3, 5×5 nebo 7×7). Podle definice konvoluce se jádro navíc musí otočit o 180°. Důsledkem toho má výsledek operace stejnou orientaci, jako vstupní data a konvoluce má potom užitečnější matematické vlastnosti. Takto popsaný vzorec konvoluce sice již lze využít, ale ve skutečnosti se nejedná o operaci, která je v konvolučních sítích primárně používána. Vzorec operace která se využívá vypadá takto:

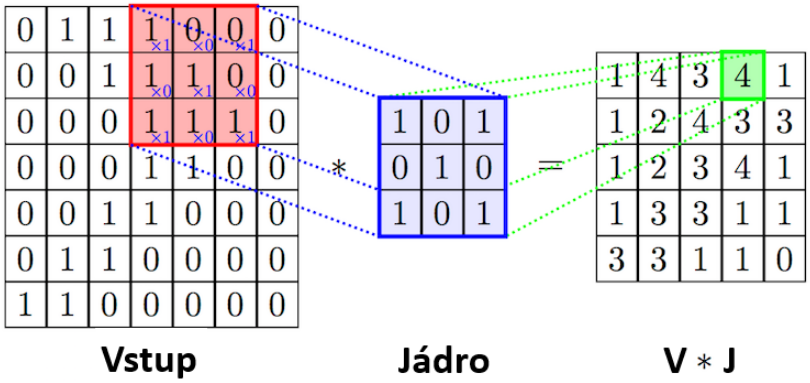
$$S(i,j) = (K * I)(i,j) = \sum_{m=1}^{K_w} \sum_{n=1} K_h I(i+m, j+n)K(m,n). \tag{3.6}$$

Tato operace se nazývá vzájemná korelace (cross–correlation) a jediným rozdílem je neotočení jádra, což způsobuje že výsledek je otočený. Proč se tedy používá korelace? Otočení jádra sice není složitou operací, ale přidává nadbytečnou výpočetní komplexitu, která se následně šíří do dalších operací a vrstev neuronové sítě. Na samotnou výslednou funkcionalitu sítě to ovšem

nemá důsledek, vnitřně se síť naučí otočené hodnoty, které proces konečného vyhodnocování neovlivní. V dalším textu bude pod slovem konvoluce myšlena právě vzájemná korelace.

Neformální popis

Pokud na konvoluci přestaneme pohlížet pouze matematickými vzorci, můžeme ji vysvětlit poměrně jednoduchým způsobem. Jádro v konvoluci slouží k detekci různých věcí, jako jsou např. hrany, textury a tvary. Jeho hodnoty se mění v průběhu učení, podobně jako u neuronů. Na vstupní matici pak přikládáme matici jádra, kde se roznásobí na sobě ležící prvky a jejich součet se zapíše do výstupní matice mapy vlastností. Jádro se dále posouvá vertikálně či horizontálně o specifikovanou délku kroku (stride). Na obrázku 3.3 je vyobrazena ukázka konvoluce.



Obrázek 3.3: Ukázka konvoluce.

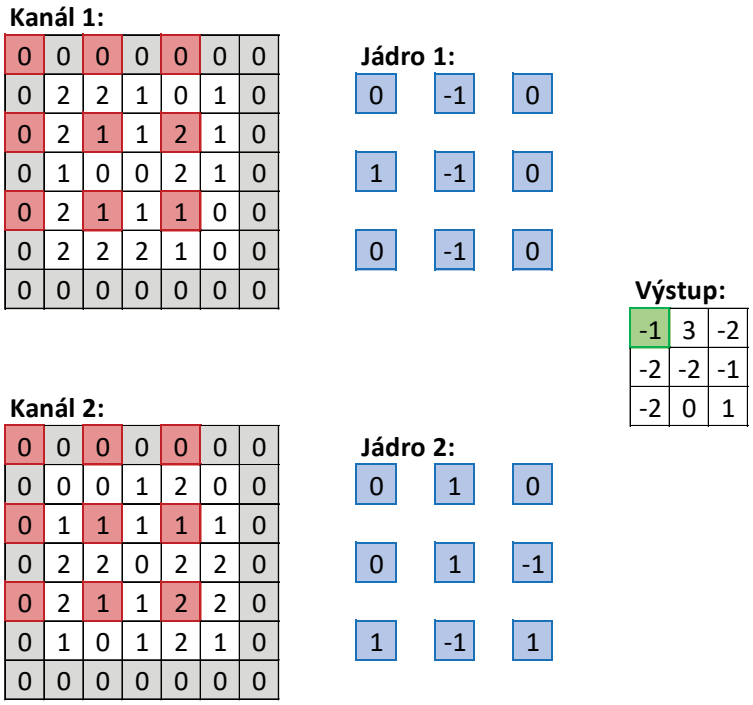
Za povšimnutí stojí, že se rozměr výstupní matice, v porovnání se vstupní, zmenšil a tím došlo k potencionálně nežádoucí ztrátě informací. Tento problém lze zamezit přidáním výplně (padding) kolem vstupní matice. Tato výplň je typicky nulová a může mít několik vrstev. Velikost výstupní matice je případně možné spočítat následujícím vzorcem:

$$\frac{W - K + 2P}{S} + 1, \tag{3.7}$$

kde W je velikost vstupní matice, K je velikost jádra, P je počet vrstev výplně a S je délka kroku.

Další možnou modifikací je rozšíření vzdálenosti mezi prvky jádra (dilation). To umožňuje získání dat z větší plochy obrazu, aniž by se jádro samotné muselo zvětšit a tím dochází k ušetření výpočtů. Protože je rozšířením uměle navýšena velikost jádra, logicky to má za důsledek zmenšení výstupu.

V neposlední řadě může konvoluce probíhat tak, že je na jeden stejný vstup použito několik rozdílných jader, jejichž jednotlivé výsledky můžou sloužit k navýšení počtu kanálů mapy vlastností. Více jader se obecně používá za účelem detekování rozdílných věcí. Obdobně je možné provést konvoluci na více kanálech současně, s jedním nebo více jádry, a naopak tak počet kanálů snížit. Výstup konvoluce se poté typicky získá sečtením výstupů z jednotlivých kanálů. Na obrázku 3.4 je zobrazena konvoluce s více kanály a rozdílným jádrem pro každý kanál, včetně přidané výplně a rozšíření jader.



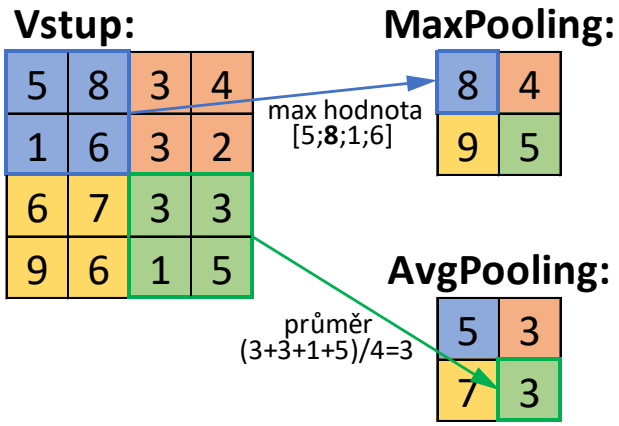
Obrázek 3.4: Konvoluce s více kanály, vstup má přidanou jednu vrstvu výplně a jádra jsou o jeden krok rozšířena.

Zpětná konvoluce

Zpětná konvoluce, někdy také transponovaná konvoluce, je inverzní operací ke konvoluci. Operace se snaží o zpětné sestrojení původních dat, které již konvolucí prošli. Kvůli zobecnění, které během konvoluce proběhne, se výsledná data liší od těch původních. Tento problém se může vyřešit například uložením dat před konvolucí a jejich pozdějšímu využití pro zpřesnění výstupu této operace.

3.3.2 Pooling vrstva

Pooling vrstvy se využívají pro zmenšení původních vstupních dat za účelem snazšího zpracování a snížení výpočetní složitosti. Obdobně jako u konvoluce zde máme jádro (nejčastěji 2×2), které se posouvá o specifikovaný krok po vstupní matici (zde nejčastěji použita velikost jádra). Hodnota pro výslednou matici se potom získá z hodnot, které jádro překrývá. Může se zde použít nejvyšší hodnota anebo průměrná hodnota. Podle toho se potom rozlišuje buď *Maxpooling* nebo *AvgPooling*. Oba přístupy jsou ukázány na obrázku 3.5. Stejně jako u konvoluce se zde na vstup může aplikovat výplň a jádro se může rozšířit.



Obrázek 3.5: Ukázka *MaxPoolingu* a *AvgPoolingu*.

3.4 Učení neuronových sítí

Proto, aby neuronová síť vůbec fungovala, musí se nejdříve naučit informace o datech, které vůbec bude zpracovávat. K tomu je potřebné mít trénovací sadu dat (dataset), ze kterých se síť naučí vzory a příznaky podle kterých bude určovat co nejpřesnější výstup. Přístupů k učení je několik, pro segmentaci se však nejčastěji využívá tzv. učení s učitelem (supervised learning), pro které je potřebné mít takový trénovací dataset, kde je kromě vstupních dat k dispozici i očekávaný výstup.

3.4.1 Průběh učení

Nejprve je nutné v síti inicializovat všechny váhy neuronů a konvolučních jader na počáteční hodnoty, obvykle náhodné s normálním nebo rovnoměrným rozdělením. Neuronová síť je potom trénovaná tak, že skrze ní postupně prochází vstupní data a výstup sítě je porovnáván s očekávaným výstupem.

Porovnání probíhá výpočtem chyby za pomocí chybové funkce, jejíž výsledek určí, jak moc se výstup ze sítě liší od toho očekávaného. Typicky se využívá funkce střední kvadratické odchylky (Mean Square Error) nebo křížové entropie (Cross–entropy), ale je jich i mnoho dalších a výběr záleží na konkrétním využiti sítě.

Na základě vypočtené chyby se potom prochází síť odzadu a upravují se váhy neuronů a konvolučních jader. Ke zpětnému průchodu se využívá algoritmu zpětného šíření, který zpětně prochází vrstvu po vrstvě a vypočítává gradient chybové funkce vzhledem k vahám sítě a jejich vlivu na chybě.

Z vypočteného gradientu se potom za pomocí optimalizačního algoritmu určí, jakým směrem a o jaký krok se mají váhy posunout tak, aby se chyba minimalizovala. Mezi používané optimalizační algoritmy patří gradientní sestup (Gradient Descent) a jeho varianty nebo Adam (Adaptive Moment Estimation) [15] či rozšířený NAdam [5].

Konečná velikost úpravy vah se potom vypočte za pomocí parametru rychlosti učení (learning rate), která vynásobením s krokem z optimalizačního algoritmu určí, jak velká část toho kroku se skutečně provede. Pokud je rychlost učení moc velká, může se stát, že se síť nikdy nedostane k optimálním vahám, protože bude optimální hodnoty přeskakovat. Na druhou stranu pokud je rychlost učení příliš malá, učení bude velmi pomalé a síť se bude muset učit dlouhou dobu. Tento problém je řešen v adaptivních optimalizačních algoritmech (např. zmíněný Adam), které umí hodnotu rychlosti učení dynamicky měnit.

Po zpracování celé množiny datasetu a úpravě vah proběhla jedna trénovací iterace, často pojmenovávána jako epocha. Po dokončení epochy se trénování buď může zastavit nebo může být opakováno. Pro opakované trénování lze použít dataset, který byl použit původně za účelem jeho lepšího naučení, nebo lze použít jiný, aby síť uměla lépe generalizovat. Jiný dataset by ovšem měl být svým obsahem stále dostatečně podobný tomu původnímu, aby na něj síť mohla své již získané znalosti správně uplatnit. K použití jiného datasetu potom může dojít, pokud se síť na původním datasetu již nedokáže zlepšit a jsou od ní požadované lepší výsledky než ty, kterých zatím dosáhla. Další možností pro využití jiného datasetu je v situaci, kde je již naučený model využit pro tzv. přenesené učení (transfer learning). Model se zde nejdříve učil na více generických datech a nový dataset ho potom má specializovat.

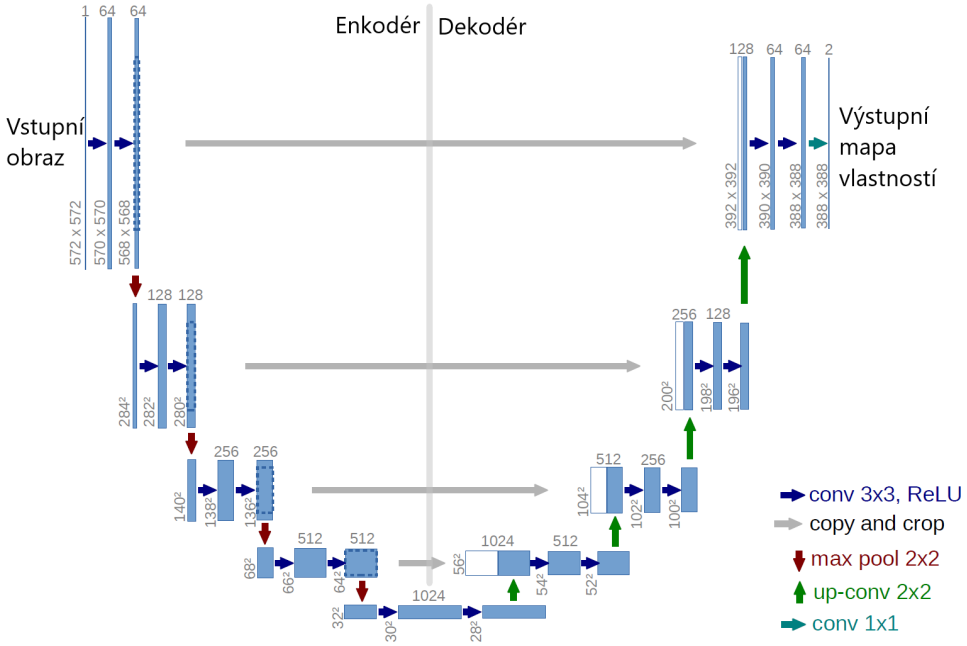
3.5 Nástroje pro realizaci neuronových sítí

Pro realizaci neuronových sítí se v dnešní době převážně používají různé knihovny programovacích jazyků, nejčastěji napsané v jazyce Python (Pytorch, TensorFlow, Keras) nebo C++ (Caffe2, TensorFlow C++ API). Tyto knihovny jsou vybaveny funkcemi pro sestavení neuronových sítí, včetně jejich učení a operací, které se v sítích používají. Se zadavatelem práce jsme se domluvili na využití Python knihovny PyTorch.

4 Analýza vybraných CNN k segmentaci obrazu

4.1 U-Net

U–Net (2015) [20] je konvoluční neuronovou sítí, navrženou pro účely segmentace biomedicínských snímků. Je pojmenovaná podle struktury sítě, která, jak lze vidět na obrázku 4.1, připomíná písmeno „U".



Obrázek 4.1: Struktura sítě U–Net [20].

Hlavní myšlenkou této architektury byla kombinace zmenšovací (contracting) a zvětšovací (expanding) cesty, často označované jako enkodér a dekodér. V enkodéru dochází k postupné redukci rozměru vstupního obrazu a získání kontextu o jeho struktuře. V dekodéru naopak probíhá zpětná rekonstrukce rozlišení, za využití transponovaných konvolucí a tzv. *skip connections* (v obrázku 4.1 šedé šipky), které předávají kontextové informace z průchodu enkodérem. Výsledkem je potom přesnější segmentace, která oproti svým předchůdcům dokáže zachytit mnohem menší detaily.

4.1.1 Enkodér

V enkodéru je za sebou několik bloků, které se skládají ze dvou 3×3 konvolucí, každá následována ReLU aktivační funkcí. Dále se provede 2×2 max–pooling hodnot s velikostí kroku 2 a dojde ke zdvojnásobení počtu kanálů mapy vlastností.

4.1.2 Dekodér

Dekodér je obdobně sestavený z bloků. V každém z nich dochází k převzorkování mapy vlastností za užiti 2×2 transponovaných konvolucí, během které se sníží počet kanálů na polovinu. Mapa vlastností se dále spojí s mapou, která vznikla na stejné úrovni v enkodéru. Stejně jako v enkodéru se zde využije dvou 3×3 konvolucí následovanými aplikací ReLU. V posledním kroku dojde k 1×1 konvoluci, během které se namapuje všech 64 kanálů s vlastnostmi na požadované třídy.

4.1.3 Chybová funkce a optimalizační algoritmus

Architektura v publikaci jako chybovou funkci používá křížovou entropii a vybraným optimalizačním algoritmem je stochastický gradientní sestup.

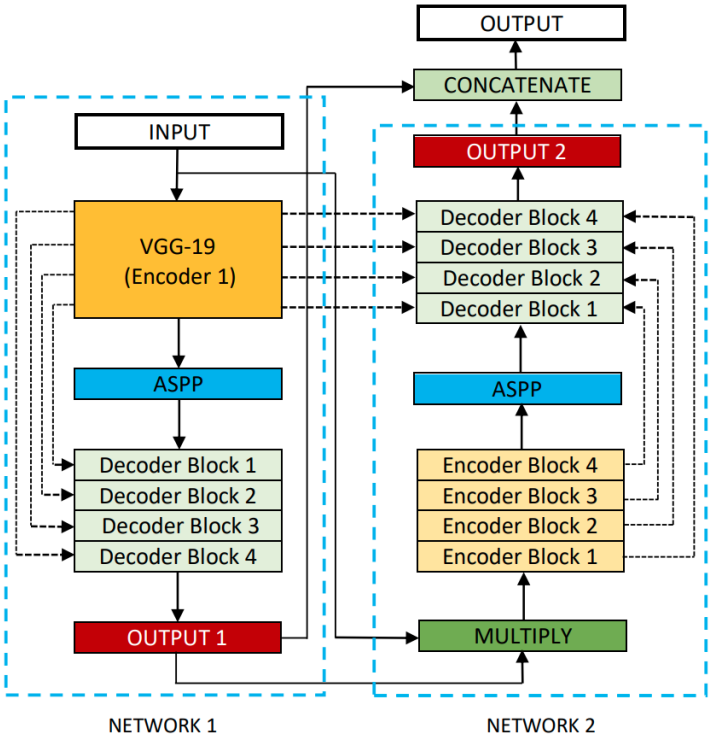
4.2 DoubleU–Net

DoubleU–Net (2020) [14] vychází z původní U–Net architektury. Skládá se ze dvou sekvenčně propojených sítí, založených na U–Net architektuře. Propojení a celou strukturu sítí lze vidět na obrázku 4.2.

Původní U–Net architektura je v obou sítích mírně poupravena. V první sítí se levá část U–Net architektury nahrazuje před–trénovanou VGG-19 [24] sítí. Důvod k tomu je takový, že VGG–19 model je mezi před–trénovanými sítěmi méně výkonově náročný, a v porovnání s enkodérem z U–Netu má větší hloubku, díky které dosahuje lepší výsledné segmentace.

Enkodéry a dekodéry se v obou sítích spojují tzv. *Atrous Spatial Pyramid Pooling* [25] (ASPP) modulem. Ten je složený z několika konvolucí s různě rozšířeným jádrem a následným poolingem výstupů. Tímto způsobem lze extrahovat informace o místě v obrazu a jeho blízkého okolí na různých škálách a následně je kombinovat.

Další změnou je využití tzv. *Squeeze–and–Excite* [11] (SE) modulu, který je použit na konci enkodéru první sítě a dekodérů obou dvou sítí. V síti potom umožňuje adaptivně zvýšit váhy informací, které mají větší dopad



Obrázek 4.2: Struktura DoubleU–Net architektury [14].

na výsledek segmentace, a snížit váhy těch, které nejsou relevantní. Tento proces využívá různých lineárních transformací a aktivačních funkcí.

Zajímavostí této architektury je i to, že vstupní data pro druhou síť se vytváří vynásobením výstupní mapy vlastností první sítě s původními vstupními daty. V posledním kroku se výstupy první a druhé sítě spojí a tím vzniká finální výstup, který je po zkombinování přesnější.

4.2.1 Enkodér

Enkodér první sítě je zmíněná VGG–19 síť. Druhý enkodér se celkově skládá ze čtyř bloků. V každém z nich se provedou dvě 3×3 konvoluce, po každé se navíc provede normalizace a aplikuje se ReLU. Po konvolucích se aplikuje SE modul pro zvýšení kvality mapy vlastností. Nakonec se provede 2×2 max–pooling s velikostí kroku 2.

4.2.2 Dekodér

Architektura využívá dvou dekodérů. Každý blok dekodéru provádí 2×2 bi–lineární převzorkování, čímž zdvojnásobuje dimenzi mapy vlastností. Podobně jako v U–Netu se zde využívá *skip–connections*. V první sítí se chovají

stejně jako v U–Netu, ale v té druhé se dodatečně využívají i mapy z první sítě. Po spojení map vlastností se stejně jako v enkodéru provedou konvoluce, normalizace a ReLU. Po všech dekodér blocích se využije SE blok a finální konvoluce se sigmoid aktivační funkcí, která vygeneruje výstup dané sítě.

4.2.3 Chybová funkce a optimalizační algoritmus

Architektura DoubleU–Netu v publikaci použila buďto kombinaci chybové funkce křížové entropie a optimalizačního algoritmu NAdam nebo chybová funkce využívající Dice koeficientu (viz sekce 6.1.3) s algoritmem Adam. Která kombinace vedla k lepším segmentačním výsledkům je závislé na použitém datasetu, v testovací části se proto vyzkouší obě možnosti.

4.3 ResUNet++

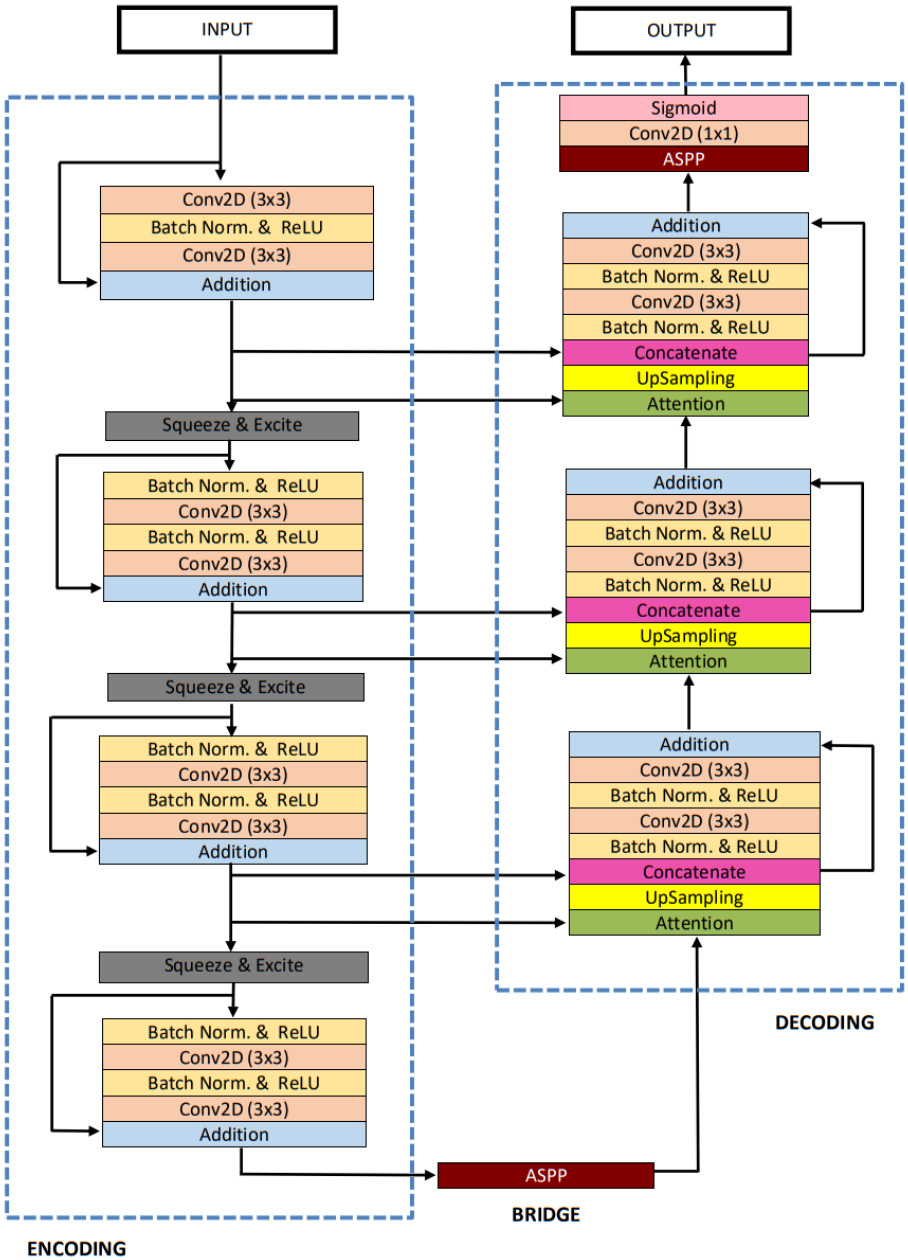
ResUNet++ (2019) [13] architektura rozšiřuje ResUNet [27], který kombinuje prvky ze síťových architektur ResNet [9] a U–Net. Přesněji využívá reziduálních spojení z ResNetu a *skip–connections* z U–Netu. Reziduální spojení pracují na podobném principu jako *skip–connections*, kde se informace z předcházejícího nebo dokonce aktuálního bloku, užívá pro zpětné získání informací, které mohly během průchodu blokem zaniknout. Celou architekturu sítě lze vidět na obrázku 4.3.

Obdobně jako u DoubleU–Netu se zde enkodér a dekodér spojuje za využití ASPP modulu. Dalším přidaným modulem je tzv. *attention* modul v dekodéru, který podobně jako SE modul adaptuje váhy důležitých informací. Na rozdíl od lineárních transformací se v něm ovšem využívá Softmax aktivační funkce nebo různých metrik podobnosti vektorů.

4.3.1 Enkodér

Na začátku enkodéru je, autory pojmenovaný, „stem blok", ve kterém proběhne 3×3 konvoluce, následovaná normalizací, ReLU a další 3×3 konvolucí. Výstup stem bloku je potom součet výstupu druhé konvoluce a původního vstupu.

Za stem blokem následují tři enkodér bloky. Skládají se ze tří dvakrát po sobě opakujících se vrstev: normalizace, aplikace ReLU a 3×3 konvoluce. Na konci bloku se sečte původní vstup do onoho bloku s výstupem poslední konvoluce. Výstup každého bloku je poté ještě předán SE modulu.



Obrázek 4.3: Struktura ResUNet++ architektury [13].

4.3.2 Dekodér

Dekodér se skládá ze tří dekodér bloků. Začínají *attention* modulem, následovaným převzorkováním mapy vlastností za užiti metody nejbližšího souseda. Mapa se dále za využití *skip–connection* spojí s příslušnou mapou z enkodéru. Následuje v podstatě stejná struktura jako v enkodéru. Opět se i sčítá původní vstup s výstupem poslední konvoluce, jediný SE modul se zde nevyužívá. Po posledním bloku výstup projde ASPP modulem, 1×1 konvolucí a konečnou sigmoid funkcí, ze které vznikne konečný výstup.

4.3.3 Chybová funkce a optimalizační algoritmus

Pro architekturu ResUNet++ byl v publikaci jako chybová funkce využit Dice koeficient a vybraným optimalizačním algoritmem byl Adam.

5 Návrh a realizace

V této kapitole je popsána implementace analyzovaných CNN do výsledného programu, který uživateli umožňuje sítě natrénovat a následně využít pro segmentování. V sekcích této kapitoly budou popsány použité knihovny, struktura programu samotného a funkcionalita jednotlivých částí.

5.1 Použité knihovny

Kromě knihovny Pytorch[1] byly v programu použité následující knihovny:

Torchmetrics

Torchmetrics[2] je kolekce Pytorch implementací různých metrik, které se využívají pro vyhodnocování výkonu modelu během trénování či testování.

Pillow

Knihovna Pillow[3] umožňuje jednoduchou práci a manipulaci s obrázky v různých formátech. Základní funkcionalitou, která se v programu používá je pouze načítání obrázků, ale poskytuje i mnoho jednoduchých možností transformace či přímého kreslení na obrázek.

Albumentations

Albumentations[4] je knihovna, která implementuje mnoho různých operací a metod pro augmenmtaci a úpravu obrazu. Tato knihovna je přímo zaměřena na využití u počítačového vidění a proto nabízí více lépe optimalizovaných transformací než Pillow.

NumPy

Knihovna NumPy[5] je jednou z nejpoužívanějších Python knihoven, která poskytuje obrovské množství matematických funkcí a efektivní manipulaci s datovými poli. Velká část této knihovny je napsaná v jazyce C s velmi optimalizovaným kódem, což knihovně umožňuje dosáhnout vysokého výkonu.

[1] pytorch.org
[2] torchmetrics.readthedocs.io/en/latest
[3] pillow.readthedocs.io/en/stable
[4] albumentations.ai
[5] numpy.org

tqdm

Tqdm[6] je malou knihovnou, umožňující vizualizaci průběhu programového cyklu do podoby progress baru. Trénování sítě obecně zabere nezanedbatelný čas a proto je tato knihovna v programu využita pro vizualizaci trénovací smyčky.

5.2 Struktura programu

Jak již bylo zmíněno, program je realizován v jazyce Python za využití knihovny PyTorch. Počáteční struktura programu byla převzata z github repozitáře implementace U–NETu s moduly k jejímu tréninku dostupné na adrese:

`github.com/aladdinpersson/Machine-Learning-Collection`,

pod složkovou strukturou `ML/Pytorch/image_segmentation/`. Program byl dále postupně rozšiřován a upravován pro potřeby práce. Výsledný program je dostupný na adrese:

`github.com/Mariok123/Image-segmentation-using-selected-CNNs`

a skládá se z několika modulů, které lze rozdělit podle jejich funkcionality a obsahu:

- Načítání dat – `dataset.py`
- Trénování modelu – `train.py`
- Mechanismus předčasného zastavení – `early_stopping.py`
- Samostatná segmentace – `predict.py`
- Implementace architektur modelů – `doubleunet_model.py`, `resunetpp_model.py` a `unet_model.py`
- Specifické moduly použité ve více architekturách – `modules.py`
- Funkce používané v různých bodech programu – `utils.py`

V následujících několika sekcích budou jednotlivé moduly a jejich funkcionality přiblíženy.

[6] tqdm.github.io

5.2.1 Načítání dat

Pro efektivní načítání dat a následnou práci s nimi, nabízí knihovna Pytorch dvě datová primitiva: `Dataset` a `Dataloader`.

Třída `Dataset` je abstraktní třídou, s konkretizací v modulu `dataset.py` a definuje, jak má být načtený dataset strukturován, jak má být zajištěn přístup k jeho jednotlivým položkám a případně jaké augmentace se na přistupovaná data aplikují. Načtená data mohou být tvořena například obrázky nebo textovými daty. Celý dataset se navíc většinou rozděluje do menších datasetů určených pro trénování, validaci a testování, a to většinou v poměru kolem 80%, 10% a 10%. Protože se data načítají do paměti a jejich velikost může převyšovat dostupnou paměť, často se zde využívá tzv. líné načítání (lazy loading), za pomoci kterého se do paměti načítají pouze data, která jsou aktuálně potřeba.

Třída `Dataloader` pak konkrétní instanci `Dataset`u obalí do objektu, přes který lze iterovat a podle specifikací je i načítat. Načítání zde lze ovlivnit různými způsoby, například načítat data po dávkách o specifikované velikosti (batch size), zamíchání dat po průchodu celým datasetem (shuffle), nebo dokonce určit počet procesů pro paralelní načítání dat.

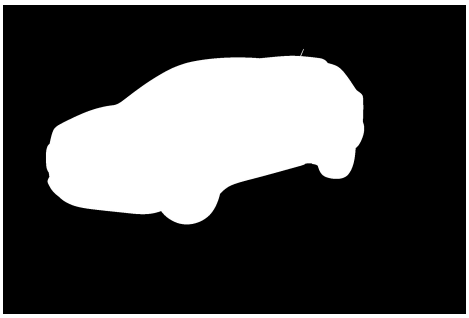
Použité datasety

Sítě byly učeny a testovány na dvou různých datasetech. Prvním z nich je dataset od firmy Carvana, dostupný po registraci z webových stránek kaggle na adrese:

`kaggle.com/c/carvana-image-masking-challenge`.

Jedná se o sadu fotografií různých modelů automobilů na otáčecí plošině, díky níž je každý automobil zachycen celkově ze 16 úhlů. V trénovací sadě je potom celkem 5088 snímků se stejným množstvím binárních obrázků segmentačních masek. Cílem segmentace je zde právě automobil. Na obrázku 5.1 je uveden příklad z Carvana datasetu.

(a) Původní snímek



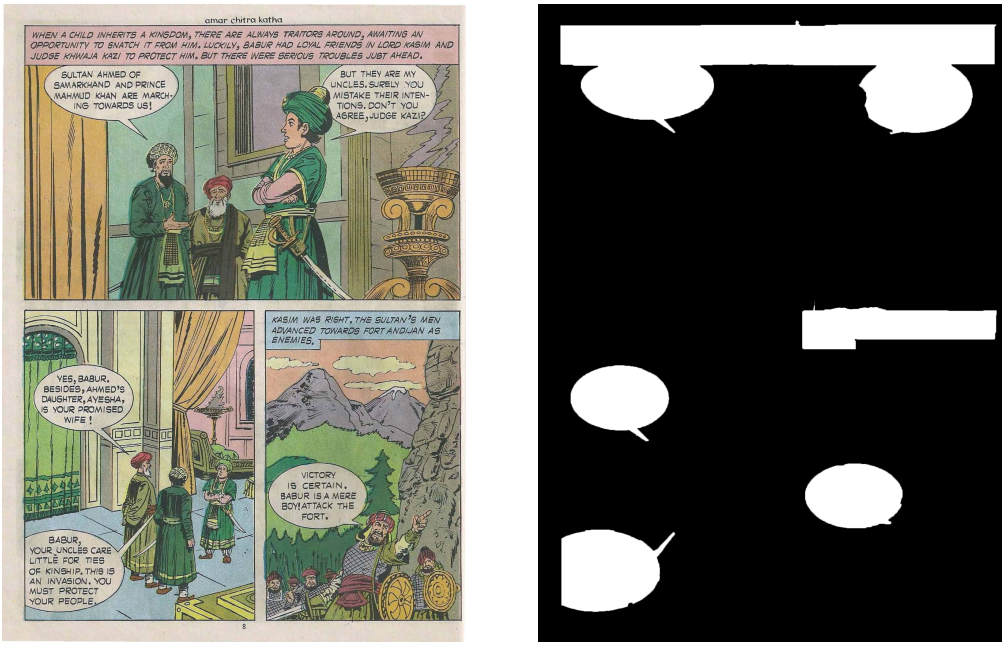
(b) Segmentační maska

Obrázek 5.1: Příklad snímku a masky z Carvana datasetu.

Druhý testovaný dataset byl původně poskytnut zadavatelem. Jedná se o menší sbírku digitalizovaných komiksů o indických mýtech, kde jsou cílem segmentace textové bubliny. Dataset je pojmenován IMCDB (Indian Mythological Comic Database) a originálně byl použit v publikaci zaměřující se na konverzi komiksů do audio–vizuální podoby [8]. Jedním z autorů byl následně poskytnut na jeho githubu, dostupného na adrese:

`github.com/gesstalt/IMCDB`.

Součástí datasetu jsou obrázky stránek komiksů a segmentační masky textových bublin. Kromě obrázků dataset obsahuje i textové soubory s lokacemi a obsahem samotných bublin, pro potřeby práce však nejsou důležité. Celkově má dataset 516 validních párů obrázků a masek. Validních z toho důvodu, že pro některé komiksy či masky chybí komiks nebo maska do páru. Na obrázku 5.2 je uveden příklad z tohoto datasetu.



(a) Původní stránka (b) Segmentační maska

Obrázek 5.2: Příklad komiksu a masky z IMCDB datasetu.

Kromě výše popsaných datasetů, pro které je v programu načítání přímo specifikováno, by měl být program schopen načíst i jakýkoliv dataset, který má obrázky a masky v oddělených složkách a při spuštění jsou k nim cesty specifikovány v argumentech při spuštění.

Augmentace dat

Pro větší diversifikaci trénovacích dat, se použitá data většinou s určitou pravděpodobností různými způsoby modifikují. Tím se množství trénovacích dat uměle navýší a umožní se síti naučit i více neobvyklé příklady. Mezi možné modifikace patří například zvětšení či zmenšení obrazu, rotace, zrcadlení nebo přidáni šumu. Modifikovaná data by však stále měla být dostatečně rozeznatelná, jinak by mohly mít negativní dopad na trénování.

5.2.2 Trénování modelu

Trénování modelu probíhá tak, jak bylo popsáno v sekci 3.4.1. Po každé trénovací epoše se navíc provede průběžná kontrola výkonu na validačním datasetu. V té se modelu předkládají data jako v trénovací smyčce, ale na datasetu, který síť nezná a nedochází ani k zpětnému šíření chyby. Během této kontroly se navíc vypočtou různé metriky popsané v sekci 6.1 a uloží

se do souboru pro případnou pozdější analýzu. Chyba která se během této kontroly vypočte se potom může využít pro určení jestli trénováním nezastavit za pomoci předčasného zastavení. Bližší popis předčasného zastavení je v následující sekci.

Trénování samotné probíhá v modulu `train.py`, který slouží jako jeden z možných vstupních bodů programu a má několik upravitelných hyperparametrů, kterými lze trénování ovlivnit:

- `LEARNING_RATE` – rychlost učení sítě
- `DEVICE` – určuje zdali má trénování probíhat na GPU nebo CPU
- `BATCH_SIZE` – velikost dávky dat v trénovací smyčce
- `NUM_WORKERS` – počet procesů pro načítání dat
- `IMAGE_HEIGHT` – výška do které se obraz transformuje před naučením
- `IMAGE_WIDTH` – šířka do které se obraz transformuje před naučením
- `EARLY_STOP_PATIENCE` – po kolika epochách nezlepšení validační chyby se ukončí trénování
- `DATASET_SPLIT` – jak velká část datasetu se použije pro trénování, zbylá část je potom použita pro validaci

K šířce a výšce do které se obraz transformuje je vhodné podotknout, že se primárně používá za účelem konzistentní velikosti obrazu, ke snížení potřebné výpočetní paměti a celkovému urychlení trénování. Je nutné si uvědomit, že změna velikosti vede ke ztrátě a zkreslení informací a detailů v obrazu, ale zároveň může síti pomoci lépe generalizovat. Při volbě hodnot by se proto ideálně měl alespoň skoro zachovat poměr stran původních dat.

5.2.3 Předčasné zastavení

Předčasné zastaveni je technika, která se využívá pro minimalizaci přeučení sítě, což je stav, kdy se síť naučila trénovací dataset moc konkrétně a při použití na jiných datech své znalosti nedokáže správně uplatnit. Princip spočívá v průběžném vyhodnocování chyby sítě na validačním datasetu po dokončení každé epochy. Jakmile se chyba na validačním datasetu po několika epochách nezlepší, trénování se zastaví a do modelu se zpětně načtou váhy, které dosáhly nejlepších výsledků. Typicky lze specifikovat, po kolika epochách nezlepšování se má trénování zastavit a o kolik by se měla chyba snížit

než je model brán jako že se nezlepšil. Implementace třídy s touto funkcionalitou je v modulu `early_stopping.py` a byla k tomu s menší modifikací využita implementace dostupná u kurzu o aplikaci neuronových sítí [10] na adrese:

`github.com/jeffheaton/t81_558_deep_learning/tree/pytorch`.

5.2.4 Samostatná segmentace

Samostatnou segmentací je myšleno využiti již natrénovaného modelu k segmentaci uživatelem určených dat. Segmentace probíhá na jednotlivých obrázcích a výsledné segmentační masky jsou ukládány do složky podle toho, jaký model byl použit. Je k tomu využit modul `predict.py`, který zároveň slouží jako druhý možný vstupní bod programu a obdobně jako u modulu pro trénování má několik hyperparametrů, kterými lze predikci ovlivnit:

- `DEVICE` – určuje zdali má segmentace probíhat na GPU nebo CPU
- `NUM_WORKERS` – počet procesů pro načítání dat
- `IMAGE_HEIGHT` – výška do které se obraz transformuje před predikcí a vlastně i výška výsledné segmentace
- `IMAGE_WIDTH` – šířka do které se obraz transformuje před predikcí a vlastně i šířka výsledné segmentace

Opět je zde vhodné zmínit něco k velikostem. Měli by být ideálně nastavené na stejnou velikost jako ta, na které se model učil. Segmentování jinak sice může proběhnout, ale výsledky mohou být značně horší.

5.2.5 Implementace modelů

Publikace o architekturách mnohdy vynechávají některé detaily, které jsou důležité pro správnou implementaci. Může to být například počet kanálů, který má vzniknout po jednotlivých konvolucích nebo jak přesně jsou implementovány některé moduly, které se v architektuře použily (např. ASPP či SE modul). Naštěstí autoři publikací svou implementaci často poskytují k dispozici na github, odkud byly vybrané implementace i využity. Problém je, že implementace nemusí být vždy napsané v knihovně Pytorch, ale třeba, jak tomu bylo u těch použitých v této práci, v knihovně TensorFlow. Pytorch implementaci autoři poskytli až dodatečně, bohužel s občasnými menšími nesrovnalostmi v porovnání s původní implementací. Proto bylo nutné implementace porovnat a případně upravit. Výsledné struktury implementovaných modelů potom odpovídají tomu, jak byly popsány v kapitole 4.

U–Net

Implementace modelu U–Net dostupná u programu, který byl využit jako počáteční struktura, byla po bližším seznámení se s knihovnou Pytorch od základu přepsána tak, aby více odpovídala kódové struktuře ostatních dvou modelů.

DoubleU–Net

Pro implementaci DoubleU–Netu modelu byl využit kód z github repozitáře dostupného na adrese:

`github.com/DebeshJha/2020-CBMS-DoubleU-Net`.

Výsledná struktura kódu byla lehce upravena a byl předělán výstup ze sítě, kde v Pytorch implementaci od autorů chybělo spojení výstupů dvou vnitřních sítí do výstupu finálního.

ResUNet++

Pro implementaci ResUNet++ modelu byla využita implementace z repozitáře na adrese:

`github.com/DebeshJha/ResUNetPlusPlus`.

Kód byl opět mírně přestrukturován, ale žádné jiné zásadní změny v něm provedeny nebyly.

5.2.6 Utilitní funkce

V modulu `utils.py` jsou funkce převážně řešící vstup a různé výstupy z programu. Jsou zde funkce pro obsluhu uživatelem zvolených argumentů při spuštěni programu, správné přiřazení chybové funkce a optimalizačního algoritmu podle vybraného modelu sítě a vytvoření `Dataloader`u z vybraného datasetu. Dále je v modulu funkce pro průběžnou kontrolu výkonu trénovaného modelu a několik funkcí pro ukládání výsledků trénování či samostatné segmentace.

6 Vyhodnocování výsledků

V této kapitole budou vyhodnoceny výsledky získané trénováním jednotlivých modelů na dříve popsaných datasetech a metriky, které byly k vyhodnocení použity.

6.1 Použité metriky

Metrik k vyhodnocení výsledků a kvality segmentace existuje velké množství [12]. Vzhledem k tomu, že segmentace probíhá klasifikací jednotlivých pixelů, většina z nich vychází z porovnávání pixelů výstupní masky a referenční masky. Je proto vhodné si nejdříve definovat několik případů, které mohou pro pixel nastat:

- True–Positive (TP): pixel je klasifikován stejnou třídou jak ve výstupu tak v referenci.
- True–Negative (TN): pixel do třídy nepatří ani ve výstupu ani v referenci.
- False–Positive (FP): pixel není třídou označen ve výstupu, ale v referenci je.
- False–Negative (FN): pixel je třídou označen ve výstupu, ale v referenci ne.

Dále budou popsány metriky, které byly k vyhodnocování použity.

6.1.1 Přesnost

Přesnost (accuracy) svou podstatou patří mezi nejjednodušší metriky hodnocení výsledku segmentace. Jedná se o poměr správně klasifikovaných pixelů (tedy TP a FP) k počtu všech pixelů. Zápis by vypadal takto:

$$accuracy = \frac{TP + FP}{TP + TN + FP + FN}. \tag{6.1}$$

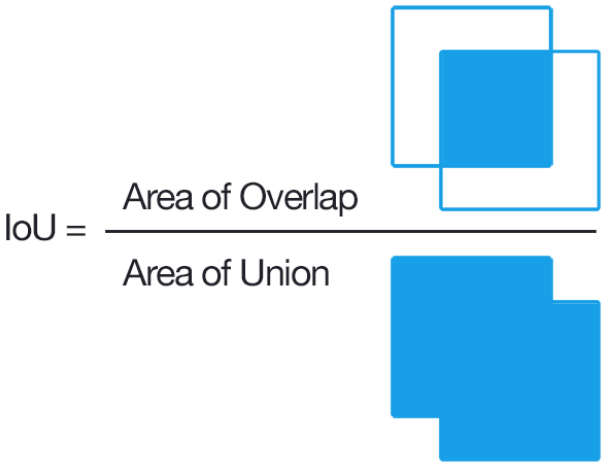
Přestože je tato metrika velmi intuitivní, není vhodná pro případy, kde jsou segmentované objekty pouze malou částí celého obrazu, neboli třídy nejsou vyvážené. Velká část referenční masky může patřit do neobjektové třídy a i kdyby síť nefungovala a ve výsledku nic nesegmentovala, přesnost by stále zůstala velká.

6.1.2 Jaccardův index

Jaccardův index, také známý jako Intersection over Union (IoU), je metrika která počítá podobnost mezi segmentovanou maskou a referenční maskou. Počítá se jako poměr plochy překryvu segmentované masky s referenční maskou a spojení těchto dvou masek. Jedná se o metriku, která je k vyhodnocování výsledků segmentace používána asi nejčastěji a to díky své jednoduchosti a nezávislosti na vyváženosti tříd. Zápis vypadá následovně:

$$IoU = \frac{TP}{TP + FP + FN}. \tag{6.2}$$

Na obrázku 6.1 je potom výpočet dodatečně znázorněný.



Obrázek 6.1: Vizualizace Jaccardova indexu [21]

6.1.3 F1–skóre (Dice koeficient)

F1–skóre (někdy Dice koeficient) je metrika definována jako harmonický průměr mezi precizností (precision) a úplností (recall) segmentace. Preciznost lze popsat jako poměr počtu TP pixelů vůči celkovému počtu pixelů co mělo být klasifikováno (tedy TP a FP pixelů), kde zápis vypadá takto:

$$precision = \frac{TP}{TP + FP}. \tag{6.3}$$

Úplnost je poměr TP pixelů k celkovému počtu pixelů co bylo klasifikováno (tedy TP a FN pixely), zápis je následující:

$$recall = \frac{TP}{TP + FN}. \tag{6.4}$$

Konečné F1–skóre pak lze vypočítat tímto způsobem:

$$F1 = 2 * \frac{precision * recall}{precision + recall} = \frac{2 * TP}{2 * TP + FP + FN}. \tag{6.5}$$

Obdobně jako Jaccardův index, F1–skóre není závislé na vyváženosti tříd a je proto používané poměrně často.

6.2 Dosažené výsledky

V tabulce 6.1 jsou vypsány hyperparametry užité při trénování sítí na datasetu Carvana, v tabulce 6.2 jsou pak použité hyperparametry pro trénování na IMCDB datasetu. U každého trénování bylo využito předčasného zastavení, pro testovací účely se však při jeho aktivaci učení nezastavilo, pouze se uložil extra model s aktuálními vahami. Důvodem k tomu bylo získání informace o vývoji výkonu přes stejné množství epoch u všech modelů. Pro Carvana dataset byly modely trénovány po dobu 30 epoch, u IMCDB datasetu to bylo epoch 60. Množství epoch bylo zvoleno po několika testech, ze kterých bylo zjištěno, že se trénování modelů většinou zastavilo samovolně dříve díky předčasnému zastavení, nebo se výkon zlepšoval již pouze o téměř zanedbatelné hodnoty.

Trénování probíhalo na grafické kartě NVIDIA GeForce RTX 2060 s 6GB paměti. Z důvodu nedostatečně velké paměti, z pohledu trénování sítí za účelem segmentace, byly hyperparametry `BATCH_SIZE` a velikosti učených obrázků značně omezeny.

Hyperparametr	**Hodnota**
`LEARNING_RATE`	1e-4
`BATCH_SIZE`	16
`NUM_WORKERS`	2
`IMAGE_HEIGHT`	160
`IMAGE_WIDTH`	240
`EARLY_STOP_PATIENCE`	3
`DATASET_SPLIT`	0.8

Tabulka 6.1: Použité hyperparametry pro trénování sítí na Carvana datasetu.

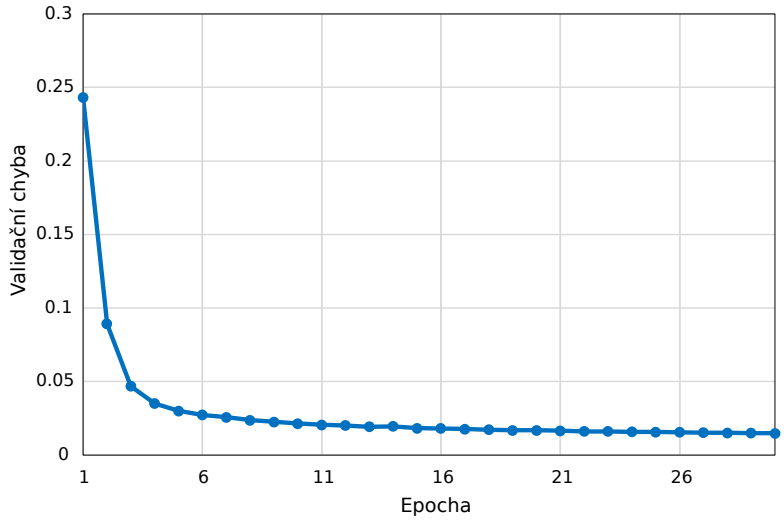
Hyperparametr	Hodnota
LEARNING_RATE	1e-4
BATCH_SIZE	8
NUM_WORKERS	2
IMAGE_HEIGHT	320
IMAGE_WIDTH	240
EARLY_STOP_PATIENCE	3
DATASET_SPLIT	0.8

Tabulka 6.2: Použité hyperparametry pro trénování sítí na IMCDB datasetu.

6.2.1 Výsledky U–Netu

Carvana dataset

Na obrázku 6.2 lze vidět vývoj validační chyby během učení. V tabulce 6.3 jsou zapsané dosažené výsledky s dříve popsanými metrikami. Dodatečně tabulka obsahuje i průměrnou dobu jedné epochy. Na obrázcích 6.3 je pak uveden příklad segmentace.



Obrázek 6.2: Vývoj validační chyby U–Netu na Carvana Datasetu.

Číslo epochy	Průměrný čas epochy[s]	Přesnost [%]	F1–skóre	Jaccard
30	148.56	99.43	0.9865	0.9734

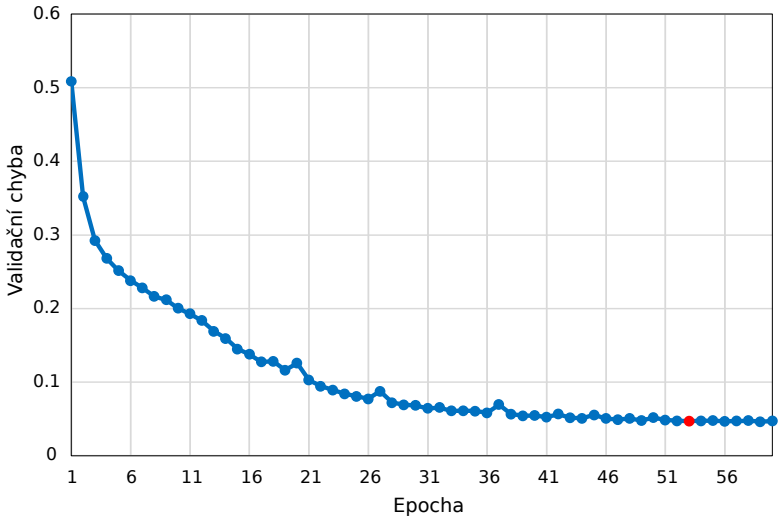
Tabulka 6.3: Dosažené výsledky U–Netu na Carvana datasetu.

(a) Segmentovaný obraz (b) Referenční maska (c) Výsledek segmentace

Obrázek 6.3: Příklad segmentace Carvana datasetu U–Netem.

IMCDB dataset

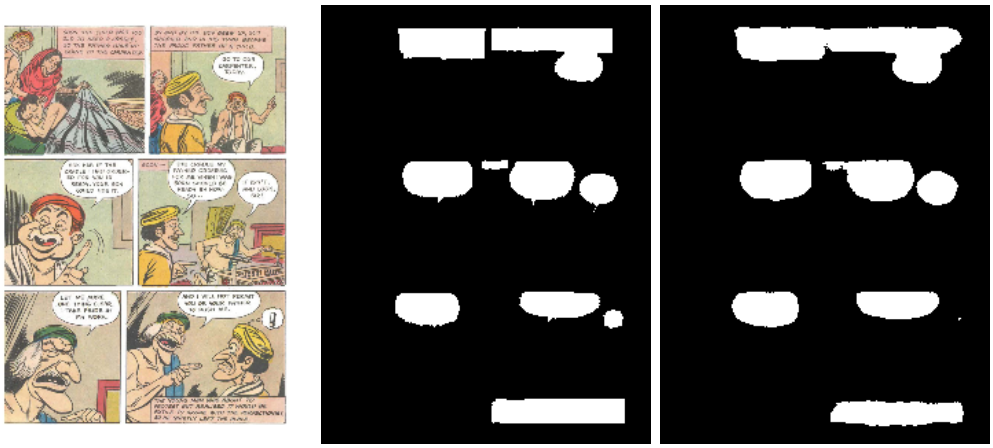
Na obrázku 6.4 je graf vývoje validační chyby během učení, červeně označený bod je místo, kde se aktivovalo předčasné zastavení. Výsledky v tabulce 6.4 jsou potom ty, kterých bylo dosaženo v tomto bodu a navíc na konci učení. Výkon se pokračováním učení změnil pouze minimálně. Na obrázcích 6.5 je pak uveden příklad segmentace.



Obrázek 6.4: Vývoj validační chyby U–Netu na IMCDB datasetu.

Číslo epochy	**Průměrný čas epochy[s]**	**Přesnost [%]**	**F1–skóre**	**Jaccard**
53	44.45	98.43	0.932	0.8736
60	44.51	98.39	0.9321	0.8738

Tabulka 6.4: Dosažené výsledky U–Netu na IMCDB datasetu.



(a) Segmentovaný obraz (b) Referenční maska (c) Výsledek segmentace

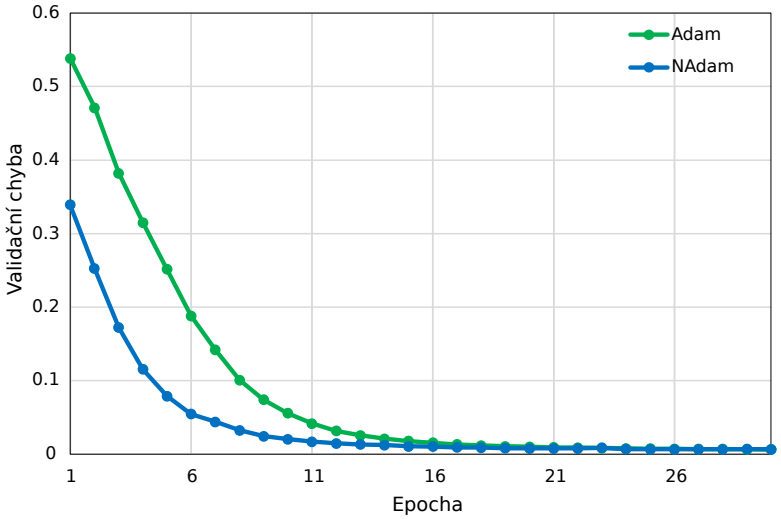
Obrázek 6.5: Příklad segmentace IMCDB datasetu U–Netem

6.2.2 Výsledky DoubleU–Netu

Jak bylo zmíněno v sekci 4.2.3, autoři v publikaci o DoubleU–Netu uvedly dvě možné kombinace chybové funkce a optimalizačního algoritmu pro možné lepší výsledky na různých datasetech. Proto byl model učen na obou datasetech dvakrát, jednou s kombinací chybové funkce křížové entropie a optimalizačního algoritmu NAdam, podruhé s kombinací funkce Dice a algoritmu Adam. Ve výsledcích budou použité kombinace označovány podle použitého optimalizačního algoritmu, tedy NAdam a Adam respektive.

Carvana dataset

Na obrázku 6.6 lze vidět vývoj validační chyby během učení, modře pro kombinaci NAdam, zeleně pro kombinaci Adam. V tabulce 6.5 jsou zapsané dosažené výsledky. Lze si v nich všimnout, že rozdíl mezi kombinacemi je pro tento dataset minimální, ale kombinace Adam se zdá podle metrik o něco lepší, má ovšem znatelně pomalejší čas epochy. Na obrázcích 6.7 jsou pak uvedeny příklady segmentace stejného snímku pro srovnání.



Obrázek 6.6: Vývoj validačních chyb DoubleU–Netu na Carvana datasetu.

	Číslo epochy	**Průměrný čas epochy[s]**	**Přesnost [%]**	**F1–skóre**	**Jaccard**
NAdam	30	188.83	99.73	0.9938	0.9876
Adam	30	209.75	99.75	0.9941	0.9883

Tabulka 6.5: Dosažené výsledky DoubleU–Netu na Carvana datasetu.

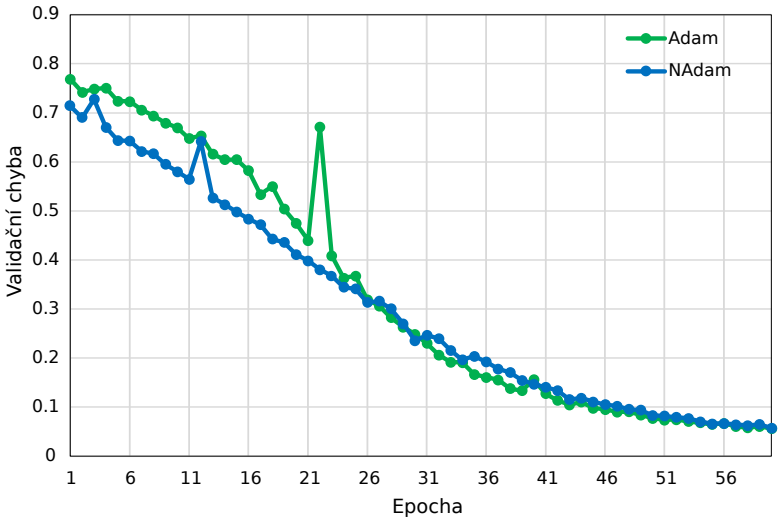
(a) Segmentovaný obraz (b) Referenční maska

(c) Výsledek segmentace pro NAdam kombinaci (d) Výsledek segmentace pro Adam kombinaci

Obrázek 6.7: Příklad segmentace Carvana datasetu DoubleU–Netem.

IMCDB dataset

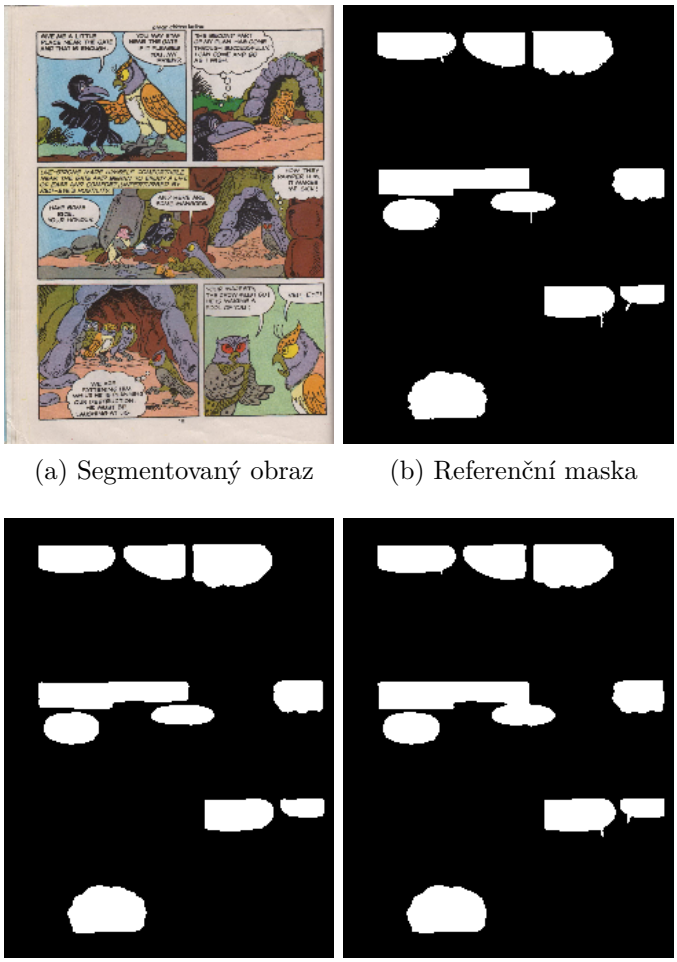
Na obrázku 6.8 je graf vývojů validační chyby během učení, modře pro kombinaci NAdam, zeleně pro kombinaci Adam. V tabulce 6.6 jsou zapsané dosažené výsledky. Rozdíl mezi kombinacemi byl opět docela malý, kombinace Adam se zdá být o trochu lepší i pro tento dataset. Velký rozdíl zde byl opět u času epochy, ale oproti Carvana datasetu byl delší pro kombinaci NAdam. Důvodem by k tomu mohl být menší `BATCH_SIZE` nebo větší velikost učeného obrázku. Na obrázcích 6.9 jsou opět uvedeny příklady segmentace stejného obrazu pro srovnání.



Obrázek 6.8: Vývoj validačních chyb DoubleU–Netu na IMCDB datasetu.

	Číslo epochy	**Průměrný čas epochy[s]**	**Přesnost [%]**	**F1–skóre**	**Jaccard**
NAdam	60	58.11	99.18	0.9614	0.9258
Adam	60	39.4	99.31	0.9681	0.9383

Tabulka 6.6: Dosažené výsledky DoubleU–Netu na IMCDB datasetu.



(a) Segmentovaný obraz (b) Referenční maska

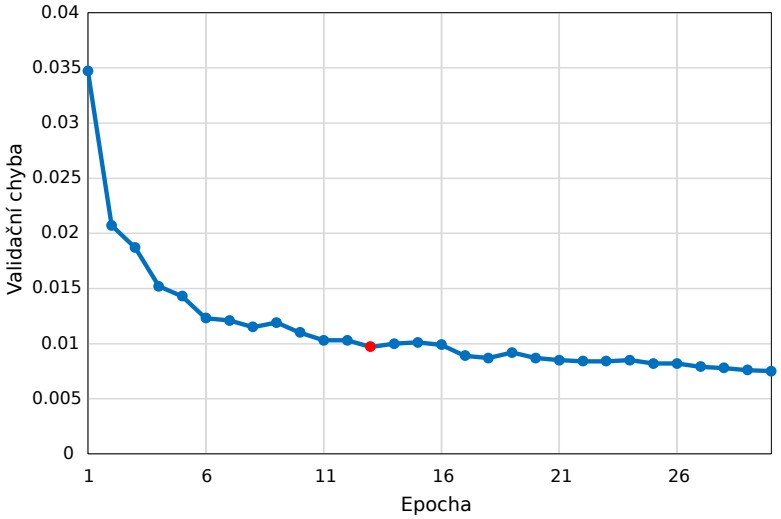
(c) Výsledek segmentace pro NAdam kombinaci (d) Výsledek segmentace pro Adam kombinaci

Obrázek 6.9: Příklad segmentace IMCDB datasetu DoubleU–Netem.

6.2.3 Výsledky ResUNet++ modelu

Carvana dataset

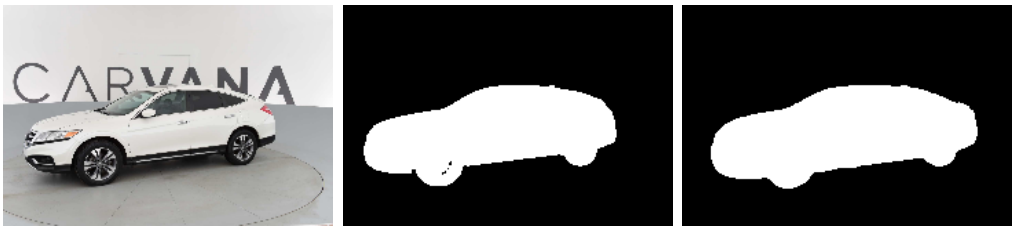
Na obrázku 6.10 lze vidět vývoj validační chyby během učení s červeně označeným bodem, kde se aktivovalo předčasné zastavení. Výsledky v tabulce 6.4 jsou potom ty, kterých bylo dosaženo v tomto bodu a na konci učení. Lze si všimnout, že se výkon za dalších 17 epoch učení sice zvedl, ale pouze o malé množství. Na obrázcích 6.11 je pak uveden příklad segmentace.



Obrázek 6.10: Vývoj validační chyby ResUNet++ modelu na Carvana datasetu.

Číslo epochy	**Průměrný čas epochy[s]**	**Přesnost [%]**	**F1–skóre**	**Jaccard**
13	153.11	99.6	0.9905	0.9812
30	143.65	99.68	0.9925	0.9852

Tabulka 6.7: Dosažené výsledky ResUNet++ modelu na Carvana datasetu.



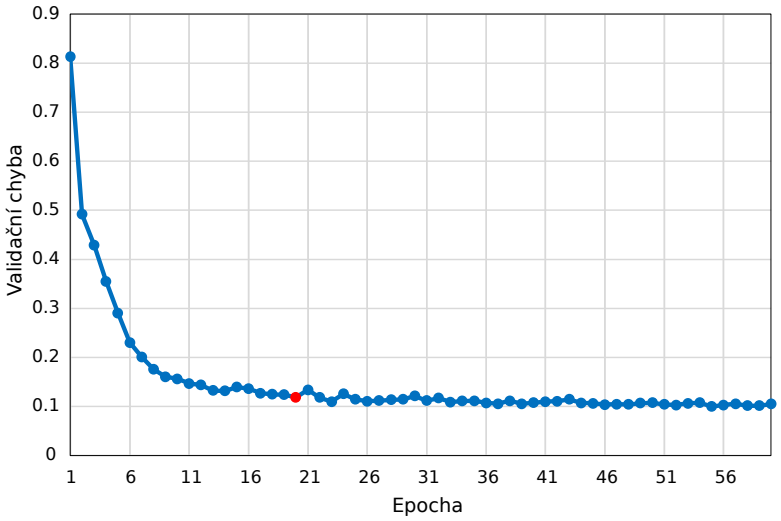
(a) Segmentovaný obraz (b) Referenční maska (c) Výsledek segmentace

Obrázek 6.11: Příklad segmentace Carvana datasetu ResUNet++ modelem.

IMCDB dataset

Na obrázku 6.12 je opět graf validační chyby během učení s červeně označeným bodem, kde se aktivovalo předčasné zastavení. Výsledky v tabulce 6.8 jsou potom ty, kterých bylo dosaženo v tomto bodu a na konci učení. Zde

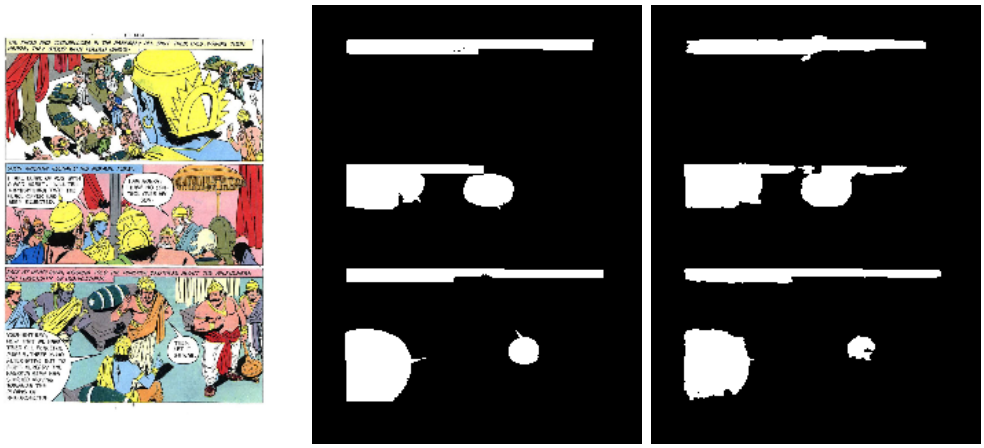
další učení sice vedlo k přijatelnému zlepšení metrik, ale za cenu trojnásobného množství epoch. Na obrázcích 6.13 je uveden příklad segmentace.



Obrázek 6.12: Vývoj validační chyby ResUNet++ modelu na IMCDB datasetu.

Číslo epochy	**Průměrný čas epochy[s]**	**Přesnost [%]**	**F1–skóre**	**Jaccard**
20	37.74	97.53	0.885	0.7977
60	37.69	97.75	0.8962	0.8156

Tabulka 6.8: Dosažené výsledky ResUNet++ modelu na IMCDB datasetu.



(a) Segmentovaný obraz (b) Referenční maska (c) Výsledek segmentace

Obrázek 6.13: Příklad segmentace IMCDB datasetu ResUNet++ modelem

6.3 Srovnání výsledků

6.3.1 Carvana dataset

V tabulce 6.9 jsou sepsány výsledky všech modelů na Carvana datasetu. Pokud se trénování modelu zastavilo předčasně, je u modelu uvedeno ze které epochy výsledky jsou, jinak jsou výsledky z celé doby učení 30 epoch.

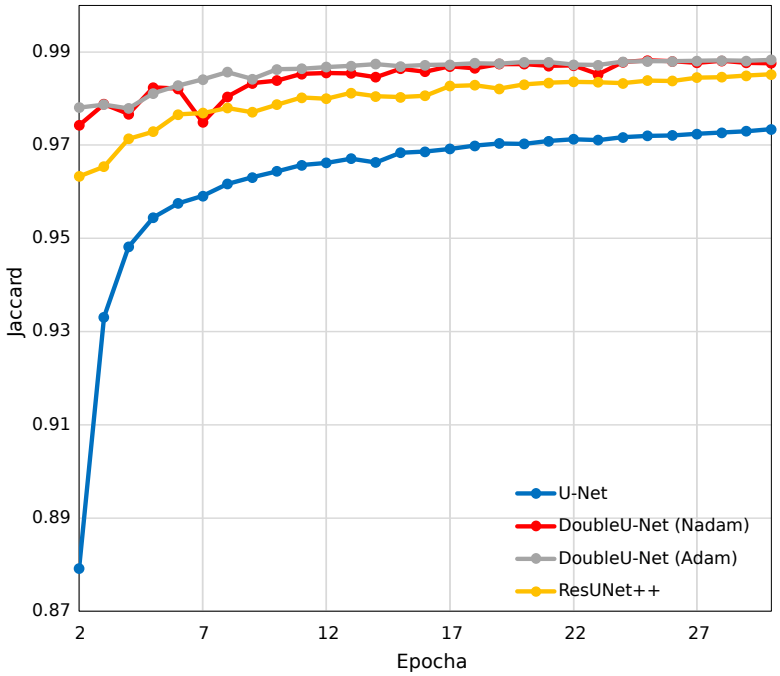
Model	**Průměrný čas epochy**[s]	**Přesnost** [%]	**F1–skóre**	**Jaccard**
U–Net	148.56	99.43	0.9865	0.9734
DoubleU–Net$_{Nadam}$	188.83	99.73	0.9938	0.9876
DoubleU–Net$_{Adam}$	209.75	99.75	0.9941	0.9883
ResUNet++ (13)	153.11	99.6	0.9905	0.9812

Tabulka 6.9: Dosažené výsledky všech modelů na Carvana datasetu.

V tabulce si můžeme povšimnout, že nejlepších výsledků dosáhl model DoubleU–Net s kombinací Adam optimalizačního algoritmu a Dice chybové funkce. Problém je u délky epochy, která je v porovnání se všemi ostatními modely nejhorší.

Zajímavé je, že model ResUNet++ dosáhl lepších výsledků než model U–Net, přestože se učil mnohem menší množství epoch. Protože z výsledků v tabulce není vidět jak množství epoch ovlivní výkon sítě, je v obrázku 6.14 zobrazen graf vývoje Jaccardova indexu jednotlivých modelů. V grafu není

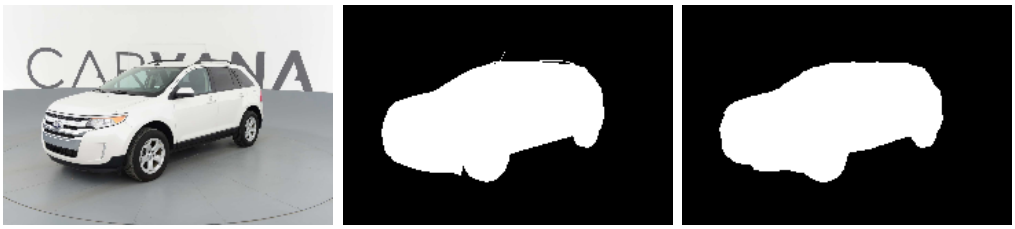
znázorněna první epocha, neboť se v ní váhy sítě teprve nově inicializovaly a výsledek epochy je proto značně zkreslený.



Obrázek 6.14: Vývoj dosaženého Jaccardova indexu všech modelů na Carvana datasetu.

Na grafu je možné vidět, že model U–Net je znatelně horší oproti ostatním modelům, přestože má nejrychlejší průběh jedné epochy. DoubleU–Net se zdá být celkově nejlepší v obou svých verzích, protože už od začátku učení dosahoval lepších výsledků než ostatní modely dosáhly na konci. ResUNet++ je potom svým výkonem uprostřed, ale ke konci 30 epochy se docela blízko přiblížil výkonu DoubleU–Netu a s ohledem na krátší dobu epochy, ho lze považovat za poměrně kompetetivní.

Na obrázcích 6.15 je, pro další srovnání modelů, uveden příklad segmentace stejného snímku všemi modely.



(a) Segmentovaný obraz (b) Referenční maska (c) Výsledek segmentace U–Netem

(d) Výsledek segmentace NAdam kombinace DoubleU–Netu (e) Výsledek segmentace Adam kombinace DoubleU–Netu (f) Výsledek segmentace ResUNet++ modelu

Obrázek 6.15: Příklad segmentace Carvana datasetu všemi modely.

Na těchto obrázcích si můžeme všimnout, že DoubleU–Net s NAdam kombinací dokázal nejlépe segmentovat malé detaily, viz. příčníky na střeše auta a mezera mezi předním kolem a blatníkem.

6.3.2 IMCDB dataset

V tabulce 6.10 jsou výsledky všech modelů na IMCDB datasetu. Pokud se trénování modelu zastavilo předčasně, je u modelu uvedeno, ze které epochy výsledky jsou, jinak jsou výsledky z celé doby učení 60 epoch.

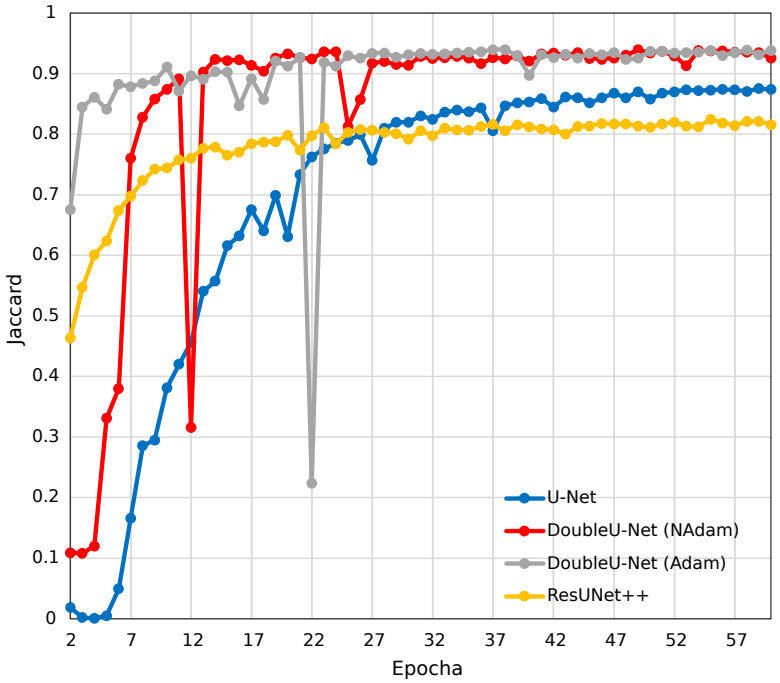
Model	**Průměrný čas epochy**[s]	**Přesnost** [%]	**F1–skóre**	**Jaccard**
U–Net (53)	44.45	98.43	0.932	0.8736
DoubleU–Net$_{Nadam}$	58.11	99.18	0.9614	0.9258
DoubleU–Net$_{Adam}$	39.4	99.31	0.9681	0.9383
ResUNet++ (20)	37.74	97.53	0.885	0.7977

Tabulka 6.10: Dosažené výsledky všech modelů na IMCDB datasetu.

Z tabulky je zřetelné, že i pro tento dataset dosáhl nejlepších výsledků model DoubleU–Net s kombinací Adam optimalizačního algoritmu a Dice

chybové funkce. V kontrastu s výsledky z Carvana datasetu je pro něj i čas epochy druhým nejlepším, překonán pouze ResUNet++ modelem, který je zde nejrychlejší, ale o pouze o necelé dvě vteřiny. Přesto že má ResUNet++ nejrychlejší dobu epochy, dosáhl pro tento dataset nejhorších výsledků.

Na obrázku 6.16 je zobrazen graf vývoje Jaccardova indexu jednotlivých modelů. V grafu není, stejně jako u výsledků pro Carvana dataset, znázorněna první epocha z důvodu zkresleného výsledku.

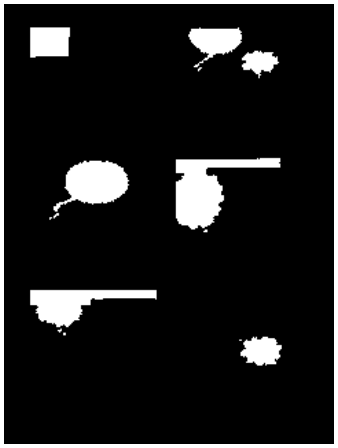


Obrázek 6.16: Vývoj dosaženého Jaccardova indexu všech modelů na IMCDB datasetu.

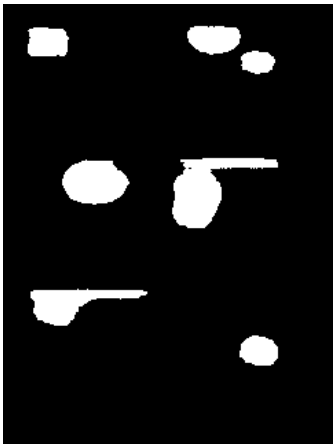
Na grafu je možné vidět, že učení modelů u IMCDB datasetu probíhalo poměrně kostrbatě. Důvodem by k tomu mohlo být poměrně malé množství dat v porovnání s Carvana datasetem. Obě varianty DoubleU–Netu už skoro od začátku mají úplně nejlepší výsledky, ale zároveň obě varianty v jedné epoše zaznamenaly prudký propad ve kvalitě výsledku, který se ale v příští epoše hned opravil. Propad je možná způsoben tím, jak optimalizační algoritmus Adam a NAdam pracuje s menší dávkou dat, která byla pro učení tohoto datasetu zvolena. Přesný důvod se mi ale nepodařilo zjistit, protože ResUNet++, který algoritmus Adam používá také, drastický propad nikde nemá. Zajímavé je, že model U–Net po téměř třiceti epochách učení překonal svými výsledky model ResUNet++, přestože u Carvana datasetu byl svými výsledky značně horší než ostatní modely po celou dobu učení.

Na obrázcích 6.17 je, pro další srovnání modelů, uveden příklad segmentace stejného snímku za využití všech modelů.

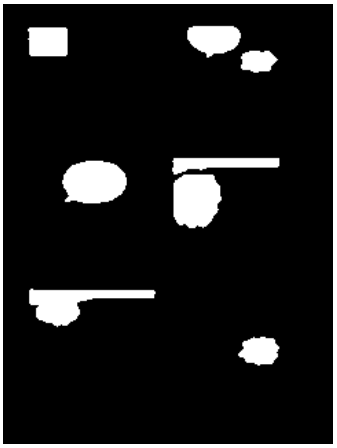
(a) Segmentovaný obraz



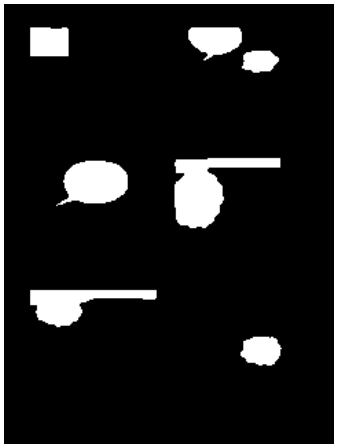
(b) Referenční maska



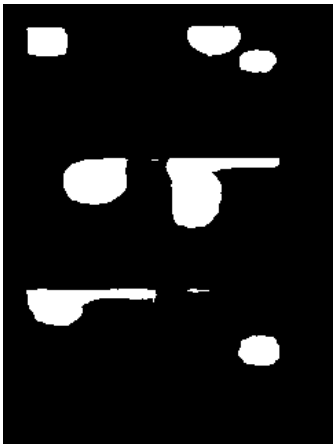
(c) Výsledek segmentace U–Netem



(d) Výsledek segmentace NAdam kombinace DoubleU–Netu



(e) Výsledek segmentace Adam kombinace DoubleU–Netu



(f) Výsledek segmentace ResUNet++ modelu

Obrázek 6.17: Příklad segmentace IMCDB datasetu všemi modely.

Z těchto obrázků je patrné, že složité a ostré tvary textových bublin dělají problém modelům všem. Nejlépe jsou na tom obě kombinace DoubleU–Netu, kterým se nejlépe podařilo zachovat tvar bublin a z některých dokonce segmentovat i část ocásků, které z nich vedou.

7 Závěr

Cílem bakalářské práce bylo seznámení se s problematikou segmentace, neuronových sítí a jejich využití za účelem segmentace. Pro přiblížení segmentace byly představeny metody, které k segmentaci využívají prahování a podobnosti obrazových regionů. Seznámení se s neuronovými sítěmi proběhlo jejich obecným popisem a přehledem o historií jejich využití pro zpracování obrazu. To vedlo ke zjištění, že nejčastěji používaným typem neuronových sítí, které se k segmentaci obrazu používají, jsou konvoluční neuronové sítě. Po dalším seznámení se s nimi, bylo několik architektur konvolučních neuronových sítí, které byly přímo navrženy k segmentaci, analyzováno a následně implementováno do programu, který uživateli umožňuje sítě natrénovat a využít je pro segmentaci na různých datech. Analyzovanými a implementovanými síťovými architekturami byly U–Net (2015), DoubleU–Net (2020) a ResUNet++ (2019).

Tyto architektury byly po implementaci otestovány na dvou různých datových sadách a výsledky testování byly v práci prezentovány. Cílem první datové sady bylo segmentování automobilu ze snímku. Cílem druhé byla segmentace textových bublin z komiksů. Z konečných výsledků vyplynulo, že architektura DoubleU–Net byla pro segmentaci obou datasetů nejlepší variantou, což odpovídá očekávání, neboť architektura je i nejvíce aktuální. Konečné nejlepší výsledky DoubleU–Net architektury s metrikou Jaccardova indexu byly 0.9883 pro Carvana dataset a 0.9383 pro IMCDB dataset.

Literatura

[1] Ahire, J. B. *The Artificial Neural Networks Handbook: Part 4* [online]. 2018. Dostupné z: https://dzone.com/articles/the-artificial-neural-networks-handbook-part-4.

[2] Baheti, P. *Activation Functions in Neural Networks* [online]. V7, 2021. [cit. 2023/05/02]. Dostupné z: https://www.v7labs.com/blog/neural-networks-activation-functions.

[3] Chen, C. et al. A Survey on Deep Learning for Localization and Mapping: Towards the Age of Spatial Machine Intelligence. *ArXiv*. 2020, abs/2006.12567.

[4] Dougherty, G. *Digital Image Processing for Medical Applications*. Cambridge University Press, 2009. doi: 10.1017/CBO9780511609657.

[5] Dozat, T. Incorporating Nesterov Momentum into Adam. 2016.

[6] Fitch, F. B. McCulloch Warren S. and Pitts Walter. A logical calculus of the ideas immanent in nervous activity. Bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133. *Journal of Symbolic Logic*. 1944, 9, s. 49 – 50.

[7] Goodfellow, I. – Bengio, Y. – Courville, A. *Deep Learning*. 9 Convolutional Networks. MIT Press, 2016. http://www.deeplearningbook.org.

[8] Gupta, V. et al. C2VNet: A Deep Learning Framework Towards Comic Strip to Audio-Visual Scene Synthesis. In *IEEE International Conference on Document Analysis and Recognition*, 2021.

[9] He, K. et al. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, s. 770–778.

[10] Heaton, J. Applications of Deep Neural Networks, 2020.

[11] Hu, J. et al. Squeeze-and-Excitation Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2017, 42, s. 2011–2023.

[12] Huynh, N. *Understanding Evaluation Metrics in Medical Image Segmentation* [online]. Medium, 2023. [cit. 2023/05/01]. Dostupné z: https://medium.com/mlearning-ai/understanding-evaluation-metrics-in-medical-image-segmentation-d289a373a3f.

[13] Jha, D. et al. ResUNet++: An Advanced Architecture for Medical Image Segmentation. *2019 IEEE International Symposium on Multimedia (ISM)*. 2019, s. 225–230.

[14] Jha, D. et al. DoubleU-Net: A Deep Convolutional Neural Network for Medical Image Segmentation. *2020 IEEE 33rd International Symposium on Computer-Based Medical Systems (CBMS)*. 2020, s. 558–564.

[15] Kingma, D. P. – Ba, J. Adam: A Method for Stochastic Optimization. *CoRR*. 2014, abs/1412.6980.

[16] Krizhevsky, A. – Sutskever, I. – Hinton, G. E. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*. 2012, 60, s. 84 – 90.

[17] LeCun, Y. et al. Gradient-based learning applied to document recognition. *Proc. IEEE*. 1998, 86, s. 2278–2324.

[18] Minsky, M. – Papert, S. Perceptrons - an introduction to computational geometry. 1969.

[19] Otsu, N. A threshold selection method from gray level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*. 1979, 9, s. 62–66.

[20] Ronneberger, O. – Fischer, P. – Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *ArXiv*. 2015, abs/1505.04597.

[21] Rosebrock, A. *Intersection over Union (IoU) for object detection* [online]. pyimagesearch, 2016. [cit. 2023/05/01]. Dostupné z: `https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`.

[22] Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*. 1958, 65 6, s. 386–408.

[23] Rumelhart, D. E. – Hinton, G. E. – Williams, R. J. Learning representations by back-propagating errors. *Nature*. 1986, 323, s. 533–536.

[24] Simonyan, K. – Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*. 2014, abs/1409.1556.

[25] Sun, C. – Lu, H. – Yang, M.-H. Learning Spatial-Aware Regressions for Visual Tracking. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2017, s. 8962–8970.

[26] Varoquaux, G. et al. *scipy-lectures/scipy-lecture-notes: Release 2017.1* [online]. Zenodo, October 2017. Dostupné z: `https://scipy-lectures.org/packages/scikit-image/auto_examples/plot_threshold.html`.

[27] Zhang, Z. – Liu, Q. – Wang, Y. Road Extraction by Deep Residual U-Net. *IEEE Geoscience and Remote Sensing Letters*. 2017, 15, s. 749–753.