Univerzita Pardubice
Fakulta elektroniky a informatiky

Segmentace obrazu pomocí konvolučních neuronových sítí

Bc. Milan Horák

Diplomová práce
2023

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Milan Horák**
Osobní číslo:	**I20204**
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Segmentace obrazu pomocí konvolučních neuronových sítí**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

Cílem diplomové práce je návrh konvoluční neuronové sítě pro segmentaci obrazu a použití této sítě jako umělého filtru obrazových dat. Student navrhne vlastnosti umělého filtru, provede sběr a označení obrazových dat pro potřeby učení neuronových sítí, provede experimenty s různými topologiemi neuronových sítí a metodicky porovná zkoumané topologie pomocí běžně používaných metrik.

Teoretická část: Rešerše metod segmentace obrazu pomocí hlubokých konvolučních neuronových sítí a existujících prostředí pro učení hlubokých sítí (Tensorflow, Caffe, atp.).

Praktická část: Návrh architektury hluboké konvoluční sítě vhodné pro definovanou segmentaci obrazových dat a její porovnání s používanými topologiemi.

Rozsah pracovní zprávy: **cca 50 stran**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

LU, Le, Yefeng ZHENG, Gustavo CARNEIRO a Lin YANG, [2018]. *Deep Learning and Convolutional Neural Networks for Medical Image Computing*. 1. United States: Springer International Publishing. ISBN 3319429981.

GONZALEZ, Rafael C. a Richard E. WOODS, [2018]. *Digital image processing*. Fourth edition. New York: Pearson. ISBN 978-013-3356-724.

Vedoucí diplomové práce: **Ing. Dominik Štursa**
Katedra řízení procesů

Datum zadání diplomové práce: **8. listopadu 2022**
Termín odevzdání diplomové práce: **18. května 2023**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 30. listopadu 2022

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem k práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na mou práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména ze skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla, podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou, nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 1. 8. 2023

Bc. Milan Horák

PODĚKOVÁNÍ

Tímto bych rád poděkoval celé své rodině za trpělivost a psychickou podporu během celého studia a psaní této práce. V neposlední řadě také děkuji za udělení mého tématu a možnost konzultovat odbornou problematiku mému vedoucímu diplomové práce, kterým je Ing. Dominik Štursa.

ANOTACE

Diplomová práce se zabývá návrhem a tvorbou vhodné konvoluční neuronové sítě pro segmentaci obrazu. Čtenář se v práci seznámí s konceptem neuronových sítí, s postupnými kroky, které jsou nezbytné pro tvorbu vlastního modelu, dále s otevřenými knihovnami pro práci s neuronovými sítěmi a výběrem vhodné topologie pro danou problematiku.

Praktická část se věnuje návrhu a implementaci konvoluční neuronové sítě pro predikci přesné polohy pixelů konkrétních objektů a tvorby vlastního filtru aplikovaného na původní obrazová data v oblasti těchto objektů.

KLÍČOVÁ SLOVA

umělá inteligence, neuronová síť, segmentace obrazu, strojové vidění, strojové učení, hluboké učení

TITLE

Image segmentation using convolutional neural networks

ANNOTATION

The following diploma thesis describes the creation of suitable convolutional neural network for picture segmentation. Reader will learn principal of neural network together with described steps necessary to create own model, working with open libraries required for neural network projects and choosing right topology for given topic.

Practical part shows design and implementation of convolutional neural network, used for recognition of particular objects, describes also creation of a filter which is applicated on picture data produces with neural network.

KEYWORDS

artificial intelligence, neural network, image segmentation, machine vision, machine learning, deep learning

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK

AI	Artificial Intelligence
ML	Machine Learning
ReLU	Rectified Linear Unit
PReLU	Parametric Rectified Linear Unit
CNN	Convolutional Neural Network
DCNN	Deep Convolutional Neural Network
FCN	Fully Convolutional Network
VGG	Visual Geometry Group
PSPN	Pyramid Scene Parsing Network
IDE	Integrated Development Environment
CUDA	Compute Unified Device Architecture
CPU	Central Processing Unit
GPU	Graphics Proccessing Unit
RAM	Random Access Memory
SISD	Single Instruction, Single Data
AVX	Advanced Vector Extension
GNU	Gnu's Not Unix
GIMP	GNU Image Manipulator Program
LuaJIT	Just-In-Time Compiler for Lua
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
IoU	Intersection over Union

ÚVOD

Během posledního desetiletí se umělá inteligence stala součástí naší společnosti a je využívána každým dnem, a to i v běžném lidském životě. Ve své nejjednodušší podobě je umělá inteligence oborem, který kombinuje informatiku a obrovské datové sady, aby umožnila řešení i problémům, u kterých nelze snadno a jednoznačně určit kritéria, podle kterých by měla být úloha zpracovávána. Jinými slovy, umělá inteligence umožňuje řešit i takové úlohy, u kterých člověk dokáže snadno rozeznat přijatelné odchylky, ale pro běžný stroj, který potřebuje pro své vyhodnocení přesné hodnoty je takové úloha problematická. Umělá inteligence zahrnuje také dílčí oblasti strojového učení. Cílem umělé inteligence je vytvoření expertních systémů, které provádějí předpovědi nebo klasifikace na základě vstupních dat.

K odstranění problémů s nepřesnostmi v umístění jednotlivých objektů lze využít algoritmy strojového vidění, které v posledních letech zaznamenaly velmi bouřlivý rozvoj. Jejich rychlost a přesnost dosahuje extrémních hodnot, díky kterým lze provádět experimenty i ve velmi náročných úlohách. Tento velmi rychlý vývoj je dán značným pokrokem v oblasti hlubokých neuronových sítí, mezi nimiž v současné době hrají hlavní roli konvoluční neuronové sítě, na které se tato práce zaměřuje.

V teoretické části budou představeny základní úlohy umělé inteligence pro počítačové vidění se zaměřením na sémantickou segmentaci obrazu. Dále zmapování současné technologie a topologie modelů pro tvorbu konvoluční neuronové sítě a vybrány nejvhodnější z nich pro tvorbu praktické části této práce.

Cílem praktické části této práce je tvorba vlastní topologie hluboké konvoluční neuronové sítě, která bude detekovat konkrétní objekty a přesně určovat každý jejich pixel. Na tyto detekované pixely bude následně aplikován filtr, který objekty zvýrazní v původních obrazových datech. Výstup z neuronové sítě nám tedy vrátí masku pro aplikaci filtru.

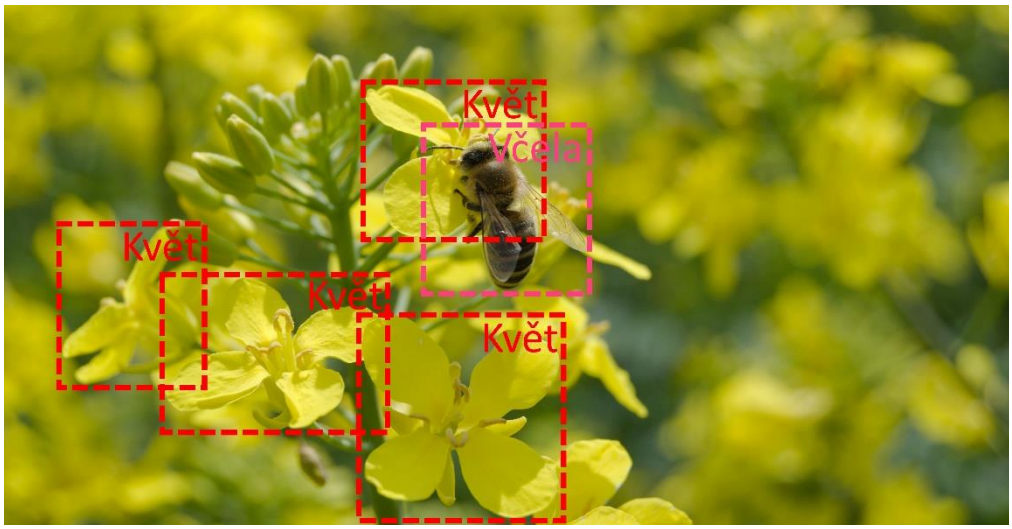
1 ÚLOHY STROJOVÉHO VIDĚNÍ

Strojové vidění je v oblasti umělé inteligence považováno za úlohy zabývající se zpracováním obrazových dat. Na rozdíl od lidského vidění má strojové vidění nevýhodu absenci celoživotního trénování, jak rozlišit předměty od sebe, jejich vzdálenost, zda se pohybují nebo zda není v obrazových datech něco špatně. [24]

Cílem strojového vidění je obvykle získat dodatečné informace z obrazových dat, které by běžně poskytnul lidský mozek. Počítačové vidění tedy napodobuje proces lidského vidění, kde je jako náhrada trojjediné sítě použita neuronová síť. V následujících podkapitolách jsou stručně popsány druhy úloh, kterými se v praxi strojové vidění běžně zabývá. Práce se dále bude více věnovat sémantické segmentaci. [24]

1.1 Detekce

Detekce zjišťuje třídu a polohu jednotlivých objektů v obrazových datech. Objektů může být v obrazových datech nalezeno větší množství o různých klasifikačních třídách. [24]



***Obrázek 1**: Ukázka detekce*

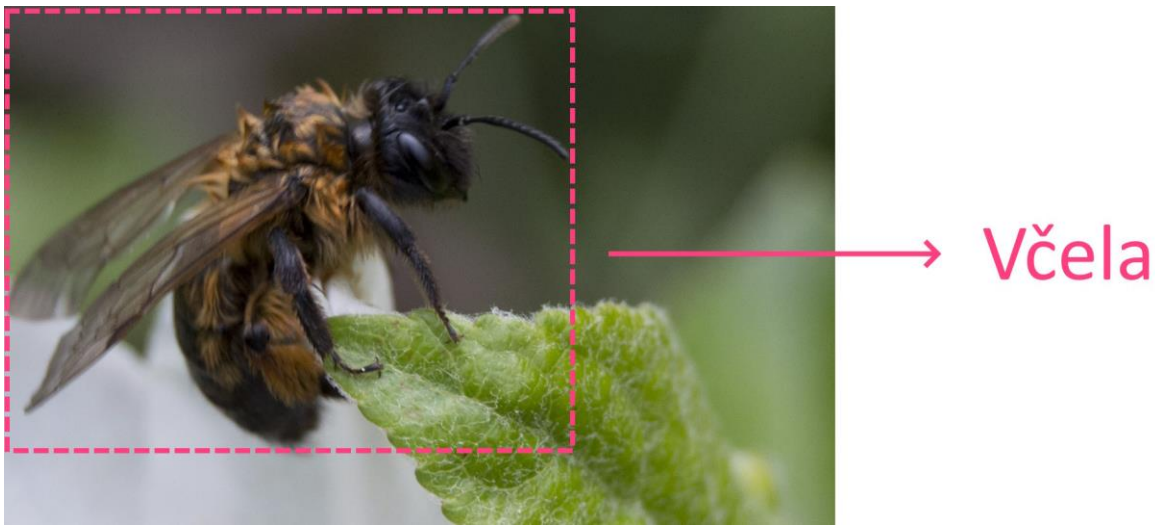
1.2 Klasifikace

Klasifikace určuje, zdali se v obrazových datech nachází hledaný objekt. Neurčuje však jeho polohu nebo množství. Výstup tedy pouze rozřadí obrazová data do určité kategorie. [24]

***Obrázek 2**: Ukázka klasifikace*

1.3 Klasifikace s lokalizací

Obdobné jako prostá klasifikace. K výstupu ale navíc přidává vektorovou informaci o poloze objektu. Vektor, který udává polohu objektu je obvykle ve tvaru (x, y, w, h), kde (x, y) udává středové souřadnice, (w) šířku a (h) výšku rámečku ohraničující objekt. Poloha objektu je tedy obvykle určena obdélníkovým výřezem. [24]



***Obrázek 3**: Ukázka klasifikace s lokalizací*

1.4 Sémantická Segmentace

Sémantická segmentace předpovídá přesnou polohu všech pixelů podle klasifikace třídy objektů v obrazových datech. Nerozlišuje více objektů stejné klasifikační třídy od sebe. Výstup je tedy matice s příznakem všech pixelů, kde každý pixel udává určitou klasifikaci. V následujícím obrázku by mohla například hodnota 0 (černá barva) reprezentovat pozadí obrázku a hodnota 1 (bílá barva) květ. [25]

***Obrázek 4**: Ukázka sémantické segmentace*

1.5 Instanční Segmentace

Obdobné jako sémantická segmentace, navíc však rozlišuje jednotlivé instance stejné klasifikační třídy od sebe. V následujícím obrázku představuje černá barva pozadí a ostatní barvy odlišné instance květu. [25]

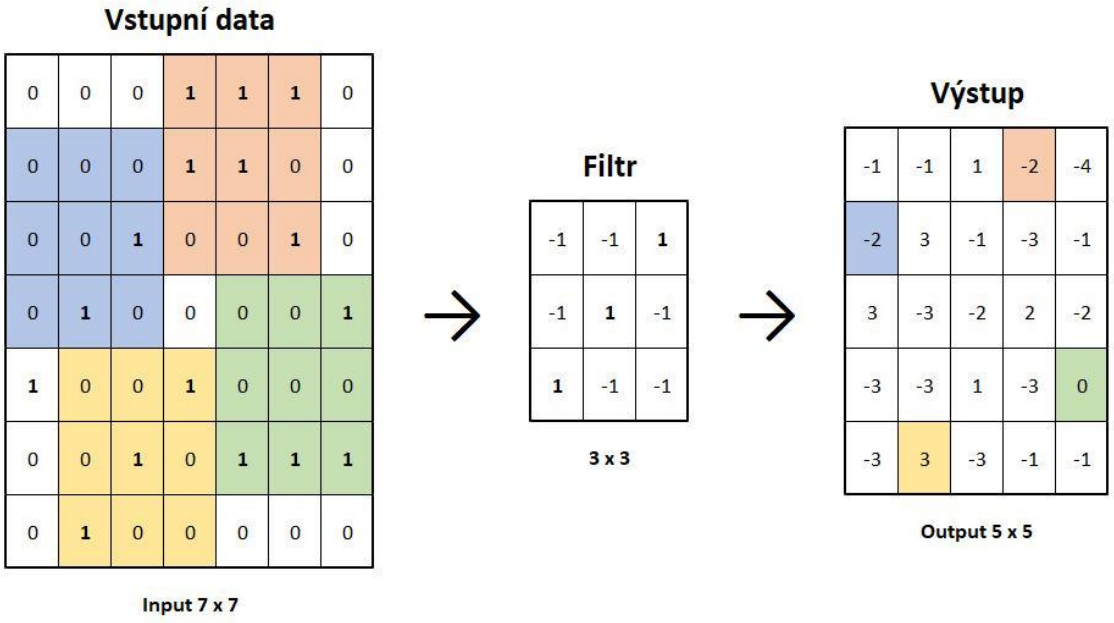
***Obrázek 5**: Ukázka instanční segmentace*

2 KONVOLUČNÍ NEURONOVÉ SÍTĚ

Konvoluční neuronové sítě jsou specifickou podmnožinou neuronových sítí, které jsou využívány pro zpracování dat s více dimenzionální strukturou známé velikosti. V této práci budou využity pro zpracování obrazových dat, lze však využít i pro zpracování rozdílných vstupních signálů (např. zvuku pro rozpoznání řeči). Tento druh neuronových sítí je v současné době v centru zájmu. Výhodou CNN je možnost navržení tak, aby za nás automaticky provedla extrakci vlastností. Extrakce vlastností je v obrazové analýze proces, u kterého jsou vstupní obrazová data zjednodušována a zároveň jsou v něm zachovány důležité vzory. Jinými slovy, proces dokáže snížit rozlišení obrazových dat způsobem, aniž bychom ztratily důležité informace. Takto připravená data lze daleko rychleji zpracovávat. Není potřeba analyzovat každý pixel původního rozlišení. [30]

2.1 Konvoluční vrstva

Konvoluční vrstva představuje množinu filtrů, které provádějí operaci konvoluce. Filtr je tvořen z matice o libovolné velikosti s určitými parametry. Parametry filtru si konvoluční síť dokáže sama optimalizovat během procesu trénování, což je velikou výhodou oproti jiným metodám počítačové grafiky, pro extrakci vlastností, kde je obvykle potřeba spoustu parametrů nastavit empiricky. Tyto filtry detekují klíčové vzory z původních dat podle svých parametrů. Postup je následující. Procházíme filtrem okénko přes vstupní data a v každém kroku spočteme výstup pomocí operace skalární součin. [30]



***Obrázek 6**: Princip aplikování konvolučního filtru*

2.2 Aktivační vrstva

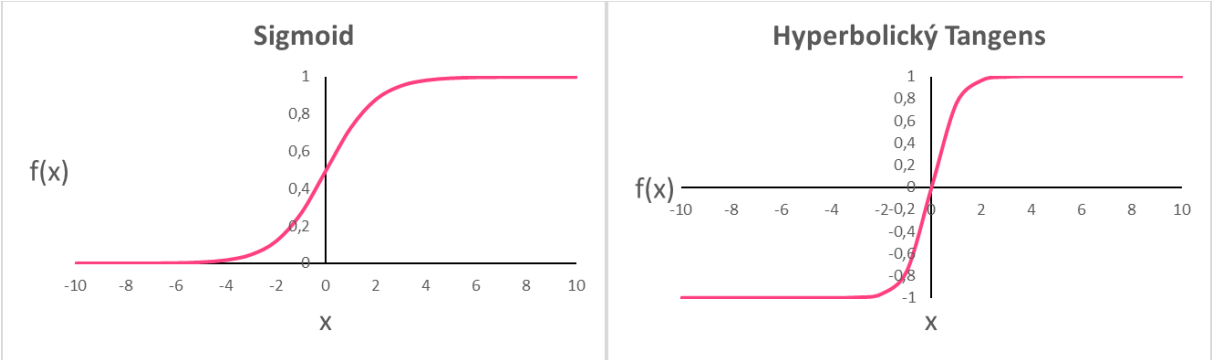
Aktivační vrstva plní roli aktivační funkce. Přijme vstupní signál z předchozí úrovně vrstvy, převede jej do jiné podoby podle svých charakteristik a předá dál do další úrovně vrstvy neuronové sítě. U CNN je tato funkce typicky oddělena v samostatné vrstvě. Jako typická aktivační funkce se běžně používá funkce *sigmoid*, *hyperbolický tangens*, *ReLU*, *Leaky ReLU* a *PReLU*. [7]

Sigmoid je aktivační funkce, která je převážně využívaná pro plně propojené vrstvy. Výstupem této funkce jsou reálná čísla v rozsahu 0 až 1 a je vycentrován v hodnotě 0,5. Nevýhodou této funkce je relativně vysoká výpočetní náročnost oproti ostatním funkcím a také jev zvaný *vanishing gradient*, při kterém jsou příliš velké vstupy saturovány na hodnotu 1. Další nevýhodou je, že výstup není centrován na hodnotu 0. [7]

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

Hyperbolický tangens je aktivační funkce, která je rovněž převážně využívána pro plně propojené vrstvy. Výstupem této funkce jsou reálná čísla v rozsahu -1 až 1. Funkce je vycentrována v hodnotě 0. Nevýhodou této funkce je stejně jako u funkce sigmoid relativně vysoká výpočetní náročnost oproti ostatním funkcím a *vanishing gradient*, kde jsou navíc ještě příliš malé vstupy saturovány na hodnotu -1. [7]

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



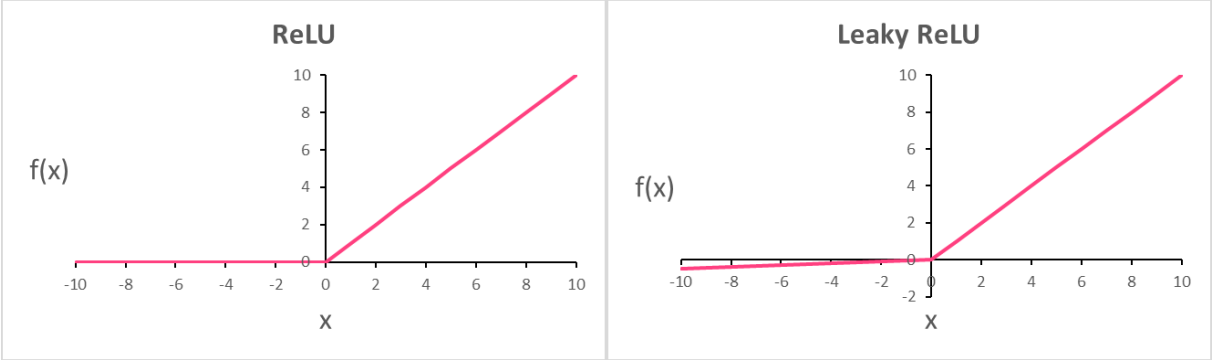
***Obrázek 7**: Ukázka grafu aktivační funkce sigmoid a hyperbolický tangens*

ReLU je velmi oblíbenou aktivační funkcí pro konvoluční neuronové sítě, a to hlavně kvůli své nízké náročnosti na výpočetní výkon. Výstupem této funkce je reálné nezáporné číslo. ReLU funguje skvěle ve většině aplikací, není však dokonalá. Trpí problémem známým jako *Dying ReLU*, což je situace, při které určitá část neuronů přestane vydávat jakýkoliv jiný výstup než 0. Tato situace nastane, když se váhy neuronu upraví způsobem, kdy se vážený součet vstupů stane záporný pro všechny instance v trénovací sadě. Jelikož funkce vrátí nulu pro jakékoliv záporné číslo, neuron již nic jiného nevrátí. [7]

$$f(x) = \begin{cases} x \; pokud \; x > 0 \\ 0 \; pokud \; x \leq 0 \end{cases} = max\,(0, x)$$

Leaky ReLU funguje podobným způsobem jako funkce ReLU, navíc však zachovává i záporné hodnoty, díky kterým se zbavuje problému *Dying ReLU*. Záporné hodnoty jsou redukovány takzvaným *leak factorem*, který je obvykle nastavován jako velmi malé číslo v řádu setin až tisícin jednotek. V následujícím grafu je *leak factor* nastaven na hodnotu 0,05 kvůli přehlednější vizualizaci v daném rozsahu hodnot. [7]

$$f(x) = \begin{cases} x \; pokud \; x > 0 \\ \alpha x \; pokud \; x \leq 0 \end{cases} = max\,(\alpha x, x), kde\; \alpha \; je \; zvolený \; leak \; factor$$



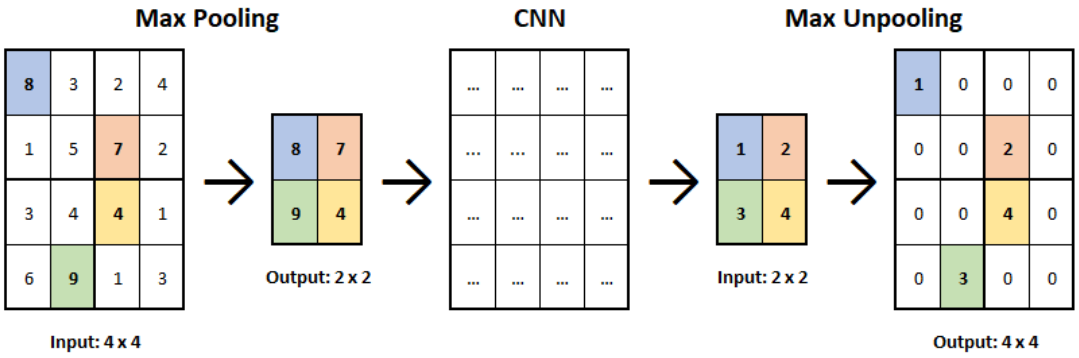
***Obrázek 8**: Ukázka grafu aktivační funkce ReLU a Leaky ReLU*

PReLU je variantou Leaky ReLU, kde *leak factor* je automaticky nastavován během tréningu zpětným šířením chyby, stejně tak jako ostatní parametry v neuronové síti. Tato aktivační funkce je obecně výhodná, pokud máme k dispozici obrovské datové sady pro trénování sítě. [7]

2.3 Pooling vrstva

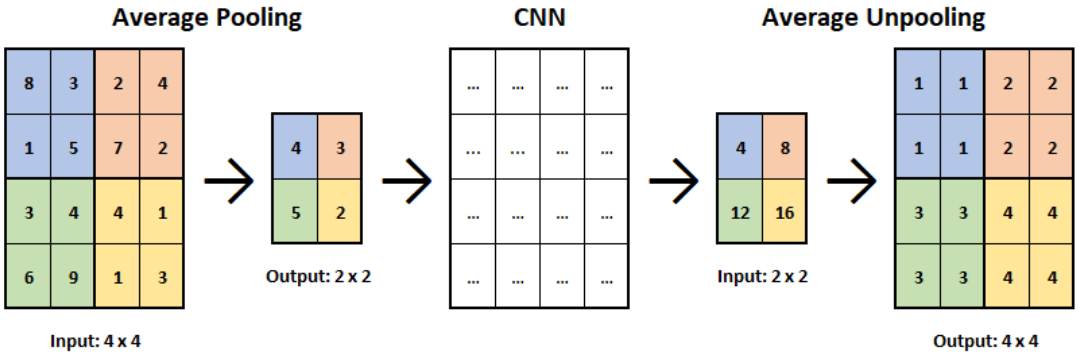
Pooling vrstva obvykle následuje po konvoluční vrstvě a má za úkol zmenšit rozměry matice vstupních dat. Hlavní účel této vrstvy je snížit výpočetní a paměťovou náročnost neuronové sítě. Základní parametry této vrstvy jsou *velikost okna* (také označována jako *pool size* nebo *filtr size*), který určuje rozměr čtvercového okna pro výběr hodnot ze vstupních dat. Dále *krok* (*stride*), který se skládá z vektoru (*x*, *y*), kde *x* udává o kolik pozic se má filtr posouvat v horizontální ose, y udává o kolik pozic se má filtr posouvat ve vertikální ose. Při každém kroku jsou hodnoty v daném místě agregovány do jedné hodnoty podle zvolené agregační funkce. Jako typická agregační funkce se běžně používá funkce *max pooling* nebo *average pooling*. [4]

Agregační funkce *max pooling* vybere nejvyšší hodnotu z daného rozsahu hodnot. Navíc si zapamatuje pozici této hodnoty, kterou využije během operace *max unpooling*.



***Obrázek 9**: Princip max pooling vrstvy (velikost okna: 2, krok: (2, 2))*

Agregační funkce *average pooling* vypočítá aritmetický průměr z daného rozsahu hodnot. Reverzní operace *average unpooling* rozprostře hodnotu mezi celou oblast. Přidělí tedy danou hodnotu vydělenou velikostí oblasti všem buňkám v určené oblasti.

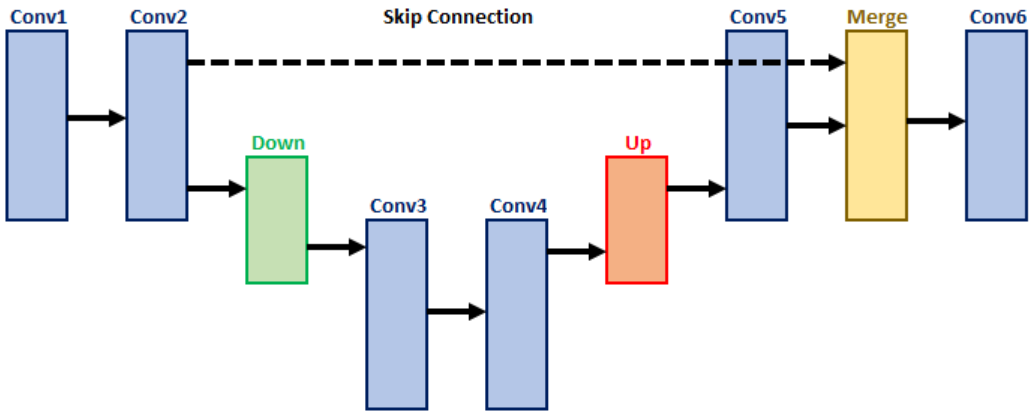


***Obrázek 10**: Princip average pooling vrstvy (velikost okna: 2, krok: (2, 2))*

2.4 Skip connection

Skip connection, jak již název napovídá, přeskakuje některé vrstvy v neuronové síti a přivede výstup z jedné vrstvy jako vstup do dalších vrstev. Tato technika byla představena pro řešení určitých specifických problémů v různých architekturách CNN. Často je touto vrstvou řešen problém degradace, který může vznikat v příliš složitých DCNN. Degradace znamená, když se kvůli příliš velké hloubce sítě začnou ztrácet některé důležité vzory, což zapříčiní zhoršení přesnosti předpovědi. [8]

V následujícím obrázku je ukázka vrstvy skip connection. Žlutě znázorněná vrstva merge představuje slučovací vrstvu, která spojuje vrstvu *Conv2* s vrstvou *Conv5*. Spojení mezi vrstvami lze provádět různými způsoby pomocí různých matematických operací. Podrobnější detaily lze čerpat z následujícího zdroje. [29]



***Obrázek 11**: Ukázka vrstvy skip connection*

3 ARCHITEKTURY PRO SÉMANTICKOU SEGMENTACI

Zde jsou stručně představeny moderní architektury konvolučních neuronových sítí, které se v současné době používají pro řešení úloh typu sémantická segmentace. Naivním přístupem ke konstrukci architektury neuronové sítě pro tento úkol by bylo jednoduše naskládat několik konvolučních vrstev za sebou a vytvořit finální mapu segmentace. Takto navržená architektura by si však zachovala plné rozlišení v celé síti, což by vedlo k relativně vysoké výpočetní a paměťové náročnosti.

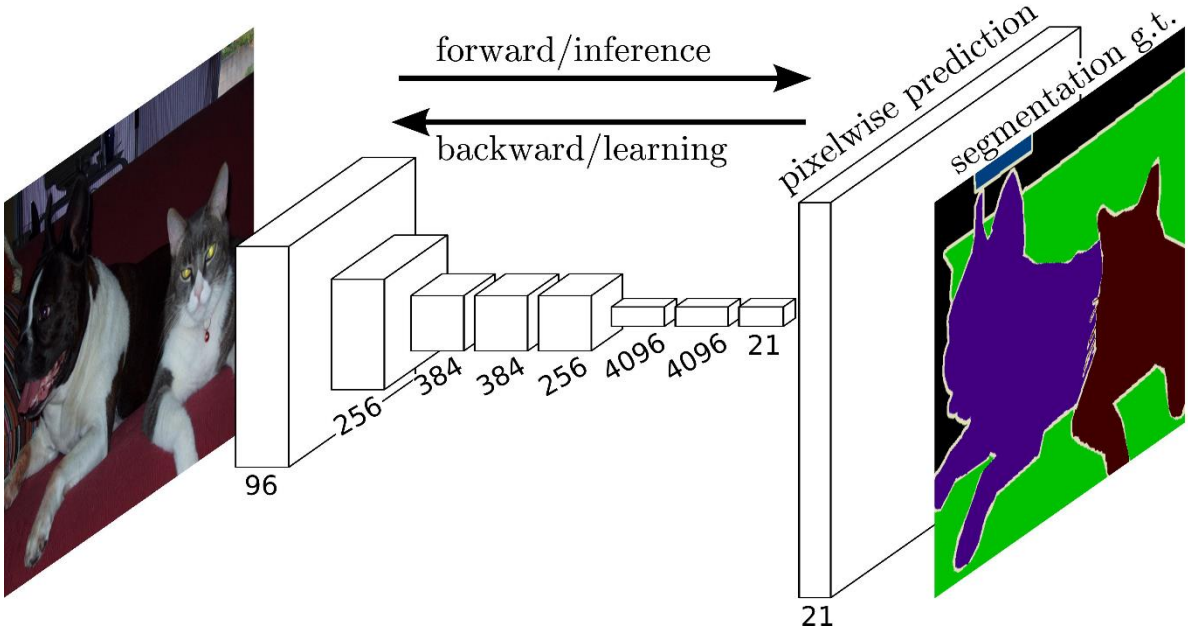
Jedním z populárních přístupů k modelům pro segmentaci obrazu je vytvořit strukturu, kde postupně snižujeme původní rozlišení (*downsampling*) vstupu a vyvineme mapování prvků s nižším rozlišením, které je časově a paměťově méně náročné. Následně zvyšujeme rozlišení (*upsampling*) prvků na mapu segmentace, v ideálním případě až do rozlišení původních vstupních dat, kde každý pixel získá svůj příznak podle klasifikační třídy. Takto navržená struktura konvoluční neuronové sítě se nazývá kodér/dekodér.

V praxi se obvykle nenavrhuje od základu nová konvoluční neuronová síť pro konkrétní úlohu. Konvoluční sítě mají příliš obrovské množství variací a parametrů, které by uživatel musel optimalizovat pro svou práci. Běžným postupem je vybrat architekturu konvoluční neuronové sítě z literatury, která je osvědčená pro daný typ problému.

3.1 FCN

FCN je architekturou používanou výhradně pro sémantickou segmentaci. Používá pouze lokálně propojené vrstvy, jako je konvoluce, sdružování a převzorkování. Vyhnutí se použití skip connections znamená méně parametrů k trénování. Jelikož zde neexistuje žádná nelokální vazba mezi vrstvami, síť nemá problém pracovat s libovolným rozlišením vstupních obrazových dat. [2]

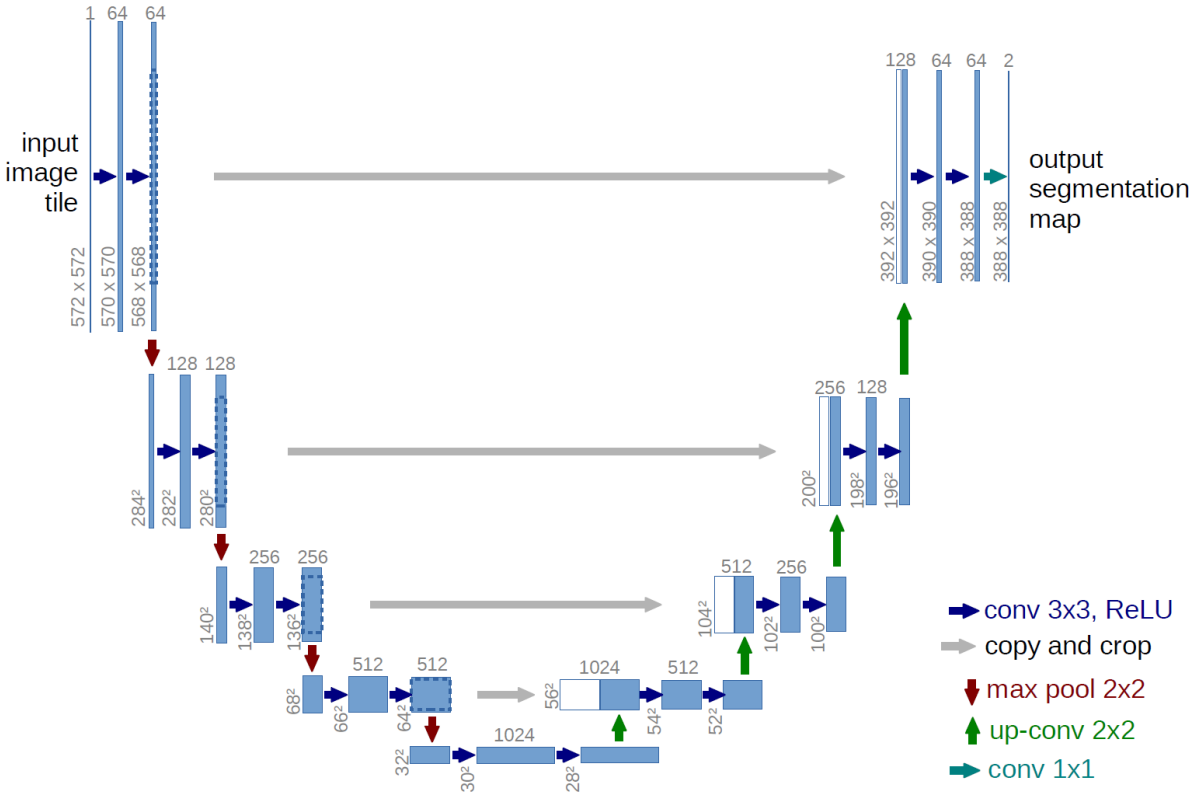
Přístup využívající FCN síť pro úlohu segmentace obrazu představil Jonathan Long na konci roku 2014. Princip je založen na přizpůsobení stávajících dobře prostudovaných sítí, pro klasifikaci obrazu, jako je například AlexNet. Jelikož tyto CNN výrazně snižují rozlišení původních obrazových dat, výstup pro segmentaci obrazu by byl příliš hrubý. Princip FCN tedy spočívá ve využití známé CNN pro klasifikaci obrazu jako kodér, kde následně přidává další konvoluční vrstvy, které jsou použity jako dekodér pro obnovení původního rozlišení, pro které chceme určit klasifikaci každého pixelu. [4]



***Obrázek 12**: Architektura FCN, zdroj: [2]*

3.2 U-net

Skládá se z opakované aplikace dvou (3x3) konvolučních vrstev, po každé následuje aktivační vrstva ReLU a (2x2) max pooling vrstva s krokem (2, 2). V každém kroku převzorkování je zdvojnásoben počet kanálů funkcí. Každý krok v expanzivní cestě se skládá z převzorkování mapy příznaků, po které následuje konvoluce (2x2), která snižuje počet kanálů prvků na polovinu. Dále dvě 3x3 konvoluční vrstvy. Po každé následuje aktivační vrstva ReLU. Oříznutí je nutné kvůli ztrátě hraničních pixelů v každé konvoluci. Na poslední vrstvě se použije konvoluce (1x1) k mapování každého 64 složkového vektoru příznaků na požadovaný počet tříd. Síť má celkem 23 konvolučních vrstev. Graficky znázorněná architektura U-net připomíná písmeno "U", podle kterého je tato síť pojmenovaná. [1]



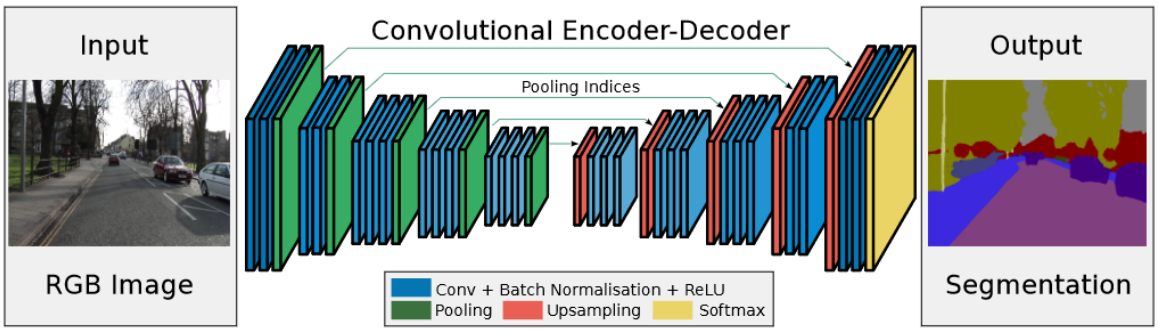
Obrázek 13: Architektura U-net, zdroj: [1]

3.3 SegNet

Architektura sítě kodéru je topologicky shodná s třinácti konvolučními vrstvami v síti VGG16. Úlohou sítě dekodéru je opět mapovat mapy funkcí kodéru s nízkým rozlišením na mapy funkcí s plným vstupním rozlišením pro klasifikaci podle pixelů. Navíc však přidává způsob, jakým dekodér převzorkuje své vstupní mapy funkcí s nižším rozlišením. [3]

Dekodér používá sdružovací indexy vypočítané v kroku max pooling odpovídajícího kodéru k provádění nelineárního převzorkování. To eliminuje potřebu učit se převzorkovat. Převzorkované mapy jsou řídké a jsou pak spojeny s filtry, aby se vytvořily husté mapy prvků. [3]

Architektura je navržena tak, aby byla efektivní jak z hlediska paměti, tak výpočetního času během inference. Nejsou zde žádné plně propojené vrstvy. Architektura má menší počet parametrů k trénování než jiné konkurenční architektury pro stejný typ úlohy. SegNet poskytuje dobrou časovou efektivitu a velmi dobrou paměťovou efektivitu ve srovnání s jinými konkurenčními architekturami, jako je například široce používaná architektura FCN. [3]



Obrázek 14: SegNet Architektura, zdroj: [3]

3.4 Pyramid Scene Parsing Network

Rozšiřuje a odstraňuje specifické problémy, které nastávají při použití FCN. Nově je zde představen takzvaný pyramidový poolingový modul, který se empiricky ukázal jako výhodný. V hluboké neuronové síti může velikost receptivního pole zhruba udávat, jak moc využíváme kontextové informace. [6]

FCN problémy:

- Neshodný vztah

  Nedostatek schopnosti shromažďovat kontextové informace zvyšuje pravděpodobnost chybné klasifikace. Například když naše CNN vyhodnotí objekt dle jeho vzhledu jako automobil, ale ve skutečnosti se jedná o loď plovoucí po vodě. Vzhledem ke kontextu, že je vyhodnocovaný objekt na vodě, je nepravděpodobné, že by se skutečně jednalo o auto.

- Záměna kategorií

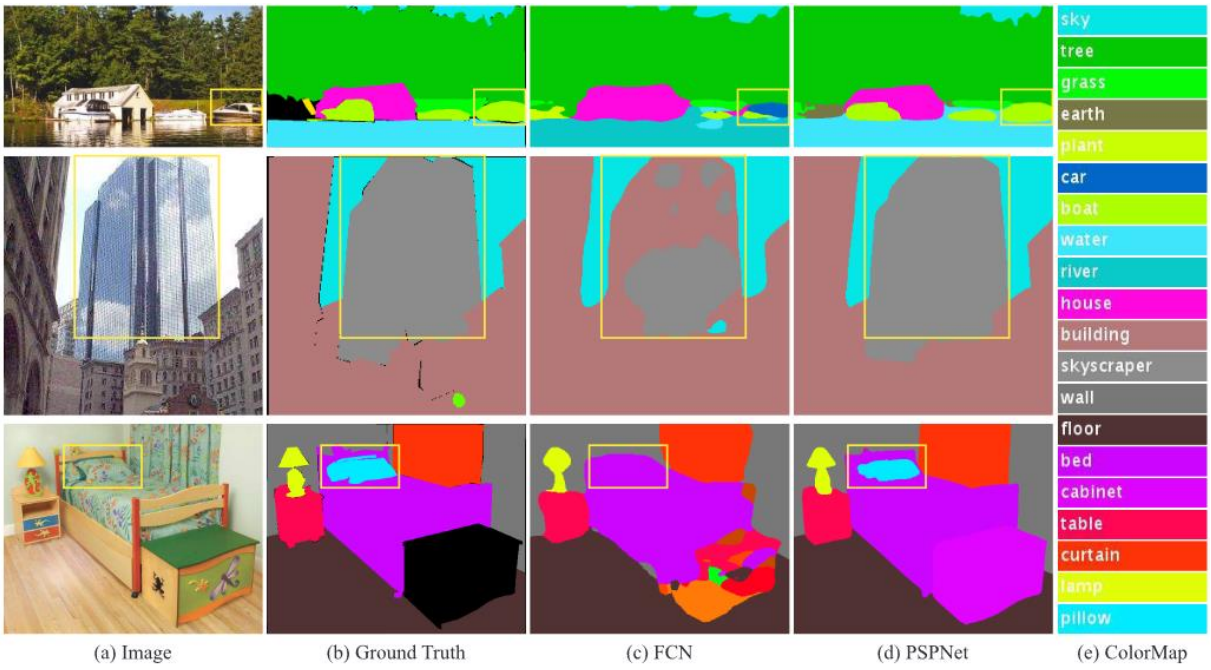
  Vyskytuje se v případě, že je příliš mnoho podobných kategorií k analýze. Například hora a kopec, pole a země nebo budova a mrakodrap. Jde o klasifikační třídy, které mají podobný charakter a naše CNN je může nejednoznačně klasifikovat, kdy půlka pixelů se může jevit jako jedna třída a druhá půlka zase jako jiná třída. Tyto výsledky je třeba vyloučit, aby celý objekt byl mrakodrap nebo budova, ale ne obojí.

- Nepřehledné třídy

  Scéna obsahuje objekty libovolné velikosti. Několik věcí malých rozměrů, jako jsou pouliční lampy a vývěsní štíty, je těžké najít, přestože mohou mít velký význam. Naopak velké objekty nebo věci mohou přesahovat. Tyto nenápadné objekty jsou snadno chybně klasifikovány pomocí FCN a způsobují tak nespojitou predikci.

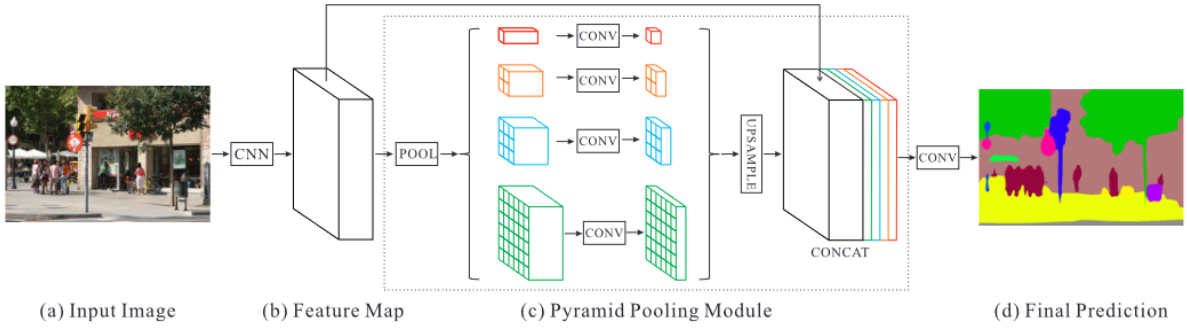
  Pro zlepšení výkonu u pozoruhodně malých nebo velkých objektů je třeba věnovat velkou pozornost různým podoblastem, které obsahují věci nenápadné kategorie.

Shrneme-li tato pozorování, mnoho chyb částečně nebo zcela souvisí s kontextovými vztahy a globálními informacemi pro různá receptivní pole. Hluboká síť s vhodnou prioritou na úrovni globální scény tak může výrazně zlepšit výkonnost rozboru scény. [6]



***Obrázek 15**: Ukázka běžných problémů FCN s klasifikací, zdroj: [6]*

Tento problém je řešen návrhem efektivní reprezentace globální priority. Globální sdružování průměrů je dobrým základním modelem, neboť globální kontextuální priorita, která se běžně používá v obrazových klasifikačních úlohách, byla úspěšně aplikována na sémantickou segmentaci. Pokud však jde o komplexní scény, je tato strategie nedostatečná k tomu, aby byly pokryty všechny potřebné informace. Pixely v těchto snímcích scény jsou anotovány s ohledem na mnoho věcí a objektů. Přímo jejich sloučení do jediného vektoru může ztratit prostorovou souvislost a způsobit nejednoznačnost. Globální kontextové informace spolu s kontextem dílčího regionu jsou v tomto ohledu užitečné k rozlišení mezi různými kategoriemi. Silnější reprezentace by mohly být sloučené informace z různých subregionů s těmito receptivními poli. [6]



***Obrázek 16**: Ukázka architektury PSPN, zdroj: [6]*

4 POUŽITÉ TECHNOLOGIE

Zde jsou představeny veškeré použité technologie a nástroje pro tvorbu vlastní CNN, která slouží pro segmentaci obrazu v této práci. Dále jsou zde představeny alternativní platformy, ve kterých lze pracovat s CNN.

4.1 Python

Python je vyšší interpretovaný objektově orientovaný programovací jazyk. Jeho jednoduchá a snadno naučitelná syntaxe klade důraz na čitelnost zdrojového kódu. Zajímavostí syntaxe tohoto jazyku je, že i bílé znaky mají svůj vlastní význam. Je tedy potřeba dodržovat určité odsazení a odřádkování kódu, jinak by se výsledný program nemusel chovat podle očekávání programátora. [10]

Python podporuje moduly a balíčky, což podporuje modularitu programu a opětovného použití kódu. Interpret Pythonu a rozsáhlá standardní knihovna jsou k dispozici ve zdrojové nebo binární podobě zdarma pro všechny hlavní platformy a lze je volně šířit.

4.2 Pip

Pip je balíčkovací systém pro správu knihoven pro programovací jazyk Python. Pomocí tohoto balíčkovacího systému lze snadno doinstalovat potřebné Python knihovny, jako je například knihovna NumPy pro práci s číselnými daty a jejich transformacemi. [11]

4.3 PyCharm

PyCharm je integrované vývojové prostředí specializované pro práci s programovacím jazykem Python. I když dokáže pracovat i s jinými jazyky, jako je například JavaScript. Podpora jazyku Python je jak pro starší verzi 2 (2.7), tak i pro novější 3 (3.5 a vyšší). PyCharm je vydávaný jak v placené edici Professional, tak i v bezplatné edici Community, která je však omezená svými funkcemi. Pro studenty a učitele existuje ještě navíc vzdělávací edice Edu, která nemá žádné omezení a je zdarma pro nekomerční účely.

Velkou výhodou je nativní podpora pro desktopové platformy Linux, MacOS i Windows. Prostředí disponuje velkým množstvím integrovaných nástrojů, jako je zvýraznění syntaxí zdrojového kódu, napovídání během psaní kódu, práce s verzovacím systémem Git, integrovaní jednotkových testů, práce s vícero databázemi najednou, automatické generování kódu, kontrola syntaxe a mnoho dalších. Kromě integrovaných nástrojů je možné i doinstalovat vlastní pluginy, díky kterým lze toto vývojové prostředí značně přizpůsobit svým vlastním potřebám. [9]

4.4 Tensorflow

TensorFlow je platforma s otevřeným zdrojovým kódem pro numerické výpočty, která zrychluje a usnadňuje strojové učení a vývoj neuronových sítí. Tato platforma je vyvinutá společností Google. Spojuje spousty modelů a algoritmů strojového a hlubokého učení. Platforma je schopna zobrazit datovou strukturu graf, která reprezentuje architekturu vámi vytvořeného modelu, kde jednotlivé vrcholy reprezentují matematické operace a jednotlivé hrany reprezentují tenzory, které fungují jako vstupy a výstupy pro dané vrcholy, s kterými jsou propojeny. Takto organizovaná struktura je pro člověka relativně jednoduchá na pochopení. [12]

Pro výrazné zrychlení výpočtů podporuje platforma Tensorflow hardwarovou akceleraci pro NVIDIA grafické karty, které obsahují CUDA jádra. Pro přesunutí výpočtů na podporované GPU je zapotřebí doinstalovat určité knihovny dle svého operačního systému. Různé způsoby instalace, včetně využití již zmíněného balíčkovacího systému pip, lze dohledat z následujícího zdroje. [13]

Pokud nejste vlastníkem těchto podporovaných GPU, může vám do značné míry zrychlit výpočet i moderní Intel nebo AMD CPU využívající architekturu x86, které podporují instrukční sadu AVX. Jedná se o dodatečnou instrukční sadu, která umožňuje procesorům vykonávat náročnější funkce při použití s kompatibilním softwarem. AVX je hardwarově závislá funkce, aktualizace softwaru nebo firmwaru ji nemohou zavést, pokud ji procesor nepodporuje. Tato instrukční sada je SW podporována platformou Tensorflow, která dokáže na rozdíl od SISD provádět paralelní vektorové výpočty, což v této problematice výrazně urychluje výpočet. SISD dokáže provádět výpočet jen nad jedním datovým prvkem najednou. Zároveň se jedná o úlohy, které jsou efektivně řešeny více jádrovým výpočtem. Více jader CPU má tedy také velký vliv na rychlost výpočtu. [26]

4.5 Keras

Keras je API pro hluboké učení napsané v programovacím jazyku Python, které běží nad platformou strojového učení TensorFlow. Byl vyvinut se zaměřením na umožnění rychlého experimentování a vývoje. Keras poskytuje vyšší úroveň abstrakce oproti Tensorflow, ovšem se zaměřením pouze na hluboké učení. Pro programátora to tedy znamená méně práce a rychlý vývoj během psaní zdrojového kódu. Používají ho i velké organizace a společnosti jako je NASA, YouTube nebo Waymo. [14]

Základními datovými strukturami Kerasu jsou vrstvy a modely. Nejjednodušším typem modelu je sekvenční model, lineární zásobník vrstev. Pro složitější architektury by mělo být použito funkční rozhraní API Keras, které umožňuje vytvářet libovolné grafy vrstev nebo psát modely zcela od nuly pomocí podtříd. Keras se řídí principem postupného odhalování složitosti. Lze tedy snadno začít, ale zároveň umožňuje zvládnout libovolně pokročilé případy použití, přičemž v každém kroku vyžaduje pouze postupné učení, což velmi zlehčuje práci začátečníkům. [14]

Zvláštní pojmenování této API slovem Keras pochází z řecké literatury. Keras (κέρας) znamená v řečtině roh. Je to odkaz na literární obraz ze starověké řecké a latinské literatury, poprvé nalezený v Odyssey, kde se objevují duchové snů (takzvaní Oneiroi). Duchové jsou rozděleni na dva druhy. Duchové, kteří přicházejí branou slonoviny, klamou snílky klamavými vizemi. Ti druzí přicházejí branou rohu a ohlašují události budoucnosti, které se opravdu naplní. Vlastnosti duchů jsou vlastně hříčkami se slovy κέρας (roh) / κραίνω (naplnit) a ἐλέφας (slonovina) / ἐλεφαίρομαι (klamat). [14]

4.6 Caffe

Caffe je framework pro hluboké učení vytvořený s ohledem na výraznost, rychlost a modularitu a je uvolněn pod licencí BSD 2-Clause. Vyvíjí ho společnost Berkeley AI Research (BAIR), Learning Center (BVLC) a komunitní přispěvatelé. Yangqing Jia vytvořil projekt během svého doktorského studia na Kalifornské univerzitě v Berkeley. [15]

Tento framework podporuje grafickou akceleraci pouze pro Linux se 64bitovou architekturou s využitím grafických karet NVIDIA s CUDA jádry. Caffe dokáže běžet až o 65 % rychleji na nejnovějších grafických procesorech NVIDIA Pascal™ a škáluje se na více grafických procesorech v rámci jednoho uzlu. Nyní lze trénovat rozsáhlé modely v řádu hodin namísto dnů. Z následujícího zdroje může čtenář nalézt dodatečné informace ohledně podpory a manuál pro instalaci potřebných knihoven včetně příkladů použití se specifickými architekturami CNN. [16]

Modely a optimalizace jsou definovány konfigurací bez nutnosti pevného kódování. Je umožněno přepínat mezi CPU a GPU nastavením jediného příznaku pro trénování na stroji s GPU a následným nasazením na komoditních clusterech nebo mobilních zařízeních. Rozšiřitelný kód podporuje aktivní vývoj. Caffe již pohání akademické výzkumné projekty, prototypy startupů, a dokonce i rozsáhlé průmyslové aplikace v oblasti počítačového vidění, řeči a multimédií. [16]

4.7 PyTorch

PyTorch je open source knihovna programovacího jazyka Python pro strojové učení, založená na frameworku Torch, která se specializuje na tenzorové výpočty, automatickou diferenciaci a akceleraci na GPU. Z hlediska četnosti využití se jedná o přímého konkurenta pro TensorFlow a Keras. [21]

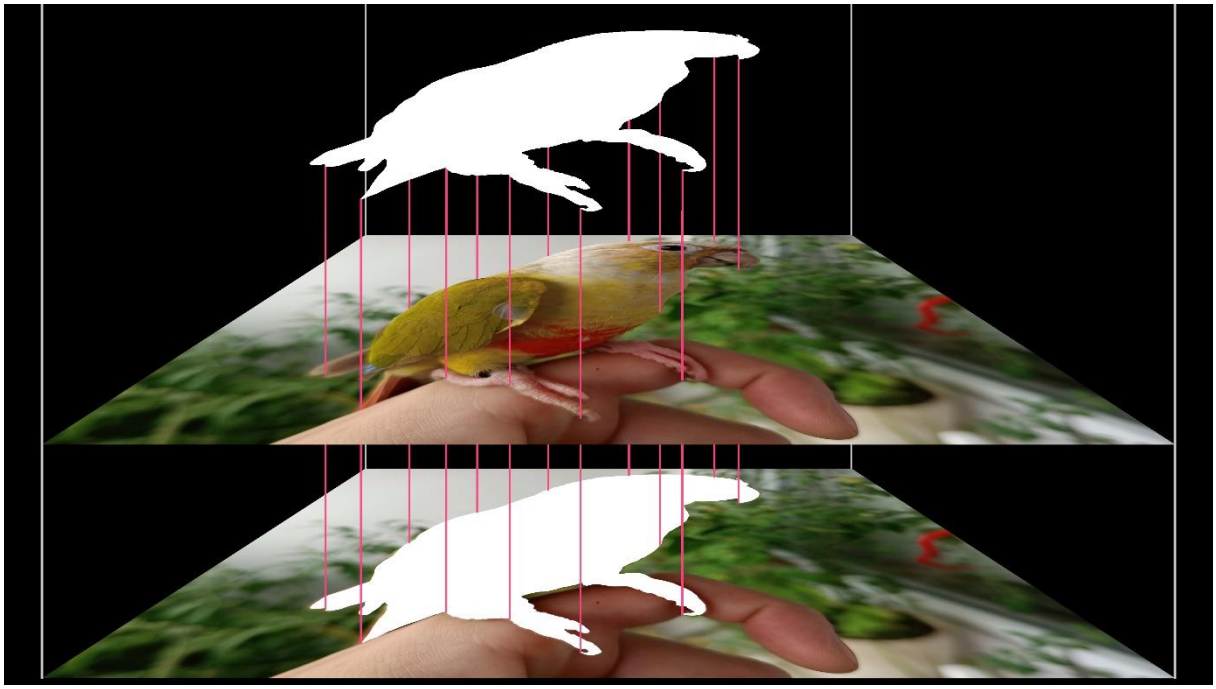
Torch je vědecký výpočetní framework pro programovací jazyk Lua se širokou podporou algoritmů strojového učení, který na prvním místě staví GPU. Je snadno použitelný a efektivní díky snadnému a rychlému skriptovacímu jazyku LuaJIT a základní implementaci CUDA. [20]

PyTorch původně vznikal kvůli problémům, které měly Keras a Tensorflow v dané době. Tensorflow nebylo snadné používat, protože rozhraní API nebylo intuitivní a tím pádem složité pro začátečníky. Keras se sice dal používat relativně velice snadno, neumožňoval však konstrukce některých nízkoúrovňových funkcí, které výzkumníci potřebovali. V současné době se tyto dvě konkurenční platformy svými možnostmi využití začaly více podobat. [21]

4.8 GIMP

GIMP je multiplatformní editor obrázků dostupný pro desktopové operační systémy GNU/Linux, MacOS, Windows, Sun OpenSolaris a FreeBSD. Jedná se o svobodný software, jehož zdrojový kód je možné měnit a změny šířit. [18]

Nástroj podporuje práci s vrstvami, což je v grafických editorech velmi užitečné. Dají se pak provádět nedestruktivní operace (operace, které nezmění původní obrazová data), protože veškeré změny lze aplikovat na další vrstvy, kde není nutno editovat přímo pixely původních obrazových dat.



***Obrázek 17**: Princip grafických vrstev*

V práci je tento grafický editor použit pro tvorbu masek vlastní datové sady. Obsahuje spoustu užitečných nástrojů pro správný výběr pixelů, které chceme určit jako objekt zkoumání. Mezi nejužitečnější nástroje pro tuto úlohu patří: přibližný výběr, volný výběr, plechovka a obecná práce s vrstvami. Přibližný výběr dokáže vybrat spojité oblasti na základě barvy. Tento nástroj je velmi užitečný v případě, kdy objekt, který je zapotřebí vybrat, je více barevně kontrastní od jeho okolí. Volný výběr je užitečným nástrojem, který umožňuje vybírat volně od ruky nebo pomocí mnohoúhelníkových segmentů včetně vzájemné kombinace. Tam kde selže nástroj přibližný výběr je vhodné použít volný výběr. Ve většině praktických případů je však potřeba volný výběr alespoň pro manuální opravu výběru. Dalším užitečným nástrojem ke tvorbě masek je nástroj plechovka pro vyplnění dané oblasti určitou barvou.

5 NÁVRH A IMPLEMENTACE SYSTÉMU

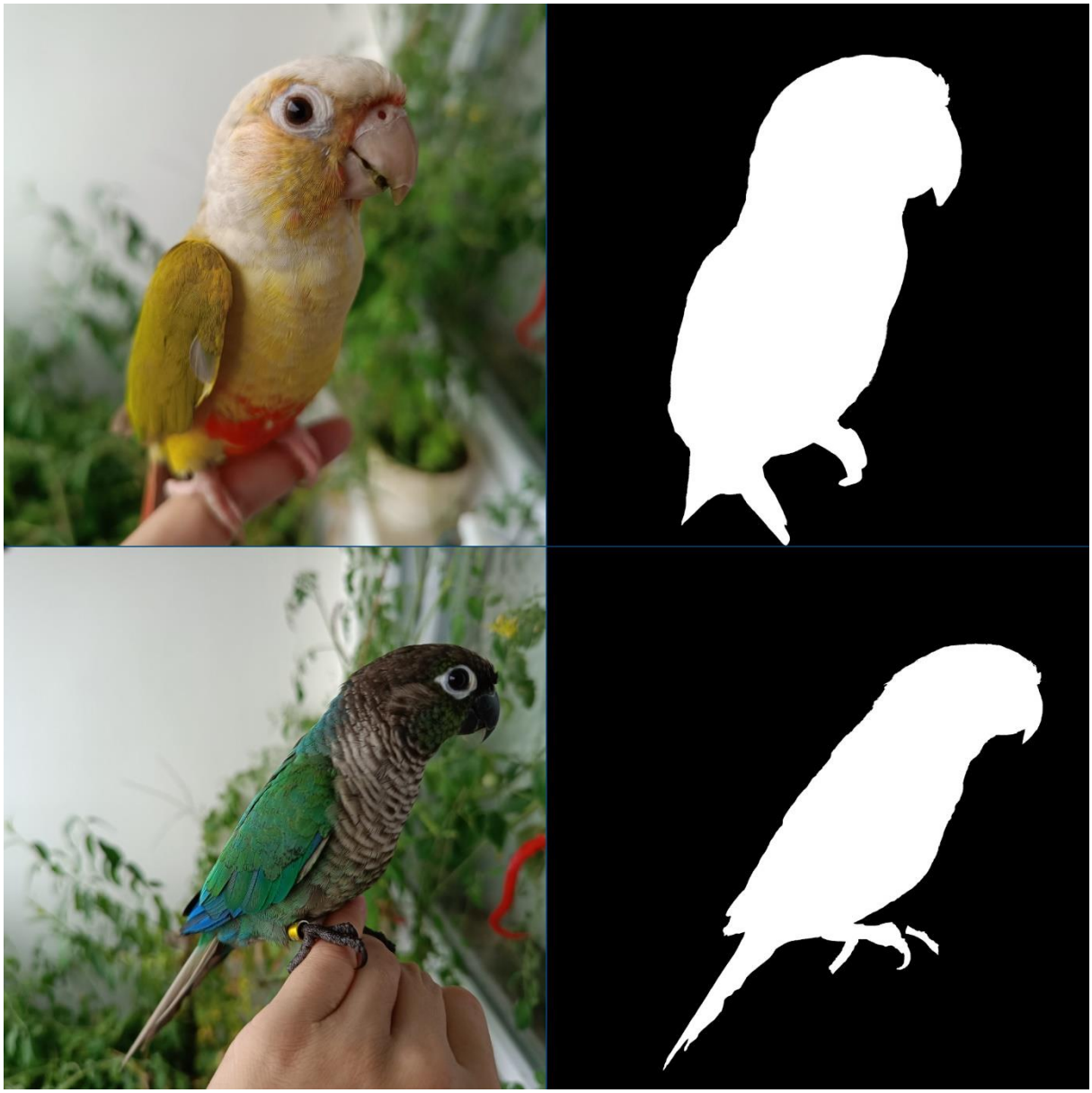
Celkový proces návrhu CNN pro segmentaci obrazu se skládá z několika částí. Nejprve je potřeba si ujasnit, jaké objekty budeme chtít model naučit rozpoznávat a segmentovat. Následně je potřeba podle těchto objektů připravit data pro trénování modelu. Dalším krokem je zvolit technologii pro budování modelu a následně jeho implementace. Po dokončení modelu CNN je potřeba model natrénovat dle připravených dat. Pak už zbývá jen otestovat model na datech, které nebyly použity během trénování a sledovat, jak moc správně náš model data vyhodnocuje. Pokud nejsme spokojeni s výsledkem, je potřeba změnit strukturu CNN, nebo zlepšit naše trénovací data (zlepšení trénovacích dat bývá obvykle častější problém).

5.1 Příprava Dat

Před tvorbou samotné CNN je potřeba přichystat vhodná obrazová data pro trénování modelu. Data pro trénování se skládají ze vstupního obrázku a mapy příznaků, kde je správně určena klasifikace každého pixelu vstupního obrázku (obvykle bílou barvou objekt hledání a černou pozadí). Aby náš model CNN fungoval spolehlivě na předem neznámá obrazová data, je třeba mít těchto dat co největší množství. Mapy příznaků se připravují zpravidla manuálně v různých grafických editorech. Tento proces může být časově velmi náročný, protože takto přichystaných dat je zapotřebí velké množství, řádově tisíců. Snížení nutnosti manuální tvorby mapy příznaků u tak velkého množství dat je možno pomocí grafických transformací a jiných operací, kde vytvoříme mapu příznaků například pouze pro 200 obrázků. Na každý z připravených obrázků opakovaně aplikujeme několik nahodilých grafických transformací (zkosení, otočení, zakřivení, překlopení, posunutí, změna jasu atd.) jak na původní obrázek, tak na mapu příznaků. Použité transformace se musejí přesně shodovat pro každou dvojici originálního obrázku a mapy příznaků, jinak by se model nenaučil správně klasifikovat jednotlivé pixely. Tímto způsobem získáme uměle více variací a náš model má pak mnohem více dat pro trénování bez většího lidského úsilí. Tomuto procesu se říká augmentace dat.

Data pro trénování jsou velmi důležitá a mají významný vliv na kvalitu vyhodnocování vstupních dat. Obecně platí čím více dat k trénování, tím spolehlivější výsledky můžeme očekávat. Musíme však dbát na to, aby všechna trénovací data neobsahovala chyby nebo nepřesnosti. Každá nevalidní trénovací data mohou negativně ovlivnit výsledky vyhodnocování.

Pro tuto práci byla vytvořena vlastní datová sada, která se skládá z 200 zdrojových obrázků. Obrazová data se skládají z vlastních fotek papoušků konkrétního druhu pyrura zelenolící, kteří pochází z Jižní Ameriky. Všechny fotky jsou v poměru 1:1 s rozlišením 2944 x 2944 pixelů. Ke všem těmto fotkám byla potřeba ručně vytvořit maska, ve které je bílou barvou znázorněn papoušek, který je cílem segmentace v této práci. Černou barvou je zbytek obrázku. Ke tvorbě těchto masek byl použit grafický editor GIMP.



Obrázek 18: *Ukázka zdrojových dat s vytvořenou maskou*

5.2 Budování modelu pomocí knihovny Keras

K implementaci CNN pomocí Python knihovny Keras je v práci využit IDE PyCharm. Pycharm dokáže vytvořit virtuální vývojové prostředí, do kterého se instalují všechny závislosti a knihovny pro běh programu. Pro vývoj to může být často výhodné, protože je tím eliminována potřeba instalovat všechny potřebné knihovny pro projekt do samotného systému, kde by pro jiný účel, než vývoj samotného projektu nebyly potřeba.

5.2.1 Image Data Generator

Knihovna Keras obsahuje velmi užitečný nástroj, který nám pomůže s prací se vstupními daty. Ne vždy má uživatel to štěstí, že má k dispozici všechny vstupní obrazová data ve stejném formátu i rozlišení (ať už pro trénování nebo pro testování). Image Data Generátor má funkci pro změnu cílového rozlišení celé datové sady. Zároveň nám dokáže unifikovat hodnoty. Pro konvoluční neuronové sítě je ideální, když se hodnoty pohybují v rozsahu od 0 do 1. Pixely ve vstupních obrázcích dosahují hodnot v rozsahu od 0 do 255. Image data generator nám pomocí parametru *rescale* dokáže změnit rozsah hodnot tím, že je nastaven na 1.0/255. Vydělením všech pixelů číslem 255 je získán požadovaný rozsah hodnot. [28]

Pro správné propojení mezi zdrojovými obrázky a jejich maskami je potřeba s tímto nástrojem dodržet několik pravidel. Je potřeba zvolit stejnou hodnotu *seed* jak pro generátor zdrojových dat, tak pro generátor masek, aby obě dvojice dat byly zpracovávány stejným způsobem. Hodnota *seed* udává pseudonáhodný řetězec čísel, podle kterých se budou data zpracovávat. Tento parametr je tedy velmi důležitý pro augmentaci dat. Jinak by docházelo k rozdílné transformaci zdrojového obrázku a masky, což by vedlo ke špatnému natrénování naší CNN. [28]

Dále název souboru se zdrojovým obrázkem musí být stejný jako název souboru s jeho maskou. Zároveň musejí být umístěny v rozdílném adresáři se stejným názvem, protože název adresáře určuje klasifikační třídu. V této práci jsou vstupní data uložena v následující struktuře.

5.2.2 Struktura zdrojových dat v projektu

```
..
└── data
    ├── fit
    │   ├── src
    │   │   └── parrots
    │   │       ├── file01.jpg
    │   │       ├── file02.jpg
    │   │       └── ...
    │   └── mask
    │       └── parrots
    │           ├── file01.jpg
    │           ├── file02.jpg
    │           └── ...
    ├── val
    │   ├── src
    │   │   └── parrots
    │   │       ├── file01.jpg
    │   │       ├── file02.jpg
    │   │       └── ...
    │   └── mask
    │       └── parrots
    │           ├── file01.jpg
    │           ├── file02.jpg
    │           └── ...
    └── test
        ├── src
        │   └── parrots
        │       ├── file01.jpg
        │       ├── file02.jpg
        │       └── ...
        └── mask
            └── parrots
                ├── file01.jpg
                ├── file02.jpg
                └── ...
```

5.2.3 Augmentace dat

Samotná knihovna Keras obsahuje velice kvalitní nástroje pro augmentaci dat. Funkce *ImageDataGenerator* o které se práce zmiňuje obsahuje mimo jiné i parametry, které augmentují vstupní data. Funkce obsahuje následující parametry.

- ***rotation_range*** (určuje maximální možné natočení obrázku)
- ***width_shift_range*** (určuje maximální posun v horizontální ose)
- ***height_shift_range*** (určuje maximální posun ve vertikální ose)
- ***brightness_range*** (určuje procentuální rozsah jasu)
- ***shear_range*** (určuje maximální zkosení)
- ***zoom_range*** (určuje procentuální rozsah přiblížení)
- ***channel_shift_range*** (určuje maximální posunutí náhodného barevného kanálu)
- ***horizontal_flip*** (povoluje horizontální překlopení)
- ***vertical_flip*** (povoluje vertikální překlopení)

[28]

Nad každým parametrem je potřeba se zamyslet, jestli je vhodné ho použít pro naše zkoumaná data. Například vertikální překlopení obrázku nebude muset vždy odpovídat reálné situaci, když bychom chtěli segmentovat například lidské postavy chodící po chodníku. Lidé neumějí chodit po chodníku vzhůru nohama, proto by v této situaci měl být parametr pro vertikální překlopení vynechán. Stejně tak by se v tomto případě měl rozumně volit parametr pro natočení obrázku. V této práci bylo použito všech dostupných parametrů pro augmentaci dat, včetně parametru pro posunutí náhodného barevného kanálu. Jelikož tento druh papouška, který je předmětem zkoumání segmentace má veliké množství barevných mutací.

5.3 Vlastní architektura CNN

5.3.1 Popis

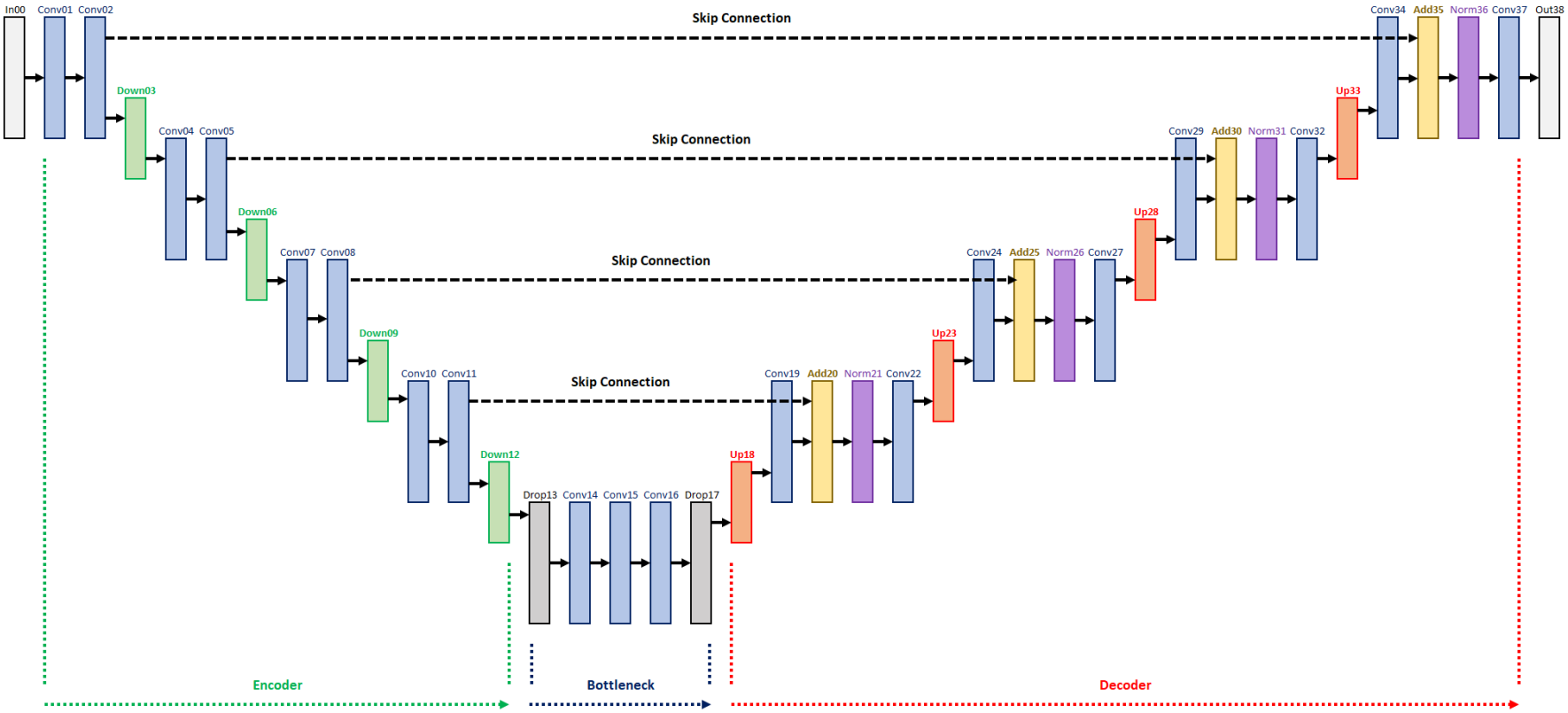
Vlastní architektura se skládá ze tří částí. Dekodér, bottleneck a enkodér. Dekodér se skládá ze čtyřech trojic vrstev. První dvě jsou vždy konvoluční vrstvy a třetí je max pooling vrstva, která snižuje rozlišení. Konvoluční vrstvy mají počet filtrů 64, velikost filtru 3 a využívají aktivační funkci ReLU. Každá konvoluční vrstva v další trojici má dvojnásobný počet těchto filtrů až do počtu 512.

Bottleneck označuje oblast architektury, která pracuje s nejnižším rozlišením. Začíná a končí speciální vrstvou Dropout, která nastavuje s určitou frekvencí náhodné vstupní hodnoty na 0 v každém kroku během doby trénování, což pomáhá zabránit nadměrnému přizpůsobení sítě pro trénovací množinu. Obě Dropout vrstvy mají frekvenci nastavenou na 0,25. Mezi těmito vrstvami jsou umístěny tři konvoluční vrstvy s počtem filtrů 512.

Dekodér se skládá ze čtyřech pětic vrstev. První je inverzní pooling vrstva, která zvyšuje rozlišení. Druhá je konvoluční vrstva. Třetí je slučovací vrstva, která sčítá hodnoty vah z předchozí konvoluční vrstvy a druhé konvoluční vrstvy dekodéru ze stejné hloubky sítě. To pomáhá bránit zkreslovat důležité vzory během snižování rozlišení. Následující vrstva BatchNormalization udržuje střední hodnotu výstupu blízko 0 a směrodatnou odchylku výstupu blízko 1. Z pravidla je výhodnější, když se hodnoty vah pohybuji kolem těchto normalizovaných hodnot. Po předchozí slučovací vrstvě, která sčítá váhy dvou vrstev je pravděpodobné, že dojde k překročení těchto hodnot. Normalizace je tedy po této operaci výhodná. Poslední vrstva z pětice je další konvoluční vrstva. Konvoluční vrstvy dekodéru začínají s počtem filtrů 512, kde každá další pětice snižuje svůj počet filtrů na polovinu až do počtu 64. Na konci sítě se nachází výstupní konvoluční vrstva, která má počet filtru 1 a velikost filtru 1 s aktivační funkcí sigmoid. Funkce sigmoid zajistí výstup hodnot v rozsahu 0 až 1. Počet filtrů 1 zajistí počet výstupních kanálů 1.

S výchozím vstupním tvarem signálu, který je nastaven na 288x288x3, kde 288x288 znamená rozlišení obrázku a 3 určuje počet barevných kanálů (RGB), je nejužší místo sítě s rozlišením 32x32. Síť dokáže pracovat i s vyšším rozlišení a vrací vždy signál ve tvaru původního rozlišení s jedním barevným kanálem. Hodnoty vráceného kanálu jsou v rozsahu hodnot od 0 do 1. Jedná se tedy o obrázek v odstínech šedi. Pro získání mapy segmentace, která obsahuje pouze jednu klasifikační třídu (papoušek), je tedy zapotřebí výstup prahovat.

VLASTNÍ ARCHITEKTURA KONVOLUČNÍ NEURONOVÉ SÍTĚ



***Obrázek 19**: ILustrace vlastní architektury CNN*

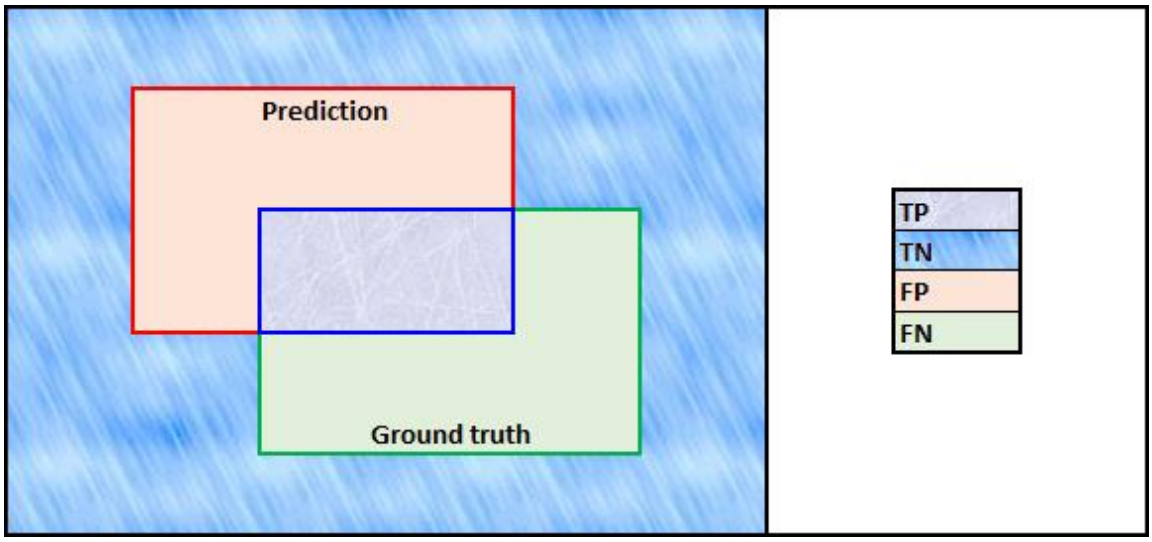
5.3.2 Zdrojový kód

```
def custom(input_shape=(288, 288, 3)):
  # input
  input_00 = Input(shape=input_shape)

  # encoder
  conv_01 = Conv2D(filters=64, kernel_size=3, activation='relu', padding='same')(input_00)
  conv_02 = Conv2D(filters=64, kernel_size=3, activation='relu', padding='same')(conv_01)
  pool_03 = MaxPooling2D()(conv_02)

  conv_04 = Conv2D(filters=128, kernel_size=3, activation='relu', padding='same')(pool_03)
  conv_05 = Conv2D(filters=128, kernel_size=3, activation='relu', padding='same')(conv_04)
  pool_06 = MaxPooling2D()(conv_05)

  conv_07 = Conv2D(filters=256, kernel_size=3, activation='relu', padding='same')(pool_06)
  conv_08 = Conv2D(filters=256, kernel_size=3, activation='relu', padding='same')(conv_07)
  pool_09 = MaxPooling2D()(conv_08)

  conv_10 = Conv2D(filters=512, kernel_size=3, activation='relu', padding='same')(pool_09)
  conv_11 = Conv2D(filters=512, kernel_size=3, activation='relu', padding='same')(conv_10)
  pool_12 = MaxPooling2D()(conv_11)

  # FCN
  drop_13 = Dropout(0.25)(pool_12)
  conv_14 = Conv2D(filters=512, kernel_size=3, activation='relu', padding='same')(drop_13)
  conv_15 = Conv2D(filters=512, kernel_size=3, activation='relu', padding='same')(conv_14)
  conv_16 = Conv2D(filters=512, kernel_size=3, activation='relu', padding='same')(conv_15)
  drop_17 = Dropout(0.25)(conv_16)

  # decoder
  up_18 = UpSampling2D()(drop_17)
  conv_19 = Conv2D(filters=512, kernel_size=3, activation='relu', padding='same')(up_18)
  add_20 = Add()([conv_11, conv_19])
  norm_21 = BatchNormalization()(add_20)
  conv_22 = Conv2D(filters=512, kernel_size=3, activation='relu', padding='same')(norm_21)

  up_23 = UpSampling2D()(conv_22)
  conv_24 = Conv2D(filters=256, kernel_size=3, activation='relu', padding='same')(up_23)
  add_25 = Add()([conv_08, conv_24])
  norm_26 = BatchNormalization()(add_25)
  conv_27 = Conv2D(filters=256, kernel_size=3, activation='relu', padding='same')(norm_26)

  up_28 = UpSampling2D()(conv_27)
  conv_29 = Conv2D(filters=128, kernel_size=3, activation='relu', padding='same')(up_28)
  add_30 = Add()([conv_05, conv_29])
  norm_31 = BatchNormalization()(add_30)
  conv_32 = Conv2D(filters=128, kernel_size=3, activation='relu', padding='same')(norm_31)

  up_33 = UpSampling2D()(conv_32)
  conv_34 = Conv2D(filters=64, kernel_size=3, activation='relu', padding='same')(up_33)
  add_35 = Add()([conv_02, conv_34])
  norm_36 = BatchNormalization()(add_35)
  conv_37 = Conv2D(filters=64, kernel_size=3, activation='relu', padding='same')(norm_36)

  # output
  out_38 = Conv2D(filters=1, kernel_size=1, activation='sigmoid')(conv_37)

  return Model(input_00, out_38, name='custom')
```

5.4 Metriky pro zhodnocení modelu

Zde jsou představeny často používané metriky pro zhodnocení kvality natrénovaného modelu CNN pro segmentaci obrazu. Metriky počítají se čtyřmi stavy, které mohou během vyhodnocení nastat.

- **TP**: Představuje počet pixelů, které byly správně klasifikovány jako papoušek.
- **TN**: Představuje počet pixelů, které byly správně klasifikovány jako pozadí.
- **FP**: Představuje počet pixelů, které byly nesprávně klasifikovány jako papoušek.
- **FN**: Představuje počet pixelů, které byly nesprávně klasifikovány jako pozadí.



Obrázek 20: *Možné stavy během předpovědi CNN*

5.4.1 Precision

Tato metrika je určena počtem všech pozitivně vyhodnocených pixelů předpovědi dělena počtem veškerých pixelů předpovědi. [27]

$$Preccision = \frac{TP}{TP\ +\ FP} = \frac{prediction\ \cap ground\_truth}{prediction}$$

5.4.2 Recall

Tato metrika je určena počtem všech pozitivně vyhodnocených pixelů předpovědi dělena počtem veškerých pixelů pravdivého stavu. [27]

$$Recall = \frac{TP}{TP + FN} = \frac{prediction\ \cap ground\_truth}{ground\_truth}$$

5.4.3 Accuracy

Accuracy je také označována jako Rand index. Tato metrika je určena počtem všech pixelů správných výsledků předpovědi dělena počtem všech pixelů v množině. [27]

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{\neg(prediction\ \Delta\ ground\_truth)}{full\_area}$$

5.4.4 Dice Coefficient

Dice Coefficient je také označován jako F1 Score. Tato metrika je určena dvojnásobkem počtu všech pozitivně vyhodnocených pixelů předpovědi dělena celkovým počtem pixelů předpovědi a pravdivého stavu. [27]

$$Dice = \frac{2TP}{2TP + FP + FN} = \frac{2\ (prediction\ \cap\ ground\_truth)}{prediction + ground\_truth}$$

5.4.5 Jaccard Index

Jaccard Index je také označován zkratkou IoU. Tato metrika je určena počtem všech pozitivně vyhodnocených pixelů předpovědi dělena počtem pixelů sjednocené předpovědi a pravdivého stavu. [27]

$$IoU = \frac{TP}{TP + FP + FN} = \frac{prediction\ \cap ground\_truth}{prediction\ \cup ground\_truth}$$

5.4.6 Implementace

Zde je ukázka konkrétní implementace výpočtu všech již zmíněných metrik v jedné funkci. V prvním kroku jsou vstupní masky funkce nejprve prahovány. Prahování je proces, při kterém jsou hodnoty obrazu zredukovány na dvě hodnoty (0 je černá, 1 je bílá v tomto konkrétním případě). To zamezí chybám během výpočtu v případě, kdyby měl uživatel své vlastní připravené masky v odstínech šedi.

Funkce přijímá dva vstupní parametry. *Ground truth* a *prediction*, které jsou reprezentovány maskami (*ground truth* je předpřipravená maska a *prediction* je předpovězená maska). Z těchto dvou vstupních množin jsou nejprve získány pomocí maticových operací všechny potřebné hodnoty stavů (TP, TN, FP a FN), se kterými počítají již zmíněné rovnice daných metrik. Po získaní všech těchto hodnot jsou dosaženy do rovnic a vypočítány. Funkce vrací výsledky všech metrik v poli.

```
def calculate_metrics(ground_truth, prediction):
  ret, ground_truth = cv2.threshold(src=ground_truth,
                                    thresh=0.5,
                                    maxval=1,
                                    type=cv2.THRESH_BINARY)

  ret, prediction = cv2.threshold(src=prediction,
                                  thresh=0.5,
                                  maxval=1,
                                  type=cv2.THRESH_BINARY)

  full_area = prediction.size

  tp = numpy.sum(prediction * ground_truth)
  fp = numpy.sum(prediction) - tp
  fn = numpy.sum(ground_truth) - tp
  tn = full_area - tp - fp - fn

  precision = tp / (tp + fp)
  recall = tp / (tp + fn)
  accuracy = (tp + tn) / full_area
  dice = (2 * tp) / (2 * tp + fp + fn)
  iou = tp / (tp + fp + fn)

  return [precision, recall, accuracy, dice, iou]
```

5.5 Trénování modelu

Po dokončení CNN je potřeba model natrénovat pomocí připravených trénovacích dat. Je zapotřebí mít správně spárované všechny dvojice dat (původní obrázek a mapu segmentace).

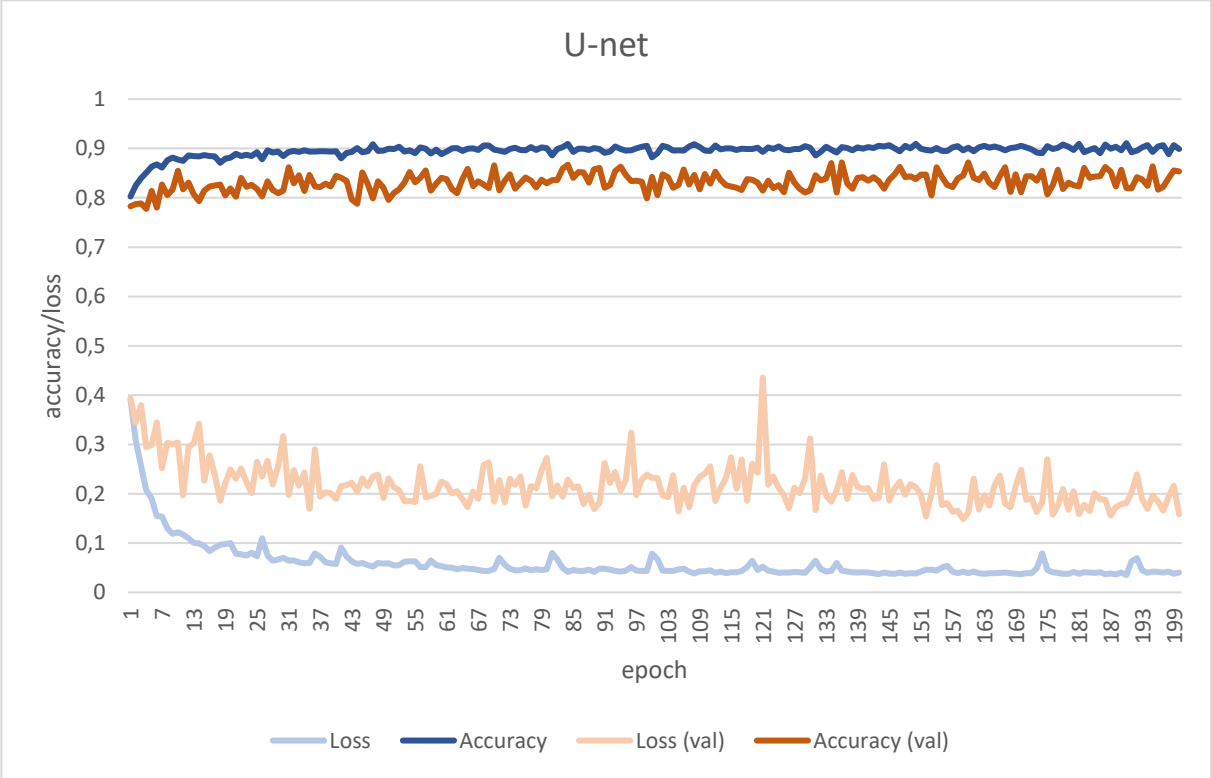
Data k trénování je vhodné rozdělit na takzvanou trénovací a validační množinu. Validační množina je určena ke kontrole nadměrného přizpůsobení vah modelu pro trénovací data. K nadměrnému přizpůsobení dochází tehdy, když je model naučen pouze specifikami tréninkových dat a není schopen dobře zobecnit data, na kterých nebyl trénován. Model takto trénuje své parametry z vlastností dat v trénovací množině, a následně předpovídá na validační množině. Během každé epochy uvidíme nejen výsledky ztrát a přesnosti pro trénovací množinu, ale také pro validační množinu. To umožňuje zjistit, jak dobře se model zobecňuje na datech, na kterých nebyl trénován, protože validační data nejsou součástí trénovacích dat. [19]

Před zahájením trénování je dobrým zvykem odebrat část dat z trénovací množiny a umístit ji do validační množiny. V knihovně Keras stačí použít parametr *validation_split,* který je možné nastavit přímo ve funkci *fit*. Funkce *fit* je určena pro natrénování modelu CNN. Parametr lze nastavit jako reálné číslo v rozsahu 0 až 1, které určuje relativní hodnotu množiny, která má být určena jako validační (0,1 rozdělí trénovací množinu na 10% validační a 90% trénovací). Tento parametr však není dostupný pro vstupní data z *ImageDataGenerator*. Další možností je odděleně vytvořit validační množinu a do funkce *fit* ji celou vložit jako parametr *validation_data,* nebo rozdělit data již v *ImageDataGenerator* pomocí stejně pojmenovaného parametru *validation_split*. Pokud je tato validační množina vkládána přímo z generátoru, je potřeba dodatečně nastavit parametr *validation_steps* ve funkci *fit*. Tento parametr nastaví celkový počet kroků, tedy počet zpracovaných vstupních dat před zastavením validace. Jelikož data z generátoru mohou představovat neomezený počet variací dat, trénování by takto skončilo v nekonečné smyčce, protože bez nastavení tohoto parametru je ve výchozím stavu prováděna validace, dokud není vstupní validační množina vyčerpána. [19]

Trénování modelu může trvat různě dlouhou dobu a závisí na složitosti modelu, na nastavení parametrů trénování, počtu trénovacích dat, rozlišení vstupních dat a rychlosti hardwaru, na kterém je trénování prováděno. Všechny CNN byly trénovány na stejné trénovací množině s rozlišením 288x288 pixelů o třech barevných kanálech.

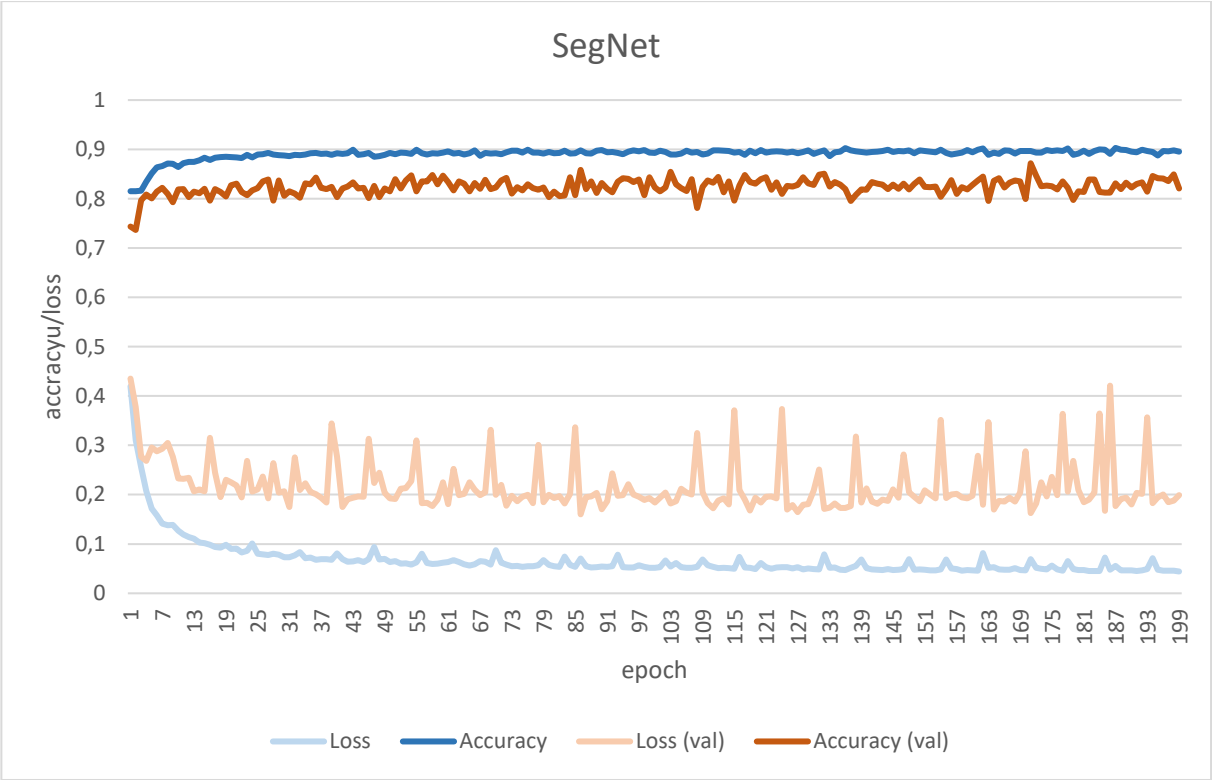
- CPU: AMD Ryzen 9 5950X (16 – Core)
- RAM: Corsair Vengeance LPX Black (2x 32 GB) DDR4 3200 MHz CL16
- MoBo: ASUS ProArt X570-CREATOR WIFI – AMD X570

K trénování byl tedy využit relativně výkonný CPU. Nebyla dostupná žádná kompatibilní GPU, která by splňovala požadavky pro využívaný software v této práci. V následujících přílohách lze vidět grafy s průběhem trénování konkrétních CNN.



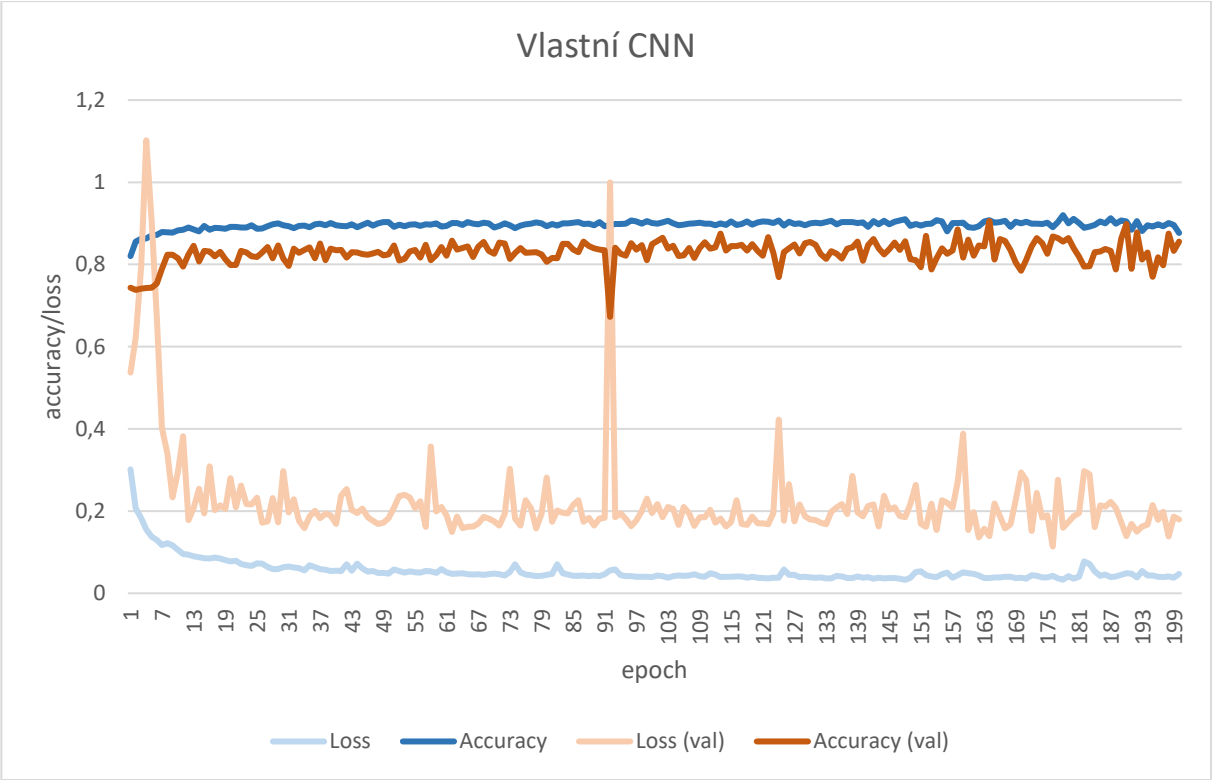
Obrázek 21: *Graf průběhu trénování sítě U-net*

Metrika accuracy na trénovací množině byla v nejlepším případě 91,07 % a hodnota ztrátové funkce 0,0353. Pro validační množinu byla accuracy v nejlepším případě 87,18 % a hodnota ztrátové funkce 0,1486. Accuracy se na validační množině od epochy 150 již nezlepšuje. Dle grafu by další epochy trénování zlepšovali přesnost sítě už jen ve vlastní trénovací sadě. Síť lze tedy prohlásit za dostatečně natrénovanou.



Obrázek 22: *Graf průběhu testování sítě SegNet*

Z grafu lze pozorovat velmi podobné charakteristiky jako při trénování CNN U-Net. Accuracy na trénovací množině byla v nejlepším případě 90,3 % a hodnota ztrátové funkce 0,0442. Pro validační množinu byla v nejlepším případě accuracy 87,18 % a hodnota ztrátové funkce 0,1601.



Obrázek 23: *Graf průběhu testování vlastní sítě*

Z grafu lze pozorovat občasné výkyvy ztrátové funkce v určité epoše. Na trénování však nemá žádný vliv. Model byl trénován 4x po padesáti epochách. Přibližně od sté epochy již nedocházelo k žádnému zlepšení na validační množině. Accuracy na trénovací množině byla v nejlepším případě 92,0 % a hodnota ztrátové funkce 0,0327. Pro validační množinu byla v nejlepším případě přesnost 90,59 % a hodnota ztrátové funkce 0,1137.

CNN	Time per epoch [s]	Samples [počet]	Batch size [počet]	Input [px]	Accuracy (val) [%]
U-Net	817	1024	8	288x288x3	87,1798
SeGnet	272	512	8	224x224x3	87,1845
Custom	479	800	10	288x288x3	90,5917

Tabulka 1*: Souhrn údajů jednotlivých CNN z trénování*

5.6 Testování Modelu

Testování pomáhá zjistit kvalitu natrénovaného modelu CNN. K otestování je nutné použít taková vstupní obrazová data, která nebyla použita během trénování. Data, která byla použita během trénování by pozitivně zkreslovala výsledky testování.

V knihovně Keras je pro tento účel přichystaná funkce *evaluate*, jejichž vstupní parametry jsou data zdrojových obrázků a jejich masek. Data z generátoru jsou také podporována. Při použití dat z generátoru je opět zapotřebí zvolit počet vzorků, který má funkce započíst do vyhodnocení pomocí parametru *steps*. Jinak by se program zasekl v nekonečné smyčce stejně tak jako u trénování s validační množinou. Tato funkce však vypočítá pouze metriku accuracy a hodnotu ztrátové funkce, která je zajímavá spíše během trénování. V práci je proto využita vlastní implementace pro výpočet ostatních metrik, která je již popsaná v jedné z předchozích kapitol na straně *40*.

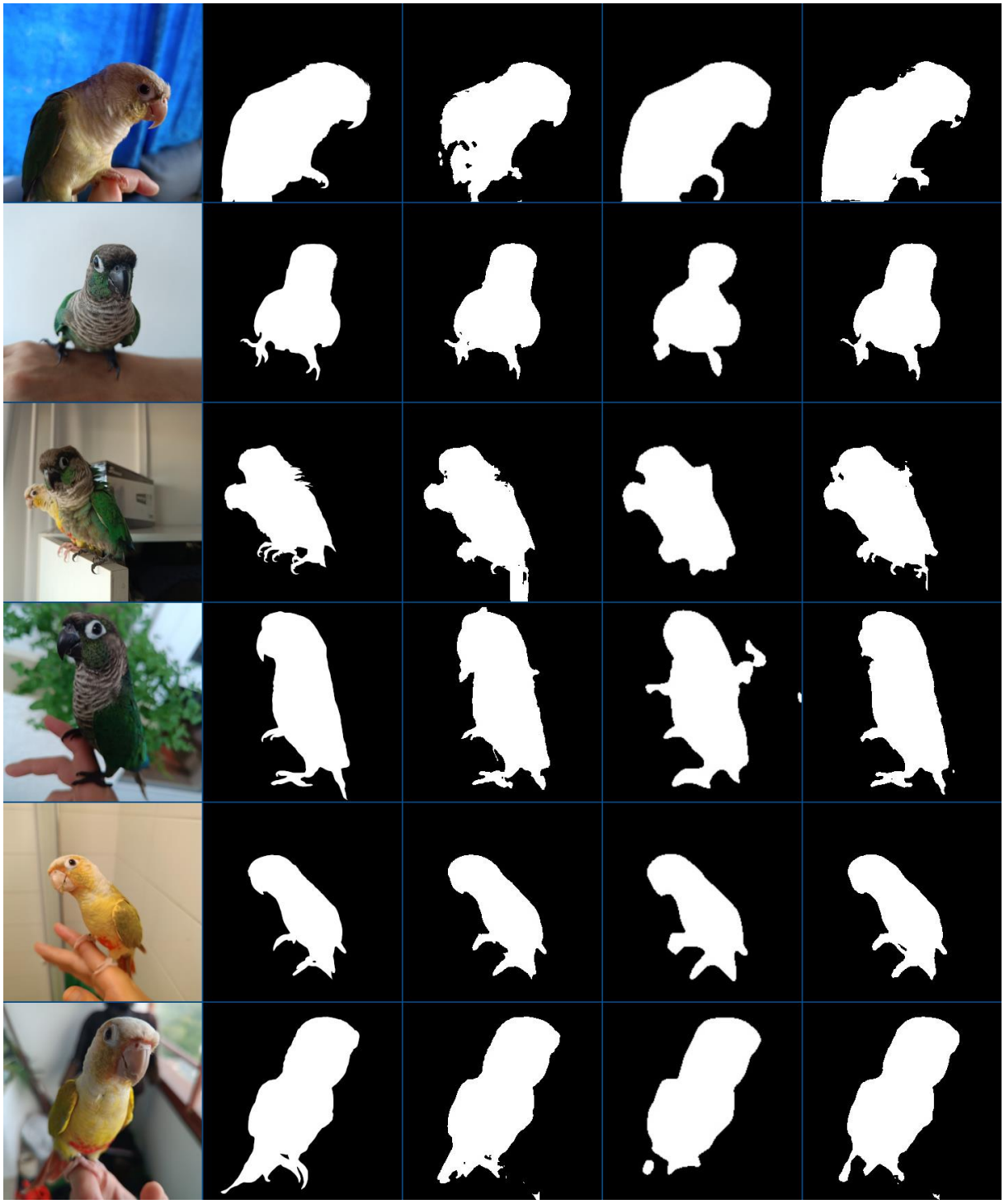
V následující tabulce jsou ukázány průměrné výsledky metrik natrénovaných modelů CNN z rozsahu celé testovací množiny. Všechny modely byly testovány na stejné testovací množině. Vlastní CNN dosáhla nejlepších výsledků u všech metrik kromě metriky preccision, u které dopadla naopak nejhůře.

CNN	Preccision	Recall	Accuracy	Dice Coefficient	Jaccard Index
U-Net	0,9183026	0,8112136	0,9488084	0,8486840	0,7592023
SegNet	0,8892470	0,7464398	0,9305283	0,8010284	0,6916630
Vlastní	0,8248310	0,9177127	0,9548134	0,8618948	0,7780690

Tabulka 2: Průměrné hodnoty metrik natrénovaných modelů z celé testovací množiny

5.6.1 Visuální porovnání

Zde je ukázka kvality segmentace z různých modelů CNN. V každém řádku představuje první obrázek originální vstupní data, druhý obrázek představuje manuální mapu segmentace, třetí obrázek představuje výstup z CNN U-net, čtvrtý obrázek představuje výstup z CNN SegNet a poslední obrázek představuje výstup z vlastní CNN.



***Obrázek 24**: Visuální porovnání výstupů mezi jednotlivými modely CNN*

5.7 Aplikování Filtru

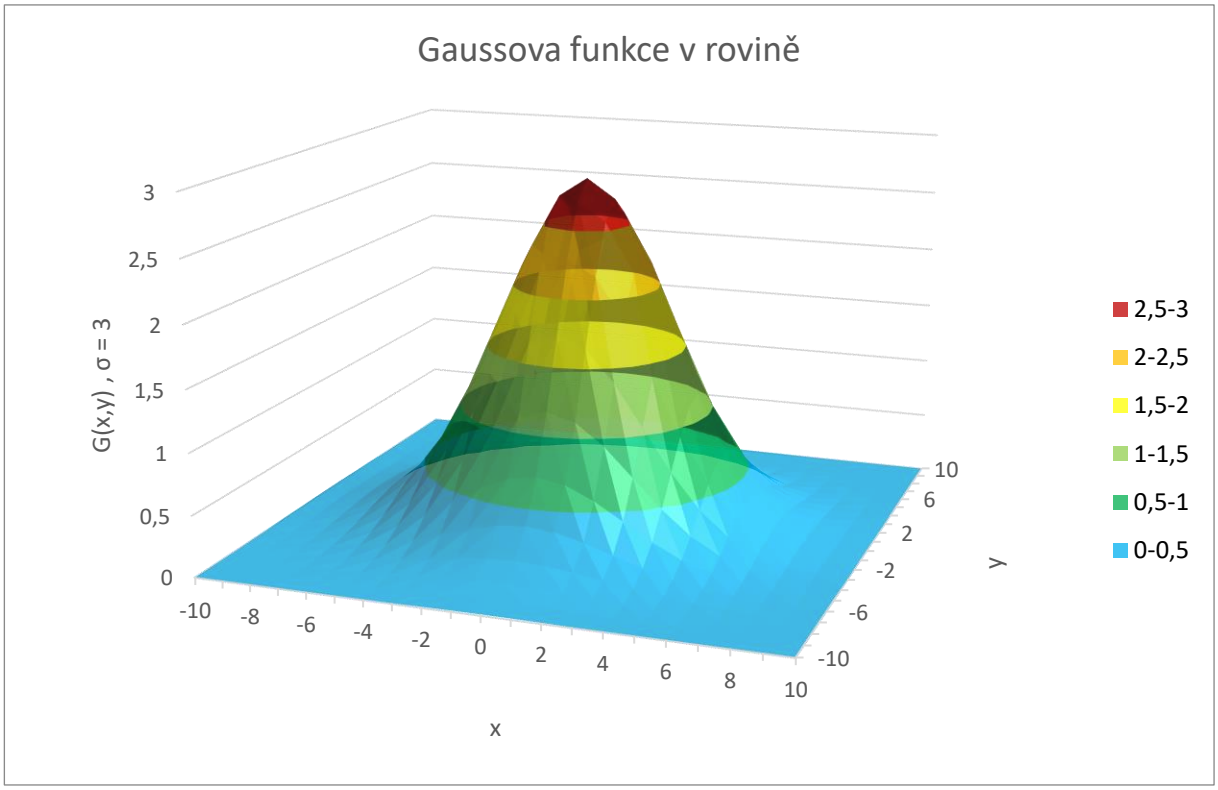
Výstup CNN vrací předpovězenou masku, s kterou je možné dále pracovat. V práci je maska dále použita jako vstupní parametr pro grafický filtru, který je aplikován na originální vstupní data. Celkový proces je složen ze dvou fází.

V první fázi je aplikováno Gaussovo rozostření na pixely originálního obrázku, které CNN vyhodnotí jako pozadí (černá barva v předpovězené masce). Gaussovo rozostření je počítáno vždy z okolí daného pixelu. Velikost okolí je možné libovolně zvolit. Vzdálenější pixely méně ovlivňují výslednou změnu. V praxi tedy není dobré používat příliš veliké vzdálenosti okolí. Zvyšuje se výpočetní náročnost s minimálním vlivem na výslednou hodnotu. Rovnice Gaussovy funkce v rovině vypadá následovně. [23]

$$G_{(x,y)} = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Kde ***x*** a ***y*** určují relativní index polohy pixelu od středového bodu, který rovnice počítá. Je-li zvolena velikost okolí na hodnotu 3 a středové souřadnice počítaného pixelu jsou [5, 5], bude se výpočet vztahovat na všechny pixely v rozsahu od [4, 4] do [6, 6].

σ je směrodatná odchylka, která určuje, jak moc okolní pixely ovlivňují hodnotu středového pixelu.



Obrázek 25: *Graf Gaussovy funkce v rovině*

Pomocí rovnice jsou spočítány váhy okolních pixelů. Váha určuje, jak moc daný pixel ovlivní počítaný středový bod. Výsledek středového bodu je určen pomocí operace konvoluce mezi původním obrázkem a váhami okolních pixelů, které jsou získány Gaussovou funkcí v rovině. Jestliže okolí přesahuje hranu obrázku, počítá se obvykle s hodnotou krajového pixelu. [23]

V druhé fázi je získán jas v každém obrazovém bodu podle následujícího vzorce.

```
brightness = 0.299 * red + 0.587 * green + 0.114 * blue
```

Pomocí tohoto vzorce je získána mnohem spolehlivější hodnota jasu, než kdyby byl každý barevný kanál rozdělen na třetiny, protože lidské oko je od přírody citlivější na zelenou barvu a zároveň není tak citlivé na modrou barvu. Po získání hodnoty jasu je touto hodnotou nahrazena hodnota červeného barevného kanálu u množiny pixelů, kterou CNN klasifikovala jako papoušek a modrého barevného kanálu u množiny pixelů, kterou CNN klasifikovala jako pozadí. Ostatní barevné kanály všech pixelů jsou nastaveny na 0.

Tímto postupem by mělo být docíleno efektu modrého lehce rozmazaného pozadí s červeně zvýrazněným papouškem v původní ostrosti. V následujícím obrázku jsou ukázky výsledného filtru spolu s originálním obrázkem a jeho maskou, kterou předpověděla CNN.

Obrázek 26: *Ukázka výsledného obrazu s aplikováním filtru*

5.7.1 Implementace pomocí cyklu

Implementace tohoto filtru pomocí cyklu je relativně intuitivní záležitost, kde stačí procházet pixel po pixelu a přepočítávat jeho hodnotu. Výkonově se však jedná o velmi pomalý proces, zvláště u obrazových dat vysokého rozlišení.

```
def custom_filter_cycle(origin, mask):
  result = numpy.zeros(shape=origin.shape)
  blur_effect= cv2.GaussianBlur(src=origin, ksize=(7, 7), sigmaX=3)

  for w in range(len(result)):
      for h in range(len(result[w])):
          if mask[w][h]:
              brightness = 0.299 * origin[w][h][2]
                         + 0.587 * origin[w][h][1]
                         + 0.114 * origin[w][h][0]
              result[w][h][2] = brightness
          else:
              brightness = 0.299 * blur_effect[w][h][2]
                         + 0.587 * blur_effect[w][h][1]
                         + 0.114 * blur_effect[w][h][0]
              result[w][h][0] = brightness

    return result
```

5.7.2 Implementace pomocí maticových operací

Pomocí maticových operací lze dosáhnout stejného výstupu jako při procházení v cyklu, ovšem za výrazně kratší výpočetní dobu. Knihovna NumPy má k dispozici velké množství maticových operací, které jsou optimalizovány pro specifické instrukce CPU.

```
def custom_filter_matrix(origin, mask):
  mask_inv = numpy.ones(shape=mask.shape) - mask
  blur_effect = cv2.GaussianBlur(src=origin, ksize=(7, 7), sigmaX=3)

  brightness_orig = origin[:, :, 2] * 0.299
                  + origin[:, :, 1] * 0.587
                  + origin[:, :, 0] * 0.114

  brightness_blur = blur_effect[:, :, 2] * 0.299
                  + blur_effect[:, :, 1] * 0.587
                  + blur_effect[:, :, 0] * 0.114

    result = numpy.zeros(origin.shape)
    result[:, :, 2] = brightness_orig * mask
    result[:, :, 0] = brightness_blur * mask_inv

    return result
```

6 UŽIVATELSKÝ MANUÁL

Program je tvořen především pro vývoj a experimentování. Je proto napsaný pouze jako konzolová aplikace, která se spouští hlavním skriptem main.py. Skript má k dispozici několik příkazů a povinných i volitelných argumentů.

6.1 Příkazy

- **train**: Natrénuje zvolenou CNN, která je následně uložena do souboru. Soubor je uložen do adresáře *./models/* s názvem *{net_name_index.keras}*. Název souboru se generuje automaticky podle typu vybrané CNN a zvyšuje index o jedničku, pokud se v adresáři najde stejně pojmenovaný soubor. Tím je zajištěno, aby program nepřepsal již uložený model.
- **predict**: Nahraje a následně použije zvolenou uloženou CNN v souboru na predikci obrazových dat v určitém adresáři. Je možné volit mezi grafickým zobrazením pro manuální porovnání, nebo uložení grafických výstupů do adresáře. Příkaz dále umožňuje použít barevný filtr, který je závislý na předpovězenou masku.
- **retrain***:* Nahraje zvolenou uloženou CNN v souboru a začne ji dále trénovat. Funkce splňuje stejnou roli jako funkce *train* s rozdílem využití již přichystaného modelu (místo výběru nového modelu, který program podporuje), který může mít již natrénované váhy. Nově natrénovaný model je uložen jako nový soubor.
- **evaluate**: Porovná připravené a předpovězené masky mezi sebou. Pro porovnání použije několik základních metrik (accuracy, recall, preccision, dice coefficient, Jaccard index), o kterých se tato práce již zmiňuje. Výsledek metrik lze vypsat do konzole, nebo uložit do souboru dle parametrů.

6.2 Použití

- *python3 ./main.py train <model_type>*
  [-o] [<origin_folder>]
  [-m] [<mask_folder>]
  [-e] [<epoch_count>]
  [-b] [<batch_size>]
  [-v] [<verbose_level>]
  [-s] [<seed_value>]

- *python3 ./main.py predict <model_name>*
  [-o] [<origin_folder>]
  [-m] [<mask_folder>]
  [-r] [<result_folder>]
  [-g]
  [-f]

- *python3 ./main.py retrain <model_name>*
  [-o] [<origin_folder>]
  [-m] [<mask_folder>]
  [-e] [<epoch_count>]
  [-b] [<batch_size>]
  [-v] [<verbose_level>]
  [-s] [<seed_value>]

- *python3 ./main.py evaluate <-o> <origin_mask_folder>*
  <-p> <predict_mask_folder>
  [-s] [<image_size>]
  [-r] [<result_filepath>

6.3 Agrumenty příkazu train

- **Povinné Argumenty**
    - ***model_type*** {unet, segnet, custom}: Vybere jednu programem podporovanou CNN pro trénování.
- **Volitelné Argumenty**
    - ***-h*** nebo ***–help:*** Zobrazuje nápovědu k použití.
    - ***-o*** nebo ***–origin:*** Nastavuje adresář ke zdrojovým obrázkům. Výchozí cesta je nastavena na *./data/fit/src/*.
    - ***-m*** nebo ***–mask:*** Nastavuje adresář k připraveným maskám. Výchozí cesta je nastavena na *./data/fit/mask/*.
    - ***-e*** nebo ***–epochs:*** Nastavuje počet epoch během trénování. Výchozí hodnota je nastavena na 50.
    - ***-b*** nebo ***–batch:*** Nastavuje velikost dávky pro trénování. Výchozí hodnota je nastavena na 8.
    - ***-v*** nebo ***–verbose:*** Nastavuje úroveň informací, které se zobrazí v konzoli během trénování. Jedná se o celočíselnou hodnotu, kde vyšší hodnota zvyšuje úroveň informací. Výchozí hodnota je nastavena na 1.
    - ***-s*** nebo ***–seed:*** Nastavuje hodnotu seed. Výchozí hodnota je nastavena na 22.

6.4 Agrumenty příkazu predict

- **Povinné argumenty**
    - ***model_name***: Určuje název souboru bez přípony, který se má načíst jako připravený model CNN pro segmentaci dat. Soubor by se měl nacházet v následující relativní cestě v kořenovém adresáři projektu *./models/*. Program pracuje pouze se soubory s příponou *.keras*.
- ***Volitelné argumenty***
    - ***-h*** nebo ***–help:*** Zobrazuje nápovědu k použití.
    - ***-o*** nebo ***–origin:*** Nastavuje adresář ke zdrojovým obrázkům. Výchozí cesta je nastavena na *./data/test/src/*.
    - ***-m*** nebo ***–mask:*** Nastavuje adresář k připraveným maskám. Pokud je nastaven, ke grafickému výstupu je navíc zobrazena připravená maska pro vizuální porovnání s předpovězenou maskou.
    - ***-g*** nebo ***–graphics:*** Výstup neuronové sítě se při nastavení tohoto příznaku začne postupně zobrazovat graficky do systémového okna. Přepínat mezi dalšími výstupy lze libovolnou klávesovou zkratkou. Jako grafický výstup je zobrazen původní obrázek a jeho předpovězená maska, která je výstupem použité CNN. Všechny obrazové výstupy jsou zobrazeny v cílovém rozlišení modelu.
    - ***-f*** nebo ***–filter:*** Zadáním tohoto argumentu jsou k výstupu přidána obrazová data navíc. Přidaná obrazová data jsou složena z originálního obrázku, na kterém je navíc použit vlastní filtr, který má ovlivněné vlastnosti podle předpovězené masky z CNN (tento výstup je buď zobrazen v okně nebo uložen do cílového adresáře dle ostatních argumentů).
    - ***-r*** nebo ***–result:*** Adresář pro ukládání předpovězených masek obrazových dat. Při nezadání tohoto argumentu se data nikam neukládají.

6.5 Agrumenty příkazu retrain

- **Povinné argumenty**
    - ***model_name***: Určuje název souboru bez přípony, který se má načíst jako připravený model CNN pro segmentaci dat. Soubor by se měl nacházet v následující relativní cestě v kořenovém adresáři projektu *./models/*. Program pracuje pouze se soubory s příponou *.keras*.
- **Volitelné argumenty**
    - ***-h*** nebo ***–help:*** Zobrazuje nápovědu k použití.
    - ***-o*** nebo ***–origin:*** Nastavuje adresář ke zdrojovým obrázkům. Výchozí cesta je nastavena na *./data/fit/src/*.
    - ***-m*** nebo ***–mask:*** Nastavuje adresář k připraveným maskám. Výchozí cesta je nastavena na *./data/fit/mask/*.
    - ***-e*** nebo ***–epochs:*** Nastavuje počet epoch během trénování. Výchozí hodnota je nastavena na 50.
    - ***-b*** nebo ***–batch:*** Nastavuje velikost dávky pro trénování. Výchozí hodnota je nastavena na 8.
    - ***-v*** nebo ***–verbose:*** Nastavuje úroveň informací, které se zobrazí v konzoli během trénování. Jedná se o celočíselnou hodnotu, kde vyšší hodnota zvyšuje úroveň informací. Výchozí hodnota je nastavena na 1.
    - ***-s*** nebo ***–seed:*** Nastavuje hodnotu seed. Výchozí hodnota je nastavena na 22.

6.6 Argumenty příkazu evaluate

- **Povinné argumenty**
    - ***-o*** nebo ***–origin***: Je určen pro výběr adresáře, ze kterého se vezmou předem připravené masky obrazových dat.
    - ***-p*** nebo ***–predict***: Je určen pro výběr adresáře, ze kterého se vezmou obrazová data, která vrátila CNN.
- **Volitelné argumenty**
    - ***-h*** nebo ***–help:*** Zobrazuje nápovědu k použití.
    - ***-s*** *nebo* ***–size:*** Nastavuje velikost čtvercového rozlišení v pixelech. Výchozí hodnota je nastavena na 288.
    - ***-r*** nebo ***–result:*** Zvolí cestu k souboru, kde mají být uloženy výsledky metrik. Při nezadání tohoto argumentu se výsledné metriky zobrazí do konzole.

6.7 Příklady

Kompiluje programem definovanou CNN U-Net, která je následně trénována dle výchozích parametrů. Po natrénování je model uložen do adresáře *./models/unet_{index}*. Data pro trénování jsou použita z výchozího adresáře *./data/src/* pro zdrojové obrázky a *.data/mask/* pro mapy příznaků.

- `python3 ./main.py train unet`

Kompiluje programem definovanou CNN SegNet, která je následně trénována s počtem epoch 100 a s velikostí dávky 10.

- `python3 ./main.py train segnet -e 100 -b 10`

Kompiluje programem definovanou CNN SegNet, která je následně trénována dle výchozích parametrů bez dodatečných informačních hlásek do konzole.

- `python3 ./main.py train segnet -v 0`

Nahrává připravený model ze souboru ./models/unet_1.keras, který je následně použit pro predikci dat z výchozího adresáře *./data/test/src*. Výstup spolu s originálem transformovaným do rozlišení, se kterým pracuje CNN je postupně zobrazen v systémovém okně. Žádná data nejsou uložena do souborů.

- `python3 ./main.py predict unet_1 -g`

Nahrává připravený model ze souboru ./model/segnet_1.keras, který je následně použit pro predikci dat z výchozího adresáře *./data/test/src*. Výstupní predikované masky z CNN jsou uloženy do adresáře ./result.

- `python3 ./main.py predict segnet_1 -r ./result`

Nahrává připravený model ze souboru ./models/unet_1.keras, který je následně použit pro predikci dat z výchozího adresáře *.data/test/src*. K výstupu je navíc přidána vlastní maska z adresáře *./data/test/mask/* pro vizuální porovnání s výstupem CNN.

- `python3 ./main.py predict unet_1 -m ./data/test/mask -g`

Nahrává připravený model ze souboru ./models/unet_1.keras, který je následně použit pro predikci dat z výchozího adresáře *./data/test/src*. Výstup spolu s originálem transformovaným do rozlišení, s kterým pracuje CNN postupně zobrazí v systémovém okně. K výstupu navíc přidává další zobrazení s aplikováním barevného filtru.

- `python3 ./main.py predict -g -f`

Nahrává připravený model ze souboru ./model/segnet_1.keras, který je následně použit pro predikci dat z výchozího adresáře *./data/test/src*. Výstupní predikované masky z CNN uloží do adresáře ./result/. Kromě masek jsou do stejného adresáře uloženy i originály s aplikováním barevného filtru.

- `python3 ./main.py predict segnet_1 -r ./result/ -f`

Nahrává připravený model ze souboru ./model/segnet_1.keras (model již může mít natrénované váhy), který je následně trénován s počtem epoch 50 a velikostí dávky 8. Po natrénování je model uložen jako nový soubor do adresáře *./models/unet_{index}*. Data pro trénování se použijí z výchozího adresáře *./data/src/* pro zdrojové obrázky a *.data/mask/* pro mapy příznaků.

- `python3 ./main.py retrain unet_1 -e 50 -b 8`

Porovnává připravené a předpovězené masky ze zadaných adresářů (*./data/test/mask* pro připravené a *./data/test/out* pro předpovězené). Do konzole vypíše výsledky běžně používaných metrik pro vyhodnocování natrénovaného modelu CNN.

- ```
  python3 ./main.py evaluate -o ./data/test/mask \
  -p ./data/test/out
  ```

Porovnává připravené a předpovězené masky ze zadaných adresářů (*./data/test/mask* pro připravené a *./data/test/out* pro předpovězené). Výsledky běžně používaných metrik jsou uloženy do souboru *./result/metrics.txt*.

- ```
  python3 ./main.py evaluate -o ./data/test/mask \
  -p ./data/test/out -r ./result/metrics.txt
  ```

ZÁVĚR

V práci byly nejprve představeny základní úlohy počítačového vidění. Popsány základní prvky, ze kterých se skládají konvoluční neuronové sítě a ukázány některé ze známých topologií z literatury pro sémantickou segmentaci. Dále byly představeny nástroje a technologie pro potřebné kroky ke tvorbě vlastní konvoluční neuronové sítě, pro segmentaci obrazu a v poslední řade vysvětlen postup samotné implementace s návrhem a vytvořením vlastního modelu CNN s aplikováním vlastního barevného filtru.

Konvoluční neuronové sítě se ukázaly jako velmi silný nástroj pro problematiku počítačového vidění. U segmentace je však problematický sběr dat pro trénovací množinu. Manuální tvorba masek pro určení správného výstupu může být časově velmi náročná, protože je těchto dat potřeba opravdu velké množství, aby model předpovídal s co největší přesností i na neznámých datech. Zároveň samotné trénování se ukázalo jako výpočetně velmi náročný proces i pro výkonnější CPU. Rozsáhlejší vývoj by se tedy neobešel bez akcelerace na podporované GPU.

Pro natrénování vybraných topologií byla vytvořena vlastní datová sada, která je složena z vlastních fotek papoušků druhu pyrura zelenolící. Ke každé fotce bylo potřeba manuálně dodělat i správné řešení ve formě mapy příznaků podle které se může CNN učit a optimalizovat tak své parametry. Mapa příznaků je tvořena dvoubarevným obrázkem se stejným rozlišením jako zdrojový obrázek, kde jsou pixely představovaného papouška znázorněny bílou barvou a zbytek obrázku černou barvou.

V praktické části byla vytvořena konzolová aplikace určená pro experimentování s různými topologiemi CNN (trénování, předpovídání, vyhodnocování atd.) s možností aplikování barevného filtru, který je počítán na základě vstupního obrázku a masky, kterou vrátí zvolená CNN. Veškeré výstupy lze ukládat do souboru i zobrazovat v okně pro visuální kontrolu.

Výstupem této práce rozhodně není model s optimalizovanými váhami, který by dokázal předpovídat s naprostou přesností. Práce je určena spíše pro experimentování s různou topologií. Navzdory tomu byly u některých topologií získány relativně slušné výsledky předpovědi masek, když je brána v potaz složitost rozpoznávaných tvarů.

POUŽITÁ LITERATURA

[1] RONNEBERGER, Olaf, Philipp FISCHER a Thomas BROX. U-Net: Convolutional Networks for Biomedical Image Segmentation. In: NAVAB, Nassir, Joachim HORNEGGER, William M. WELLS a Alejandro F. FRANGI, ed. *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* [online]. Cham: Springer International Publishing, 2015, 2015-11-18, s. 234-241 [cit. 2023-02-14]. Lecture Notes in Computer Science. ISBN 978-3-319-24573-7. Dostupné z: doi:10.1007/978-3-319-24574-4_28

[2] SHELHAMER, Evan, Jonathan LONG a Trevor DARRELL. Fully Convolutional Networks for Semantic Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. 2017, **39**(4), 640-651 [cit. 2023-02-14]. ISSN 0162-8828. Dostupné z: doi:10.1109/TPAMI.2016.2572683

[3] BADRINARAYANAN, Vijay, Alex KENDALL a Roberto CIPOLLA. SegNet: a Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. 2017, **39**(12), 2481-2495 [cit. 2023-02-14]. ISSN 0162-8828. Dostupné z: doi:10.1109/TPAMI.2016.2644615

[4] JORDAN, Jeremy. An overview of semantic image segmentation. *Data Science* [online]. 2018 [cit. 2023-02-15]. Dostupné z: https://www.jeremyjordan.me/semantic-segmentation/

[5] WANG, Li, Xingxing CHEN, Liangyuan HU a Hui LI. Overview of Image Semantic Segmentation Technology. In: *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)* [online]. IEEE, 2020, 2020-12-11, s. 19-26 [cit. 2023-02-15]. ISBN 978-1-7281-5244-8. ISSN 26932865. Dostupné z: doi:10.1109/ITAIC49862.2020.9338770

[6] ZHAO, Hengshuang, Jianping SHI, Xiaojuan QI, Xiaogang WANG a Jiaya JIA. Pyramid Scene Parsing Network. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. IEEE, 2017, 2017, s. 6230-6239 [cit. 2023-03-08]. ISBN 978-1-5386-0457-1. Dostupné z: doi:10.1109/CVPR.2017.660

[7] CHEN, B. 7 popular activation functions you should know in Deep Learning and how to use them with Keras and TensorFlow 2: a practical introduction to Sigmoid, Tanh, ReLU, Leaky ReLU, PReLU, ELU, and SELU. *Towards Data Science* [online]. 2021 [cit. 2023-02-23]. Dostupné z: https://towardsdatascience.com/7-popular-activation-functions-you-should-know-in-deep-learning-and-how-to-use-them-with-keras-and-27b4d838dfe6

[8] SIVARAM, T. All You Need to Know About Skip Connections. *Analytics Vidhya* [online]. 2021 [cit. 2023-02-23]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/08/all-you-need-to-know-about-skip-connections/

[9] INTELLIPAAT, Shlok. What is PyCharm?. *Intellipaat.com* [online]. [cit. 2023-03-09]. Dostupné z: https://intellipaat.com/blog/what-is-pycharm/

[10] What is Python? Executive Summary. *Python.org* [online]. Python Software Foundation [cit. 2023-03-09]. Dostupné z: https://www.python.org/doc/essays/blurb/

[11] CALANES LUNA, Caviel. Pip Python Tutorial for Package Management. *DataCamp* [online]. DataCamp, Inc., a Delaware corporation., 2023 [cit. 2023-03-09]. Dostupné z: https://www.datacamp.com/tutorial/pip-python-package-manager

[12] YEGULALP, Serdar. What is TensorFlow? The machine learning library explained. *InfoWorld - Technology insight for the enterprise* [online]. 2022 [cit. 2023-03-09]. Dostupné z: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html

[13] Install TensorFlow 2. *TensorFlow* [online]. 2022 [cit. 2023-03-09]. Dostupné z: https://www.tensorflow.org/install

[14] Merging layers. *Keras: Deep Learning for humans* [online]. [cit. 2023-08-28]. Dostupné z: https://keras.io/about/

[15] JIA. Caffe: Deep Learning Framework. *Caffe: Convolutional Architecture for Fast Feature Embedding* [online]. Johannesburg, 2014 [cit. 2023-03-30]. Dostupné z: https://caffe.berkeleyvision.org/

[16] Caffe Deep Learning Framework and GPU Acceleration: NVIDIA Data Center. *NVIDIA* [online]. Santa Clara, California: NVIDIA Corporation [cit. 2023-03-30]. Dostupné z: https://www.nvidia.com/en-au/data-center/gpu-accelerated-applications/caffe/

[17] GONZALEZ, Rafael C. a Richard E. WOODS. *Digital image processing*. New York, NY: Pearson, [2018]. ISBN 978-013-3356-724.

[18] *GIMP: GNU Image Manipulation Program* [online]. 1997 [cit. 2023-08-07]. Dostupné z: https://www.gimp.org/

[19] Build a Validation Set With TensorFlow's Keras API. *Deeplizard: Building Collective Intelligence* [online]. [cit. 2023-08-17]. Dostupné z: https://deeplizard.com/learn/video/dzoh8cfnvnI

[20] *Torch: Scientific computing for LuaJIT* [online]. Ronan Collobert [cit. 2023-08-17]. Dostupné z: http://torch.ch/

[21] What is PyTorch? *PyImageSearch: You can master Computer Vision, Deep Learning, and OpenCV* [online]. Adrian Rosebrock, c2023 [cit. 2023-08-17]. Dostupné z: https://pyimagesearch.com/2021/07/05/what-is-pytorch/

[22] GONZALEZ, Rafael C. a Richard E. WOODS. *Digital image processing*. New York, NY: Pearson, [2018]. ISBN 978-013-3356-724.

[23] Gaussian Blur Algorithm. *PixelsTech: a place for geeks* [online]. c2023 [cit. 2023-08-20]. Dostupné z: https://www.pixelstech.net/article/1353768112-Gaussian-Blur-Algorithm

[24] What is Computer Vision? *V7: The AI Data Engine for Computer Vision* [online]. London: V7, 2018 [cit. 2023-08-24]. Dostupné z: https://www.v7labs.com/blog/what-is-computer-vision#h3

[25] Difference Between Semantic and Instance Segmentation. *Roboflow Blog* [online]. Des Moines: Roboflow, c2023 [cit. 2023-08-24]. Dostupné z: https://blog.roboflow.com/difference-semantic-segmentation-instance-segmentation/

[26] JUANXI LI, Casey. Building TensorFlow from source for SSE/AVX/FMA instructions: worth the effort? *Medium: Where good ideas find you* [online]. San Francisco: Evan Williams, 2012 [cit. 2023-08-24]. Dostupné z: https://medium.com/@sometimescasey/building-tensorflow-from-source-for-sse-avx-fma-instructions-worth-the-effort-fbda4e30eec3

[27] HUYNH, Nghi. Understanding Evaluation Metrics in Medical Image Segmentation. *Medium: Where good ideas find you* [online]. San Francisco: Evan Williams, 2012 [cit. 2023-08-24]. Dostupné z: https://medium.com/mlearning-ai/understanding-evaluation-metrics-in-medical-image-segmentation-d289a373a3f

[28] Tf.keras.preprocessing.image.ImageDataGenerator. *TensorFlow* [online]. California: Google researchers [cit. 2023-08-25]. Dostupné z: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

[29] Merging layers. *KerasKeras: Deep Learning for humans* [online]. [cit. 2023-08-28]. Dostupné z: https://keras.io/api/layers/merging_layers/

[30] How Do Convolutional Layers Work in Deep Learning Neural Networks? *Machine Learning Mastery* [online]. Victoria: Guiding Tech Media, c2023 [cit. 2023-08-28]. Dostupné z: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/